ТРУДЫ

ИНСТИТУТА СИСТЕМНОГО ПРОГРАММИРОВАНИЯ РАН

PROCEEDINGS OF THE INSTITUTE FOR SYSTEM PROGRAMMING OF THE RAS

ISSN Print 2079-8156
Том 34 Выпуск 4

ISSN Online 2220-6426
Volume 34 Issue 4

Институт системного
программирования
им. В.П. Иванникова РАН

Москва, 2022

ИСП РАН

ISSN 2079-8156 (Print), ISSN 2220-6426 (Online)

Труды Института системного программирования РАН
Proceedings of the Institute for System Programming of the RAS

Труды ИСП РАН – это издание с двойной анонимной системой рецензирования, публикующее научные статьи, относящиеся ко всем областях системного программирования, технологий программирования и вычислительной техники. Целью издания является формирование научно-информационной среды в этих областях путем публикации высококачественных статей в открытом доступе.
Издание предназначено для исследователей, студентов и аспирантов, а также практиков. Оно охватывает широкий спектр тем, включая, в частности, следующие:

- операционные системы;
- компиляторные технологии;
- базы данных и информационные системы;
- параллельные и распределенные системы;
- автоматизированная разработка программ;
- верификация, валидация и тестирование;
- статический и динамический анализ;
- защита и обеспечение безопасности ПО;
- компьютерные алгоритмы;
- искусственный интеллект.

Журнал издается по одному тому в год, шесть выпусков в каждом томе.
Поддерживается открытый доступ к содержанию издания, обеспечивая доступность результатов исследований для общественности и поддерживая глобальный обмен знаниями.
Труды ИСП РАН реферируются и/или индексируются в:

Proceedings of ISP RAS are a double-blind peer-reviewed journal publishing scientific articles in the areas of system programming, software engineering, and computer science. The journal's goal is to develop a respected network of knowledge in the mentioned above areas by publishing high quality articles on open access.
The journal is intended for researchers, students, and practitioners. It covers a wide variety of topics including (but not limited to):

- Operating Systems.
- Compiler Technology.
- Databases and Information Systems.
- Parallel and Distributed Systems.
- Software Engineering.
- Software Modeling and Design Tools.
- Verification, Validation, and Testing.
- Static and Dynamic Analysis.
- Software Safety and Security.
- Computer Algorithms.
- Artificial Intelligence.

The journal is published one volume per year, six issues in each volume.
Open access to the journal content allows to provide public access to the research results and to support global exchange of knowledge. **Proceedings of ISP RAS** is abstracted and/or indexed in:

УДК004.45

Адрес: 109004, г. Москва, ул. А. Солженицына, дом 25.
Телефон: +7(495) 912-44-25
E-mail: info-isp@ispras.ru
Сайт: http://www.ispras.ru/proceedings/

Address: 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.
Tel: +7(495) 912-44-25
E-mail: info-isp@ispras.ru
Web: http://www.ispras.ru/en/proceedings

Содержание

Table of Contents

DOI: 10.15514/ISPRAS-2022-34(4)-1

Case study: Source code static analysis for performance issues detection

[1] *A.Y. Gerasimov, ORCID: 0000-0001-9964-5850 <gerasimov.alexander@huawei.com>*
[1] *A.A. Kanakhin, ORCID: 0000-0000-0000-000 <kanakhin.alexey@huawei.com>*
[1] *P.A. Privalov, ORCID: 0000-0002-8939-5824 <petr.privalov@huawei.com>*
[2] *A.A. Zhukov, ORCID: 0000-0002-2788-4542 <andrey.zhukov@huawei-partners.com>*
[2] *E.A. Kaminsky, ORCID: 0000-0002-5040-0999 <evgeny.kaminsky1@huawei-partners.com>*

[1] *Chong-Ming Software and Technology Center, Huawei Technologies Co. Ltd.,*
17k2, Krylatskaya st., Moscow, Russia, 121614
[2] *Coleman Services,*
Bld. 2, Shchipok st. 5/7, Moscow, Russia, 155054

Abstract. Source code static analysis is widely used for program errors detection. Mostly it is used for finding critical issues like security vulnerabilities, critical program defects leading to runtime errors like crash and unexpected behavior of programs. Many SCSA tools are used for checking code conformance to different coding style guides. In this case study we present results of applying SCSA techniques for checking performance coding rules of Huawei and evaluate whether manually fixing found issues in accordance with the guidelines could impact performance, or if the compiler already applies all necessary optimizations during compilation.

Keywords: source code static analysis; program performance; compilers

For citation: Gerasimov A.Y., Kanakhin A.A., Privalov P.A., Zhukov A.A., Kaminsky E.A. Case study: source code static analysis for performance issues detection. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 4, 2022. pp. 7-20. DOI: 10.15514/ISPRAS-2022-34(4)-1

Acknowledgements. This paper presents a work of the team. Authors has done a contribution to the paper, but Cooddy development team responsible for engine and checkers implementation and ideas and code review should be pointed out: Pavel Mezhuev, Aleksey Demidov, Veronika Butkevich, Natalya Chernova, Damir Gimatdinov.

Применение статического анализа исходного кода для поиска проблем с производительностью: примеры из практики

[1] *А.Ю. Герасимов, ORCID: 0000-0001-9964-5850 <gerasimov.alexander@huawei.com>*
[1] *А.А. Канахин, ORCID: 0000-0001-9800-2722 <kanakhin.alexey@huawei.com>*
[1] *П.А. Привалов, ORCID: 0000-0002-8939-5824 <petr.privalov@huawei.com>*
[2] *А.А. Жуков, ORCID: 0000-0002-2788-4542 <andrey.zhukov@huawei-partners.com>*
[2] *Е.А. Каминский, ORCID: 0000-0002-5040-0999 <evgeny.kaminsky1@huawei-partners.com>*

[1] *ООО "Техкомпания Хуавэй",*
Россия, 121614, Москва, ул. Крылатская 17к2
[2] *Coleman Services,*
Россия, 155054, Москва, ул. Щипок 5/7, стр. 2

Аннотация. Статический анализ исходного кода программ широко используется для обнаружения ошибок. В основном он используется для обнаружения критических недостатков программ, таких как уязвимости безопасности, критических ошибок времени исполнения, таких как разрушение программы и неожиданное поведение. Многие инструменты статического анализа кода программ используются для проверки кода программ на соответствие правилам кодирования. В этой работе мы представляем результаты применения техник анализа кода программ для обнаружения ошибок производительности из руководства по программированию производительных программ компании Huawei и результаты проверки, влияет ли исправление программы в соответствии с этими правилами на результирующую производительность программ, или компилятор в состоянии автоматически оптимизировать программу.

Ключевые слова: статический анализ исходного кода; производительность программ; компиляторы

Для цитирования: Герасимов А.Ю., Канахин А.А., Привалов П.А., Жуков А.А., Каминский Е.А. Применение статического анализа исходного кода для поиска проблем с производительностью: примеры из практики. Труды ИСП РАН, том 34, вып. 4, 2022 г., стр. 7-20. 10.15514/ISPRAS-2022-34(4)-1

Благодарности: Эта статья представляет результат работы команды исследователей. Авторы статьи внесли основной вклад в её написание, но также должен учитываться вклад участников команды исследований и разработки в создание ядра и анализаторов инструмента Cooddy, инспекцию кода и идей, положенных в основу реализации инструмента: Павла Межуева, Алексея Демидова, Вероники Буткевич, Натальи Черновой и Дамира Гиматдинова.

1. Introduction

Programming languages like C and C++ are commonly used in performance-critical applications. Both of them are compiled languages—they pass through a compilation and an optimization step before being assembled into an executable binary. Early on, compilers were not proficient enough to optimize some code constructs like double checks and repeated calculations. To account for this, various coding guidelines placed the burden of this optimization on programmers, which sometimes affected code readability. Today, compiler optimization capabilities are much wider because of the evolution of their optimization algorithms and an increased performance budget for compilation. As an example, GCC, which is the most widely used C++ compiler at the time of writing, provides over 100 distinct optimization flags [1-3].

In spite of the advancements in automatic optimization, there are still cases where manual optimization is required to achieve maximum performance. In this paper, we demonstrate our program for automatic source code analysis capable of diagnosing for many of the rules from various coding guides, especially Huawei coding guidelines. Our research goal is to evaluate whether changing source code in accordance with analysis results could visibly impact performance on a

computation-heavy open source project. This evaluation takes into account that optimizations made by a modern compiler do not need to be manually implemented by the developer, and an additional goal is to review each category of issues and find out whether it is already optimizable by the compiler. As our compiler of choice we selected GCC over other compilers because it is the most widely used inside the industry.

2. Experiment

As our target for analysis we wanted to choose a C++-based, open-source, command-line application which performs a lot of calculations. Firstly, an application with a command-line interface and no graphical user interface would be easy to analyze and measure performance of. Secondly, an application performing a lot of calculations (as opposed to spending most of the time waiting for user input, sending/answering web requests, etc.) could be significantly optimized by changing the source code to reduce redundant data copies, redundant loop calculations, cache misses and other time sinks under programmer's control.

After some consideration we settled on a project called “Yosys”, a framework for Verilog RTL synthesis (i.e. synthesis of a logic circuit based on some specification of how such a circuit should operate; such specification is written at the register transfer level (RTL)) [4]. In addition to the requirements outlined above, we chose "Yosys" due to a high number of detected issues by our application.

Although 5210 issues were found, most of these were not worth analyzing in detail, either because they were caused by a common problem or because they were obvious false positives (FPs). After an initial filtering we narrowed the amount of interesting issues down to 1845. For review we have split found issues into categories based on the kind of problem they describe. Each category was assessed separately to determine whether the detected issue affects performance in a meaningful way, and if so, if it is optimized by the compiler. For comparing source code, as compiled into assembler, we used “Compiler Explorer”, a widely used web tool for inspecting results of compilation [5]. Unless specified otherwise, all ASM examples are compiled with GCC 12.1, with `-O2` optimizations.

3. Results

When describing results, each section is titled after a particular class of issue, with our internal detect class name in parenthesis for later reference.

3.1 Replace multiple if-else statements with a switch statement (RedundantMultipleIfElseChecker)

The guideline states that a tree of if-else conditions should be replaced with a switch statement where possible.

A switch statement is syntactically simpler than an equivalent if-else tree. First, using a switch ensures that each branch is taken based on an equality comparison with the same value or expression. Secondly, case values in a switch are required to be constant expressions (known at compile time), which further simplifies the optimization job for the compiler. Indeed, compilers do optimize switches better than if-else trees [6].

Project analysis found 36 defects of this type, with 100% TP rate. Unfortunately, none of the defects were located on performance-critical code paths, so fixing them will not improve application performance.

3.2 Replace access to a few arrays with the same index to one array of structs (RedundantCacheAccessChecker)

Defines the proximity of data accessed at the same time to improve data access. The idea is to put data in the same cache line and basic purpose is to find the cases where 2 or more arrays accessed with same indexes.

Specifically, it may be considered that, in the data structure, a data field that is frequently accessed is defined before, and a data field that is seldom accessed is defined after. In this way, most accesses need to be processed only once by loading the cache; otherwise, multiple times of loading are required. The idea is to encapsulate the field assignment in the extracted hotspot structure to ensure that the non-hotspot fields are assigned values at the same time to avoid omission.

The main FP sources are:

- Detect on very small arrays like 2-32 elements which fully fits in cache and have aggressive random access in cycle. In addition to caching, often such an array is used as a way to address variables by indexes instead of by names (for example a common use-case is to store X and Y coordinates as an array of two elements). Other times an array is simply a source of constant data which is likely to be optimized away completely.
- Values read from different arrays is not synchronized by index, and grouping items by structure in one array will have negative effect in performance.

Only 4 cases appear to be true positive, others cases fall into one of FP categories described above, which make it just ~2.5% TP rate.

3.3 Reorder condition sub-expressions to avoid redundant heavy-weight calculations (WeightingConditionChecker)

This checker leverages the short-circuit evaluation principle, implemented in C and C++. Short-circuit evaluation guarantees that the right-hand operand of built-in `&&` (logical AND) and `||` (logical OR) operands will not be evaluated if evaluation of the left-hand operand already determined expression result. This means that a programmer can reorder operands in if conditions so that more expensive to compute operands appear last, which will yield an increase in performance in cases where these operands are never evaluated.

The heuristic by which the approximate cost of an operator is determined is as follows. Each operand has a "cost" value, determined by the most expensive operation performed (fig. 1).

Cost	Operation
1	Literal expression
2	Variable access expression
3	Conditional expression
4	Call expression

Fig. 1. Weighting condition checker operation cost

Operands are sorted by their cost, with lower cost operands being placed first. Care is taken to avoid reordering operands which share variables, to avoid cases like reordering `nullptr` checks and pointer dereferencing. Sadly, no similar algorithm exists for preventing reordering function calls with side effects which affect each other.

Checker could be especially useful in cases where a condition with many operands is inside a loop.

Analysis of Yosys found 138 defects, with 87 of them being true positives (63% TP rate). The main sources of false positives (51 cases) are short functions inlined by the compiler. Function calls have

a cost of 4, yet the real operation inside is often much cheaper. Reordering such conditions will have no effect or sometimes an opposite effect.

82 cases contain external functions which made these cases difficult to analyze and optimize. Due to branch predictor and speculative execution such optimizations generally will have no effect.

Other cases do not lie on critical execution paths, and fixing them requires deep knowledge of the project to know with certainty that reordering conditions will produce no undesired side-effects.

3.4 Avoid redundant heap allocations (RedundantHeapAllocChecker)

The rule states that one should avoid redundant heap allocations, i.e. situations where neither manual lifetime management, nor big memory blocks are required. Frequent memory allocation and deallocation can be a serious performance issue. In some cases, a better approach might be to re-use a block of memory allocated once.

Analysis produced 67 defects, with 6 of them being true positives (9% TP rate). The main FP sources are:

- Pointer leaves the scope of the function where memory is allocated, i.e. manual memory management is, in fact, required (33/67 ≈ 49% of all cases);
- There are common recommendations not to allocate more than 16KB per function on the stack and use heap allocation instead. Our tool uses a limit of 4KB. There are cases where allocated size is constant and greater than 4KB, which can be considered a FP (12/67 ≈ 18% of all cases);
- Cases where we cannot assume the allocated memory size is FP due to insufficient knowledge of the code. (16/67 ≈ 24% of all cases).

True positive cases that we decided not to fix:

- Issues in non-performance critical code, mostly used for debugging and error messaging functions;
- Console output of statistics;
- File output dump functions;
- Dead code (functions not used by the application);
- Lookup of libraries and files by path;
- One case where a 20-byte structure was allocated on heap and was freed at the end of the function. But the defect is not on any performance critical path and has no measurable performance impact.

3.5 Avoid double checking the same value (DoubleCheckChecker)

The rule states that if a pointer validity check is performed at some point in the code and enters a “safe code block”, any subsequent checks within the safe code block are redundant. For safety and security reasons, SEI CERT C Coding Standard [7] recommends that any called function validate its parameters.

Influence of the multiple null checks on performance is questionable for two reasons. First, if both checks are visible by the compiler, it will optimize the latter check during global common subexpression elimination (available in GCC under the `-fgcse` flag). Additionally, if the check is not removed for whatever reason, branch prediction will minimize the second branch to a no-op. There are, however, several older methods of branch prediction that can create situations where an incorrect branch is taken [8]. Static prediction is the simplest branch prediction technique because it does not rely on information about the dynamic history of code executing. Instead, it predicts the

outcome of a branch based solely on the branch instruction. With static prediction all decisions are made at compile time, before the execution of the program. The early implementations of SPARC [9] and MIPS [10] always predict that a conditional jump will not be taken, so they always fetch the next sequential instruction. And this is probably even source of recommendation to handle exceptional cases inside if with return. A more advanced form of static prediction presumes that backward branches will be taken and that forward branches will not. A backward branch is one that has a target address that is lower than its own address.

GCC uses `-fdelete-null-pointer-checks` flag (commonly enabled under `-O2`) to enable global dataflow analysis that eliminates useless checks for null pointers. The optimization algorithm assumes that a `nullptr` can never be dereferenced, since dereferencing it would lead to undefined behavior under C++ Standard, and a trap in most real-world cases. This means that if a pointer has already been dereferenced, any later checks for `nullptr` can be discarded.

These optimizations make found issues irrelevant. However, such a rule can still be useful with very old versions of compiler or some specific architectures where the optimization passes like the ones described above are not available.

3.6 Avoid redundant memory zeroing (RedundantZeroMemoryChecker)

The rule states that redundant calls to `memset`, such as after allocating memory with `calloc`, should be avoided. The memory operation functions `memset/memset_s` involve system calls and have a relatively high overhead. If the memory is used to store a string, you can avoid zeroing the entire block, since the '\0' terminator prevents reading the unused tail.

Analysis found 7 defects. We classified 3 cases as FPs. These cases had no efficient fix—a zero-initialized structure, with some fields being individually initialized afterwards, was not possible to optimize.

Other 4 cases had a similar structure: a call to `calloc` followed by a call to `memset`.

Source	GCC 6.1	GCC 4.1.2

`struct SomeStruct {`
`  float a;`
`  double b;`
`  char* c;`
`  int d[10];`
`};`
 	
`SomeStruct* Create() {`
`  SomeStruct* s;`
`  s =`
`(SomeStruct*)calloc(1,`
 	
`sizeof(SomeStruct));`
`  memset(s, 0,`
 	
`sizeof(SomeStruct));`
`  return s;`
`}`
	`Create():`
	`  mov  esi, 64`
	`  mov  edi, 1`
	`  jmp  calloc`
		`Create():`
		`  sub  %rsp, 8`
		`  mov  %edi, 1`
		`  mov  %esi, 64`
		`  call  calloc`
		`  cld`
		`  mov  %rdx, %rax`
		`  mov  %ecx, 8`
		`  xor  %eax, %eax`
		`  mov  %rdi, %rdx`
		`  rep  stosq`
		`  mov  %rax, %rdx`
		`  add  %rsp, 8`
		`  ret`

Fig. 2. Optimization of memory zeroing under different versions of GCC

Additionally, this code is well optimized by a modern compiler, and only an old version of GCC 4.1.2 do not (fig. 2).

3.7 Avoid passing function arguments by value (RedundantArgCopyChecker)

This class of issues stems from a syntax feature of the C and C++ languages: when passing an argument to a function, the default operation is to take a copy of the passed value. This can cause a performance issue when big structures are copied by accident.

There are two alternatives to pass-by-copy. In C, one can modify the function signature to take a pointer (or const pointer if the value is not meant to be modified), and take an address of a value when passing it to the function. In C++, a more streamlined option is to pass a reference or const reference, which does not require changing the caller code and also disallows null values.

Of course, both of these methods involve pointer indirection, which can introduce a pessimization into the callee code. Most codestyle rules recommend passing values by copy only when they are small enough to be copied in registers. Our research found that 16 bytes is the maximum structure size which can be safely like this (fig. 3).

12 bytes structure

```
foo():
  sub    rsp, 24
  lea    rdi, [rsp+4]
  call   T::T()
  mov    rdi, QWORD PTR [rsp+4]
  mov    esi, DWORD PTR [rsp+12]
  call   bar(T)
  add    rsp, 24
  ret
```

16 bytes structure

```
foo():
  sub    rsp, 24
  mov    rdi, rsp
  call   T::T()
  mov    rdi, QWORD PTR [rsp]
  mov    rsi, QWORD PTR [rsp+8]
  call   bar(T)
  add    rsp, 24
  ret
```

20 bytes structure

```
foo():
  sub    rsp, 40
  mov    rdi, rsp
  call   T::T()
  sub    rsp, 32
  movdqa xmm0, XMMWORD PTR [rsp+32]
  mov    eax, DWORD PTR [rsp+48]
  movups XMMWORD PTR [rsp], xmm0
  mov    DWORD PTR [rsp+16], eax
  call   bar(T)
  add    rsp, 72
  ret
```

Fig. 3. Assembly generated from function foo, which creates structure of specified size and passes that structure to function bar by value

Our analysis found only a single issue, where a structure of 32 bytes was copied when passed into a print function. Since the cost of printing greatly outweighs the cost of pushing values on the stack, we consider this case not performance critical.

3.8 Avoid using RTTI (RttiChecker)

RTTI (Run-Time Type Information) is a special mechanism in the C++ language which allows the user to retrieve type information at runtime (mainly the type name), as well as traverse inheritance trees [11, sect. 17.8]. This is the mechanism behind features like `typeid` and `dynamic_cast`. This language feature, along with exceptions, has long been a polarizing discussion point [12, sect. C.146], and a notorious breaker of the "zero-overhead abstraction" rule [13], since for polymorphic classes type information now needs to be stored alongside the value, regardless of whether `dynamic_cast` is actually used or not. For this reason, RTTI is disabled in many high-performance projects [14], and the rule to disable RTTI via compiler flags is present in many C++ style guides [15, 16].

To explain the options a programmer has when avoiding RTTI, let us conduct a one-paragraph review of different kinds of polymorphism. Dynamic polymorphism, the very same that is being used by virtual inheritance in C++, is the practice of using dynamic method dispatch when calling methods of polymorphic objects. Dynamic method dispatch is called dynamic since the work of selecting an appropriate derived method happens at runtime (using what in essence is just pointers to functions) [17]. This approach does not restrict the amount of derived classes that can be created and allows derived classes to be declared in separate translation units. This means that the C++ virtual polymorphism model is openly extensible without modifying the base class, and so it is an example of open type set polymorphism (referred to later as simply open polymorphism). A polar opposite to open polymorphism is closed polymorphism, where no extension of the class hierarchy is allowed. An example of closed polymorphism is the variant data type; a variable of such type is a wrapper around one of N types specified during declaration [18, p.24]. Of course, to call variant types polymorphic we also need to implement method dispatch, and here, since we know all types in advance, we can avoid using runtime pointer indirection and just create a branch structure, which checks which type is stored inside and call its appropriate method. This logic is generated statically, and so variant-based polymorphism is an example of static polymorphism.

A general recommendation of the C++ community when replacing RTTI with other mechanisms is to implement static, closed polymorphism, as opposed to dynamic, open polymorphism that virtual methods and inheritance represent [19, p. 18][18, p. 80].

3.9 Other rules

Analysis was also performed on several other rules. Most of these deserve only a passing mention, since all detects generated by them were optimized by the compiler.

These additional rules were:

- Avoid repeated variable initialization (RedundantInitializationChecker);
- Avoid complex calculations inside the loop condition (RedundantLoopCondCalcChecker);
- Avoid repeated nested dereferencing, such as nested member variable access via a chain of pointers (RedundantAddressCalculationChecker);
- Do not mark global variables as volatile (RedundantVolatileGlobalVarChecker). Detects from this rule were not optimized, but it was impossible to determine accurately whether they were TP or FP without deep knowledge of the code.

4. Related Works

4.1 Proprietary tools

PVS-Studio, trial version is used [20].

PVS-Studio has 35 rules on performance optimization in C++. 12 of them intersected with the rules detected by Cooddy.

After running this tool on Yosys we found 128 warnings, detected by 10 rules. 81 of these warnings are true positive (FP rate of 37%). The only warning that was common with Cooddy was the one from RedundantArgCopyChecker.

4.2 Open source projects

CppCheck has 11 performance checkers that apply to CWE398, CWE597, CWE628, CWE704. Most of them are related to std::string. No rules are common with Cooddy checkers [21].

4.3 Non-commercial article-based tools

CARAMEL is a novel static technique that detects and fixes performance bugs that have non-intrusive fixes likely to be adopted by developers. Each performance bug detected by CARAMEL is associated with a loop and a condition. When the condition becomes "true" during the loop execution, all the remaining computation performed by the loop is wasted. CARAMEL analyses C/C++/Java applications [22].

Other tools are Toddler [23], Clarity [24], LDoctor [25].

5. Conclusion

We can split the code guidelines we reviewed into 4 main categories: valid and useful rules, outdated rules, rules that can be implemented in SCSA but were not possible to be properly evaluated using our method and software, and finally rules that cannot be implemented in SCSA at all.

5.1 Useful rules

Out of the rules reviewed RedundantArgCopyChecker is the only one we consider useful when working on a modern architecture and with a modern compiler. Although we did not find many issues on the “Yosys” project, it’s likely because on any project with significant attention such issues are quickly fixed when on performance-critical paths. SCSA can be used during development to more quickly spot such issues.

5.2 Outdated rules

Rules in this category are outdated in the sense that they are no longer a programmer’s burden—today’s compiler technology is advanced enough that all of the possible performance issues are optimized away. The checkers in this category are:

- RedundantMultipleIfElseChecker;
- DoubleCheckChecker;
- RedundantInitializationChecker;
- RedundantLoopCondCalcChecker;
- RedundantAddressCalculationChecker;
- RedundantZeroMemoryChecker.

5.3 Poor method

These issues are useful and it is possible to gain performance by fixing them, but the current implementation in our SCSA tool is not advanced enough to evaluate their impact properly. These issue categories are:

- RedundantCacheAccessChecker, due to a high number of false positives;
- WeightingConditionChecker, because in its current implementation it does not consider inlined functions and functions with side-effects when reordering conditional operands;
- RedundantHeapAllocChecker, due to a high number of false positives, mainly when pointers to allocated memory leave the function scope.

5.4 Not implementable in automatic SCSA

We consider two checkers unimplementable in SCSA engines in general: RedundantVolatileGlobalVarChecker and RttiChecker.

RedundantVolatileGlobalVarChecker is essentially a "code knowledge" rule. The reason for its existence is due to a very broad misuse of the volatile keyword as an erroneous way to create atomic variables [26; 27, sect. 5.1.2.3 para.2; 12, section CP.8]. The volatile keyword should only be used in very specific circumstances, and in these circumstances its necessity is a fact hidden from the compiler and known only by the programmer.

RttiChecker is not possible to implement properly because even in cases when no code uses any functionality dependent on RTTI, a compiler does not know whether the source files are compiled into an object file are linked to an object file with RTTI enabled. Some compilers do not support linking object files in such a way [28].

5.5 Final observations

The analysis performed shows that more often than not, compiler technology covers the common issues with the source code and allows the programmer to write code for readability and simplicity as a first priority. Additionally, automatic SCSA could be of use when configured to exclude classes of issues that are no longer relevant in a modern environment. Cooddy in particular was often not able to properly filter out false positives, but the approach in general is viable and further research is ought to improve Cooddy's ability to cover issues still affecting today's code. On the other hand there are some specific architectures which has no branch prediction module or sophisticated memory management units. In this case checking performance coding rules has a sense and SCSA can be helpful.

References

[1] GCC, the GNU Compiler Collection. Available at: https://web.archive.org/web/20220913154847/http://gcc.gnu.org/, accessed 2022-09-13.
[2] The State of Developer Ecosystem 2021: C++. Available at: https://web.archive.org/web/20220609172327/https://www.jetbrains.com/lp/devecosystem-2021/cpp/, accessed 2022-06-09.
[3] GCC: Options That Control Optimization. Available at: https://web.archive.org/web/20220910030424/https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html, accessed 2022-09-10.
[4] Yosys Open SYnthesis Suite. Available at: https://github.com/YosysHQ/yosys, accessed 2022-09-10.
[5] Compiler Explorer. Available at: https://godbolt.org/
[6] V. Lazarenko. From Switch Statement Down to Machine Code. Available at: https://web.archive.org/web/20220313040503/http://lazarenko.me/switch/, accessed 2022-03-13.

[7] SEI CERT C Coding standard. API00-C. Functions should validate their parameters. Available at: https://wiki.sei.cmu.edu/confluence/display/c/API00-C.+Functions+should+validate+their+parameters, accessed 2022-03-13.
[8] J.E. Smith. A study of branch prediction strategies. In Proc. of the 8th Annual Symposium on Computer Architecture, 1981, pp. 135-148.
[9] SPARC International. Available at: https://sparc.org/, accessed 2022-03-13.
[10] D. Patterson. Computer Organization and Design, Fifth Edition. Morgan Kaufmann, 2013, 800 p.
[11] ISO/IEC N4860 – Programming Language C++. [Working draft]. Available at: https://web.archive.org/web/20220901051849/https://isocpp.org/files/papers/N4860.pdf, accessed 2022-09-01.
[12] C++ Core Guidelines. Available at: https://web.archive.org/web/20220913102723/https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines, accessed 2022-09-13.
[13] B. Stroustrup. Foundations of C++. Lecture Notes in Computer Science, vol. 7211, 2013, pp. 1-25.
[14] Electronic Arts Standard Template Library. Available at: https://github.com/electronicarts/EASTL/blob/master/doc/FAQ.md#info11-what-c-language-features-does-eastl-use-eg-virtual-functions, accessed 2022-09-01.
[15] Google C++ Style Guide. Available at: https://google.github.io/styleguide/cppguide.html#Run-Time_Type_Information__RTTI_, accessed 2022-09-01.
[16] LLVM Coding Standards. Available at: https://llvm.org/docs/CodingStandards.html#do-not-use-rtti-or-exceptions, accessed 2022-09-01.
[17] S. Milton, H.W. Schmidt. Dynamic Dispatch in Object-Oriented Languages. Australian National University, TR-CS-94-02, 1994.
[18] J.R. Bandela. Polymorphism != Virtual. Flexible Runtime Polymorphysm wihtout Inheritance. In the CppCon 2019 Presentation Materials, 2019, available at: https://github.com/CppCon/CppCon2019/blob/master/Presentations/polymorphism__virtual/polymorphism__virtual__john_bandela__cppcon_2019.pdf, accessed 2022-09-01.
[19] L. Dionne. Runtime polymorphism: back to the basics. In the CppCon 2017 Presentation Materials, 2017, available at: https://github.com/CppCon/CppCon2017/blob/master/Presentations/Runtime%20Polymorphism%20-%20Back%20to%20the%20Basics/Runtime%20Polymorphism%20-%20Back%20to%20the%20Basics%20-%20Louis%20Dionne%20-%20CppCon%202017.pdf, accessed 2022-09-01.
[20] PVS-Studio. Available at: https://pvs-studio.com/en/docs/warnings/#MicroOptimizationsCPP, accessed 2022-09-01.
[21] CPPCheck. Available at: https://github.com/danmar/cppcheck, accessed 2022-09-01.
[22] A. Nistor, P-C. Chang et al. Caramel: detecting and fixing performance problems that have non-intrusive fixes. In Proc. of the 37th International Conference on Software Engineering, vol. 1, 2015, pp. 902-912
[23] A. Nistor, L. Song et al. Toddler: Detecting Performance Problems via Similar Memory-Access Patters. In Proc. of the 35th International Conference on Software Engineering (ICSE), 2013, pp. 562-571.
[24] O. Olivio, I. Dilling, C. Lin. Static detection of asymptotic performance bugs in collection traversals. In Proc. of the ACM SIGPLAN Conference on Programming Language Design and Implementation, 2016, pp. 369-378.
[25] L. Song, S. Lu. Performance Diagnostics for Inefficient Loops. In Proc. of the IEEE/ACM 39th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET), 2017, pp. 370-380.
[26] SEI CERT C Coding Standard. 3 Recommendations. Rec. 14. Concurrency (CON). CON02-C. Do not use volatile as a synchronization primitive. Available at: https://web.archive.org/web/20220916130555/https://wiki.sei.cmu.edu/confluence/display/c/CON02-C.+Do+not+use+volatile+as+a+synchronization+primitive, accessed 2022-09-16.
[27] ISO International Standard ISO/IEC 9899:201x – Programming Language C. [Working draft]. Geneva, Switzerland: International Organization for Standardization (ISO). Available at: https://web.archive.org/web/20220831233111/https://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf, accessed 2022-08-31.

[28] XL C/C++ for Linux: -qrtti, -fno-rtti. Available at: https://web.archive.org/web/20220916134348/https://www.ibm.com/docs/en/xl-c-and-cpp-linux/13.1.1?topic=descriptions-qrtti-fno-rtti-qnortti-only, accessed 2022-09-16.

Информация об авторах / Information about authors

Александр Юрьевич ГЕРАСИМОВ – кандидат физико-математических наук, руководитель группы анализа программ в Московском исследовательском центре Российского исследовательского института компании Huawei с 2020. Сфера научных интересов: автоматический анализ программ, обеспечение качества программ, компиляторные технологии, цикл разработки безопасного ПО.

Alexander Yurievich GERASIMOV – Doctor of Philosophy in Computer Sciences, Head of the Program Analysis team of Moscow Research Center of Russian Research Institute of Huawei Company from 2020. Research interests: automatic program analysis, quality assurance, compiler construction technologies, secure software development cycle.

Алексей Алексеевич КАНАХИН – кандидат физико-математических наук, ведущий инженер-исследователь в Московском исследовательском центре Российского исследовательского института компании Huawei с 2019 г. Сфера научных интересов: высокопроизводительные вычисления, архитектура компьютера, отказоустойчивость компьютерных систем, автоматический анализ программ.

Alexey Alexeyevich KANAKHIN – Doctor of Philosophy in Physics of Semiconductors, Senior Research Engineer in Moscow Research Center of Russian Research Institute of Huawei Company from 2019. Research interests: high-performance computing, computer architecture, fault-tolerant computer systems, automatic program analysis.

Петр Алексеевич ПРИВАЛОВ – магистр прикладной математики и физики, ведущий инженер группы анализа программ в Московском исследовательском центре Российского исследовательского института компании Huawei с 2020. Сфера научных интересов: автоматический анализ программ, обеспечение качества программ, компиляторные технологии, цикл разработки безопасного ПО, архитектура программ, многопоточные алгоритмы.

Petr Alekseevich PRIVALOV – Master of Science in applied mathematics and physics, Senior Software Engineer at the Program Analysis team of Moscow Research Center of Russian Research Institute of Huawei Company from 2020. Research interests: automatic program analysis, quality assurance, compiler construction technologies, secure software development cycle, program architecture and design, multithread algorithms.

Андрей Александрович ЖУКОВ – магистр по специальности "информатика и вычислительная техника", сотрудник группы анализа программ в Московском исследовательском центре Российского исследовательского института компании Huawei с 2022. Сфера научных интересов: автоматический анализ программ, компиляторные технологии, метапрограммирование, архитектура программного обеспечения.

Andrey Alexandrovich ZHUKOV – Master of Science in information and computation systems, member of the Program Analysis team of Moscow Research Center of Russian Research Institute of Huawei Company from 2022. Research interests: automatic program analysis, compiler technology, metaprogramming, software architecture.

Евгений Аркадьевич КАМИНСКИЙ – старший разработчик в команде анализа программ в Московском исследовательском центре Российского исследовательского института компании Huawei с 2022. Сфера научных интересов: автоматический анализ программ,

обеспечение качества программ, компиляторные технологии, цикл разработки безопасного ПО.

Evgenii Arkadievich KAMINSKII – senior developer of the Program Analysis team of Moscow Research Center of Russian Research Institute of Huawei Company from 2022. Research interests: automatic program analysis, quality assurance, compiler construction technologies, secure software development cycle.

DOI: 10.15514/ISPRAS-2022-34(4)-2

Подходы, направленные на повышение эффективности фаззинг-тестирования компонентов защищенной ОС

В.В. Егорова, ORCID: 0000-0003-1286-3631 <vegorova@astralinux.ru>
А.С. Панов, ORCID: 0000-0002-0046-0766 <apanov@astralinux.ru>
В.Ю. Тележников, ORCID: 0000-0002-6192-2856 <vtelezhnikov@astralinux.ru>
П.Н. Девянин, ORCID: 0000-0003-2561-794X <pdevyanin@astralinux.ru>

ООО «РусБИТех-Астра»,
117105, г. Москва, Варшавское ш., д. 26, стр.11

Аннотация. Фаззинг-тестирование в рамках цикла непрерывной разработки является необходимым инструментом, направленным в первую очередь на обеспечение доверия к разрабатываемому ПО. При этом при наличии значительных объемов кодовой базы фаззинг-тестирование становится ресурсозатратной задачей и именно поэтому важным направлением исследований является повышение эффективности фаззинг-тестирования с целью наиболее быстрого достижения интересующих участков кода без снижения качественных показателей. В рамках данной статьи рассмотрены подходы, направленные на повышение эффективности фаззинг-тестирования как для ядерных модулей, так и для ПО, входящего в пространство пользователя. С другой стороны, на отмеченных объемах программного кода инструментальные средства статического анализа выдают колоссальное количество предупреждений о возможных ошибках, при этом основные ресурсы для такого тестирования требуются не для получения результатов работы анализаторов, а для их полноценной аналитической обработки. В связи с этим, в статье значительное внимание уделяется подходу корреляции результатов статического и динамического анализа с помощью разработанного авторами инструментального средства, позволяющего в том числе реализовать направленное фаззинг-тестирование с целью подтверждения срабатываний статических анализаторов, что значительно повышает эффективность проведения тестирования компонентов защищенной ОС Astra Linux.

Ключевые слова: динамический анализ; направленное фаззинг-тестирование; фаззинг; операционная система; Astra Linux

Для цитирования: Егорова В.В., Панов А.С., Тележников В.Ю., Девянин П.Н. Подходы, направленные на повышение эффективности фаззинг-тестирования компонентов защищенной ОС. Труды ИСП РАН, том 34, вып. 4, 2022 г., стр. 21-34. 10.15514/ISPRAS-2022-34(4)-2

Approaches for improving the efficiency of protected OS components fuzzing

V.V. Egorova, ORCID: 0000-0003-1286-3631 <vegorova@astralinux.ru>
A.S. Panov, ORCID: 0000-0002-0046-0766 <apanov@astralinux.ru>
V.*Y.* Telezhnikov, *ORCID: 0000-0002-6192-2856 <vtelezhnikov@astralinux.ru>*
P.N. Devyanin, ORCID: 0000-0003-2561-794X <pdevyanin@astralinux.ru>

RusBITech-Astra
26, Varshavskoe, Moscow, 117105, Russia

Abstract. Fuzzing as a part of the continuous integration is a necessary tool, aimed primarily at the providing confidence in the software being developed. At the same time, in the presence of significant amounts of the source code, fuzzing becomes a resource-intensive task. That's why increasing the efficiency of fuzzing to reach needed code sections more quickly without reducing quality becomes an important line of research. The article deals with approaches to improve the efficiency of fuzzing both for kernel and for user-space software. On the other hand, on these amounts of program code, static code analysis produces a huge number of warnings about possible errors, and the main resources within this type of analysis are required not to obtain to result, but for analytical processing. In this regard, in the article considerable attention is paid to the approach of correlating the results of static and dynamic code analysis using the developed tool, which also allows to implement directed fuzzing in order to confirm the warnings of static analyzer, which significantly increases the efficiency of testing components of the protected OS Astra Linux.

Keywords: dynamic analysis; directed fuzzing; fuzzing; operating system; Astra Linux

For citation: Egorova V.V., Panov A.S., Telezhnikov V.Y., Devyanin P.N. Approaches for improving the efficiency of protected OS components fuzzing. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 4, 2022. pp. 21-34 (in Russian). DOI: 10.15514/ISPRAS-2022-34(4)-2

1. Введение

Согласно утвержденной ФСТЭК России «Методике выявления уязвимостей и недекларированных возможностей в программном обеспечении» [1] одним из этапов разработки безопасного ПО для обеспечения соответствия ее требованиям, начиная с минимального уровня доверия, является выполнение динамического анализа программного кода объекта оценки, в том числе с применением фаззинг-тестирования. Данный метод динамического анализа является наиболее распространенным и эффективным, благодаря чему появилось множество инструментальных средств, реализующих различные технологии тестирования, а также множество инструментов, направленных на повышение эффективности фаззинга.

При проведении тестирования операционной системы Astra Linux [2] (далее – ОС Astra Linux), разрабатываемой в ГК «Астра» и сертифицированной по наивысшему, первому уровню доверия, необходимо применять наиболее передовые методы и программные средства фаззинга. При этом важно выбирать технологии и средства анализа в соответствии с исследуемым ПО, опираясь на его функциональные особенности и специфику для достижения наибольшей эффективности фаззинг-тестирования.

Исходя из этого, объектами исследования при проведении авторами фаззинг-тестирования в первую очередь являются собственные средства защиты информации (далее – СЗИ), интерфейсы которых являются важнейшей составляющей поверхности атаки ОС Astra Linux. При этом эти СЗИ функционируют как в пользовательском пространстве, так и на уровне ядра ОС, где имеется подсистема безопасности PARSEC, реализованная в модулях ядра ОС linux-astra-modules на основе мандатной сущностно-ролевой ДП-модели управления доступом и информационными потоками (МРОСЛ ДП-модели) [3]. Помимо тестирования СЗИ в ГК «Астра» также проводится фаззинг-тестирование ПО с открытым исходным кодом,

которое играет важную роль в реализации функций защиты информации. Среди такого ПО, например, подключаемые модули аутентификации PAM, представляющие собой одну из частей стандартного механизма аутентификации UNIX-подобных систем, которые также дополнены собственными модулями принятия решений. Помимо PAM-модулей также проводится тестирование ядра ОС и его модулей безопасности, в том числе с использованием различных санитайзеров (KASAN, KCSAN и др.), что в сочетании с разработанными авторами подходами к фаззинг-тестированию позволяет обнаруживать уникальные ошибки, не отраженные в результатах тестирования ядер ОС семейства Linux в рамках других проектов – общедоступной базы syzbot от компании Google [4] и Технологического центра исследования безопасности ядра Linux [5]. С целью сравнения обнаруженных на собственных стендах ошибок с ошибками, обнаруженными в рамках syzbot используется разработанное в ГК «Астра» инструментальное средство syzStats, позволяющее с помощью графического интерфейса отсеивать уникальные «падения» ядра ОС, ранее не отраженные в syzbot.

В ходе фаззинг-тестирования авторами также используются различные технологии расширения полученного покрытия исходного кода, позволяющие одновременно сократить время выполнения фаззинг-тестирования интересующих модулей. Также апробируются технологии по корреляции результатов различных инструментальных средств, предназначенных для анализа программного кода, например, сопоставление результатов динамического и статического анализа с использованием корпусов и наборов системных вызовов, сгенерированных с помощью syzkaller [6], что в результате позволяет с помощью направленного фаззинг-тестирования достоверно подтверждать ошибки, обнаруженные статическими анализаторами.

В данной работе изложены применяемые в ГК «Астра» подходы по повышению эффективности фаззинг-тестирования с использованием как собственных инструментов, так и инструментов с открытым исходным кодом. При этом важно отметить применяемые авторами критерии, которые позволяют охарактеризовать процедуру фаззинг-тестирования как эффективную:

- количество обнаруженных ошибок на стенде фаззинг-тестирования с учетом времени работы стенда, а также количество подтвержденных срабатываний, полученных в ходе статического анализа;
- количество обнаруженных уникальных ошибок в ядре Linux, не выявленных ранее другими исследователями;
- процент покрытия кода, полученный в результате работы стенда или консолидированное покрытие на нескольких стендах, а также время его достижения;
- степень автоматизации создания, запуска и обработки результатов, полученных на стендах фаззинг-тестирования.

Статья организована следующим образом. В разд. 2 содержится обзор применяемых в ходе исследований безопасности кода инструментальных средств, как для ядерного, так и для пользовательского пространства, а также используемых при этом подходов, позволяющих не только увеличить уровень покрытия кода, но и сократить время выполнения фаззинг-тестирования интересующих частей кода. В разд. 3 излагаются подходы к фаззингу ядра ОС Astra Linux и ее подсистемы безопасности, реализованной в модулях linux-astra-modules. В разд. 4 описан новый подход к корреляции результатов статического и динамического анализа, реализованный авторами в инструменте syzCore. Разд. 5 посвящен применяемым подходам в рамках фаззинг-тестирования ПО, входящего в пространство пользователя, в том числе с использованием алгоритмов машинного обучения. Заключение завершает статью, в нем приводятся возможные направления дальнейших исследований.

2. Используемые инструментальные средства

Для реализации фаззинг-тестирования компонентов ядра ОС, как правило, могут использоваться общедоступные средства динамического анализа, среди которых наибольшее распространение получило специально предназначенное для ядер ОС программное средство syzkaller [6] – проект с открытым исходным кодом, с применением которого было выявлено множество уязвимостей в различных версиях ядра Linux. Он является наиболее перспективным и динамично развивающимся средством фаззинг-тестирования ядра ОС, драйверов и модулей. Данный инструмент имеет встроенную поддержку ядра Linux и является наиболее функциональным средством для тестирования ядра ОС, драйверов и модулей, в том числе и выгружаемых модулей linux-astra-modules, специально включаемых в состав ядра при фаззинг-тестировании, и в рамках которых функционируют механизмы защиты подсистемы безопасности PARSEC.

С целью тестирования отдельных драйверов, библиотек и приложений пользовательского пространства, в том числе входящих в подсистему безопасности PARSEC и являющихся важной составляющей поверхности атаки, в ГК «Астра» применяются положительно зарекомендовавшие себя средства динамического анализа, среди которых различные модификации фаззера American Fuzzy Lop (AFL) [7], libFuzzer [8] – инструмент, предназначенный для фаззинга библиотек, комплекс динамического анализа программ Crusher [9], разработанный ИСП РАН. С применением данных инструментальных средств авторами также создана единая автоматизированная система фаззинга приложений пользовательского пространства, которая непрерывно улучшается и адаптируется под новые условия, инструменты и подходы. Это делается с целью ускорения развертывания стендов, автоматизации фаззинг-тестирования и сбора покрытия в рамках цикла непрерывной разработки и регрессионного тестирования.

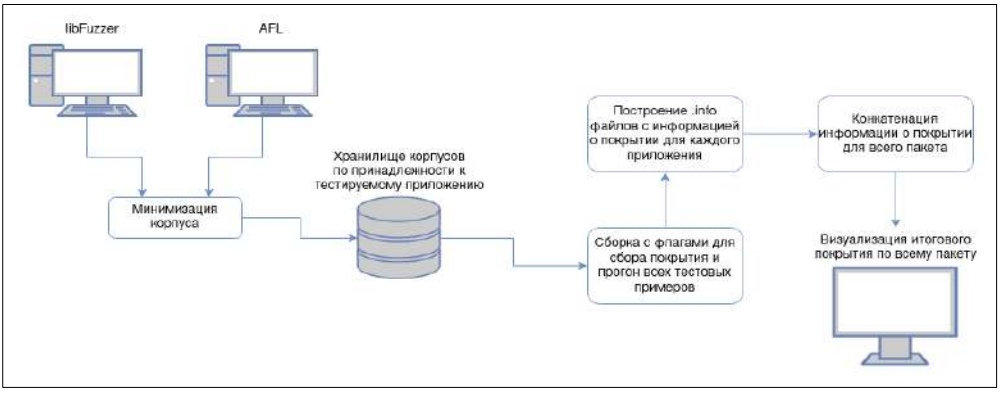


Рис. 1. Общая схема работы системы для сбора покрытия
Fig. 1. General scheme of coverage collecting system

Для сбора и анализа покрытия в ходе работы фаззеров применяется разработанная авторами система (рис.1), основанная на свободно распространяемой утилите для построения покрытия исходного кода Gcov [10] и ее расширении lcov [11]. Эта система минимизирует исходный корпус с целью снижения времени, требуемого на воспроизведение всех тестовых примеров, создаваемых инструментами фаззинг-тестирования, а после использует их для генерации результатов покрытия кода. Эти результаты сохраняются для каждого файла отдельно и в дальнейшем сливаются в один файл, демонстрирующий итоговое покрытие тестами кода всей анализируемой библиотеки или подсистемы. Покрытие кода интерпретируется от одного тестового значения к другому, чтобы определить, какие новые ветви, функции и строки покрываются фаззером с новым тестовым примером. Получение полных и исчерпывающих данных о покрытии кода в сочетании с ручным анализом помогает максимизировать эффективность фаззинга.

Таким образом, применяются различные инструменты и их комбинации, совместно с другими апробированными технологиями безопасной разработки и обеспечения доверия к ОС Astra Linux, а также проводится сбор и анализ выявленных ошибок и результаты их устранения. Весь перечисленный комплекс инструментальных средств, вместе с инструментами, предназначенными для других видов анализа и верификации объединяется в единый комплекс – стенд доверия (рис. 2), развернутый в рамках практики непрерывной интеграции в инструментальной среде GitLab [12] с целью автоматизации процессов сборки, тестирования и верификации программного кода в соответствии с требованиями нормативных документов [1, 12].

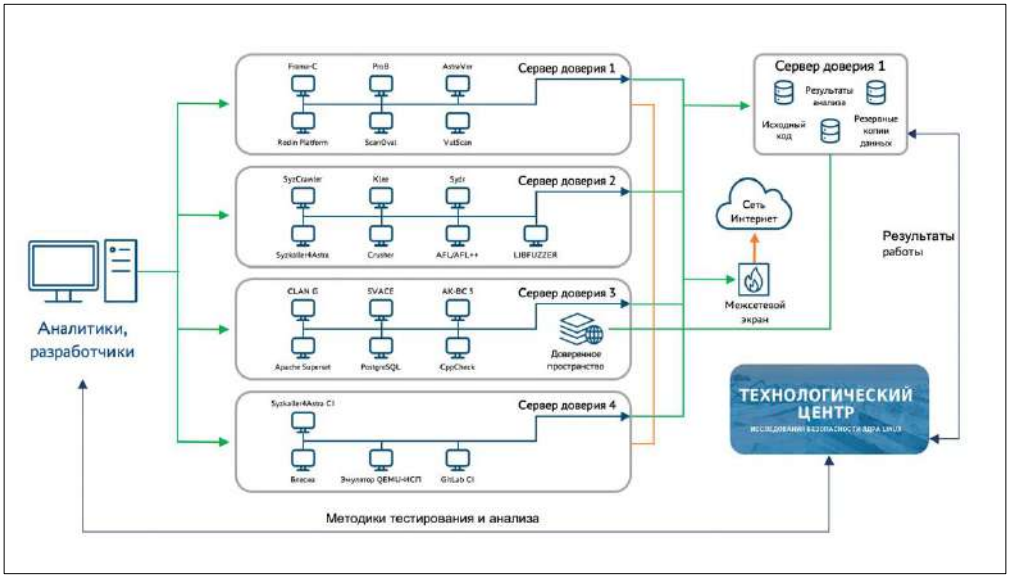


Рис.2. Масштабируемая структура стенда доверия
Fig. 2. Scalable structure of the Trust Bench

3. Фаззинг ядра ОС

Инструмент syzkaller обладает достаточно широкими возможностями для динамического анализа ядра ОС, драйверов и модулей и реализует эффективные алгоритмы фаззинг-тестирования: генерационные, для составления начальных тестовых данных на основе предоставленных ему описаний системных вызовов, мутационные для генерации новых тестовых значений с целью достижения большего покрытия, генетические для отбора наиболее подходящих экземпляров набора сгенерированных корпусов входных данных – расширяющих покрытие или чаще приводящих к падениям.

С учетом специфики ОС на основе фаззера syzkaller в ГК «Астра» разработан комплекс непрерывного фаззинг-тестирования syzkaller4astra, обеспечивающий встраивание в тестируемое ядро необходимых модулей подсистемы безопасности PARSEC, и их настройку, а также включающий в себя все созданные в компании инструментальные средства, направленные на повышение эффективности фаззинг-тестирования. На рис. 3 изображена обобщенная схема функционирования стендов динамического анализа кода syzkaller4astra.

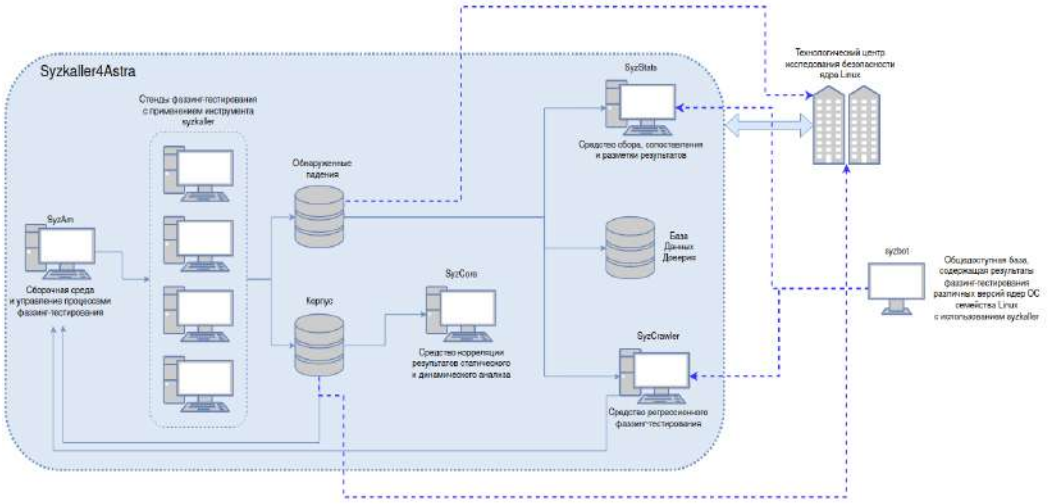


Рис.3. Динамический анализ кода ОС Astra Linux с использованием стенда syzkaller4astra
Fig. 3. Dynamic analysis of the OS Astra Linux code using the syzkaller4astra bench

При использовании данного инструмента авторами применяются следующие подходы, позволяющие увеличить уровень покрытия кода, одновременно сократив время выполнения фаззинг-тестирования в первую очередь для подсистемы безопасности PARSEC:

- ограничение тестируемых системных вызовов ядра ОС с целью ускорения процесса фаззинг-тестирования и увеличения покрытия программного кода linux-astra-modules, для чего было разработано специальное автоматизированное средство сбора и анализа статистики базового покрытия в общем и по каждому системному вызову в частности, а также для отслеживания динамики изменения покрытия по каждому системному вызову с течением времени;
- уточнение существующих и описание новых прототипов системных вызовов, что позволяет увеличить покрытие исходного кода при фаззинг-тестировании и выполнить направленный фаззинг для сокращения времени тестирования модулей подсистемы безопасности PARSEC, интерфейсы которой являются важнейшей составляющей поверхности атаки к ОС Astra Linux;
- с целью использования результатов фаззинг-тестирования ядер ОС семейства Linux, полученных сторонними организациями, авторами был написан инструмент syzcrawler, который осуществляет сбор и обработку данных из общедоступной базы syzbot [4], содержащей результаты фаззинг-тестирования различных версий ядер ОС семейства Linux с применением инструментального средства syzkaller, и данных, полученных в ходе взаимодействия с Технологическим центром исследования безопасности ядра Linux [5], после чего эти данные тестируются на ядре ОС Astra Linux, а корпусы, содержащие последовательность системных вызовов и позволяющие увеличить покрыте кода, добавляются к собственным корпусам и используются на стендах syzkaller4astra;
- регрессионное фаззинг-тестирование, также реализованное в рамках проекта syzcrawler, позволяющее обеспечить автоматизированное тестирование обнаруженных ранее падений на стендах фаззинга на доработанных версиях ядра ОС с целью подтверждения их устранения;
- обнаружение аналогичных уже найденным ошибок по шаблонам, что позволяет находить типовые ошибки, не дожидаясь их выявления с помощью инструментов динамического

анализа;

- динамический анализ модулей ядра ОС при имитации исключительных ситуаций, например, нехватки системных ресурсов, с помощью инструментального средства KEDR, разработанного ИСП РАН [14];
- корреляция результатов статического и динамического анализа с помощью разработанной авторами автоматизированной системы syzcore, что позволяет подтверждать наличие ошибки, обнаруженной статическим анализатором;
- сбор покрытия модулей ядра одновременно с фаззинг-тестированием модулей защиты, входящих в пространство пользователя, что позволяет получать более полную информацию о покрытии подсистемы безопасности PARSEC в пространстве ядра во время выполнения программ в пространстве пользователя, и, как следствие, повышать эффективность фаззинг-тестирования.

С целью упрощения анализа обнаруженных на стендах syzkaller4astra ошибок авторами разработан инструмент syzStats, позволяющий автоматически сопоставлять найденные на стендах syzkaller4astra ошибки с ошибками, содержащимися в общедоступной базе syzbot, что дает возможность быстрее обнаруживать уникальные ошибки, ранее не выявленные другими исследователями.

Для применения инструментального средства syzkaller в рамках цикла непрерывной разработки авторами развивается автоматизированная система для фаззинг-тестирования ядер ОС – syzAm, которая позволяет упростить и ускорить развертывание стендов, предоставляет балансировку нагрузки серверов, консолидированную информацию о производительности стендов и покрытии кода тестами, автоматически генерирует файлы-конфигураций. Помимо этого, в данную автоматизированную систему включены инструменты тестирования syzcrawler и сопоставления ошибок syzStats.

Для администрирования данных процессов разработана панель управления стендами фаззинга ядра ОС, которая позволяет не только централизовано хранить и просматривать информацию со всех стендов тестирования, отслеживать покрытие кода, анализировать обнаруженные ошибки и их уникальность относительно общедоступной базы данных syzbot, но и контролировать их работу, запуская регрессионное тестирование, а также создавая дополнительные и управляя уже существующими стендами фаззинга.

Рис. 4. Обобщенная статистика стендов фаззинга, построенная с помощью Grafana и Prometheus
Fig. 4. General fuzzing benches stats built using Grafana and Prometheus

Рис. 5. Статистика стендов фаззинга, построенная с помощью Grafana и Prometheus
Fig. 5. Fuzzing benches stats built using Grafana and Prometheus

Для сбора статистических данных по всем стендам фаззинг-тестирования в этой автоматизированной системе используется набор инструментов для сбора метрик Prometheus [15] с открытым исходным кодом, позволяющий собирать и хранить метрики в виде временных рядов. На рис. 4 и 5 приведены примеры построения статистики с помощью

Prometheus, интегрированной с Grafana [16]. Применение монитора состояния фаззинг-тестирования позволяет своевременно выполнять балансировку нагрузки, задавать и отслеживать множественные критерии определения эффективности работы стендов, осуществлять своевременную выгрузку и реагирование на полученные результаты тестирования.

В результате применения описанных подходов удалось повысить эффективность фаззинг-тестирования. В частности, покрытие по базовым блокам для ядерных модулей подсистемы безопасности PARSEC превышает 60% (более 95% для монитора управления доступом), а скорость его достижения в результате применения описанных подходов сократилась вдвое, что, несомненно, влечет за собой значительное сокращение накладных расходов ресурсов серверов стенда доверия. При этом также важны и качественные показатели: в ходе применения предложенных подходов обнаруживаются и устраняются ошибки в PARSEC, а также уникальные ошибки для других компонентов ядра ОС Linux, которые не были отражены в syzbot, а благодаря применению автоматизированной системы регрессионного тестирования и системе обнаружения аналогичных уже существующим падениям по шаблонам удалось также сократить время на устранение и подтверждение устранения обнаруженных ранее ошибок.

4. Корреляция результатов статического и динамического анализа

При наличии значительных объемов кодовой базы ручная разметка кода и ручной анализ потенциальных ошибок, обнаруженных статическим анализатором, является затруднительным в первую очередь из-за большого количества срабатываний (предупреждений), в том числе имеющих высокий уровень критичности. На отмеченных объемах программного кода применяемые инструментальные средства выдают тысячи, а иногда, десятки тысяч предупреждений о возможных ошибках, большая часть из которых, как правило, оказываются ложными или не учитывающими специфику анализируемого кода, что наиболее актуально для модулей или драйверов ядра ОС [17]. В связи с этим, одним из важнейших направлений научно-исследовательской деятельности в ГК «Астра» является решение задачи разработки инструментальных средств, автоматизирующих обработку этих предупреждений. В рамках этих исследований авторами была выдвинута идея, что в качестве метода подтверждения обнаруженных статическим анализатором ошибок возможно использовать направленное фаззинг-тестирование.

Эта идея основывается на следующем предположении: если в ходе работы syzkaller было получено покрытие потенциально опасной функции (т.е. функции, в которой с помощью статического анализа были обнаружены ошибки высокого уровня критичности), то возможно ограничить тестируемые системные вызовы до тех, что позволили получить покрытие этой функции, а корпус до тех тестовых примеров, которые позволили достигнуть интересующих участков кода. При этом участки кода, являющиеся в данном случае «интересными», могут быть вручную выбраны и расширены исследователем. В случае, если syzkaller не удалось получить покрытие на тех участках, где статический анализатор обнаружил ошибку, подбор направленных системных вызовов остается на усмотрение исследователя.

При этом важно отметить, что предложенный подход не позволяет достоверно подтвердить отсутствие ошибки, обнаруженной статическим анализатором. Несмотря на это, среди большого количества срабатываний статического анализатора для тех ошибок, для которых был организован направленный фаззинг, но которые не были подтверждены на стендах фаззинг-тестирования, снижается уровень приоритета. Это означает, что при ручной разметке аналитик в первую очередь обратит внимание на те предупреждения, для которых либо еще не было предпринято попытки подтверждения с помощью направленного фаззинг-тестирования, либо на те, которые уже были обнаружены (подтверждены) на стенде фаззинг-

тестирования. Такой подход позволяет эффективно распределять ресурсы для анализа и устранения обнаруженных ошибок.

Для внедрения описанного подхода в масштабируемую автоматизированную систему верификации и анализа кода, поддерживаемую в ГК «Астра» на базе стенда доверия, авторами было написано инструментальное средство syzcore, общая схема функционирования которого отражена на рис. 6.

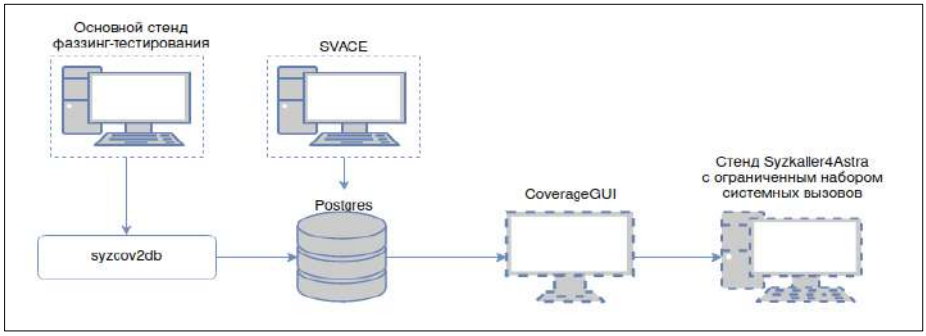


Рис. 6. Общая схема функционирования инструмента syzcore
Fig. 6. General scheme of the sycore tool

На текущем этапе этим инструментальным средством осуществляется сопоставление покрытия, полученного в результате фаззинг-тестирования ядра ОС с помощью syzkaller, и результатов работы статического анализатора SVACE [18], разработанного ИСП РАН.

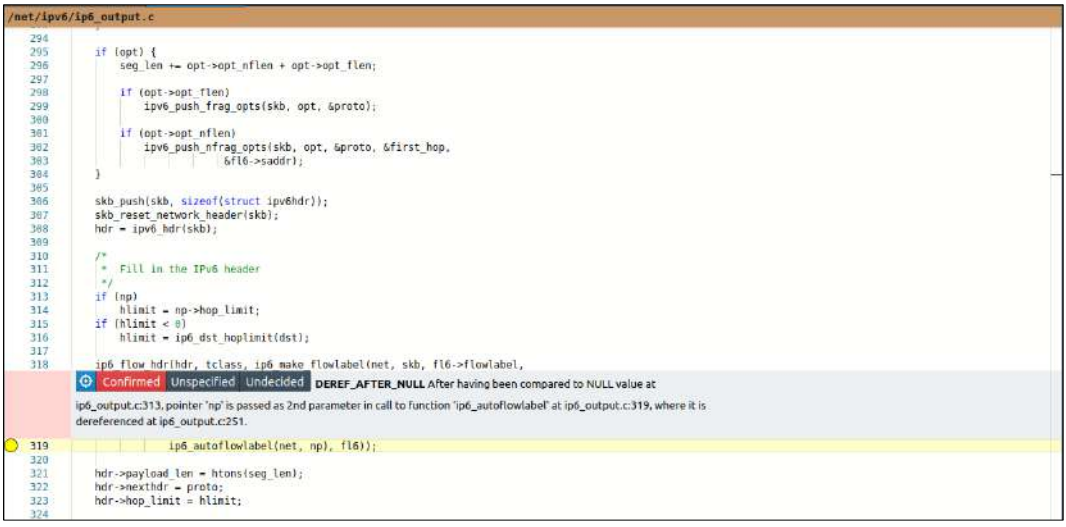


Рис. 7. Пример срабатывания SVACE, обнаруженного в рамках работы Технологического центра исследования безопасности ядра Linux
Fig. 7. An example of error, detected using SVACE as a part of the work of The Linux Kernel Security Technology Research Center

Для сбора покрытия ядра syzkaller использует kcov, а само покрытие представляется с помощью добавления на этапе инструментации вызова __sanitizer_cov_trace_pc в каждый базовый блок, сгенерированный во время процесса сборки. В ходе апробации различных подходов к выгрузке покрытия и информации со стенда syzkaller4astra был выбран способ получения «сырых» адресов, так как в условиях большого объема кода такой способ является наиболее эффективным. Далее с помощью addr2line и объектного файла ядра vmlinux полученные данные возможно преобразовать в строки исходных файлов. SVACE, в свою

очередь, предоставляет информацию о принадлежности ошибки к конкретной функции в файле.

С помощью графической визуализации (рис. 7), можно проанализировать совместное покрытие двух инструментов, а также тестовые примеры и системные вызовы, зашедшие в ходе фаззинг-тестирования в то или иное ветвление. В результате, после экспертного анализа пересечений обнаруженных SVACE ошибок и полученных данных о покрытии syzkaller создаются новые стенды фаззинг-тестирования с усеченными корпусами и системными вызовами.

Предупреждение, сформированное статическим анализатором SVACE и отраженное на рис. 7 (строка 319), было подтверждено с помощью направленного фаззинга инструментом syzkaller (рис. 8). С целью подтверждения системные вызовы были ограничены только вызовами, вошедшими в функцию с предупреждением, также был собран усеченный корпус, состоящий только из релевантных тестовых примеров.

Рис. 8. Подтверждение ошибки, обнаруженной SVACE, с помощью syzkaller
Fig. 8. Confirming an error detected by SVACE with sykaller

Таким образом, на текущем этапе разработан инструмент, позволяющий сопоставлять результаты, полученные в ходе статического анализа, с покрытием, полученным со стендов фаззинг тестирования и системными вызовами, позволяющими попасть в тот или иной участок кода. При этом благодаря автоматизированным технологиям по корреляции результатов статического и динамического анализа успешно подтверждаются срабатывания статических анализаторов, и на очередной итерации исследования кода модулей безопасности ядра удалось подтвердить 5 новых срабатываний. При этом благодаря применяемым подходам направленного фаззинг-тестирования, количество воспроизводимых ошибок неуклонно увеличивается.

5. Фаззинг программного обеспечения пользовательского пространства

В ходе динамического анализа приложений пользовательского пространства, в том числе утилит и библиотек, входящих в подсистему безопасности PARSEC, являющуюся важнейшей составляющей поверхности атаки из пространства пользователя, для них были авторами написаны специальные «программы-обертки», которые получают на вход данные, сгенерированные фаззером, и передают их в целевую программу или конкретную функцию. При этом важным фактором для обертки является возможность принимать и обрабатывать

любые искаженные значения, создавать и изменять используемые в ходе работы программ файлы. Также для достижения эффективности фаззинг-тестирования важно, чтобы обертка имела узкую цель, по этой причине одной библиотеке часто соответствует несколько оберток, каждая из которых отвечает за анализ своего модуля. Помимо индивидуальных оберток для каждой программы с помощью системы автоматической генерации словарей создаются свои словари, при этом в словарь включаются флаги программ, длинные текстовые и числовые значения, и другие константы, сравнение с которыми содержится в исходном коде программы и подбор которых в ходе фаззинг-тестирования займет длительное время. При этом для сложных программ, работающих, например, с анализом содержимого объемных файлов, составленные словари авторами просматриваются вручную. Это связано с большим количеством сравнений в исходном коде программы, так как включение в словарь задействованных при этом данных в конечном итоге снизят эффективность от использования словаря.

С целью повышения эффективности фаззинг-тестирования приложений пользовательского пространства применяется подход, основанный на имитации исключительных ситуаций, направленный в первую очередь на то, чтобы покрыть те пути исполнения, которые возникают в таких ситуациях, например, при нехватке системных ресурсов, когда запрошенная программой память не выделилась из-за ее недостатка или какой-либо ошибки. Данный подход позволяет не только обнаружить ошибки, которые могут возникнуть в исключительной ситуации, но и расширить покрытие по ветвям, обрабатывающим данную исключительную ситуацию. Также такой подход используется для фаззинг-тестирования утилит и библиотек, входящих в подсистему безопасности PARSEC, работающих с системными файлами, и позволяет обработать случаи, когда файл был намеренно изменен или удален. В результате, применение данного подхода позволило увеличить покрытие утилит и библиотек PARSEC, входящих в пространство пользователя, в среднем на 10%.

Существует ряд готовых наборов инструментов, созданных с целью отслеживания и автоматизации процессов сборки и анализа кода с применением фаззинг-тестирования, среди них, например, набор утилит для аудита безопасности приложений BugBane [19]. Однако с учетом особенностей сборки и функционирования собственных СЗИ, а также необходимости создания и внедрения систем отслеживания состояния серверов, функционирующих в рамках стенда доверия, балансировки их нагрузки и потребности в удаленном администрировании всех стендов фаззинг-тестирования, возникает необходимость создания собственной автоматизированной системы, удовлетворяющей всем требованиям, предъявляемым к подобной системе в организации. Таким образом, для обеспечения применимости перечисленных инструментов для фаззинг-тестирования приложений пользовательского пространства в рамках цикла непрерывной разработки безопасного программного обеспечения в ГК «Астра» реализован проект по автоматизации процессов фаззинг-тестирования приложений пользовательского пространства, что обеспечивает целевую сборку анализируемого ПО сразу после внесения изменений разработчиками, генерацию словарей, распределение стендов тестирования по серверам. Аналогично с автоматизированной системой фаззинга ядра ОС, для фаззинга приложений пользовательского пространства разработана отдельная панель для управления стендами фаззинга, позволяющая отслеживать нагрузку на сервера, хранить и просматривать информацию со всех стендов, контролировать их работу, создавать новые стенды фаззинга и управлять уже существующими.

Процент покрытия, полученный в результате применения описанных подходов составляет свыше 85% по строкам для утилит и библиотек пользовательского пространства, входящих в подсистему безопасности PARSEC. Полученные результаты позволяют охарактеризовать процедуру фаззинг-тестирования как эффективную.

Кроме того, существует ряд исследований [20-22], подтверждающих эффективность применения алгоритмов машинного обучения в рамках фаззинг-тестирования, а также представляют ряд задач фаззинг-тестирования, которые могут решаться с использованием алгоритмов машинного обучения среди них [23]: генерация исходного тестового примера, создание тестового набора и его фильтрация, выбор эффективных мутаций и др. В конечном итоге решение этих задач непосредственно повлияет на покрытие кода и позволит в целом повысить результативность фаззинг-тестирования. Качество исходного тестового примера непосредственно влияет на результаты фаззинг-тестирования, а использование алгоритмов машинного обучения позволяет исследовать общие закономерности в тестируемых файлах и выделить те, которые приводят к большему покрытию исходного кода и увеличению количества обнаруженных ошибок [24]. Правильная генерация тестового корпуса напрямую влияет на результативность фаззинга и иногда только от него зависит, будет ли обнаружена та или иная ошибка. Фильтрация, в свою очередь, позволяет выделить среди всего (иногда достаточно объемного) корпуса минимальный набор тестовых примеров, позволяющий с наибольшей вероятностью обнаружить новые пути исполнения программы, что в конечном итоге не только расширит покрытие, но и может повысить вероятность обнаружение ранее не обнаруженной ошибки в коде исследуемой программы [25, 26]. Выбор эффективных мутаций в фаззинге также является важной задачей, так как влияние мутаторов в AFL не одинаково и их распределение напрямую зависит от тестируемой программы [27].

В рамках исследований, проводимых в ГК «Астра», проходят апробацию подходы по применению машинного обучения для создания и фильтрации тестового набора, а также для генерации исходного тестового примера с целью повышения результативности фаззинг-тестирования.

6. Заключение

Совокупность описанных подходов и реализующих их инструментальных средств, применяемых авторами как к ядру ОС и встроенным в него модулям подсистемы безопасности PARSEC, так и к приложениям пользовательского пространства, позволяет минимизировать участие аналитика в настройке и запуске стендов фаззинг-тестирования, а также сократить время, затрачиваемое на анализ обнаруженных срабатываний анализаторов. Предварительные результаты апробации говорят о корреляции между собой не менее 30% результатов, полученных в результате применения двух методов анализа кода (статического и динамического). Разработанное в ГК «Астра» инструментальное средство корреляции позволяет не только эффективно расставлять приоритеты при ручной разметке обнаруженных срабатываний, но и в конечном итоге повысить результативность проведения тестирования компонентов защищенной ОС Astra Linux. Помимо этого, важным критерием эффективности также является время выявления ошибок, которое в результате применения всех описанных подходов сократилось втрое по сравнению с классическим методом фаззинг-тестирования, как для пользовательского пространства, по имеющимся ранее оберткам, так и для модулей ядра ОС, благодаря направленному фаззинг-тестированию, позволяющему реализовывать целевое воспроизведение ошибок. В итоге за счет применения перечисленных подходов общее покрытие возросло на 15%, а время его достижение сократилось вдвое. А, значит, совокупность всех применяемых подходов обеспечивает повышение эффективности тестирования и большую уверенность в качестве и достоверности полученных результатов. Таким образом, предложенные подходы к фаззинг-тестированию позволяют обеспечивать минимизацию ошибок в коде СЗИ на протяжении всего жизненного цикла разработки, устраняя большинство программных ошибок на его ранних этапах, что создает условия для разработки безопасного системного ПО на базе ядра Linux и обеспечивает высокий уровень доверия непосредственно к ОС Astra Linux.

В качестве направлений дальнейших исследований наиболее приоритетными являются:

- оценка результативности подхода корреляции результатов статических и динамических анализаторов для анализа ПО, входящего в пользовательское пространство, в том числе СЗИ, реализуемых подсистемой безопасности PARSEC;
- создание инструмента для автоматизированного уточнения системных вызовов, что позволит быстрее расширять покрытие с целью достижения интересующих участков кода;
- построение графа вызовов ядерных функций, позволяющих достигать интересующих частей кода и сопоставлять их с системными вызовами.

Список литературы / References

[1] Информационное сообщение ФСТЭК России от 10.02.2021 № 240/24/ / Informational message of FSTEK Russia of 10th February 2021 #240/24/647. Available at: https://fstec.ru/normotvorcheskaya/informatsionnye-i-analiticheskie-materialy/2171-informatsionnoe-soobshchenie-fstek-rossii-ot-10-fevralya-2021-g-n-240-24-647, accessed 12.09.2022 (in Russian).

[2] Девянин П.Н., Тележников В.Ю., Третьяков С.В. Основы безопасности операционной системы Astra Linux Special Edition. Управление доступом. Учебное пособие. М., Горячая линия – Телеком, 2022 г., 148 стр. / Devyanin P.N., Telezhnikov V.Y., Tret'yakov S.V. Astra Linux Special Edition security basics. Access control. Hotline-Telecom, 2022, 148 p. (in Russian).

[3] Девянин П.Н. Модели безопасности компьютерных систем. Управление доступом и информационными потоками. Учебное пособие для вузов. М., Горячая линия – Телеком, 2020 г., 352 стр. / Devyanin P.N. Security models of computer systems. Control for access and information flows. Hotline-Telecom, 2020, 352 p. (in Russian).

[4] Проект syzbot / Syzbot project. Available at: https://syzkaller.appspot.com, accessed 12.09.2022.

[5] Технологический центр исследования безопасности ядра Linux / The Linux Kernel Security Technology Research Center. Available at: https://portal.linuxtesting.ru, accessed 12.09.2022 (in Russian).

[6] Syzkaller / Syzkaller. Available at: https://github.com/google/syzkaller, accessed 12.09.2022.

[7] Инструментальное средство фаззинг-тестирования American Fuzzy Lop / American Fuzzy Lop. Available at: https://github.com/google/AFL, accessed 12.09.2022.

[8] Инструментальное средство фаззинг-тестирования libFuzzer / libFuzzer. Available at: https://llvm.org/docs/LibFuzzer.html, accessed 12.09.2022.

[9] Комплекс динамического анализа программ Crusher / ISP Crusher: a dynamic analysis toolset. Available at: https://www.ispras.ru/en/technologies/crusher/, accessed 12.09.2022 (in Russian).

[10] Инструментальное средство сбора и подсчета покрытия Gcov / Gcov: source code coverage analysis and statement-by-statement profiling tool. Available at: https://gcc.gnu.org/onlinedocs/gcc-4.5.2/gcc/Gcov.html, accessed 12.09.2022.

[11] Инструментальное средство визуализации покрытия Lcov / Lcov: graphical front-end for Gcov. Available at: https://github.com/linux-test-project/lcov, accessed 12.09.2022.

[12] GitLab. Веб-инструмент жизненного цикла DevOps / GitLab. A full DevOps tool. Available at: http://gitlab.com, accessed 12.09.2022.

[13] ГОСТ Р 56939-2016 «Защита информации. Разработка безопасного программного обеспечения. Общие требования» / GOST R 56939-2016 «Information protection. Secure Software Development. General requirements». Federal Agency for Technical Regulation and Metrology, 2016 (in Russian).

[14] KEDR. Расширяемая система для динамического анализа модулей ядра Linux / KEDR. An extensible system for dynamic analysis and verification Linux kernel modules. Available at: https://github.com/euspectre/kedr, accessed 12.09.2022.

[15] Набор инструментов для мониторинга Prometheus / Prometheus. An open source monitoring system. Available at: https://prometheus.io, accessed 12.09.2022.

[16] Платформа для визуализации данных Grafana / Grafana. Multi-platform open-source analytics and interactive visualization web application. Available at https://grafana.com, accessed 12.09.2022.

[17] Девянин П.Н., Хорошилов А.В., Тележников В.Ю. Формирование методологии разработки безопасного системного программного обеспечения на примере операционных систем. Труды ИСП РАН, том 33, вып. 5, 2021 г., стр. 25-40 / Devyanin P.N., Telezhnikov V.Y., Khoroshilov V.V. Building a methodology for secure system software development on the example of operating systems. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 5, 2021, pp. 25-40 (in Russian). DOI: 10.15514/ISPRAS–2021–33(5)–2.

[18] Статический анализатор Svace / SVACE static analyzer. Available at: http://www.ispras.ru/technologies/svace, accessed 12.09.2022 (in Russian).
[19] Набор утилит для аудита безопасности приложений BugBane / BugBane. A set of utilites for auditing application security. Available at: https://github.com/gardatech/bugbane, accessed 12.09.2022 (in Russian).
[20] Cheng L., Zhang Y. et al. Optimizing seed inputs in fuzzing with machine learning. In Proc. of the 41st International Conference on Software Engineering: Companion Proceedings, 2019, pp. 244-245.
[21] Cummins C., Petoumenos P. et al. Compiler fuzzing through deep learning. In Proc. of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, 2018, pp. 95-105.
[22] Godefroid P., Peleg H., Singh R.. Learn&Fuzz: Machine learning for input fuzzing. In Proc. of the 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), 2017, pp. 50-59.
[23] Wang Y., Jia P. et al. A systematic review of fuzzing based on machine learning techniques. PLoS ONE, vol. 15, issue 8, 2020, article no. e0237749, 37 p.
[24] Lyu C., Ji S. et al. SmartSeed: Smart Seed Generation for Efficient Fuzzing. arXiv preprint arXiv:1807.02606, 2018, 17 p.
[25] Gong W, Zhang G, Zhou X. Learn to Accelerate Identifying New Test Cases in Fuzzing. Lecture Notes in Computer Science, vol. 10656, 2017, pp. 298-307.
[26] Rajpal M, Blum W, Singh R. Not all bytes are equal: Neural byte sieve for fuzzing. arXiv preprint arXiv: 1711.04596, 2017, pp. 1-10.
[27] Lyu C., Ji S. et al. MOPT: Optimized Mutation Scheduling for Fuzzers. In Proc. of the 28th USENIX Security Symposium (USENIX Security 19), 2019, pp. 1949–1966.

Информация об авторах / Information about authors

Виктория Вячеславовна ЕГОРОВА – старший научный сотрудник. Область интересов: анализ программ, динамический анализ, фаззинг.

Victoriia Vyacheslavovna EGOROVA – senior researcher. Field of Interest: program analysis, dynamic analysis, fuzzing.

Алексей Сергеевич ПАНОВ – научный сотрудник. Область интересов: поиск уязвимостей в ПО, анализ защищенности ИС.

Alexey Sergeevich PANOV – researcher. Field of Interest: Vulnerability research, penetration testing.

Владимир Юрьевич ТЕЛЕЖНИКОВ – кандидат технических наук, начальник отдела научных исследований. Область интересов: формальные модели безопасности компьютерных систем, методы статического и динамического анализа программного кода, операционная система Linux.

Vladimir Yurevich TELEZHNIKOV, Ph.D in Technical Science, Head of Research Department. Field of Interest: formal security models of computer systems, methods of static and dynamic analysis of program code, Linux operating system.

Петр Николаевич ДЕВЯНИН – член-корреспондент Академии криптографии России, доктор технических наук, профессор, научный руководитель ООО «РусБИТех-Астра» (ГК «Астра»). Область интересов: теория информационной безопасности, формальные модели безопасности компьютерных систем.

Petr Nikolaevich DEVYANIN – Doctor of Technical Sciences, Corresponding Member of Russian Academy of Cryptography, Professor, Scientific Director in RusBITech-Astra (Astra Linux). Field of Interest: information security theory, formal security models of computer systems.

DOI: 10.15514/ISPRAS-2022-34(4)-3

Инструмент динамического анализа IoT-систем ELF с поддержкой символьных вычислений

Р.Д. Коваленко, ORCID: 0000-0002-2225-5560 <r.kovalenko@ispras.ru>
А.Н. Макаров, ORCID: 0000-0003-2237-3396 <alex1118@ispras.ru>
Институт системного программирования им. В.П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. В результате работы по направлению анализа IoT-устройств авторами был создан инструмент ELF (embedded linux fuzz), который, в частности, предоставляет функционал для применения существующих средств динамического анализа в работе с различными IoT-устройствами. В статье рассматриваются вопросы применения символьного выполнения для анализа IoT-систем, построенных на базе ядер Linux, описывается способ интеграции одного из популярных фреймворков полносистемного символьного выполнения S2E в среду инструмента ELF, а также применимость полученной связки инструментов к реализации распределенного гибридного фаззинга IoT-устройств.

Ключевые слова: фаззинг; символьное выполнение; IoT-устройство; Linux

Для цитирования: Коваленко Р.Д., Макаров А.Н. Инструмент динамического анализа IoT-систем ELF с поддержкой символьных вычислений. Труды ИСП РАН, том 34, вып. 4, 2022 г., стр. 35-48. 10.15514/ISPRAS-2022-34(4)-3

ELF dynamic analysis tool for IoT systems with symbolic execution

R.D. Kovalenko, ORCID: 0000-0002-2225-5560 <r.kovalenko@ispras.ru>
A.N. Makarov, ORCID: 0000-0003-2237-3396 <alex1118@ispras.ru>
Ivannikov Institute for System Programming of the RAS,
25, Alexander Solzhenitsyn st., Moscow,109004, Russia

Abstract. As a result of background work on analysis in embedded Linux OS, the authors created the ELF (embedded linux fuzzing) tool that provides functionality for use in conventional dynamic analysis tools working with IoT devices. The article discusses the use of full-system symbolic execution for the analysis of IoT systems based on Linux kernels, describes how to integrate S2E full-system symbolic execution frameworks into the ELF tool environment, as well as the possibility of applicability of the resulting toolchain to the implementation of distributed hybrid IoT fuzzing.

Keywords: fuzzing; symbolic execution; IoT-device; Linux

For citation: Kovalenko R.D., Makarov A.N. ELF dynamic analysis tool for IoT systems with symbolic execution support. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 4, 2022. pp. 35-48 (in Russian). DOI: 10.15514/ISPRAS-2022-34(4)-3

1. Введение

Процесс проведения анализа IoT-устройств зачастую обладает высокой трудоемкостью в силу ряда особенностей реализации и функционирования таких устройств. Архитектура аппаратных компонентов Интернета-вещей разнообразна и, как правило, различается у разных производителей, а также в рамках разных линеек одного производителя. Этот факт повышает трудоемкость анализа ПО данного класса. Другим фактором, осложняющим

анализ, является характерное для IoT-систем отсутствие исходных кодов. При этом, в отличие от десктопных систем, в IoT-устройствах бинарный код объектов анализа содержится внутри образа, формат которого, как правило, заранее не известен, и требуется предварительный анализ для их извлечения. Запуск же целевого бинарного модуля IoT-устройства вне предполагаемой производителем среды выполнения либо невозможен, либо затруднен, либо результат его работы может отличаться от результата в исходной среде запуска. Использование полносистемного эмулятора – один из подходов к решению указанной проблемы.

Для проведения динамического анализа бинарного кода IoT-устройства необходимо внесение дополнительных инструментов в среду выполнения (отладчики, профилировщики и т.д.), что не всегда возможно, а в ряде случаев довольно трудоемко в силу малого объёма памяти IoT-устройства, наличия в нем специфичных файловых систем (ФС) и т.д. При этом, как правило, требуется пересборка образа IoT-устройства, в процессе которой может потребоваться дополнительный анализ реализации загрузчиков, кода проверки целостности и т.п. При этом, даже в тех случаях, когда удалось поместить «навесной» инструмент (например, отладчик) внутрь IoT-устройства, его использование может быть затруднено по причинам ограниченности функциональности ядра.

В силу перечисленных ограничений и особенностей широкое распространение получил подход к анализу безопасности IoT-устройств, основанный на анализе отдельных библиотек с доступным исходным кодом. В этом случае, установив точную версию библиотеки, можно применять к ней средства анализа с учетом наличия исходных кодов (фаззинг, символьное выполнение, статический анализ по исходным кодам и т.д.). Развитие методов анализа по исходным кодам привело к появлению гибридного фаззинга, в котором совмещаются идеи символьных вычислений и фаззинга. Гибридный фаззинг библиотек с открытым исходным кодом показал хорошие результаты [1]. Однако и данный подход не лишен недостатков – ошибки, имеющиеся в коде библиотеки, могут не проявляться вне контекста IoT-устройства. В качестве примера такой ситуации можно привести уязвимость CVE-2018-0101 в Cisco ASA [2]. В уязвимых устройствах при обработке сетевых запросов, содержащих данные в формате xml, был задействован код библиотеки libexpat. Суть уязвимости заключается в том, что при обработке сетевых запросов VPN-сервис IoT-устройства переходит в некоторое состояние, в котором после отправки нескольких пакетов происходит аварийное завершение сервиса с возможностью выполнить произвольный код в его контексте. Примечательно, что уязвимый код, реализующий логику разбора xml, содержится не в самой библиотеке libexpat, а в callback-функциях, относящихся к IoT-устройству. Указатели на эти функции получает библиотека libexpat, и передает управление на них в процессе разбора xml. Следовательно, протестировать уязвимый код вне контекста IoT-устройства невозможно. Таким образом, обнаружение подобного рода ошибок возможно только в процессе проведения полносистемного анализа.

В данной работе рассматривается интеграция в инструмент ELF возможностей полносистемного символьного выполнения с целью поддержки гибридного фаззинга.

2. Обзор архитектуры ELF, общая схема функционирования

В предыдущей статье об инструменте ELF [3] были подробно описаны его базовая архитектура и назначение: инструмент представляет собой платформу на основе полносистемной эмуляции QEMU, обеспечивающую исследование различными инструментами (отладчиков, фаззеров, средств инструментации и т.д.) в штатной среде функционирования программного кода IoT-устройства. В качестве исходных данных для помещения IoT-устройства в среду ELF необходимо наличие образа Linux-подобного ядра, образа ФС, а также образа начальной корневой ФС (initrd). Указанные компоненты предварительно должны быть извлечены из IoT-устройства. При этом стоит обратить

внимание на то, что в случаях, когда удаётся восстановить версию ядра IoT-устройства, а также полностью или частично его конфигурацию, то ядро может быть заново собрано из исходных кодов с добавлением различных модулей, полезных для отладки и анализа.

В ходе проведенных исследований было установлено, что дополнительные инструментальные средства анализа, которые добавляются в начальную корневую ФС, как правило, не влияют на логику функционирования IoT-устройства. Таким образом вносить изменения в образ ФС IoT-устройств не требуется, поскольку весь дополнительный инструментарий помещается в образ initrd. В составе инструмента ELF содержится набор скриптов, которые обеспечивают не только запуск тестов, различных средств анализа, но и выполняют при необходимости пересборку всех компонент в зависимости от сценария тестирования. Тем самым исключаются ошибки в работе ELF, связанные с рассогласованностью отдельных компонентов и модулей.

Интерфейс запуска всех инструментов анализа в ELF унифицирован, а параметры определяются в едином конфигурационном файле. Инструмент ELF может запускаться в нескольких режимах:

- тестовый – в этом режиме IoT-устройство запускается в эмуляторе QEMU. При этом может использоваться любая «ванильная» QEMU версии 3.0 и выше, QEMU ИСП РАН или QEMU из проекта S2E. При запуске IoT-устройства пользователю предоставляется shell-консоль от имени пользователя root. В тестовом режиме возможно проведение фаззинга с помощью внешних сетевых фаззеров, а также снятие трассы.
- отладки – предоставляет те же возможности, что тестовый режим, но добавляется возможность отладки IoT-устройства через интерфейс GDB/QEMU;
- фаззинга – в данном режиме проводится фаззинг целевого ПО IoT-устройства с помощью AFL++;
- символьных вычислений – в данном режиме выполняются символьные вычисления для целевого ПО IoT-устройства.

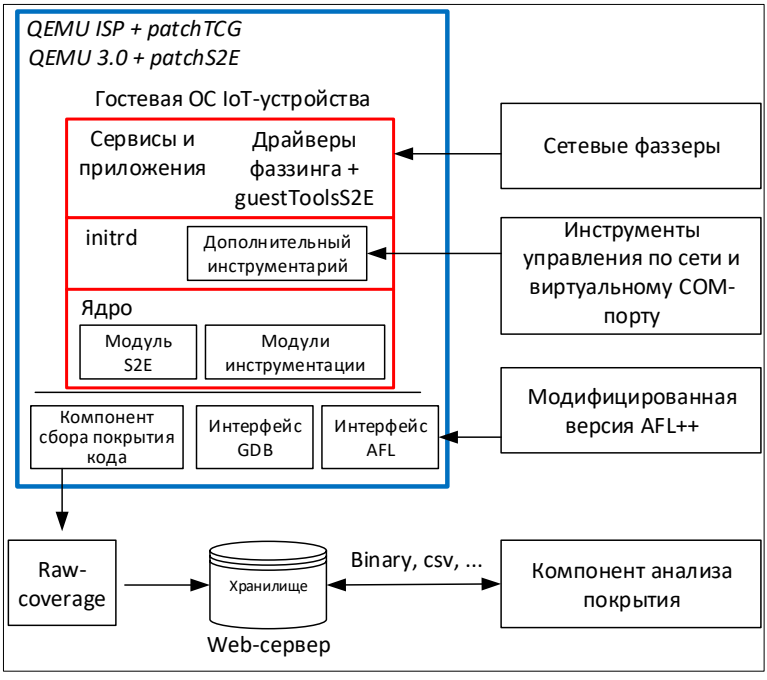


Рис. 1. Основные компоненты инструмента ELF
Fig. 1. The main components of the ELF tool

Базовая часть ELF – эмулятор QEMU. В текущей версии ELF для фаззинга и символьных вычислений поддерживаются две версии QEMU: 1) специализированная версия QEMU ИСП РАН [4], в которую внесены изменения для поддержки фаззинга (patchTCG) и 2) эмулятор на базе QEMU 3.0 из проекта S2E [5,6] с внесенными изменениями (patchS2E).

Инструмент ELF предназначен для исследования IoT-устройств с архитектурами x86_64 и ARM. На рис. 1 приведена схема функционирования ELF и его основные компоненты.

По сравнению с ELF предыдущей версии [3], в новой версии авторами обновлены и добавлены следующие возможности:

- поддержка последних версий AFL++ [7], Boofuzz [8] и Radamsa [9];
- инжектирование драйверов фаззинга в адресное пространство процессов;
- получения командной строки для управления IoT-устройством, посредством виртуальных COM-портов;
- интерфейс добавления различных утилит анализа (strace, tcpdump, gdbserver, busybox, far2l, различных сетевых инструментов и т.д);
- поддержка символьных вычислений (см. разд. 3);
- распределенный гибридный фаззинг с поддержкой docker-контейнеров (см. разд. 4).

Для использования режима отладки среда ELF запускается с GDB-сервером. Данный режим используется для работы с сетевыми фаззерами.

Механизм взаимодействия с AFL++ реализован в виде протокола взаимодействия между QEMU и фаззером, на основе вставки на уровне бинарного кода несуществующей машинной инструкции (рис. 2).

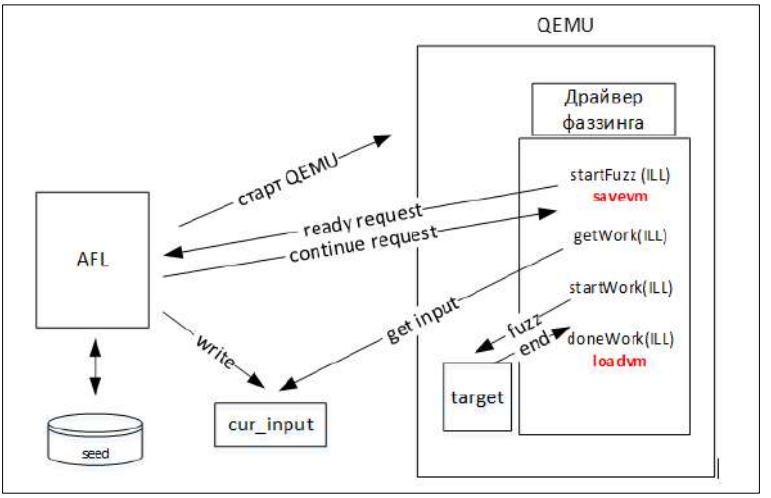


Рис. 2. Протокол взаимодействия AFL и QEMU
Fig, 2. AFL and QEMU interaction protocol

Добавлен дополнительный инструмент для загрузки драйверов фаззинга в адресное пространство IoT-устройства (утилита linuxinjector) – бинарный исполняемый модуль, который обладает функционалом полноценного инжектора разделяемых библиотек в адресные пространства других процессов и предназначен для создания контекста работы драйверов фаззинга в адресных пространствах произвольных процессов IoT-устройства.

Это позволило в новой версии ELF реализовать поддержку драйверов фаззинга в виде отдельных разделяемых библиотек, которые теперь можно внедрять в целевое приложение посредством инжектирования в его адресное пространство. Этот подход позволяет реализовывать логику сетевого фаззинга, вклиниваясь в то или иное место работы штатного

сценария работы исследуемого сетевого сервиса. Это помогает в тех ситуациях, когда, например, внутри IoT-устройства отсутствует возможность напрямую из дополнительного приложения отправлять сетевые пакеты на интересующий TCP-порт (используется dpdk, netmap и т.д.).

В состав дополнительного инструментария добавлена возможность получения командной строки IoT-устройства посредством механизма виртуального соединения COM-портов виртуальный машины и хоста. В процессе запуска QEMU на хосте открывается виртуальный COM-порт (/dev/pts/N), подключение к которому позволяет обеспечить доступ к внутренней командной оболочке (shell) IoT-устройства, или если она отсутствует (малофункциональна), то в IoT-устройство добавляется busybox, который используется в качестве shell-оболочки. Отметим, что данный функционал позволяет получить root-доступ к IoT-устройству даже в тех ситуациях, когда сеть не настроена или отключена.

При сборке компонентов ELF в зависимости от сценария тестирования в образ начальной ФС могут быть добавлены различные утилиты для упрощения изучения IoT-устройства «изнутри». Это могут быть инструменты отладки (strace, ltrace, gdbserver и т.д.), инструменты для работы сетью (tcpdump), shell-оболочки (busybox, far2l) с различными дополнительными утилитами и т.п. Для внедрения подобных утилит требуется их предварительно скомпилировать с помощью набора библиотек и компилятора, соответствующего исследуемому IoT-устройству.

3. Интеграция S2E в ELF

S2E [5, 6] – это платформа для написания инструментов анализа свойств и поведения программных систем. S2E представляет собой модульную библиотеку, которая предоставляет возможности символьного выполнения и анализа программ с использованием *полносистемной* эмуляции в QEMU. Инструмент S2E запускает *немодифицированные* программные стеки x86, x86-64 или ARM, включая программы, библиотеки, ядро и драйверы. Возможности работы с немодифицированными программными стеками в режиме полносистемной эмуляции полностью соответствуют идеологии инструмента ELF, что позволяет совместить работу этих двух инструментов. Инструмент S2E использует фрэймворк символьных вычислений KLEE [10] и решатель Z3 [11]. На рис. 3 приведена схема интеграция инструментов ELF и S2E с основными компонентами.

- Интерфейс управления ELF – унифицированный интерфейс для проведения динамического анализа IoT-устройств, в том числе с помощью символьных вычислений.
- Эмулятор QEMU из S2E, который используется при запуске в режиме символьных вычислений.
- Модифицированное ядро Linux. В S2E для исследования Linux используется исправленное ядро фиксированной четвертой версии, а также модуль ядра s2e.ko. В ELF нет привязки к конкретному ядру – модифицируется оригинальное ядро IoT-устройства, при этом исходные коды не обязательны.
- Образ файловой системы initrd, в который добавляются дополнительные компоненты для запуска S2E:
  - стартовые скрипты для запуска символьных вычислений;
  - библиотека GuestTools – библиотека для взаимодействия с модулями расширения S2E, а также указания символьных переменных;
  - драйвер символьных вычислений – играет роль, аналогичную GuestTools, но ориентирован на тестирование сетевых сервисов.
- Библиотека libs2e.so, которая содержит основной функционал S2E.

Компиляция дополнительных компонент для запуска S2E должна выполняться с помощью соответствующего набора библиотек и компилятора для выбранного IoT-устройства.

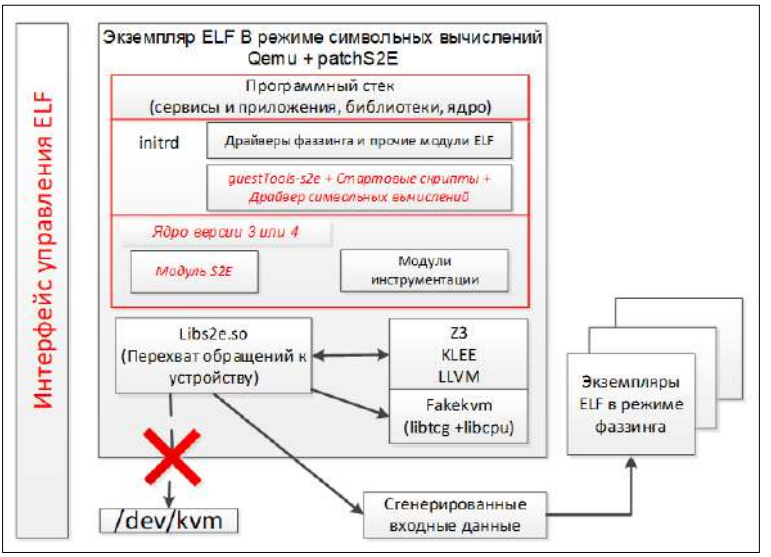


Рис. 3. Схема интеграции инструментов ELF и S2E
Fig. 3. Scheme of integration of ELF and S2E tools

Запуск QEMU в режиме символьных вычислений обладает рядом особенностей. Во-первых, требуется запускать QEMU с параметром LD_PRELOAD=libs2e.so. Данная библиотека содержит всю логику работы с символьными вычислениями, а также реализует логику работы эмулируемого процессора (вместо QEMU). Таким образом, отделяются задачи эмуляции процессора (libs2e.so) от эмуляции периферийных устройств (QEMU).

Во-вторых, QEMU запускается с ключом -enable-kvm, т.е. с точки зрения QEMU она запускается на реальном процессоре и эмулирует только аппаратные устройства посредством интерфейса KVM. Но реально до устройства /dev/kvm дело не доходит, поскольку библиотека libs2e.so перехватывает соответствующие системные вызовы к KVM. Таким образом, вторая значимая часть libs2e.so (отмечена на рис.3 как fakekvm) – это эмуляция процессора и перевод машинного кода в представление LLVM. Часто указывают, что для использования фрэймворка KLEE требуются исходные коды. Действительно, в примерах KLEE для выполнения символьных вычислений требуется использовать компилятор clang. Однако фактически для KLEE требуются не исходные коды, а представление кода в LLVM. Библиотека libs2e.so как раз это и выполняет – эмулирует процессор и получает LLVM представление, которое передаётся на вход KLEE.

В рамках проводимых исследований авторы не изменяли функционал libs2e.so для символьных вычислений, но внесли небольшие изменения в части касающейся эмуляции процессора (добавлена поддержка дополнительных наборов инструкций) для расширения номенклатуры исследуемых IoT-устройств.

4. Применение ELF для исследования сетевых сервисов с помощью символьных вычислений и распределенного фаззинга

В примерах от S2E для указания символьных данных в сетевых пакетах используется SystemTap – средство инструментации кода ядра Linux [12]. На рис. 4 представлен пример использования SystemTap из документации S2E.

Функция s2e_make_symbolic определяет, какие данные являются символьными. Следует обратить внимание на то, что сам код функции s2e_make_symbolic выполняет только вставку машинной «инструкции» с кодом «0x0F 0x3F». Этот машинный код некорректен, т.е. на самом деле вставляется несуществующая машинная инструкция, которая обрабатывается

библиотекой libs2e.so в контексте процесса QEMU, которая затем транслируется в вызов klee_make_symbolic, где и определяются символьные переменные. Такой приём полностью соответствуют принятому в ELF протоколу взаимодействия между фаззером (AFL++) и QEMU, который упомянут раннее в разд. 2.

```
static inline void s2e_make_symbolic( void *buf, int size, const char *name) {
  __asm__ __volatile__(
    ".byte 0x0f, 0x3f\n"
    ".byte 0x00, 0x03, 0x00, 0x00\n"
    ".byte 0x00, 0x00, 0x00, 0x00\n"
    : : "a" (buf), "b" (size), "c" (name) );
 } %}
# Take a pointer to the IP header, and make the options length field symbolic
function s2e_inject_symbolic_ip_optionlength (ipheader: long) %{
    uint8_t *data = (uint8_t*)((uintptr_t)(THIS->ipheader + 0));
    uint8_t len;
    s2e_make_symbolic(&len, 1, "ip_headerlength");
    *data = *data & 0xF0;
    *data = *data | ((len) & 0xF);
%}
# Instruct SystemTap to intercept the netif_receive_skb kernel function
probe kernel.function("netif_receive_skb"){
    msg = sprintf("%s: len=%d datalen=%d\n", probefunc(), $skb->len,
                                                $skb->data_len)
    s2e_message(msg)
    s2e_inject_symbolic_ip_optionlength($skb->data)
}
```

Рис. 4. Пример скрипта SystemTap для указания символьных данных
Fig. 4. An example of a SystemTap script for specifying character data

Подход с применением SystemTap целесообразен при наличии исходных кодов ядра, однако практика использования ELF предполагает, что полностью исходный код ядра и конфигурационный файл не доступны, и поэтому анализ проводится на аутентичном бинарном образе ядра (но оно может модифицироваться). Поэтому авторами ELF были разработаны драйверы символьных вычислений (по аналогии с драйверами фаззинга).

- s2esender – сетевой клиент в составе ELF (расширение guestTools S2E) для задания символьных переменных в сетевых протоколах. В качестве символьных переменных может выступать как длина пакетов, так и их содержимое. Также есть возможность работать с цепочками пакетов, т.е. потенциально «пробиваться» через состояния сетевых сервисов.
- s2einjector – инжектор в процессы для указания символьных данных. Выполняет те же действия, что и s2esender, только не запускает новый процесс, а инжектируется в уже существующий процесс. Это удобно для анализа сервисов, которые стартуют при загрузке ОС в IoT-устройстве.

Драйверы символьных вычислений выделяют в сетевом пакете символьные данные (с помощью вызова s2e_make_symbolic) и отправляют его исследуемому сетевому сервису, либо отслеживают функции приема пакетов в сети и в полученных пакетах отмечают символьные данные.

Результатом работы символьных вычислений может быть как выявленная ошибка и вызывающие ее данные, так и набор входных данных для распределенного фаззинга. В последнем случае выходные данные работы ELF в режиме символьных вычислений автоматически передаются экземплярам ELF, запущенным в режиме фаззинга. Для этого авторами ELF был внесен соответствующий программный код в библиотеку libs2e.so.

Очевидно, что в режиме фаззинга ELF должен запускаться во многих экземплярах, их работа координироваться, а результаты – централизованно обрабатываться и храниться. Инструмент ELF может запускаться во многих экземплярах как на хосте, так и в docker-контейнерах. Для управления процессом распределенного фаззинга в настоящее время разрабатывается

система оркестрации на основе Web-приложения. Пользователю инструмента ELF через браузер доступны операции как над группами экземпляров ELF, так и для каждого экземпляра ELF в отдельности. В настоящее время результаты работы ELF хранятся в файловых директориях, но предполагается их хранение в единой базе данных. На рис. 5 приведены снимки экрана со списком задач (левая часть рисунка) и окном для выбора просмотра/управления хостом, docker-контейнером или отображением статистики по текущему фаззингу (правая часть рисунка).

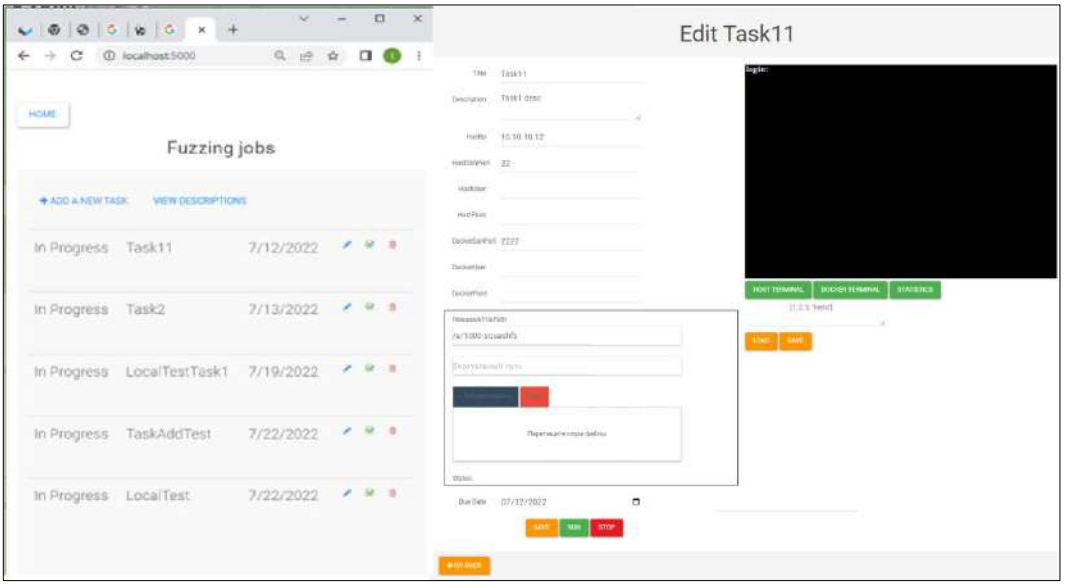


Рис. 5. Снимки экранов системы оркестрации для управления фаззингом
Fig. 5. Screenshots of the orchestration system for managing fuzzing

5. Синтетические тесты

В данном разделе приводятся результаты работы инструмента ELF на различных синтетических тестах. Инструмент ELF для проверки и контроля работоспособности символьных вычислений содержит следующие синтетические тесты:

- набор тестов от s2e;
- консольное приложение;
- сетевой сервер vulnserver;
- набор тестов для ряда известных уязвимостей;
- тест для штатного HTTP-сервера IoT-устройства.

Функционал анализируемого консольного приложения – отображение текущей версии устройства при запуске с параметром «-v». При этом добавлен ключ «-vv», который приводит к аварийному завершению приложения. В результате работы ELF восстановлены все поддерживаемые приложением аргументы, а также зафиксировано аварийное завершение работы приложения при запуске с параметром «-vv» (рис. 6). При этом логируются входные данные, вызвавшие ошибку.

На тестовом примере vulnserver показана возможность восстановления команд простого тестового FTP-подобного сетевого сервера, воспринимающего ограниченный набор простых команд. Инструмент ELF в режиме символьного выполнения успешно восстанавливает все команды в течение нескольких минут (рис. 7).

```
169 [State 2] TestCaseGenerator: generating test case at address 0x80489dd
169 [State 2] TestCaseGenerator:            v0_arg1_0 = {0x2d, 0x76, 0x2d}; (string) "-v-"
169 [State 2] Switching from state 2 to state 4
169 [State 4] LinuxMonitor: Removing task (pid=0x11e, cr3=0x177f000, exitCode=0).
169 [State 4] ProcessExecutionDetector: Unloading process 0x11e
169 [State 4] LinuxMonitor: Removing task (pid=0x11d, cr3=0x177f000, exitCode=0).
169 [State 4] LinuxMonitor: Process ./s2ecmd loaded pid=0x11f
169 [State 4] LinuxMonitor: Process ./s2ecmd loaded pid=0x11f
169 [State 4] LinuxMonitor: ModuleDescriptor Name=s2ecmd Path=./s2ecmd Size=0x6d1ac Address
169 [State 4] ModuleExecutionDetector: module loaded: ModuleDescriptor Name=s2ecmd Path=./s
169 [State 4] BaseInstructions: Killing state 4
169 [State 4] Terminating state: State was terminated by opcode
            message: "bootstrap terminated"
            status: 0x0
169 [State 4] TestCaseGenerator: generating test case at address 0x80489dd
169 [State 4] TestCaseGenerator:            v0_arg1_0 = {0x2d, 0x76, 0x76}; (string) "-vv"
169 [State 4] Switching from state 4 to state 3
169 [State 3] LinuxMonitor: Received segfault type=0 pagedir=0xdc30000 pid=0x11e pc=0x80489
169 [State 3] LinuxMonitor: Blocking searcher until state is terminated
169 [State 3] LinuxMonitor: 0x89 0x7 0x90 0x90 0x89 0x34 0x24 0xeb 0x40 0x57 0x57 0x68 0x53
0x65 0xfe 0xff 0xff 0x89 0x85 0x54 0xff 0xff 0xff 0x83 0xc4 0x14 0x53 0xe8 0xc6 0xfd 0xff 0
0x8048967:  mov     dword ptr [rdi], eax
0x8048969:  nop
0x804896a:  nop
0x804896b:  mov     dword ptr [rsp], esi
0x804896e:  jmp     0x80489b0
0x8048970:  push    rdi
0x8048971:  push    rdi
0x8048972:  push    0x8048d53
0x8048977:  push    rsi
0x8048978:  call    0x80487a0
```

Рис. 6. Результат работы ELF в режиме символьного выполнения на тестовом консольном приложении

Fig. 6. The result of running ELF in symbolic execution mode on a test console application

```
  64 [State 51] TestCaseGenerator: generating test case at address 0x80489dd
  64 [State 51] TestCaseGenerator:      v0_netpackage_0 =
{0x48, 0x45, 0x4c, 0x50, 0x48, 0x48, 0x48, 0x48, 0x48, 0x48, 0x48, 0x48, 0x48, 0x48}; (string) "HELPHHHHHH
     v1_lenpackage_1 = {0x10, 0x0, 0x0, 0x0}; (int32_t) 16, (string) "...."
  64 [State 51] Switching from state 51 to state 42

          message: "bootstrap terminated"
          status: 0x0
1430 [State 291] TestCaseGenerator: generating test case at address 0x80489dd
1430 [State 291] TestCaseGenerator:      v0_netpackage_0 =
{0x47, 0x4d, 0x4f, 0x4e, 0x47, 0x47, 0x47, 0x47, 0x47, 0x47, 0x47, 0x47, 0x47, 0xx47}; (string) "GMONGGGGGGGG
     v1_lenpackage_1 = {0x19, 0x0, 0x0, 0x0}; (int32_t) 25, (string) "...."

          message: "bootstrap terminated"
          status: 0x0
2106 [State 197] TestCaseGenerator: generating test case at address 0x80489dd
2106 [State 197] TestCaseGenerator:      v0_netpackage_0 =
{0x47, 0x44, 0x4f, 0x47, 0x47, 0x47, 0x47, 0x47, 0x47, 0x47, 0x47, 0x47, 0x47, 0x47}; (string) "GDOGGGGGGGGG
     v1_lenpackage_1 = {0x19, 0x0, 0x0, 0x0}; (int32_t) 25, (string) "...."
2106 [State 197] Switching from state 197 to state 237

           message: "bootstrap terminated"
           status: 0x0
2820 [State 83] TestCaseGenerator: generating test case at address 0x80489dd
2820 [State 83] TestCaseGenerator:      v0_netpackage_0 =
{0x4b, 0x53, 0x54, 0x45, 0x54, 0x20, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54}; (string) "KSTET TTTTTT
     v1_lenpackage_1 = {0x11, 0x0, 0x0, 0x0}; (int32_t) 17, (string) "...."
2820 [State 83] Switching from state 83 to state 441

          message: "bootstrap terminated"
          status: 0x0
2847 [State 144] TestCaseGenerator: generating test case at address 0x80489dd
2847 [State 144] TestCaseGenerator:      v0_netpackage_0 =
{0x4b, 0x53, 0x54, 0x41, 0x4e, 0x20, 0x4b, 0x4b, 0x4b, 0x4b, 0x4b, 0x4b, 0x4b, 0x4b}; (string) "KSTAN KKKKKKK
     v1_lenpackage_1 = {0x11, 0x0, 0x0, 0x0}; (int32_t) 17, (string) "...."
2847 [State 144] Switching from state 144 to state 116
```

Рис. 7. Результат работы ELF с поддержкой полносистемного символьного выполнения на модельном сетевом сервисе

Fig. 7. The result of ELF operation with support for full-system symbolic execution on a model network service

Набор тестов содержит примеры обнаружения известных уязвимостей при запуске ELF в режиме символьного выполнения. Один из таких примеров – нахождение уязвимости CVE-2018-7445 (в сетевом сервисе SMB). В тесте всё содержимое сетевого пакета, а также его длина объявляются символьными переменными. Сформированный сетевой пакет отправляется на TCP-порт SMB с номером 445 (см. рис. 8). В результате уже при проверке четвертого состояния выдаются данные сетевого пакета, приводящие к ошибке.

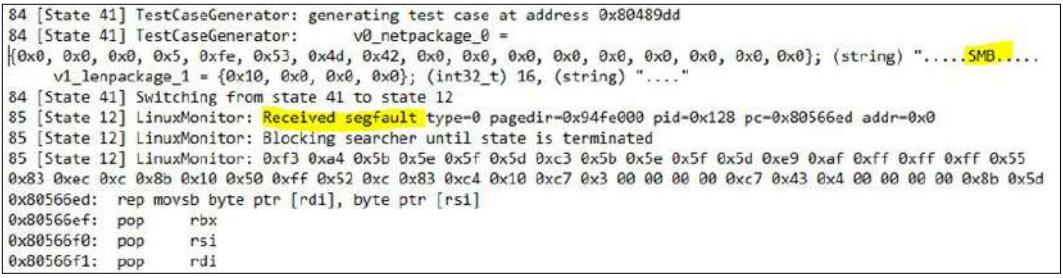


Рис. 8. Найденная уязвимость CVE-2018-7445 в результате работы ELF в режиме символьного выполнения

Fig. 8. Found vulnerability CVE-2018-7445 as a result of ELF running in symbolic execution mode

Большинство IoT-устройств содержит штатный HTTP-сервер, поэтому набор тестов в ELF содержит универсальный сетевой клиент для отправки символьных данных на TCP-порт с номером 80. На рис. 9 отображены состояния, в которых восстановлены поддерживаемые методы протокола HTTP в исследуемом IoT-устройстве до прохождения пользователем процедуры аутентификации.

Рис. 9. Восстановление ключевых команд HTTP-сервера IoT-устройства в среде ELF с поддержкой полносистемного символьного выполнения

Fig. 9. Recovery of key commands of the HTTP server of the IoT device in the ELF environment with support for full-system symbolic execution

Одним из показателей качества фаззинга является величина покрытия. Целью применения символьных вычислений при генерации входных данных для фаззеров является увеличение величины покрытия. Авторы ELF провели сравнение величин покрытия при фаззинге AFL++ и фаззинге совместно с символьными вычислениями. На всех проводимых тестах наблюдалось увеличение покрытия. Инструмент ELF позволяет вычислять покрытие на уровне QEMU с применением инструментов интроспекции. Получить покрытие для системного и прикладного ПО IoT-устройства в привычных для многих программистов форматах не всегда возможно, т.к. исходные коды отсутствуют. Средства бинарной инструментации (dynamorio, pin tools и т.п.) не всегда могут быть использованы непосредственно в среде IoT-устройства. Чтобы сравнить величины покрытия, полученные от разных фаззеров, сохранялись все сгенерированные ими входные данные, которые впоследствии передавались на эталонный экземпляр тестового ПО.

Далее приводится результат только одного теста, который является весьма показательным.

В этом примере использовалось одно из IoT-устройств, содержащее библиотеку libexpat версии 2.1.0. Для проверки инструмент ELF запускался с тремя сценариями фаззинга: Radamsa версии 0.6, AFL++ версии 4.01a и S2E совместно с AFL++. Для теста было создано небольшое приложение (fuzz_test), которое вызывало функцию разбора XML_Parse (точка входа в разбор xml в библиотеки libexpat) для входного файла seed.xml. Приложение fuzz_test запускалось на IoT-устройстве и являлось целевым приложением для фаззинга. Начальный корпус для фаззеров Radamsa и AFL++ состоял из одного файла seed.xml, содержимое которого подготовленного по результатам «ручного» анализа бинарного кода IoT-устройства. В файл seed.xml были добавлены все теги и их параметры, которые были выявлены, чтобы обеспечить фаззерам максимальное покрытие. Для сценария с символьными вычислениями файл seed.xml не использовался, а лишь указывалось, что входной файл является символьным файлом размером 128 байт.

Чтобы максимально точно выполнить замеры величин покрытия, было скомпилировано контрольно-измерительное приложение fuzz_test_control из исходных кодов fuzz_test и libexpat версии 2.1.0 с опциями компилятора для получения покрытия в формате gcov.

Контрольно-измерительное приложение fuzz_test_control использовалось следующим образом:

- в процессе фаззинга сохранялся корпус из всех сгенерированных входных данных или из тех данных, которые сам фаззер определяет как минимальный корпус для достижения заданного покрытия;
- независимо от того, вычисляет ли фаззер самостоятельно покрытие или нет, для измерения покрытия используется fuzz_test_control;
- измерение покрытия выполняется следующим образом: для каждого из трех сценариев все сохраненные входные данные последовательно подаются на вход fuzz_test_control – в результате для каждого сценария получается покрытие в формате gcda.

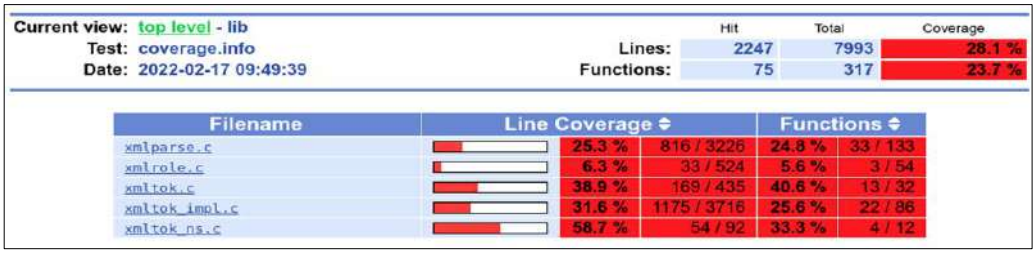


Рис. 10. Покрытие, полученное фаззером Radamsa версии 0.6
Fig. 10. Coverage obtained by Radamsa fuzzer version 0.6

На рис. 10, 11 и 12 показаны результаты измерений величин покрытия, полученные фаззерами Radamsa, AFL++ и гибридным фаззингом. В табл. 1 сведены результаты измерений для указанных фаззеров.

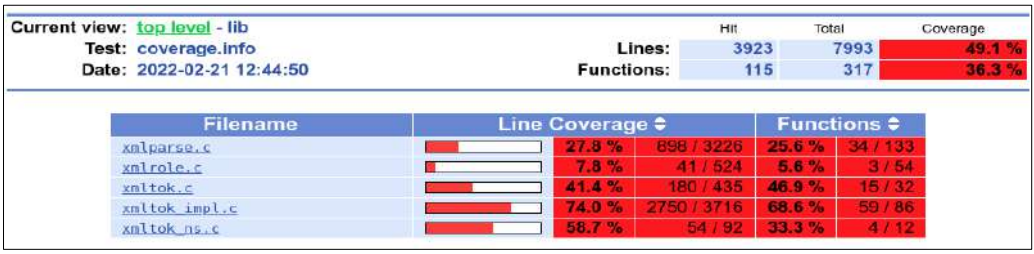


Рис. 11. Покрытие, полученное фаззером AFL++ версии 4.01a
Fig. 11. Coverage obtained by fuzzer AFL++ version 4.01a

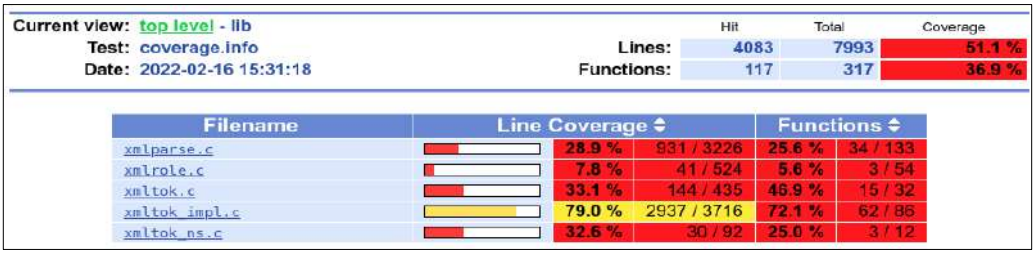


Рис. 12. Покрытие, полученное в результате гибридного фаззинга
Fig. 12. Coverage resulting from hybrid fuzzing

Табл. 1. Сравнение покрытий, полученных от различных фаззеров
Table 1. Comparison of coverages obtained from different fuzzers

фаззер	**Количество строк (всего 7993), %**	**Количество функций (всего 317), %**	**Начальный seed**	**Время (ч.)**
Radamsa 0.6	2247 28,1%	75 23,7%	seed.xml	96
AFL++ 4.01a	3923 49,1%	115 36,3%	seed.xml	96
Гибридный фаззинг: S2E+ AFL++ 4.01a	4083 51,1%	117 36,9%	отсутствует	24 часа S2E 24 часа AFL++

Как видно из табл. 1, разница между покрытием, полученным от AFL++ и гибридным фаззингом, незначительная – всего 2 %. Однако, гибридный фаззинг достиг указанной величины покрытия в два раза быстрее – суммарно за 48 часов. На практике ELF в режиме символьных вычислений запускается параллельно с экземплярами ELF в режиме фазинга и по мере генерации входных данных передает их фаззерам AFL. Важно отметить, что для того чтобы AFL++ достиг покрытия 49%, потребовался ручной анализ кода IoT-устройства для определения тегов, которые были указаны в файле seed.xml. В сценарии же с гибридным фаззером предварительного анализа не требовалось. Входной файл, как таковой, отсутствовал, была указана его длина (128 байт), а все его содержимое было объявлено символьной переменной.

5. Заключение

Версия ELF с поддержкой символьных вычислений и гибридного фаззинга успешно продемонстрировала возможность полносистемного анализа IoT-устройств с применением

символьных вычислений. Тот факт, что для некоторых известных CVE инструмент ELF в символьном режиме находит входные данные, приводящие к аварийному завершению, позволяет рассматривать символьные вычисления не только как способ генерации входных данных, но и инструмент поиска ошибок в контексте IoT-устройствах. При этом, следует отметить, что тестовые CVE подобраны для демонстрации таких ситуаций, в которых поиск ошибок в уязвимых библиотеках вне контекста их использования в IoT-устройстве не даёт результата.

В качестве развития среды ELF с полносистемным символьным выполнением можно выделить следующие направления:

- автоматизированное внесение функционала S2E в бинарный образ ядра;
- переход на новую версию KLEE;
- выбор оптимального решателя;
- улучшение функционирования моделей функций;
- отображение покрытия в формате cachegrind в символьном режиме;
- переход на более новую версию QEMU в символьном режиме работы S2E;
- создание единой базы данных для хранения и обработки результатов распределенного гибридного фаззинга.

Список литературы / References

[1]. OSS-Sydr-Fuzz: Hybrid Fuzzing for Open Source Software. Available at: https://github.com/ispras/oss-sydr-fuzz.git, accessed 16.08.2022.
[2]. C. Halbronn. The Return of Robin Hood vs Cisco ASA Available at: https://www.nccgroup.trust/globalassets/newsroom/uk/events/offensivecon2018-the-return-of-robin-hood-vs-cisco-asa.pdf, accessed: 16.08.2022.
[3]. Коваленко Р.Д., Макаров А.Н. Динамический анализ IoT-систем на основе полносистемной эмуляции в QEMU. Труды ИСП РАН, том 33, вып. 5, 2021 г., стр. 155-166. DOI: 10.15514/ISPRAS–2021–33(5)–9 / Kovalenko R.D., Makarov A.N. Dynamic analysis of IoT systems based on full-system emulation in QEMU. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 5, 2021, pp. 155-166 (in Russian).
[4]. P. Dovgalyuk. Deterministic Replay of System's Execution with Multi-target QEMU Simulator for Dynamic Analysis and Reverse Debugging. In Proc. of the 16th European Conference on Software Maintenance and Reengineerin, 2012, pp. 553-556.
[5]. V. Chipounov, V. Kuznetsov, G. Candea. S2E: a platform for in-vivo multi-path analysis of software systems. ACM SIGPLAN Notices, vol. 46, issue 3, 2011, pp 265-278.
[6]. S2E: The Selective Symbolic Execution Platform. Available at: http://s2e.systems/docs/, accessed 16.08.2022.
[7]. The fuzzer afl++. Available at: https://github.com/AFLplusplus/AFLplusplus, accessed 16.08.2022.
[8]. J. Pereyda. Boofuzz Documentation, Release 0.4.1. Available at: https://buildmedia.readthedocs.org/media/pdf/boofuzz/latest/boofuzz.pdf, accessed 16.08.2022.
[9]. radamsa: a general-purpose fuzzer. Available at: https://gitlab.com/akihe/radamsa, accessed 16.08.2022.
[10]. KLEE Symbolic Execution Engine. c https://klee.github.io/, accessed 16.08.2022.
[11]. The Z3 Theorem Prover. The Z3 Theorem Prover. Available at: https://github.com/Z3Prover/z3, accessed 16.08.2022.
[12]. Using SystemTap with S2E. Available at: http://s2e.systems/docs/Tutorials/SystemTap/index.html, accessed 16.08.2022.

Информация об авторах / Information about authors

Алексей Николаевич МАКАРОВ – научный сотрудник отдела компиляторных технологий. Сфера научных интересов: информационные технологии, безопасность ПО, анализ бинарного кода, системное программирование, фаззинг, исследования программ.

Aleksey Nikolaevich MAKAROV – Researcher at the Department of Compiler Technologies. Research interests: information technology, software security, binary code analysis, system programming, fuzzing, software research.

Роман Дмитриевич КОВАЛЕНКО – научный сотрудник отдела компиляторных технологий. Сфера научных интересов: информационные технологии, безопасность ПО, анализ бинарного кода, системное программирование, фаззинг, исследования программ.

Roman Dmitrievich KOVALENKO – Researcher, Compiler Technology Department. Research interests: information technology, software security, binary code analysis, system programming, fuzzing, software research.

DOI: 10.15514/ISPRAS-2022-34(4)-4

Автоматическое тестирование LLVM-программ со сложными входными структурами данных

[1]А.*В. Мисонижник, ORCID: 0000-0002-5907-0324 <misonijnik@gmail.com>*
[1]*А.А. Бабушкин, ORCID: 0000-0002-5661-5800 <ocelaiwo@gmail.com>*
[2]*С.А. Морозов, ORCID: 0000-0003-1160-5614 <morozov.serg901@gmail.com>*
[1]*Ю.О. Костюков, ORCID: 0000-0003-4607-039X <kostyukov.yurii@gmail.com>*
[1]*Д.А. Мордвинов, ORCID: 0000-0002-6437-3020 <mordvinov.dmitry@gmail.com>*
[1]*Д.В. Кознов, ORCID: 0000-0003-2632-3193 <d.koznov@spbu.ru>*

[1]*Санкт-Петербургский государственный университет,*
199034, Россия, Санкт-Петербург, Университетская наб., д. 7-9
[2]*Национальный исследовательский университет "Высшая школа экономики",*
190121, Россия, Санкт-Петербург, Союза Печатников ул., д.16

Аннотация. Символьное исполнение является известным подходом для автоматической генерации регрессионных тестов и поиска ошибок/уязвимостей в программах. Данная работа посвящена созданию практичного метода к символьному исполнению LLVM-программ, пригодного для работы со сложными входными структурами данных. Метод основан на известной идее ленивой инициализации, позволяющей избавить пользователя от необходимости вручную создавать ограничения на входные структуры данных и полностью автоматизировать процесс символьного исполнения программы. Предлагается два улучшения ленивой инициализации для сегментированной символьной моделей памяти – использование временных меток и информации о типах. Предложенный метод реализован в символьной виртуальной машине KLEE для платформы LLVM и протестирован на реальных С-структурах данных — списках, биномиальных кучах, AVL-деревьях, красно-чёрных деревьях, двоичных деревьях и борах (префиксных деревьях).

Ключевые слова: автоматическое тестирование; символьное исполнение; ленивая инициализация; платформа LLVM; KLEE; структуры данных

Для цитирования: Мисонижник А.В., Бабушкин А.А., Морозов С.А., Костюков, Д.А. Мордвинов Ю.О., Кознов Д.В. Автоматическое тестирование LLVM-программ со сложными входными структурами данных. Труды ИСП РАН, том 34, вып. 4, 2022 г., стр. 49-62. DOI: 10.15514/ISPRAS-2022-34(4)-4

Благодарности: Данное исследование было поддержано грантом РНФ № 22-21-00697.

Automated testing of LLVM programs with complex input data structures

Automated testing of LLVM programs with complex input data structures

[1]*A.V. Misonizhnik, ORCID: 0000-0002-5907-0324 <misonijnik@gmail.com>*
[1]*A.A. Babushkin, ORCID: 0000-0002-5661-5800 <ocelaiwo@gmail.com>*
[2]*S.A. Morozov, ORCID: 0000-0003-1160-5614 <morozov.serg901@gmail.com>*
[1]*Yu.O. Kostyukov, ORCID: 0000-0003-4607-039X <kostyukov.yurii@gmail.com>*
[1]*D.A. Mordvinov, ORCID: 0000-0002-6437-3020 <mordvinov.dmitry@gmail.com>*
D.V. Koznov, ORCID: 0000-0003-2632-3193 <d.koznov@spbu.ru>

[1]*Saint Petersburg State University,*
7/9 Universitetskaya Emb., Saint Petersburg, 199034, Russia
[2]*HSE University,*
16 Soyuza Pechatnikov Street, St Petersburg, 190121. Russia

Abstract. Symbolic execution is a widely used approach for automatic regression test generation and bug and vulnerability finding. The main goal of this paper is to present a practical symbolic execution-based approach for LLVM programs with complex input data structures. The approach is based on the well-known idea of lazy initialization, which frees the user from providing constraints on input data structures manually. Thus, it provides us with a fully automatic symbolic execution of even complex program. Two lazy initialization improvements are proposed for segmented memory models: one based on timestamps and one based on type information. The approach is implemented in the KLEE symbolic virtual machine for the LLVM platform and tested on real C data structures — lists, binomial heaps, AVL trees, red-black trees, binary trees, and tries.

Keywords: automated testing; symbolic execution; lazy initialization; LLVM platform; KLEE; data structures

For citation: Misonizhnik A.V., Babushkin A.A., Morozov S.A., Kostyukov Yu.O., Mordvinov D.A., Koznov D.V. Automated testing of LLVM programs with complex input data structures. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 4, 2022. pp. 49-62 (in Russian). DOI: 10.15514/ISPRAS-2022-34(4)-4

Acknowledgements. The work is supported by the grant of RNF № 22-21-00697.

1. Введение

Регрессионное тестирование является важным инструментом повышения надёжности программного обеспечения. Регрессионные тесты фиксируют поведение программных компонент, позволяя обнаруживать ошибки при внесении изменений в исходный код этих компонент. Однако при большом количестве различных сценариев поведения тестируемого кода написание регрессионных тестов является трудоёмкой задачей, что часто ведёт к упущенным тестовым сценариям.

Естественной идеей решения этой проблемы служит автоматизированная генерация регрессионных тестов [1]. Одним из эффективных способов достижения этой цели служит *символьное исполнение* [2], [3], которое исследует различные ветви поведения программы, используя инструменты проверки выполнимости логических формул (SMT-решатели [4]) для автоматического вывода входных данных, приводящих исполнение программы в эти ветви.

Реальные программы часто содерат *сложные структуры данных*, такие как списки и разные виды деревьев. Далее в этой статье под сложными структурами мы будем иметь в виду структуры, содержащие указатели на другие структуры или на себя. Они широко используются в реальном коде, а их автоматическое тестирование представляет особую сложность. Например, списки используются более чем в 10000 различных точках кода ядра Linux версии 5.6 [5]. Красно-чёрные деревья используются в планировщиках ядра Linux, в драйверах CD/DVD и файловой системе ext3 [6]. Боры (префиксные деревья) используются

для маршрутизации в оборудовании, например, в программном обеспечении компании CISCO [7].

Однако обработка структур данных с указателями является одной из известных проблем символьного исполнения: большинство современных SMT-решателей оказываются неэффективными для работы с динамической памятью и недетерминированными указателями, при помощи которых реализуются сложные структуры данных [8]. Одним из способов решения этой проблемы является использование «ручной» инициализации фрагментов структур данных для конкретизации тестируемого состояния программы [9], что оказывается трудоёмким и неудобным.

Перспективным методом решения этой проблемы является механизм *ленивой инициализации* [10], который наиболее эффективен для *сегментированной* модели символьной памяти [11], в рамках которой вся динамическая память программы разделяется на непересекающиеся блоки. Механизм ленивой инициализации работает следующим образом. Пусть недетерминированный указатель p ссылается на блок памяти размера k, и в динамической памяти уже выделено n объектов размера как минимум k с адресами $a_1, a_2, \dots, a_n$. При разыменовании указателя p символьное исполнение с ленивой инициализацией рассматривает $n+1$ сценарий поведения программы: n сценариев поведения, где $p = a_i$, плюс ещё один сценарий, где p ссылается на новый блок памяти, чья инициализация отложена (проводится *лениво*). Ленивая инициализация позволяет автоматически инициализировать те фрагменты сложных структур данных, которые в действительности читаются в рассматриваемом пути исполнения программы, и полностью автоматически создать входные экземпляры для сколь угодно сложных структур данных. В итоге все порождённые сценарии дадут соответствующие тесты, однако многие из них не будут воспроизводимы, например, из-за того, что символьный указатель не может указывать на фрагмент памяти, выделенный после его инициализации, или на фрагмент памяти, помеченный другим типом. Наш метод направлен на решение этих проблем.

В данной работе мы разработали улучшенную концепцию ленивой инициализации, добавив в неё *временные метки* и *типы* (разд. 0). Эта концепция может быть использована в различных символьных виртуальных машинах, которые используют сегментированную символьную модель памяти. Далее, мы взяли символьную виртуальную машину KLEE [12], входящую в состав проекта LLVM, поскольку KLEE является эффективным инструментом генерации тестовых данных для программ с примитивными параметрами функций и широко используется в исследовательском и индустриальном сообществах. Мы выполнили собственную реализацию классической ленивой инициализации для KLEE, поскольку нам не удалось найти соответствующей открытой реализации, и улучшили её, добавив временные метки и типы (разд. 4). Эффективность концепции временных меток была исследована в ходе экспериментального исследования, выполненного на наборе реализаций различных структур данных на языке C (разд. 5). Мы также выполнили обзор близких к нашему исследованию работ (разд. 6).

Таким образом, основными результатами данной статьи является следующее.

1) Улучшенный метод ленивой инициализации путём добавления временных меток и типов.
2) Реализация механизма ленивой инициализации с метками времени и типами в символьной виртуальной машине KLEE.
3) Новый формат представления тестов в KLEE, позволяющий воспроизводить сценарии исполнения, в которых была применена ленивая инициализация.
4) Выполненные эксперименты по автоматическому тестированию сложных структур данных, реализованных на языке C — списков, биномиальных куч, AVL-деревьев, красно-чёрных деревьев, двоичных деревьев, боров (префиксных деревьев).

2. Символьное исполнение с ленивой инициализацией

При классическом символьном исполнении функций с простыми входными параметрами, такими как целые числа, в символьной памяти создаются простые символьные значения, и далее при работе с ними строятся символьные термы. Классическое символьное исполнение не масштабируется на функции со сложными входными параметрами (указатели на сложные структуры данных, такие как списки и деревья), т.к. параметр может указывать на любой

```
1. typedef struct List {
2.   struct List *next;
3.   int value;
4. } List;
5.
6. unsigned length(List *list, unsigned bound) {
7.   unsigned len = 0;
8.   for (List *p = list; p && bound; len++, bound--)
9.     p = p->next;
10.  return len;
11.}
12.
13.#define BOUND 2
14.#define SIZE 1
15.
16.int main() {
17.  int *array = make_concrete_array(10, sizeof(int));
18.  List *xs = make_symbolic(List);
19.  List *sing = make_concrete_list(SIZE);
20.
21.  unsigned length1 = length(xs, BOUND);
22.
23.  assert(length1 = 0 || (xs != sing && xs != array));
24.}
```

фрагмент в символьной памяти, который также может содержать символьные указатели. Ленивая инициализация [10] является методом, который решает эту проблему путём учёта (нумерации) возможных адресов в памяти, на которые может указывать символьный указатель.

Листинг 1. Фрагмент кода С-программы с функцией `length`*, имеющей сложный входной параметр (указатель на список)*

Listing 1. Code fragment in C representing `length` *function with a complex input parameter (a list pointer)*

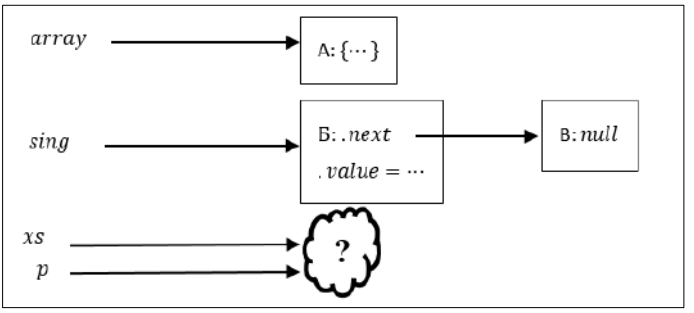


Рис. 1. Состояние символьной памяти после символьного исполнения строк 17-19 листинга 1
Fig. 1. Symbolic state after symbolic execution of 17-19 lines of code from listing 1.

Рассмотрим, как работает символьное исполнение с ленивой инициализацией на примере кода с листинга 1. Функция `main()` создаёт и инициализирует конкретный массив `array` при помощи функции `make_concrete_array`. Затем в строке 18 переменная `xs` инициализируется как символьный указатель на список, т.е. указатель на некоторую область в памяти, чьё значение нельзя определить статически – `xs` может указывать как на один из существующих выделенных фрагментов в памяти, так и некоторый новый. Далее создаётся и инициализируется список `sing` длины `SIZE`. Далее вычисляется ограниченная длина списка `xs` путём вызова функции `length`, которая проходит по ячейкам списка не более чем `BOUND` раз.

В процессе символьного исполнения строк 17-19 будет построено состояние символьной памяти, изображённое на рис. 1. Переменные `array` и `sing` указывают на конкретные фрагменты в памяти (выделенные прямоугольниками с метками «А», «Б» и «В»), а `xs` указывает на *лениво инициализируемую память* (представлена на рис. 1 в виде облака) – фрагмент памяти, который будет определён только при доступе.

Таким образом, в момент вызова функции `length` в строке 21 указатель `xs` оказывается недетерминированным. В строке 8 происходит разветвление процесса символьного исполнения по условию $p = 0$ $(null)$ для указателя p, на первой итерации равного xs (что также отражено на рисунке 1), как и при обычном символьном исполнении. Войдя в состояние с $p = xs = 0$, процесс символьного исполнения перейдёт к концу функции, вернёт значение 0 и тем самым отсечёт состояния, где `xs` указывает на блок памяти «В». Во втором состоянии ($xs \neq 0$) процесс символьного исполнения войдёт внутрь цикла, где будет выполнено разыменование лениво инициализируемой памяти.

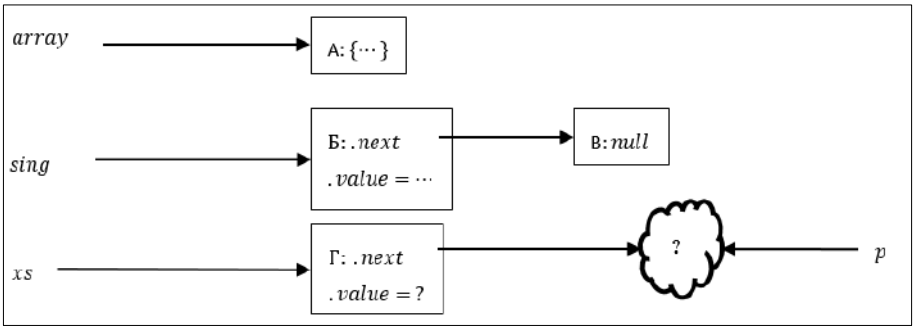


Рис. 2. Одно из состояний символьной памяти после символьного исполнения 17-21 и 7-9 строк кода с листинга
Fig. 2. One of symbolic states after symbolic execution of 17-19 and 7-9 lines of code from listing 1

Механизм ленивой инициализации работает следующим образом. При разыменовании указателя (`p->next`) в строке 9 будут созданы три символьных состояния, отличающиеся только тем, на что указывает `xs` – на массив (блок «А» на рис. 1), на существующий список (блок «Б» на рисунке 1) или на некоторый иной фрагмент памяти. Последнее символьное состояние программы представлено на рис. 2. В нём механизм ленивой инициализации создал новую структуру требуемого типа `List` и инициализировал все её поля символьными значениями примитивного типа (например, поле `value` в блоке «Г»), а все поля с типом указателя – новыми ленивыми значениями (поле `next`). Переменная `xs` по-прежнему указывает на начало списка, а переменная `p`, после выполнения инструкции в строке 9, будет указывать на `xs->next`. На следующей итерации цикла всё произойдёт аналогичным образом: указатель $p = xs \rightarrow next$ будет проверен на равенство нулю, а затем при

разыменовании `p` механизм ленивой инициализации породит четыре символьных состояния, отличающиеся тем, на что указывает `p` (`xs->next`): на массив (блок «А» на рисунке 2), на существующий список (блок «Б» на рис. 2), на порождённую предыдущей итерацией структуру (блок «Г» на рис. 2) или на некоторый иной фрагмент памяти.

Далее, если в символьной памяти находятся n объектов, то при разыменовании символьного указателя p будет создано $n + 1$ символьное состояние: в n состояниях p будет указывать на существующие объекты, а в последнем – на новый объект в памяти, все поля которого лениво инициализированы. Таким образом, данный метод расширяет символьное исполнение возможностью *систематически* анализировать все возможные состояния памяти программы с недетерминированным указателем.

3. Ленивая инициализация с временными метками и типами

Одной из проблем ленивой инициализации является комбинаторный взрыв числа путей символьного исполнения. После каждого разыменования недетерминированного указателя число исследуемых сценариев поведения программы увеличивается на число уже выделенных в динамической памяти объектов подходящего размера.

Снова вернёмся к листингу 1. В результате вызова функции `make_concrete_list` (строка 19) в символьной памяти выделится $SIZE$ объектов. Во время символьного исполнения функции `length` происходит разыменование символьного указателя `p` и при этом создаётся, как минимум, $SIZE + 1$ новых символьных состояний: перебираются объекты, созданные в функциях `make_concrete_array` и `make_concrete_list`. Полученные значения разыменовываются в следующей итерации цикла. Для сценария, в котором произошла ленивая инициализация объекта (см. рис. 2), разыменование недетерминированного указателя приведёт к увеличению числа рассматриваемых сценариев, теперь уже на $SIZE + 2$. Аналогичные действия произойдут в каждой следующей итерации цикла. Число итераций цикла равно $BOUND$, поэтому после исполнения в функции `main` вызова функции `length` число исследуемых сценариев исполнения увеличится на $O(SIZE \times BOUND)$. Таким образом, каждый последующий вызов функции `length` увеличивает количество исследуемых сценариев программы экспоненциально. Чтобы уменьшить число возможных сценариев исполнения, мы предлагаем дополнить модель символьной памяти *временными метками* и *информацией о типах объектов* в памяти. Оба этих улучшения позволяют отсечь некорректные сценарии поведения программы, которые в итоге дают невоспроизводимые тесты.

3.1 Временные метки

Объекты в символьной памяти выделяются в порядке исполнения инструкций исследуемого кода. Этот порядок гарантирует, что объекты, которые выделены раньше, не могут хранить в своей памяти адреса объектов, выделенных позже. В частности, недетерминированный указатель `xs` не может указывать на объект, который создан функцией `make_concrete_list` (например, отмеченный меткой «Б» на рисунке 1). Более того, каждый указатель, полученный из лениво инициализированных объектов в ходе исполнения функции `length` (например, `xs->next`), также не может указывать на конкретный объект, выделенный позже (например, отмеченный меткой «Б» на рисунке 2). Чтобы запретить такие разыменования, предлагается связывать каждый объект в символьной памяти с *временной меткой*, хранящей время его выделения. Метки упорядочены по возрастанию – чем позже создан объект, тем больше его временная метка.

Недетерминированные указатели имеют временные метки, равные временным меткам объектов, из которых значения этих указателей были прочитаны. Временные метки лениво инициализированных объектов равны временным меткам указателей, при разыменовании

которых они были инициализированы. Чтобы запретить разыменования указателей на объекты, созданные позже, при разыменовании недетерминированного указателя с меткой n во время перебора объектов памяти нужно отбрасывать те объекты, временные метки которых больше, чем n. Такой запрет отсекает некорректные сценарии поведения программы, тем самым ускоряя процесс символьного исполнения программы.

3.2 Информация о типах объектов

Спецификации некоторых языков могут гарантировать корректное поведение программ, в которых происходит разыменование указателей, только в том случае, если тип указателя и тип объекта в памяти совместимы друг с другом. Понятие совместимости в этом случае определяется в спецификации языка. Например, для языка C совместимость типов определяется правилами разыменования указателей, которые описаны в стандарте **Ошибка! Источник ссылки не найден.**. Таким образом, нарушение правил разыменования указателей в коде может привести к некорректному поведению программы. Если символьное исполнение будет учитывать эти правила, то это позволит отсекать некорректные сценарии исполнения программы и тем самым ослабить эффект экспоненциального взрыва числа исследуемых сценариев исполнения.

Поэтому кроме временных меток предлагается хранить типы объектов в памяти и указателей. При разыменовании недетерминированный указатель, во-первых, должен быть сопоставлен только с объектами подходящего типа, и, во-вторых, при ленивой инициализации нового объекта по этому указателю такому объекту должен быть присвоен тот же тип.

Вернемся к листингу 1. В строке 9 процесс символьного исполнения выполняет попытку разыменования недетерминированного указателя `xs` с типом `List`. При этом механизм ленивой инициализации должен рассмотреть все объекты с типом `List` и только их. Таким образом, во-первых, будет отсечен сценарий исполнения, в котором указатель сопоставляется объекту, созданному функцией `make_concrete_array` (например, отмеченный меткой «А» на рис. 1). Во-вторых, при ленивой инициализации нового объекта по указателю `xs`, полученному объекту (отмечен меткой «Г» на рисунке 2) будет присвоен тип `List`. Таким образом, на следующей итерации цикла при переборе объектов с этим типом полученный новый объект будет также учтен, поскольку он имеет подходящий тип. Заметим, что при одновременном использовании временных меток и типов проверка в строке 23 будет выполняться всегда.

Механизм ленивой инициализации и описанные в данном разделе улучшения могут быть реализованы в любой символьной виртуальной машине, основанной на *сегментированной символьной модели памяти* [11]. Так как в этой модели вся память разбита на непересекающиеся сегменты, соответствующие выделенным в памяти объектам, каждый сегмент памяти можно связать с временной меткой и типом, которые соответствуют этому сегменту. В следующем разделе будет описана реализация механизма ленивой инициализации с улучшениями в символьной виртуальной машине KLEE (основана на сегментированной символьной модели памяти).

4. Поддержка ленивой инициализации в KLEE

Мы использовали виртуальную символьную машину KLEE версии 2.3 [14], реализовав в ней механизм ленивой инициализации и оптимизации процесса работы с символьной памятью – временные метки и типы. Кроме того, были внесены модификации в процедуры генерации и воспроизведения тестов. Эти модификации описаны ниже.

Воспроизведение сгенерированных тестов в исходной версии KLEE реализовано следующим образом. Во время генерации теста KLEE находит подходящие значения для переменных, которые были объявлены символьными с помощью исполнения функции

`klee_make_symbolic`. Результатом генерации теста является файл *.`ktest`. В него записываются значения символьных переменных в том порядке, в котором произошли вызовы функции `klee_make_symbolic`. При воспроизведении теста вызовы функции `klee_make_symbolic` в исходной программе используются для записи подобранных значений в соответствующие переменные.

Однако тесты с символьными указателями не могут быть воспроизведены в оригинальной версии KLEE, поскольку во время генерации теста в символьные указатели записываются конкретные адреса, выделенные для указываемых объектов во время работы символьного исполнения. Эти адреса не соответствуют адресам объектов, выделяемых во время воспроизведения теста, так как эти адреса неизвестны на этапе генерации теста. В связи с этим мы изменили в KLEE процедуры генерации и воспроизведения тестов.

4.1 Модификация процедуры генерации тестов

В символьное состояние добавлено множество указателей: выражений, через которые происходит доступ к объектам в памяти. Эта информация используется во время генерации теста для того, чтобы определить, какие части символьных объектов представляют собой указатели, и на какие объекты эти указатели указывают. Информация об указателях впоследствии записывается в сгенерированный тест.

4.2 Модификация процедуры воспроизведения тестов

Во время воспроизведения теста информация об указателях используется для того, чтобы проинициализировать все необходимые объекты и их поля. Это происходит следующим образом. Все символьные объекты памяти инициализируются последовательно. Во время инициализации каждого объекта обрабатывается информация о его указателях. Все объекты, которые указывают на текущий объект и еще не инициализированы, инициализируются рекурсивно. У каждого инициализированного объекта запоминается его реальный адрес. После инициализации объекта его адрес записывается в соответствующее поле указывающего на него объекта. Таким образом все поля с типом указателя действительно указывают на нужные объекты в памяти.

Описанная реализация доступна по ссылке[1].

5. Эксперименты

Цель экспериментального исследования заключалась в проверке эффективности предложенного в статье метода. С этой целью сравнивались следующие инструменты: базовая версия KLEE 2.3, в которой ленивая инициализация отсутствует (KLEE-BASIC); версия KLEE 2.3 с реализованной нами классической ленивой инициализацией (CLI); версия KLEE 2.3 с классической ленивой инициализацией (KLEE-LI), а также предложенными в данной статье временными метками и типами (KLEE-LI-OPT). В качестве SMT-решателя был использован Z3 версии 4.9 [15].

Для экспериментов был разработан набор из 42-х тестовых С-программ. Эти программы инструментированы вызовом функции `klee_make_symbolic`, которая делает символьным объект, адрес которого передаётся в качестве аргумента. 27 программ содержат операции с указателями на следующие рекурсивные структуры данных: связный список, биномиальная куча, деревья (красно-чёрное, AVL, двоичное) и бор. В данных программах указатели на эти структуры делаются символьными. Остальные 15 программ содержат объекты примитивных структур данных и программы с явным выделением данных на куче посредством функции `malloc`. В них символьными делаются не указатели, а сами объекты в памяти. Это

[1] https://github.com/misonijnik/klee/tree/klee-2.3-li-opt

программы, на исполнение которых рассчитана обычная версия KLEE и которые требуются, чтобы показать, что наши оптимизации не ухудшают работу базового инструмента.

Для сравнения были выбраны следующие метрики: время работы инструмента на тестовой программе (в секундах), количество используемой при этом оперативной памяти (в мегабайтах) и процент покрытых инструментом инструкций кода тестовой программы. Процент покрытых инструкций измерялся с помощью инструмента gcov [16].

Эксперименты проводились на рабочей станции с процессором Intel Core I5-8265U с 16 ГБ оперативной памяти под управлением операционной системы Linux. Процесс символьного исполнения может не завершаться при исследовании рекурсивных и циклических программ, поэтому временные ограничения необходимы, чтобы отсечь зависания. Каждому запуску было задано ограничение 60 секунд. Для программ из набора выбранное время подобрано эмпирически: как правило, его хватает, чтобы исследовать большинство интересных сценариев исполнения. После истечения этого времени символьное исполнение следующей инструкции тестовой программы не может начаться, однако исполнение текущей инструкции обязано завершиться. Также часть времени после завершения исполнения тратится непосредственно на генерацию тестов. Из-за этого реальное время исполнения может быть больше заданного ограничения в 60 секунд. Во время символьного исполнения SMT-решатель может зависнуть, пытаясь выполнить запрос, поэтому было необходимо также ограничить его время работы. Время исполнения одного запроса SMT-решателем было ограничено 5 секундами. Это ограничение также было подобрано эмпирически для программ из предложенного набора: в нашем случае большинство завершающихся запросов завершают работу за указанное время.

Сводные результаты экспериментов представлены в табл. 1, детальное описание поведения каждого инструмента на каждой тестовой программе в соответствии с выбранными метриками представлено табл. 2 в Приложении.

Табл. 1. Сводные результаты экспериментов
Table 1. Summary results of experiment

Инструмент	**Время работы, с.**			**Оперативная память, МБ**			**Среднее покрытие кода, %**
	Мин.	**Макс.**	**Средн.**	**Мин.**	**Макс.**	**Средн.**	
KLEE-BASIC	0.02	98.8	42.3	17.3	106.5	41	48.3
KLEE-LI	0.02	130.7	55.7	17.3	124.3	35.1	81.9
KLEE-LI-OPT	0.02	128.1	43.2	17.3	120.1	28.4	86.9

Опишем результаты экспериментов.

Как следует из табл. 1, среднее время работы KLEE-LI-OPT (43.2 с) сопоставимо с KLEE-BASIC (42.3 с), при этом среднее время работы KLEE-LI оказывается больше, чем KLEE-LI-OPT — следовательно, предложенные оптимизации ленивой инициализации приводят к ускорению последней. Однако верхняя граница времени работы у KLEE с ленивой инициализацией больше, что связано с появлением новых исследуемых сценариев исполнения. Потребление оперативной памяти с применением ленивой оптимизации уменьшается, что связано с тем, что версия KLEE-BASIC тратит существенные ресурсы на исследование сценариев исполнения, в которых указатели разыменовываются в глобальные объекты памяти. Такие объекты имеют большой размер и занимают много места. При каждом разыменовании указателя для записи в память KLEE-BASIC копирует объект, на который указатель был разыменован. В дальнейшем KLEE-BASIC тратит значительное время на исследование таких сценариев исполнения. Из-за этого копирование глобальных объектов происходит чаще, чем при ленивой инициализации. Вместо этого KLEE с ленивой инициализацией большую часть времени исследуют сценарии исполнения с лениво инициализированными объектами, размер которых часто меньше, чем размер глобальных объектов. Более того, наши оптимизации ещё больше уменьшают используемую оперативную память. Однако на ряде тестов KLEE-LI-OPT показывает потребление памяти,

превосходящее KLEE-BASIC, поскольку для символьного исполнения с оптимизациями необходимо хранить в памяти дополнительную информацию о временных метках и типах. Из-за этого для простых программ, на которых KLEE-LI-OPT не даёт выигрыша во времени исполнения и покрытии кода, эта версия использует больше памяти, чем версия KLEE-BASIC. Наконец, KLEE-LI-OPT почти в два раза увеличивает покрытие кода в тестовых C-программах по сравнению с KLEE-BASIC: ленивая инициализация позволяет исследовать недостижимые для KLEE-BASIC сценарии исполнения программы, а оптимизации позволяют отсекать некорректные сценарии исполнения, снижая влияние проблемы экспоненциального взрыва числа исследуемых сценариев.

Таким образом, KLEE-BASIC не может эффективно анализировать программы, в которых происходит разыменование недетерминированных указателей на рекурсивные структуры данных, что выражается в низком проценте кодового покрытия на тестовых программах со связными списками, биномиальными кучами, деревьями и борами. Представленные оптимизации в версии KLEE-LI-OPT, в свою очередь, позволяют отсеять заведомо невоспроизводимые варианты исполнения программы, тем самым уменьшить время работы инструмента и увеличить процент тестового покрытия.

6. Близкие работы

6.1 Символьное исполнение с ленивой инициализацией

Идея ленивой инициализации в символьном исполнении была предложена в 2003 году [10] и с тех пор была реализована в различных символьных виртуальных машинах. К примеру, работа [17] описывает метод ограниченной ленивой инициализации, реализованный в инструменте SPF [18] для языка Java. В отличие от классической версии, количество лениво инициализированных объектов одного типа здесь зафиксировано заранее. Во время разыменования символьного указателя выполняется перебор заранее фиксированных вариантов, т.н. «плотных границ» (tight bounds), полученных с помощью «плотного анализа полей» (tight field analysis). «Плотные границы» хранятся как некоторое отношение на объектах, которые возможно лениво инициализировать, и результат разыменования указателя на объект должен удовлетворять этому отношению. В статье также представлен механизм, позволяющий избежать порождения изоморфных структур. Метод с ограниченной ленивой инициализацией улучшен в [19]. Однако в этих работах не представлены методы порождения исполняемых тестов на базе инстанцированных структур данных.

Идея ленивой инициализации использовалась при создании логики HEX [20] – языка спецификации пред- и постусловий для Java, удобного для работы с динамической памятью. Пользовательские предусловия позволяют уменьшить пространство перебора объектов при ленивой инициализации. Однако необходимость вручную описывать инварианты структур данных является существенным ограничением этого метода.

6.2 Тестирование структур данных в KLEE

Символьная виртуальная машина KLEE [12] является очень популярной в академии и индустрии, и уже были сделаны попытки поддержать в ней обработку сложных структуры данных. Самым близким к нам является инструмент UC-KLEE [21], который реализует фрагмент механизма ленивой инициализации, позволяющий символьное исполнение кода с входными структурами данных без алиасинга (т.е. не содержащим двух ссылок на один и тот же участок памяти). К сожалению, нам не удалось найти исходный код или публично доступную рабочую версию UC-KLEE.

Другая недавняя работа [22] фокусируется на множестве стратегий смягчения комбинаторного взрыва путей исполнения и также реализует ограниченный вариант ленивой инициализации, напоминая UC-KLEE. Однако наш метод, в отличие от UC-KLEE, может

быть применён для тестирования циклических структур данных и структур с алиасингом (например, графов).

7. Заключение

Мы представили нашу реализацию инструмента автоматического символьного исполнения LLVM-программ со сложными входными структурами данных и продемонстрировали его эффективность для реальных структур данных на языке C. Также был разработан новый формат представления KLEE-тестов с инициализацией структур данных, при этом была переработана вся инфраструктура KLEE для поддержки этого нового формата. Это позволило значительно улучшить качество кодового покрытия программ с указателями на рекурсивные структуры данных по сравнению с оригинальной версией KLEE.

В дальнейшем наша реализация может использоваться как по прямому назначению (генерация тестовых данных), так и для других академических экспериментов (вычисление слабейших предусловий, обратное символьное исполнение, вывод инвариантов циклов и т.д.).

Список литературы / References

[1] Korel B., Al-Yami A.M. Automated Regression Test Generation. ACM SIGSOFT Software Engineering Notes, vol. 23, issue 2, 1998, pp. 143-152.

[2] Cadar C., Sen K. Symbolic execution for software testing: three decades later. Communications of the ACM, vol. 56, issue 2, 2013, pp. 82-90.

[3] Baldoni R., Coppa E. et al. A survey of symbolic execution techniques. ACM Computing Surveys (CSUR), vol. 51, issue 3, 2018, pp. 1-39.

[4] Barrett C., Tinelli C. Satisfiability Modulo Theories. In Handbook of Model Checking, Springer, 2018, pp. 305-343.

[5] Volanschi N., Lawall J. The impact of generic data structures: decoding the role of lists in the linux kernel. In Proc. of the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2020, pp. 103-114.

[6] Corbet J. Trees II: red-black trees. URL: https://lwn.net/Articles/184495/, 2006.

[7] Bacthu N., Banerjee A. et al. Dynamic and compressed trie for use in route lookup. United States Patent Application 20180212876, URL: https://www.freepatentsonline.com/20180212876.pdf, 2018.

[8] Braione P., Denaro G. et al. Combining symbolic execution and search-based testing for programs with complex heap inputs. In Proc. of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2017), 2017, pp. 90-101.

[9] Galeotti J.P., Rosner N. et al. Analysis of invariants for efficient bounded verification. In Proc. of the 19th International Symposium on Software Testing and Analysis (ISSTA '10), 2010, pp. 25-36.

[10] Khurshid S., Puasuareanu C.S., Visser W. Generalized symbolic execution for model checking and testing. In Proc. of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems, 2003, pp. 553-568.

[11] Kapus T., Cadar C.A segmented memory model for symbolic execution. In Proc. of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2019, pp. 774-784.

[12] Cadar C., Dunbar D., Engler D. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs. In Proc. of the 8th USENIX Conference on Operating Systems Design and Implementation, 2008, pp. 209-224.

[13] ISO/IEC 9899:201x - N1570 - Programming languages – C. URL: https://web.cs.dal.ca/~vlado/pl/C_Standard_2011-n1570.pdf.

[14] Cadar C., Dunbar D. KLEE. URL: https://github.com/klee/klee/tree/v2.3, 2022.

[15] de Moura L., Bjorner N. Z3: An Efficient SMT Solver. Lecture Notes in Computer Science, vol. 4963, 2008, pp. 337-340.

[16] Gough B., Stallman R. M. An Introduction to GCC for the GNU Compilers gcc and g++. Network Theory Ltd, 2004, 144 p.

[17] Geldenhuys J., Aguirre N. et al. Bounded Lazy Initialization. Lecture Notes in Computer Science, vol. 7871, 2013, pp. 229-243.
[18] Puasuareanu C. S., Rungta N. Symbolic PathFinder: symbolic execution of Java bytecode. In Proc. of the IEEE/ACM International Conference on Automated Software Engineering, 2010, pp. 179-180.
[19] Rosner N., Geldenhuys J. et al. BLISS: improved symbolic execution by bounded lazy initialization with SAT support. IEEE Transactions on Software Engineering, vol. 41, issue 7, 2015, pp. 639-660.
[20] Braione P., Denaro G., Pezze M. Symbolic Execution of Programs with Heap Inputs. In Proc. of the 10th Joint Meeting on Foundations of Software Engineering, 2015, pp. 602-613.
[21] Ramos D. A., Engler D. Under-Constrained Symbolic Execution: Correctness Checking for Real Code. In Proc. of the 24th USENIX Security Symposium (USENIX Security 15), 2015, pp. 49-64.
[22] Rutledge R., Orso A. PG-KLEE: Trading Soundness for Coverage. In Proc. of the IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), 2020, pp. 65-68.

Информация об авторах / Information about authors

Александр Владимирович МИСОНИЖНИК. Получил степень магистра в области информационных технологий в Санкт-Петербургском государственном университете в 2021 году. Его исследовательские интересы включают методы эффективного поиска недостижимых состояний в символьном анализе программ.

Alexander Vladimirovich MISONIZHNIK. Received his master's degree in computer science at St. Petersburg State University in 2021. His research interests include efficient pruning of unreachable states in symbolic program analysis.

Алексей Александрович БАБУШКИН. Получил степень бакалавра в области математики и компьютерных наук в Санкт-Петербургском государственном университете в 2022 году. Его исследовательские интересы включают методы анализа программ и автоматической генерации тестов.

Alexey Alexandrovich BABUSHKIN. Received his bachelor's degree in mathematics and computer science at St. Petersburg State University in 2022. His research interests include program analysis and automatic test generation.

Сергей Антонович МОРОЗОВ. Студент 3 курса бакалавриата “Прикладная математика и информатика” в Национальном исследовательском университете “Высшая школа экономики” в Санкт-Петербурге. Его исследовательские интересы включают методы анализа программ и оптимизации символьного исполнения.

Sergey Antonovich MOROZOV. A 3rd-year undergraduate student in Applied Mathematics and Computer Science at the National Research University Higher School of Economics in St. Petersburg. His research interests include methods of program analysis and symbolic execution optimization.

Юрий Олегович КОСТЮКОВ, аспирант кафедры системного программирования Санкт-Петербургского государственного университета, получил степень магистра в области информационных технологий в Санкт-Петербургском государственном университете в 2021 г. Его исследовательские интересы включают проблему автоматического вывода индуктивных инвариантов и автоматическое порождение тестовых покрытий.

Yurii Olegovich KOSTYUKOV, PhD student of the Department of System Programming at St. Petersburg State University, received his master's degree in computer science at St. Petersburg State University in 2021. His research interests include automatic inductive invariant inference and automatic test coverage generation.

Дмитрий Александрович МОРДВИНОВ, кандидат физ.-мат. наук, доцент кафедры системного программирования Санкт-Петербургского государственного университета. Область его научных интересов включает формальную верификацию, синтез программ и решение систем дизъюнктов Хорна.

Dmitry Alexandrovich MORDVINOV, PhD in Physics and Mathematics, Associate Professor at the Department of System Programming of St. Petersburg State University. His research interests include formal verification, program synthesis, and constraint Horn clause solving.

Дмитрий Владимирович КОЗНОВ, доктор технических наук, профессор кафедры системного программирования. Научные интересы: программная инженерия, модельно-ориентированная разработка программного обеспечения, программные данные, машинное обучение.

Dmitry Vladimirovich KOZNOV, Doctor of Technical Sciences, Professor of the System Programming Department. Research interests: software engineering, model-driven software development, program data, machine learning.

Приложение

Табл. 2. Расширенное сравнение стандартной версии KLEE (KLEE-BASIC), KLEE с ленивой инициализацией (KLEE-LI) и KLEE с ленивой инициализацией и оптимизациями (KLEE-LI-OPT)
Table 2. Extended comparison of the standard version of KLEE (KLEE-BASIC), KLEE with lazy initialization (KLEE-LI), and KLEE with lazy initialization and optimizations (KLEE-LI-OPT)

Тестовые С-программы	**Время работы, с**			**Оперативная память, МБ**			**Процент покрытого кода,%**		
	KLEE-BASIC	KLEE-LI	KLEE-LI-OPT	KLEE-BASIC	KLEE-LI	KLEE-LI-OPT	KLEE-BASIC	KLEE-LI	KLEE-LI-OPT
abs.c	0.03	0.04	0.04	20	20	20	100	100	100
aggregate.c	0.02	0.02	0.02	17.3	17.3	17.4	100	100	100
array_equality.c	0.02	0.02	0.02	17.3	17.3	17.3	100	100	100
array_sum.c	0.02	0.03	0.02	17.4	17.4	17.4	100	100	100
arraylist.c	3	0	0.05	20.5	22.3	17.5	0	88	88
avl_balance.c	62.4	196	77.7	42	31	32.31	0	29	29
avl_find.c	72	222	90	44.6	36.5	33.4	66.7	83	83
avl_height.c	68.8	182	78.5	47.9	29.4	26.3	0	76.9	76
avl_insert.c	61	258	98	32	40	40	60.9	69.6	69
binomial_heap.c	69.7	79.3	61.5	43.4	38	30	0	91.2	95
boolean.c	0.04	0.06	0.04	17.3	17.4	17.4	100	100	100
get_sign.c	0.03	0.04	0.03	17.3	17.4	17.4	100	100	100
integer_series.c	7.8	0.00	56.1	21.3	26.8	23.6	0	0	100
list_nonempty.c	60.3	63.9	124	60	45.9	30.3	0	100	100
list_pointer_access.c	65.7	98.3	0.8	61.5	45.04	18.4	0	100	100
list_size.c	61	127	60.3	60.3	43.5	28.9	77.8	88.9	100
list.c	69.4	61.5	70.2	53.4	41.3	30	100	100	100

malloc_big.c	0.02	0.02	0.02	17.3	17.3	17.3	100	100	100
malloc.c	0.02	0.02	0.02	17.3	17.3	17.3	100	100	100
manyvar.c	0.05	0.06	0.05	17.4	17.4	17.4	100	100	100
narrow.c	0.02	0.02	0.02	20	20	20	57.1	57.1	57
queue_head_peek_pop.c	61.2	108	0.2	56	46	18	0	90.5	95
queue_peek_head_tail.c	59.9	101	0.3	59.54	47	21	0	78.6	100
queue_push_pop.c	73.2	67.1	0.02	57.6	40.3	17.5	0	94	93
rb_grandparent.c	61	157	78.5	30.8	32.7	33.5	0	87.5	87
rb_insert_find.c	68	222	92.3	42.47	35.6	34	0	39.8	39
rb_remove.c	98.8	210	86	57.6	38.3	33.5	0	9.7	9
recursive.c	61	60	61	106	124	120	100	100	100
regexp.c	11.2	60	13.7	42	66.3	66	100	100	100
simple.c	3	0	2.2	20.6	21.8	17.8	0	100	100
sort.c	0.02	0.03	0.02	17.5	17.6	17.6	87.5	87.5	87
structs.c	0.02	0.03	0.02	17.3	17.4	17.4	100	100	100
tree_bfs.c	62.1	77.5	123.3	60.4	43.3	29.3	80	100	100
tree_count_nodes.c	62.1	209	87.2	30.3	38	35.61	80	90	90
tree_delete_find.c	68.1	66.3	68.8	70	45	47	28.9	52.6	89
tree_find_height.c	71	76.4	128	58.4	32	30	63.6	91	90
tree_insert_find.c	63	60.7	61	59.7	40.5	30.3	60	92	100
tree_makeempty.c	63	134	98.5	54.8	40.8	29.3	66.7	100	100
trie_insert_remove.c	61	61	60.5	41.7	44.5	29.8	0	22.4	25
trie_lookup.c	78.5	61	64.5	57.7	42.3	29.2	0	81.2	93
trie_num_entries.c	67.2	69	0.07	58.5	39.4	17.5	0	100	100
trie_remove.c	81.5	68.3	68.6	58.1	45.4	31	0	38.2	44

DOI: 10.15514/ISPRAS-2022-34(4)-5

Обнаружение ошибок взаимоисключающей блокировки в программах на языке C# при помощи методов статического анализа

П.И. Рагозина, ORCID: 0000-0003-4219-7203 <pragozina@ispras.ru>
В.Н. Игнатьев, ORCID: 0000-0003-3192-1390 <valery.ignatyev@ispras.ru>

Институт системного программирования им. В.П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25
Московский государственный университет им. М.В. Ломоносова,
119991, Россия, Москва, Ленинские горы, д. 1

Аннотация. В статье рассматриваются алгоритмы статического анализа, которые направлены на поиск трех типов ошибок, связанных с понятием синхронизирующего монитора: переопределение переменной взаимоисключающей блокировки внутри критической секции; использование переменной некорректного типа при входе в монитор; блокировка с задействованием объекта, имеющего методы, которые используют для блокировки ссылку на экземпляр (this). Разработанные алгоритмы опираются на технологию символьного исполнения и опираются на межпроцедурный анализ с применением резюме функций, что обеспечивает масштабируемость, чувствительность к полям, контексту, потоку управления. Полученные методы реализованы в инфраструктуре статического анализатора SharpChecker, использующего элементы компиляторной платформы Roslyn, в виде трёх детекторов. С их помощью при тестировании на проектах с открытым кодом найдено 23 ошибки и получена доля верных срабатываний в 88.5%, в то время как потребление времени каждым детектором составляет от 0.1 до 0.7% от общего времени работы анализатора. Ошибки, для поиска которых были разработаны данные детекторы, сложно обнаружить другими способами, помимо статического анализа, из-за того, что они тесно связаны с понятием многопоточности. При этом находить их необходимо: всего один подобный дефект может привести к нестабильности работы программы и даже сделать её уязвимой для злоумышленников.

Ключевые слова: статический анализ; поиск дефектов; символьное исполнение; язык C#; ошибки синхронизации; критическая секция; межпроцедурный анализ.

Для цитирования: Рагозина П.И., Игнатьев В.Н. Обнаружение ошибок взаимоисключающей блокировки в программах на языке C# при помощи методов статического анализа. Труды ИСП РАН, том 34, вып. 4, 2022 г., стр. 63-78. 10.15514/ISPRAS-2022-34(4)-5

Detection of erroneous usage of synchronization monitor in C# via static analysis

P. Ragozina, ORCID: 0000-0003-4219-7203 <pragozina@ispras.ru>
V. Ignatyev, ORCID: 0000-0003-3192-1390 <valery.ignatyev@ispras.ru>

Ivannikov Institute for System Programming of the RAS,
25, Alexander Solzhenitsyn Str., Moscow, 109004, Russia
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russian Federation

Abstract. The paper describes static analysis algorithms aimed at finding three types of errors related to the concept of a synchronizing monitor: redefinition of a variable of mutually exclusive locking inside a critical section; use of an incorrect variable type when entering the monitor; blocking involving an object that has methods that use a reference to an instance (this) to lock. Developed algorithms rely on symbolic execution technology and involve interprocedural analysis via summary of functions, which ensures scalability, field-, context-, and flow-sensivity. Proposed methods were implemented in the infrastructure of a static analyzer in the form of three separate detectors. Testing on the set of open source projects revealed 23 errors and the true positive ratio of 88.5% was obtained, while the time consumption only made up from 0.1 to 0.7% of the total analysis time. The errors that these detectors were designed to find are difficult to detect by testing or dynamic analysis because of their multithreading nature. At the same time, it is necessary to find them: just one such defect can lead to incorrectness of the program and even make it vulnerable to intruders.

Keywords: static analysis; software error detection; symbolic execution; C# language; synchronization errors; critical section; inter-procedural analysis.

For citation: Ragozina P., Ignatyev V. Detection of erroneous usage of synchronization monitor in C# via static analysis. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 4, 2022. pp. 63-78 (in Russian). DOI: 10.15514/ISPRAS-2022-34(4)-5

1. Введение

В многопоточном программировании взаимоисключающая блокировка – это механизм, обеспечивающий синхронизацию данных. Синхронизация играет важную роль в обеспечении безопасности кода, так как её отсутствие или неправильное функционирование могут привести к *состоянию гонки* (race condition) – серьёзная ошибка, при которой работа системы или приложения зависит от того, в каком порядке выполняются разными потоками части кода.

С помощью традиционных методов диагностики ошибки блокировки трудно обнаружить и исправить ввиду осложнений воспроизводимости, вызываемых многопоточностью. Так, например, состояние гонки сделает ненадёжным применение отладки с помощью тестов, особенно в ситуациях, когда проблема возникает лишь для малой доли потоков при строго определенных условиях – тогда вероятность воспроизведения ошибки также значительно уменьшается, и возможна ситуация, когда программист просто не сможет воспроизвести возникшую у пользователя проблему. В качестве решения возможно создание значительно более громоздких тестов для повышения вероятности обнаружения значений, но даже тогда дефект может быть пропущен и не исправлен. Намного эффективнее оказываются методы статического анализа, так как они не требуют реального выполнения проверяемой программы.

Частым способом защитить данные при многопоточном программировании является создание *критической секции* – фрагмента кода, который в один момент времени может выполняться только одним потоком. В работе рассмотрен способ поиска ошибок в характерном для языка C# способе реализации критической секции – при помощи некоторой переменной синхронизации, выполняющей роль мьютекса для процессов. Изменение значения такой переменной внутри критической секции недопустимо, так как может

привести к её использованию несколькими потоками одновременно, а следовательно, и к состоянию гонки.

```
1     class A
2     {
3        static object obj = new object();
4        static int a = 0;
5        public static void Proc()
6        {
7           lock (obj)
8           {
9              obj = new object(); // !!!
10             a++;
11          }
12       }
13    }
```

Рис. 1. Простейший пример ошибки переопределения переменной блокировки
Fig. 1. Simple example of lock reassignment error

Рассмотрим пример на рис. 1. Если один из потоков, выполняющих функцию Proc, поменяет значение obj внутри критической секции (строка 9), то другой поток сможет попасть внутрь занятой первым процессом секции, так как новый объект-мьютекс obj, в отличие от старого, свободен. В свою очередь, это может привести к одновременному выполнению потоками строки 10 и некорректной работе программы.

```
 1    public class A
 2    {
 3         public void fa()
 4       {
 5          lock (typeof(int)) // !!!
 6          {
 7             /* Некоторый код */
 8             lock (typeof(float)) // !!!
 9             {
10             }
11          }
12       }
13       }
14    public class B
15    {
16         public void fb()
17       {
18          lock (typeof(float)) // !!!
19          {
20             /* Некоторый код */
21             lock (typeof(int)) // !!!
22             {
23             }
24          }
25       }
26       }
```

Рис. 2. Пример ошибочного типа переменной блокировки (System.Type)
Fig. 2. Simple example of improper lock object type (System.Type)

Также важно, чтобы используемая переменная сама не была общим ресурсом (к примеру, типом или строковой константой) – это может стать источником уязвимостей и взаимоблокировок. Рассмотрим пример на рис. 2. Здесь, если функция fb будет запущена одновременно с fa, то это приведет к взаимной блокировке (deadlock), так как ни одна из них не сможет продолжить работу без занятого другой ресурса.

Частный случай общего ресурса как объекта блокировки – ссылка на текущий экземпляр класса (this), так как экземпляр может быть легко получен методом извне. Такое решение становится особенно опасным, когда объект типа, имеющий функции с такой блокировкой, сам используется как аргумент lock в другом участке программы. Рассмотрим случай, изображенный на рис. 3. Здесь любой сторонний класс A может приостановить работу функции fb(), потенциально блокируя выполнение всех методов класса B, использующих lock(this).

```
 1     public class B
 2     {
 3     public void fb()
 4        {
 5           lock (this) // !!!
 6           {
 7              /* Некоторый код */
 8           }
 9        }
10   }
11   public class A
12   {
13      public void fa(B b)
14      {
15         lock (b) // !!!
16         {
17            Thread.Sleep (100000);
18         }
19      }
20   }
21      public class C
22      {
23      public void fc()
24      {
25           A a = new A();
26         B b = new B();
27         Task.Factory.Run(() => a.fa(b));
28         Task.Factory.Run(b.fb);
29      }
30   }
```

Рис. 3. Пример недопустимого задействования в мониторе объекта с методом, применяющим блокировку на this
Fig. 3. Example of invalid lock object with a method using 'this' as lock

Данная работа посвящена задаче обнаружения в программах, написанных на языке C#, подобных ошибок.

2. Механизм синхронизации

Основной способ реализации критической секции в программах на языке C# – при помощи класса System.Threading.Monitor, использующего в качестве мьютекса для её защиты некоторый объект. Основные методы – void Monitor.Enter(object obj), обеспечивающий вход

в критическую секцию, защищенную с помощью объекта obj, и void Exit(object obj), обеспечивающий освобождение obj и выход из секции [1]. Также можно вызывать Monitor.Enter с дополнительным булевым параметром ref _lockWasTaken. Тогда значение true этого параметра означает, что блокировка была успешно выполнена. Однако более распространённым инструментом является оператор lock(object obj), созданный для упрощения работы с классом Monitor. Он представляет собой сочетание Monitor.Enter(obj, ref _lockWasTaken), Monitor.Exit(obj) и блока try … catch … finally. В finally при истинности _lockWasTaken вызывается Monitor.Exit. Таким образом, блокировка освобождается, даже если возникает исключение в теле оператора. При анализе кода на этапе построения графа потока управления эти два способа создания критической секции будут аналогичны (табл. 1). При этом важно отметить, что при входе в монитор с использованием объекта obj роль мьютекса исполняет не переменная, а сам объект, на который она ссылается.

Табл. 1. Эквивалентное представление оператора lock
Table 1. Equivalent representation of lock operator

Функция, использующая оператор lock	**Аналогичная функция, использующая метод Monitor**

`public void f()`
`{`
`  object obj = new object();`
`  lock (obj)`
`  {`
`  /* Некоторый код */`
`  }`
`}`
	`public void f()`
	`{`
	`  object obj = new object();`
	`  bool _lockWasTaken = false;`
	`  try`
	`  {`
	`  Monitor.Enter(obj, ref _lockWasTaken);`
	`  /* Некоторый код */`
	`  }`
	`  finally`
	`  {`
	`  if (_lockWasTaken)`
	`  Monitor.Exit(obj);`
	`  }`
	`}`

Ещё один метод синхронизации – применение разделяемых ресурсов с использованием классов System.Threading.ReaderWriterLock [2] и System.Threading.ReaderWriterLockSlim [3]. Оба они позволяют получить доступ к ресурсу одному записывающему процессу (writer/"писатель") и нескольким считывающим (readers/"читатели"). При этом "читатели" могут получать доступ к ресурсу только в том случае, если в текущий момент "писатель" его не блокирует. Их принципы работы очень сходны между собой: создается объект класса, затем выполняется работа с ресурсами при помощи методов этого класса. Разница состоит в том, что ReaderWriterLockSlim проще в использовании и позволяет избежать ряда случаев, приводящих к тупикам. В связи с этим ReaderWriterLock применяется значительно реже и в данной статье рассматриваться не будет.

Основные методы класса ReaderWriterLockSlim:

- EnterReadLock(), TryEnterReadLock(), ExitReadLock() – для блокировки на чтение;
- EnterWriteLock(), TryEnterWriteLock(), ExitWriteLock() – для блокировки на запись;
- EnterUpgradableReadLock(), TryEnterUpgradableReadLock(), ExitUpgradableReadLock() – для обновляемой блокировки на чтение. Обновляемая блокировка аналогична блокировке на чтение за исключением того, что позднее она может быть расширена до блокировки на запись.

Так же, как и в случае с обычной критической секцией, используемый в блокировке объект (в данном случае типа ReaderWriterLockSlim) нельзя переопределять.

Кроме того, по уже рассмотренным ранее причинам (см. рис. 3), нежелательными аргументами являются объекты, являющиеся общими ресурсами, неограниченно доступными из других участков программы. К таким объектам относятся [4]:

- объекты Type;
- строковые константы и интернированные строки (interned string);
- ссылки на текущий экземпляр класса (this).

Их опасность в том, что любой метод – в ряде случаев даже принадлежащий сторонней программе – может бесконтрольно получить доступ к общедоступному объекту блокировки, заблокировать с его помощью (возможно, умышленно) необходимый ресурс и спровоцировать взаимную блокировку (deadlock).

3. Особенности реализации

3.1 Статический анализатор

Как уже указано выше, работа над детекторами выполнена в рамках разработки *SharpChecker* – инструмента статического анализа для программ на языке C#, основанного на компиляторной платформе Microsoft .NET Compiler Platform (Roslyn) [5]. Roslyn — компиляторная инфраструктура с открытым исходным кодом для языков C# и Visual Basic, предоставляющая интерфейсы для компиляции, анализа, рефакторинга кода. В нашем случае, Roslyn предоставляет АСД, таблицу символов и основу для ГПУ, а SharpChecker использует их для построения графов потока управления всех методов и дальнейшего анализа. Этот процесс происходит следующим образом [6][7].

A. Проводится разбор предоставленного синтаксического дерева. Производится анализ всех возможных явных и неявных вызовов. Строится иерархия наследования для классов. Выполняется поиск синтаксических ошибок, которые возможно обнаружить с помощью разбора АСД.
B. Строится статический граф вызовов.
C. Производится обход графа от вызываемой функции к вызывающей. Если в нём есть циклы, то они разрываются в произвольном месте. Для каждой функции выполняется анализ на основе символьного выполнения при помощи разработанного в SharpChecker движка статического символьного выполнения с объединением состояний. Выполняют свою работу детекторы, чувствительные к путям. При анализе часть собранной информации сохраняется в резюме метода для использования в методах, которые вызывают его. Анализ происходит параллельно.
    a. Производится построение ГПУ для каждой функции.
    b. Производится обход базовых блоков. Движок анализа генерирует используемые впоследствии события различных типов, например, присваивание значения в переменную или разыменование переменной.
    c. Детекторы, подписываясь на указанные события, могут накапливать необходимые данные и проводить проверки.
D. Вызывается обработчик завершения анализа, выдающий предупреждения на основе собранной информации.

3.2 Символьное выполнение

Рассмотрим подробнее основной метод статического анализа, обеспечивающий внутрипроцедурный анализ и активно применяемый в разработанных детекторах. Каждая функция при символьном выполнении полагается точкой входа в программу. При её непосредственном запуске параметры и состояние кучи могут быть произвольными, поэтому начальное состояние параметризуется набором символьных переменных. Множество способов запуска функции можно получить путем их замены на всевозможные значения.

Основная цель метода символьного выполнения – построение для любого выбранного пути ГПУ символьного состояния, то есть множества выражений, которое при замене символьных переменных на конкретные значения будут соответствовать верным результатам выполнения операторов функции.

Такой метод позволяет "смоделировать" работу функции для всевозможных вариантов ветвления, не запуская её, однако у него есть свои ограничения. В частности, требуется строго определить набор возможных путей. Для этого вводится понятие графа развёртки – ациклического графа, полученного из ГПУ путём допущения некоторого максимального числа k итераций циклов и представление всех возможных при таком ограничении участков кода, достигаемых через обратные рёбра, в виде отдельных подграфов, которые достигаются с помощью только прямых ребер. Такой метод представляет собой аппроксимацию и снижает находимое число ошибок, однако потенциально почти бесконечное число путей сделало бы символьное выполнение малоэффективным.

3.3 Детекторы

3.3.1 LockObjectReassignment

Для обнаружения ошибок, вызванных переопределением мьютекса блокировки, использование анализа синтаксического дерева возможно в некоторых частных случаях, однако в большинстве ситуаций синтаксического анализа недостаточно, так как он не является межпроцедурным и не может гарантировать обязательную для ошибки подобного типа чувствительность к контексту, потокам и полям, а также желательную чувствительность к путям (см. рис. 4 и 5).

```
  1     var rwls = new ReaderWriterLockSlim();
  2     rwls.EnterReadLock();
  3     try
  4     {
  5        rwls = new ReaderWriterLockSlim();
// !!!
  6        /*Некоторый код*/
  7     }
  8  finally
  9  {
10     rwls.ExitReadLock();
11  }
```

Рис. 4. Пример кода с ошибкой переопределения объекта ReaderWriterLockSlim
Fig. 4. Example of erroneous ReaderWriterLockSlim reassignmemt

```
1     class A
 2    {
 3         public object x;
 4    }
 5    class B
 6    {
 7       void f(A firstA, A secondA)
 8       {
 9          lock (secondA.x)
10          {
11             firstA = secondA;
12             firstA.x = new object();
//  !!!
13          }
14       }
15    }
```

Рис. 5. Пример кода, в котором нечувствительный к полям анализатор не обнаружит ошибку (после приравнивания в строке 11, firstA ссылается на тот же объект, что и secondA, и поле firstA.x становится эквивалентно полю secondA.x)

Fig. 5. Example code in which the field-insensitive analyzer will not detect an error (after assignment in line 11, firstA refers to the same object assecondA, and field firstA.x becomes equivalent to field secondA.x)

Таким образом, метод разбора AST способен обнаружить явное переопределение переменной, но не неявное. Более подходящим оказался метод символьного выполнения.

Поиск ошибки реализован следующим образом: в контексте каждого базового блока создаётся некоторый словарь, по которому можно получить информацию обо всех переменных, которые хотя бы на одном из путей ГПУ были задействованы в незавершённой синхронизации (lock, ReaderWriterLockSlim) и их возможных значениях с учётом прошлых переопределений.

```
1     static object obj;
2     static int num;
3     lock (obj)
4     {
5          if (num<10)
6          if (num>11)
7             obj = new object;
8     }
```

Рис. 6. Здесь нечувствительным к путям анализатором будет выдано ложное сообщение об ошибке в строке 7 (строка 7 недостижима)

Fig. 6. In this code, a path-insensitive analyzer would issue a false error message in line 7 (line 7 is unreachable)

Если несколько ветвей передают управление в один блок, то словарь в этом блоке будет объединением словарей во всех возможных блоках-предшественниках.

Среди событий, регистрируемых при анализе потока данных, фиксируются следующие.

- События вызова функции (чтобы при вызове функций, связанных с критическими секциями, корректировать словарь, а для остальных – сравнивать значения переменных из словаря до и после выполнения, обнаруживая переопределение переменной внутри данной функции).
- События присваивания – чтобы выявлять выражения, для которых с левой стороны

находится задействованный в синхронизации элемент.

- События объединения контекстов – для работы со словарями переменных из контекстов блоков.

С учётом этих данных выводятся сообщения об ошибках и сохраняются данные о новых значениях объектов блокировки, чтобы избежать лишних ложных срабатываний.

3.3.2 WrongTypeLock

При решении задачи поиска переменных-мьютексов неправильного типа (за исключением ссылок на экземпляр this, о которых будет рассказано ниже) также избран алгоритм, задействующий метод символьного выполнения, но несколько более простой, нежели в предыдущем случае. Поиск аргументов-типов и аргументов-строковых литералов осуществляется путём простого анализа синтаксического дерева. Однако этого метода оказалось недостаточно для поиска аргументов – интернированых строк (interned string), так как строка, задействованная в блокировке, могла, например, быть заранее интернирована внутри некоторого метода. Итоговый алгоритм для каждой функции фиксирует в резюме строки, которые были интернированы внутри неё (в резюме включаются в том числе и интернированные строки из вложенных функций). Единственное фиксируемое детектором событие – вызов функции. При обнаружении такого события рассматриваются три варианта.

- Вызван метод входа в критическую секцию – тогда его аргумент проверяется;
- Вызвана функция String.Intern – тогда интернируемая строка заносится в резюме вызвавшей функции;
- Вызвана другая функция – тогда из её резюме в резюме вызвавшей функции копируются данные об интернированных строках.

3.3.3 ThisLockObject

Эта задача, фактически являющаяся подзадачей поиска аргументов неправильного типа, вынесена в отдельный детектор, так как в процессе работы решено было использовать для неё другой алгоритм, нежели для других возможных неправильных типов.

Кроме того, итоговая реализация выдаёт предупреждения не на все вызовы формата lock(this), так как на этапе тестирования на проектах с открытым кодом выяснилось, что программисты часто пренебрегают правилами, предостерегающими от этой практики. По этой причине число срабатываний на крупных проектах исчислялось сотнями. Вместо этого избрана другая тактика: в текущей версии детектора выдаются предупреждения только о самой опасной ситуации, возможной при данной ошибке – о случаях, когда внешняя функция использует для блокировки объект класса, предки/потомки которого (или он сам) используют this в блокировках внутри своих функций (рис. 9). В том числе к таким функциям относятся методы с атрибутом [MethodImpl(MethodImplOptions.Synchronized)], добавление которого эквивалентно добавлению оператора lock(this), заключающего в себя всё тело метода. Такая возможность также учитывается.

Важную роль в принципе работы детектора ThisLockObject играет вспомогательный AST-анализатор, во время анализа синтаксического дерева сохраняющий в своих записях информацию о создаваемых классах с «опасными» методами и их предках. Основной детектор, как и предыдущие, является символьным, однако, как и WrongTypeLock, использует только события вызова функций. Для каждой функции входа в критическую секцию проверяется, не принадлежит ли её первый аргумент списку небезопасных для блокировки классов, составленному вспомогательным анализатором типов. Если это так, то выводится соответствующее сообщение.

4. Результаты тестирования

В процессе разработки работа детекторов проверена как на наборе из более чем 100 модульных тестов, так и на open source проектах. Разработанные синтетические тесты включают проверки:

- на чувствительность к контексту:
  - для LockObjectReassignment учитывается возможность изменения переменной с помощью функции – рассмотрены варианты передачи параметра по ссылке и по значению, изменение внутри функции статического члена; учитывается использование оператора lock или класса Monitor);
  - для WrongTypeLock – производится проверка на интернирование строки внутри функции;
- на чувствительность к полям (учитывается структура используемых данных);
- на чувствительность к потоку управления (учитываются такие особенности потока управления, как циклы и ветвления);
- на чувствительность к путям (ошибки в недосягаемых участках кода не фиксируются).

Табл. 2. Статистика работы детекторов
Table 2. Statistics of detectors' performance

	Время без новых детекторов	**Время со включенным детектором**	**Срабатываний**		
			TP	**FP**	**Пропущенных**
LockObjectReassignment	01:20:15	01:20:49 (+0.7%)	7	3	1
WrongTypeLock		01:20:19 (+<0.1%)	9	0	Нет данных
ThisLockObject		01:20:41 (+0.5%)	7	0	Нет данных

Рассмотрим подробнее тестирование на ПО с открытым исходным кодом. Оно производилось на наборе из 21 проекта общим объёмом в 6 млн. строк кода. Анализ выполнялся на машине с 8-ядерным процессором Intel(R) Core(TM) i7-6700, имеющей 32 Gb RAM.Ниже приведены несколько примеров найденных ошибок (рис. 7-9) и статистика работы детекторов (табл. 2).

Данные о необнаруженной ошибке для LockObjectReassignment получены с помощью стороннего анализатора. Эта ошибка связана с более углублённым использованием возможностей ReaderWriterLockSlim, которые текущая версия детектора не учитывает.

4.1 LockObjectReassignment

Ниже приведён пример реальной ошибки, найденной разработанным детектором в коде OpenSim – открытой платформы для создания пользовательских виртуальных симуляций [8]. Версия: 0.9.0.0.

```
Файл /OpenSim/Region/Framework/Scenes/Animation/MovementAnimationOverrides.cs:
92    public void CopyAOPairsFrom(Dictionary<string, UUID> src)
93    {
94       lock (m_overrides)
95       {
96          m_overrides.Clear();
97          m_overrides = new Dictionary<string, UUID>(src);
// !!! LOCK_OBJECT_REASSIGNMENT Lock object
MovementAnimationOverrides.m_overrides was changed in assignment
98       }
99    }
```

Рис. 7. Пример истинного срабатывания LOCK_OBJECT_REASSIGNMENT
Fig. 7. Example of true positive LOCK_OBJECT_REASSIGNMENT warning

4.2 WrongTypeLock

Эта ошибка найдена в эмуляторе с открытым кодом Wcell [9]. Версия: 9.0.

```
Файл /Core/Cell.Core/ObjectPoolMgr.cs:
59    public static bool RegisterType<T>
                            (Func<T> func) where T : class
60    {
61       long typePointer = GetTypePointer<T>();
62       lock (typeof(ObjectPoolMgr))
// !!! WRONG_TYPE_LOCK System.Type is used for locking
63       {
64          if (!Pools.ContainsKey(typePointer))
65          {
66             Pools.Add(typePointer, new ObjectPool<T>(func));
67             return true;
68          }
69       }
70       return false;
71    }
```

Рис. 8. Пример истинного срабатывания WRONG_TYPE_LOCK
Fig. 8. Example of true positive WRONG_TYPE_LOCK warning

4.3 ThisLockObject

Эта ошибка найдена в коде библиотеки Apache Lucene.NET [10]. Версия: 3.0.3.

По данным табл. 2 видно, что влияние разработанных детекторов на время выполнения анализа незначительно (<1%): с точки зрения временных затрат все они достаточно эффективны. Статистика результатов также приемлема. По таблице также можно увидеть, что детектор LockObjectReassignment выдаёт ложные срабатывания, однако их процент находится в допустимых пределах (30%), как и процент ненайденных ошибок.

Файл */src/Lucene.Net.Core/Index/BufferedUpdatesStream.cs*:

```
 49  public class BufferedUpdatesStream
 50  {
        /.../
 73        public virtual long Push(FrozenBufferedUpdates packet)
 74        {
 75        lock (this)
 76        {
              /.../
 98        }
 99        }
        /.../
718  }
```

Файл */src/Lucene.Net.Core/Index/IndexWriter.cs:*

```
 275 internal readonly BufferedUpdatesStream
BufferedUpdatesStream;
1623  public virtual bool TryDeleteDocument(IndexReader
readerIn,
                                                int docID)
1624  {
1661    lock (BufferedUpdatesStream)
// !!! THIS_LOCK_OBJECT Lucene.Net.Index.BufferedUpdatesStream contains
locks on instance (this) in its functions. It is unsafe to use variable of this type for
locking.
1662    {
/.../
1687    }
1700 }
```

Рис. 9. Пример истинного срабатывания THIS_LOCK_OBJECT
Fig. 9. Example of true positive THIS_LOCK_OBJECT warning

5. Сравнение с существующими решениями

В самом инструменте SharpChecker есть следующие детекторы ошибок, также работающие с проблемами синхронизации потоков [11]:

- выявление переменных, к которым потенциально возможен несинхронизированный доступ нескольких потоков и на которые рекомендована установка синхронизирующей блокировки (NO_LOCK.STAT);
- обнаружение участков, где возможны взаимные блокировки – бесконечное ожидание нескольких процессов, каждый из которых использует ресурсы, необходимые другому (DEADLOCK).

Можно заметить, что ни один из них не может предложить полноценного решения нашей проблемы.

В других статических анализаторах существуют следующие детекторы ошибок в области синхронизации (все представленные анализаторы применяют метод символьного выполнения).

- Svace [12] – промышленный статический анализатор для Java и C/C++/C#:
  - Детектор возможных взаимных блокировок, сходный с упомянутым выше.
  - Выявление случаев, когда блокировка семафора выполняется одним потоком два раза подряд без освобождения (DOUBLE_LOCK).
  - Детектор ситуаций, когда функция, освобождая семафор в одних ветвях, сохраняет его до своего завершения в других (NO_UNLOCK).

Обнаружение ситуаций, когда переменная из стека используется для синхронизации (LOCK_ON_STACK). Это является ошибкой, так как каждый поток использует собственный стек.

  - Обнаружение вызовов блокирующих функций внутри критической секции, что может привести к долгому ожиданию не только для данного процесса, но и для остальных (LONG_TIME_IN_LOCK).
  - Детектор синхронизации доступа к статическому полю при помощи нестатической переменной (WRONG_LOCK.STATIC). Такой доступ может создать ситуацию гонки, т.к. даже для разных объектов синхронизации обращение к статическому полю означает доступ к одному и тому же участку памяти из разных потоков выполнения.

- Infer Static Analyzer [13] – инструмент статического анализа с открытым исходным кодом, разработанный командой инженеров Facebook [14] для языков Java и C/C++/Objective-C (включает в себя инструмент RacerD [15], нацеленный на поиск потенциальных состояний гонки):
  - Поиск классов, в которых есть публичный метод, записывающий данные в некоторый член, используя блокировку и публичный метод, считывающий эти данные без использования блокировки (LOCK_CONSISTENCY_VIOLATION).
  - Поиск объектов, к которым возможен несинхронизированный доступ двух и более потоков, по крайней мере один из которых выполняет запись (THREAD_SAFETY_VIOLATION).
  - Поиск взаимных блокировок.
- Clang Static Analyzer [16] – статический анализатор, который находит ошибки в программах на языках C, C++ и Objective-C:
  - BlockInCriticalSection – обнаруживает использование внутри критической секции блокирующих функций (аналогично LONG_TIME_IN_LOCK в Svace).
  - PthreadLock– обнаруживает случаи неправильной последовательности блокировок/освобождений ресурса.
- SonarQube [17] – платформа с открытым исходным кодом для непрерывного анализа и измерения качества программного кода для большого набора языков, включающего в себя в том числе и C#.
  - Детектор использования переменных блокировки, являющихся общими ресурсами (частично соответствует сочетанию разработанных нами WrongTypeLock и ThisLockObject с некоторыми различиями – например, сигнализирует о любых случаях использования lock(this)).
- Coverity [18] – пакет программного обеспечения, состоящий из статического и динамического анализаторов кода, нацеленный на поиск ошибок и недочётов в безопасности исходных кодах программ, написанных на Си, C++, Java, C# и JavaScript.
  - OVERWRITE_OF_LOCK_FIELD_DURING_CRITICAL_SECTION – детектор переопределений объекта блокировки в критической секции – практически аналогичен LockObjectReassignment.
  - BAD_LOCK_OBJECT – обнаружение объектов блокировки неправильного типа, в частности, интернированных строк, по критериям поиска схож с WrongTypeLock.

- Поиск взаимных блокировок.

Общее сравнение анализаторов см. в табл. 3.

Табл. 3. Возможности разработанных детекторов по сравнению с существующими анализаторами (анализатор Svace также обнаруживает использование в lock неподходящих объектов, но имеет другие критерии – нацелен на поиск объектов стека и статических полей)
Table 3. Capabilities of the developed detectors compared to existing analyzers (the Svace analyzer detects the use of incorrect objects in lock as well, but has other criteria – it is aimed at searching for stack objects and static fields)

	Переопределение переменной	**Неправильный тип (this)**	**Неправильный тип (другой)**
Svace	-	-	+-
Infer Static Analyzer	-	-	-
Clang Static Analyzer	-	-	-
SonarQube	-	+	+
Coverity	+	-	+
SharpChecker	+	+	+

Таким образом, несмотря на возможность поиска ошибок, связанных с критическими секциями, ошибки описанных в статье видов не находятся большинством рассмотренных сторонних анализаторов. Тем не менее, примеры подобного переопределения несколько раз были обнаружены при исправлении других дефектов в проектах с открытым исходным кодом. Планируется, что разрабатываемый метод поможет эффективнее находить их в будущем.

6. Заключение

Был разработан, реализован и протестирован алгоритм поиска комплекса ошибок осуществления взаимоисключающей блокировки. По итогам тестирования разработанный детектор показал приемлемые скорость и точность, зачастую обнаруживая ошибки, которые не могут найти другие инструменты анализа. Так, поиск ошибки вида Lock Reassignment среди множества рассмотренных анализаторов осуществляется только в инструменте Coverity, который, в отличие от SharpChecker, является закрытым. Помимо этого, новизна проекта состоит в реализации алгоритма поиска подобных ошибок в рамках возможностей и ограничений, заданных конкретно анализатором SharpChecker, а именно, конкретной реализацией межпроцедурного чувствительного к контексту и путям статического символьного выполнения с объединением состояний.

В дальнейшем планируется ещё больше повысить эффективность работы полученного анализатора, а также продолжить расширять возможности нашего инструмента в области ошибок многопоточности, рассмотрев такие ошибки, как выход из критической секции, в которую не был совершён успешный вход, многократный захват и освобождение одной и той же переменной блокировки и т.д.

Список литературы / References

[1] Microsoft Docs: .NET API browser: System.Threading: Monitor Class. Available at: https://docs.microsoft.com/en-us/dotnet/api/system.threading.monitor, accessed: 29.07.22.
[2] Microsoft Docs: .NET API browser: System.Threading: ReaderWriterLock Class. Available at: https://docs.microsoft.com/en-us/dotnet/api/system.threading.readerwriterlock, accessed: 01.08.22.
[3] Microsoft Docs: .NET API browser: System.Threading: ReaderWriterLockSlim Class. Available at: https://docs.microsoft.com/en-us/dotnet/api/system.threading.readerwriterlockslim, accessed: 01.08.22.

[4] Microsoft Docs: .NET API browser: C# Keywords: Statement Keywords (C# Reference): lock statement. Available at: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/statements/lock, accessed: 03.08.22.
[5] The .NET Compiler Platform ("Roslyn"). Available at: https://github.com/dotnet/Roslyn, accessed: 29.07.22
[6] Кошелев В.К. Межпроцедурный статический анализ для поиска ошибок в исходном коде программ на языке С. Диссертация на соискание учёной степени кандидата физико-математических наук, Москва, ИСП РАН, 2017 г., 104 стр. / Koshelev V.K. Interprocedural static analysis for finding errors in the source code of C programs. Thesis for the degree of candidate of physical and mathematical sciences, Moscow, ISP RAS, 2017, 104 p. (in Russian).
[7] Кошелев В.К., Игнатьев В.Н., Борзилов А.И. Инфраструктура статического анализа программ на языке C#. Труды ИСП РАН, том 28, вып. 1, 2016 г., стр. 21-40 / Koshelev V.K., Ignatyev V.N., Borzilov A.I. C# static analysis framework. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 1, 2016, pp. 21-40 (in Russian). DOI: 10.15514/ISPRAS-2016-28(1)-2.
[8] OpenSimulator. Available at: http://opensimulator.org, accessed: 23.10.2021.
[9] WCell: World of Warcraft emulator written in C#/.NET 4.0, with design and extensibility in mind. Available at: https://github.com/WCell/WCell, accessed at 15.08.2022.
[10] Welcome to the Lucene.NET website! | Apache Lucene.NET 4.8.0. Available at: https://lucenenet.apache.org, accessed: 23.10.2021.
[11] Белеванцев А.А. Многоуровневый статический анализ исходного кода для обеспечения качества программ. Диссертация на соискание учёной степени доктора физико-математических наук, Москва, ИСП РАН, 2017 г., 229 стр. / Belevantsev A.A. Multi-level static analysis of the source code to ensure the quality of programs. Thesis for the degree of Doctor of Physical and Mathematical Sciences, Moscow, ISP RAS, 2017, 229 p. (in Russian).
[12] Иванников В.П., Белеванцев А.А. и др. Статический анализатор Svace для поиска дефектов в исходном коде программ. Труды ИСП РАН, том 26, вып. 1, 2014 г., стр. 231-250 / Ivannikov V.P., Belevantsev A.A. et al. Static analyzer Svace for finding of defects in program source code. Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 1, 2014, pp. 231-240 (in Russian). DOI: 10.15514/ISPRAS-2014-26(1)-7.
[13] Calcagno C., Distefano D. Infer: An automatic program verifier for memory safety of C programs. Lecture Notes in Computer Science, vol. 6617, 2011, pp. 459-465
[14] Calcagno C., Distefano D. et al. Moving fast with software verification. Lecture Notes in Computer Science, vol. 9058, 2015, pp. 3-11.
[15] Liu B., Liu P. et al. When threads meet events: efficient and precise static race detection with origins. In Proc. of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, 2021, pp. 725-739.
[16] Kremenek T. Finding software bugs with the clang static analyzer. Available at: https://llvm.org/devmtg/2008-08/Kremenek_StaticAnalyzer.pdf, accessed: 23.10.2021.
[17] Campbell G.A., Papapetrou P.P. SonarQube in action. Manning, 2013, 392 p.
[18] Synopsys Software Security | Software Integrity Group, Available at: http://www.coverity.com, accessed: 13.09.2022.

Информация об авторах / Information about authors

Полина Ильинична РАГОЗИНА – студентка бакалавратуры кафедры системного программирования ф-та ВМК МГУ, исследователь в отделе компиляторных технологий ИСП РАН. Научные интересы: статический анализ программ, символьное выполнение, поиск дефектов синхронизации в многопоточных программах.

Polina Ilyinichna RAGOZINA is a bachelor's student of the System Programming Department of the CMC Faculty of Moscow State University, a researcher in the Department of Compiler Technologies of ISP RAS. Scientific interests: static analysis of programs, symbolic execution, search for synchronization defects in multithreaded programs.

Валерий Николаевич ИГНАТЬЕВ, кандидат физико-математических наук, старший научный сотрудник ИСП РАН, старший преподаватель кафедры системного программирования

факультета ВМК МГУ. Научные интересы включают методы поиска ошибок в исходном коде ПО на основе статического анализа.

Valery Nikolayevich IGNATYEV, PhD in computer sciences, senior researcher at Ivannikov Institute for system programming RAS and senior lecturer at system programming division of CMC faculty of Lomonosov Moscow State University. He is interested in techniques of errors and vulnerabilities detection in program source code using static analysis.

DOI: 10.15514/ISPRAS-2022-34(4)-6

Big Transformers for Code Generation

G.A. Arutyunov, ORCID: 00000-0003-4537-4332 <gaarutyunov@edu.hse.ru>
S.M. Avdoshin, ORCID: 0000-0001-8473-8077 <savdoshin@hse.ru>
HSE University,
20, Myasnitskaya st., Moscow, 101000 Russia

Abstract. IT industry has been thriving over the past decades. Numerous new programming languages have emerged, new architectural patterns and software development techniques. Tools involved in the process ought to evolve as well. One of the key principles of new generation of instruments for software development would be the ability of the tools to learn using neural networks. First of all, it is necessary for the tools to learn how to write code. In this work we study the ability of Transformers to generate competition level code. The main goal is to discover whether open-source Big Transformers are "naturally" good coders**.**

Keywords: neural networks; code generation; Transformers; GPT

For citation: Arutyunov G.A., Avdoshin S.M. Big Transformers for Code Generation. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 4, 2022. pp. 79-88. DOI: 10.15514/ISPRAS-2022-34(4)-6

Acknowledgments: This research was supported in part through computational resources of HPC facilities at HSE University [1].

Большие трансформеры для генерации кода

Г.А. Арутюнов, ORCID: 0000-0003-4537-4332 <gaarutyunov@edu.hse.ru>
С.М. Авдошин, ORCID: 0000-0001-8473-8077 <savdoshin@hse.ru>
Национальный исследовательский университет «Высшая школа экономики» (НИУ ВШЭ), 101000, Россия, г. Москва, ул. Мясницкая, д. 20

Аннотация. Индустрия разработки программного обеспечения развивается стремительными темпами. Непрерывно появляются новые языки программирования, архитектурные паттерны и подходы к разработке. Развиваться должны и инструменты, используемые программистами. Среди необходимых условий появления нового семейства инструментария следует выделить способность обучаться за счет использования моделей машинного и глубинного обучения. В данной статье будут рассмотрены достижения последних лет в области применения авторегрессионых моделей для генерации программного кода из естественного языка. Основной целью исследования является оценка того, можно ли назвать Большие трансформеры с открытым исходным кодом программистами «от природы».

Ключевые слова: нейронные сети; генерация кода; Трансформеры; GPT

Для цитирования: Арутюнов Г.А., Авдошин С.М. Большие трансформеры для генерации кода. Труды ИСП РАН, том 34, вып. 4, 2022 г., стр. 79-88. 10.15514/ISPRAS-2022-34(4)-6

Благодарности. Исследование выполнено с использованием суперкомпьютерного комплекса НИУ ВШЭ [1]**.**

1. Introduction

In the recent years, there has been solid advancement in code generation using neural networks. GitHub and OpenAI, for example, launched their tool called Copilot that can generate code from

prompts written in natural language [2]. The code assistant is based on OpenAI Codex [3], a proprietary GPT-3 [4] variant fine-tuned on code from GitHub.

The success of OpenAI's GPT-3 model has encouraged open-source communities to develop large pre-trained language models. For example, EleutherAI has developed several good performing autoregressive language models: GPT-Neo [5], GPT-J [6] and finally GPT-NeoX [7]. The last of these three has 20 billion parameters and performs remarkably well on several benchmarks as shown in the original paper [7].

The goal of this work is to test the ability of the two biggest EleutherAI models, GPT-J and GPT-NeoX, to generate Python competition-level code. We will analyse its performance and compare its results with preceding works using APPS benchmark [8].

2. Problem Statement

Although the task of text-to-code generation may appear similar to text-to-text generation, it has some major differences. First, the generated code must be syntactically correct in order to be executed. Moreover, it should solve the exact task that was expected from it. Therefore, testing code generation and using same quality metrics as with text generation is not useful [8].

2.1 The APPS Dataset

Due to these issues we will use a benchmark specifically designed to validate code generation – the APPS dataset [8]. It is composed by 10000 programming tasks from platforms such as Codewars AtCoder, Kattis and Codeforces. Tasks are separated into 3 difficulty levels: introductory, interview, competition. Furthermore, each task is accompanied by 20 test-cases on average that are used for validation.

2.2 Performance Metrics

As for model quality metrics, the same as in original work is used – average test cases. In addition, we review the runtime and syntax errors rate of generated code. Strict accuracy is only used in this work to compare previous research since they used it in their articles (see Table 1). However, it is not used for our models since Transformers studied in this work are not yet good coders for such a strict metric.

Average test cases metric shows the average number of test cases that generated code has passed. It is defined as follows:

$$\frac{1}{P}\sum_{p=1}^{P}\frac{1}{C_p}\sum_{c=1}^{C_p}1\{\text{eval}(\langle \text{code}_p\rangle, x_{p,c}) = y_{p,c}\},$$

P – number of tasks,

C_p – number of test cases per task,

$\langle \text{code}_p\rangle$ – code generated by the model

$\{(x_{pc}, y_{p,c})\}$ – set of inputs-outputs for the task.

Strict accuracy demands that a task passes all test cases. It is defined as follows:

$$\frac{1}{P}\sum_{p=1}^{P}\prod_{c=1}^{C_p}1\{\text{eval}(\langle \text{code}_p\rangle, x_{p,c}) = y_{p,c}\}$$

3. Previous Works and Motivation

After Hendrycks et al. showed in their research [8] that Transformers are able to generate code that can solve competition-level code several companies have tried to use models of similar architecture for the same task.

The first one was Codex [3] which is a model by OpenAI based on GPT-3. As we can see from the Table 1 the model almost doubled the results of previous GPT-Neo model at introductory problems.

Table 1. Comparison of GPT-Neo, GPT-J, GPT-NeoX, Codex and AlphaCode Performance ON APPS dataset. Strict accuracy is calculated from 5 solutions. For Codex and AlphaCode we also present results filtered from 1000 generated solutions

Model name	**Strict accuracy @ 5, %**		
	Introductory	***Interview***	***Competition***
GPT-Neo 2.7B	5.50	0.80	0.00
GPT-J 6B	0.00	0.00	0.00
GPT-NeoX 20B	0.15	0.20	0.00
Codex 12B	9.65	0.51	0.09
Codex 12B filtered from 1000	24.52	3.23	3.08
AlphaCode 1B filtered from 1000	14.36	5.63	4.58

The next is AlphaCode by Google [9]. Authors trained 5 models: 300M, 1B, 3B, 9B, 41B. However only the one with 1B parameters was tested on APPS dataset. As shown in Table 1 AlphaCode didn't beat Codex in introduction problems, however it did in the interview and competition tasks.

Sadly, both models are proprietary and it is impossible to research them and perform more experiments. Open-source models with comparable sizes are GPT-Neo 2.7B [5], GPT-J 6B [6] and GPT-NeoX 20B [7] – all developed by EleutherAI.

The first model has already been tested in the Hendrycks et al. work and showed promising results (see Table 1). The motivation behind this work is to determine whether by just fine-tuning Big Transformers like GPT-J and GPT-NeoX with billions of parameters, as it was done in the mentioned work, directly results in high performance in a code generation task such as APPS.

4. Proposed Solution

The primary model in this research is the GPT-NeoX. We used GPT-NeoX model since it showed results comparable with GPT-3 and other Big Transformers in many NLP tasks [7]. Moreover, it is one of the biggest available open-source models with 20 billion parameters. Other reasons for choosing GPT-NeoX model is its configurability suitable for extensive research, the tokenizer and the architecture of the model that allows parallelizing the training process.

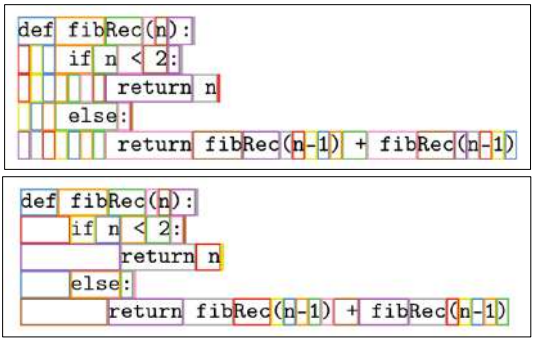


Fig. 1. Tokenization result for GPT-2 (above) and GPT-NeoX (below)

The tokenizer used in the model can process new lines spaces and tabs grouping them in one token which is very useful for code processing. For example, the code displayed on fig. 1 processed be the GPT-NeoX tokenizer has 39 tokens against 50 produced by the GPT-2 tokenizer.

In addition, the model supports model parallelism [10] and pipeline parallelism [11] with Deepspeed and PyTorch distributed. Therefore, even billions of parameters may fit in the GPUs' memory. Table 2 depicts the GPU requirements for models training. Smaller models with low sequence length can be trained with less requirements, however we doubled the number of GPUs to increase the speed of training process.

Table. 2. Comparison of GPU requirements for models training. "M" in model name stands for millions of tokens, "B" – for billions, while the number in the brackets "(2k)" shows the sequence length in thousands.

Model	**GPUs characteristics**		
	Number of GPUs	***GPU model***	***GPU memory***
165M (2k)	2	V100	32GB
165M (4k)	2	V100	32GB
165M (8k)	2	A100	80GB
6B (2)	8	A100	80GB
20B (2k)	8	A100	80GB

All the GPT-NeoX variants were trained using Adam optimizer with batch size of 64 for 100 epochs. Then the same batch size and optimizer were used for fine-tuning process during 10 epochs. The GPT-J model was fine-tuned with batch size of 256 for 10 epochs – same as GPT-Neo in the original APPS paper [8].

The development of the text-to-code generation model based on GPT-NeoX was separated into 4 different stages that are discussed below. The results and conclusions of these stages are detailed in the next section.

4.1 Fine-tuning on APPS

We started by fine-tuning both GPT-J 6B and GPT-NeoX 20B models on APPS dataset. In the original paper [8] GPT-Neo showed good results after such fine-tuning, however, neither the GPT-J nor GPT-NeoX did. As we can see in Table 1 both models yielded 0% strict accuracy in all the levels of difficulty.

4.2 Fine-tuning on English-to-Python datasets

Due to the failure of APPS-only fine-tuning we gathered 5 additional datasets. Two datasets that contain question-answer pairs scraped from Stackoverflow:

1) StaQC [12, 13] that contains around 270 thousand examples. We use the 148 thousand that are written in python.
2) CoNaLa [14] also scraped from Stackoverflow containing around 600 thousand examples.
3) Moreover, we used 2 additional datasets comprised of programming competition tasks:
4) Description2Code [15] comprised from 8 thousand tasks from CodeChef, Codeforces and Hackerearth. Codeforces data was excluded to avoid data leakage since it also appears in APPS dataset.
5) CodeNet [16, 17] that includes tasks from AIZU Online Judge and AtCoder platforms. AtCoder data was also excluded to avoid overlap with APPS dataset.
6) Finally, we add a dataset with parsed docstrings: CodeDocstringCorpus [18, 19] consisting of almost 150 thousand docstring-to-code pairs with functions and methods written in Python.

All the code in the above-mentioned datasets was processed by interpreting and transforming it into Python3 code using lib2to3. It was done to diminish syntax errors in generated code. All the code was then tokenized using a vocabulary of 52 thousand tokens.

4.3 Pre-training on Python code from GitHub

As demonstrated in other researches models pre-trained with code samples and then fine-tuned on text-to-code datasets yield promising results [9, 20]. Therefore, 150 thousand python scripts from around 5 thousand public GitHub repositories were collected. They were used to pre-train GPT-NeoX model on code.

4.4 Other models and context windows

Researchers have demonstrated that not only the size of the transformer matters but rather a good combination of size and context window [20].

In this work, we also pre-trained and fine-tuned smaller models with 165 million parameters. Furthermore, we increased the context windows from 2048 to 8192 tokens. The models were pre-trained with GitHub Python dataset and fine-tuned with English-to-Python dataset.

5. Experiment Results

In this section we will discuss the conclusions we made based on the results of all 4 stages.

5.1 Fine-tuning dataset matters

When creating a fine-tuning dataset for text-to-code generation, there are several things that need to be considered.

First, the dataset should not be too narrow for a concrete task. Otherwise, the model will be limited to the constructs it learned from the dataset. Programming is a very wide field, therefore the more examples you get, more programming skills the model will acquire.

Second, it is important to filter syntactically incorrect or inappropriate code, for example code written on another version of the programming language. Otherwise, the model will learn from bad examples which will result in crushed tests.

When conducting the experiments, the construction of the fine-tuning dataset allowed us to increase results from slightly above 0% average test cases to around 3% for the 20 billion model.

5.2 Text vs Code pre-training

Although some previous works demonstrate that code is similar to natural language [21], text pre-trained models are not expected to yield the same results as models pre-trained on code.

In our research we do not emphasize on comparing text vs code pre-trained models. However, as we can see in Table 3 GPT-NeoX models with less parameters that were pre-trained on code yield similar results as the 20 billion parameter model trained on the Pile [22].

5.3 Model size and context window

The performance of models was analysed in two manners: based on the average test cases passed and on the ratio of runtime and syntax errors.

Fig. 2 demonstrates the results of testing the code generated with the GPT-NeoX models with different sizes and context lengths. It shows very interesting results. As we can see, the smallest 165 million parameter model with sequence length of 8 thousand tokens acts almost as well as the 20 billion parameter model.

As for the errors rate, we noticed that the best models had a syntax errors rate around 5-15% and a runtime errors rate between 60 and 70%. For worst models the situation was the opposite: around 60% of the code was syntactically wrong.

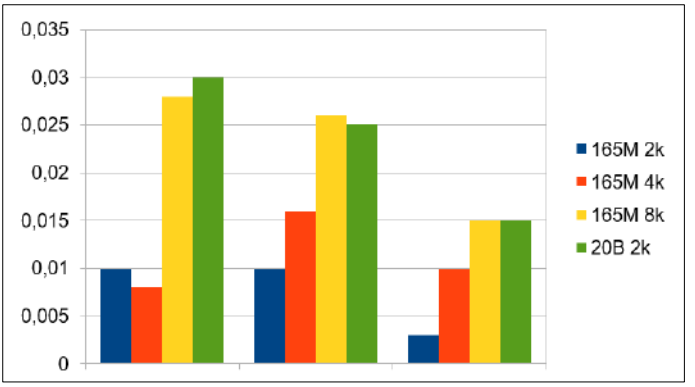


Fig. 2. Comparison of average passed test cases for different models

5.4 Best models

Table 3 shows the comparison of different models trained on code. In the case of GPT-NeoX the context window with best results for each model size are shown. As we can see, the number of parameters doesn't directly result in the best performance. Nor does the size of the pre-training dataset. The GPT-NeoX 20B model trained on the Pile performed similar to the 165M model that was trained on a comparably smaller dataset. However, the length of the context window does affect the model performance positively.

Table. 2. Comparison of EleutherAI models Performance on APPS dataset

Model name	**Avg. test cases passed, %**		
	Introductory	***Interview***	***Competition***
GPT-NeoX 165M 8k	2.80	2.60	1.50
GPT-Neo 2.7B 2k	14.68	9.85	6.54
GPT-J 6B 2k	0.09	0.01	0.00
GPT-NeoX 20B 2k	3.00	2.50	1.60

5.5 Quality of generated code

Apart from the metrics it is necessary to compare the generated code. In this section we will present some examples of code generated by both GPT-J and GPT-NeoX models. The task is formulated below. And the generated code is displayed on Figures 3 and 4.

Task: You are given a positive integer n, and an array a of positive integers. The task is to calculate the number of such pairs (i, j) that $i < j$ and $a_i \,\&\, a_j \geq a_i \oplus a_j$, where & denotes the bitwise AND operation, and $\oplus$ denotes the bitwise XOR operation.

```
t=int(input())
for i in range(t):
    n=int(input())

a=list(map(int,input().spl
it()))
    for j in range(n):
        a[j]=a[j]%2
    print(a.count(1))
```

Fig. 3. Code generated by GPT-NeoX model

On fig. 3 you can see the code generated by GPT-NeoX with 165 million parameters and context length of 8 thousand tokens. This code passes all the test cases and seems easily understandable and concise.

```
n = int(input())
a = list(map(int,
input().split()))
b = list(set(a))
b.sort()
for i in range(n-1):

print(b[b.index(a[i])+1])
```

Fig. 4. Code generated by GPT-J model

On fig. 4 you can see the code generated by GPT-J. Although this code doesn't contain syntax errors and seems pleasant overall it results in runtime errors.

As we can see, the models learnt to generate good code for simple tasks, nevertheless they could solve too few problems.

6. Future Work

This research demonstrated several key results on text-to-code generation.

First of all, the quality of a dataset matters a lot. To increase the metrics of a Transformer model it is necessary to construct a good dataset first. Future work could elaborate further on other languages – both programming and natural. For example, using datasets such as MCoNaLa [23] which contains examples on Russian, Spanish and Japanese.

Secondly, the research discovered that even smaller models pre-trained on code show similar results to bigger transformers pre-trained on text. However, analysing code as sequence may lead to information loss and increasing the context window on bigger models requires lots of computation resources. It would be a good approach to use models that support other representations of code that captures most of the information. For example, graph representation using abstract syntax tree could be used with graph transformers [24–27].

6. Conclusion

As for conclusion, in this work we demonstrated that Big Transformers are not naturally good coders. There are several points that must be taken into account to use them for code generation task.

First, it is important to collect extensive and high-quality dataset with minima incorrect code. Datasets must be verified by compiling code or running static analysis tool to exclude code that contains errors from the dataset.

Secondly, a model with appropriate context length and number of parameters must be chosen to generate functionally correct code. As shown in our results, context length is positively correlated with model performance.

References

[1] Kostenetskiy P.S., Chulkevich R.A., Kozyrev V.I. HPC Resources of the Higher School of Economics. Journal of Physics: Conference Series, vol. 1740, no. 1, 2021, article no. 012050, 11 p.

[2] Introducing GitHub Copilot: your AI pair programmer, The GitHub Blog, 2021. Available at: https://github.blog/2021-06-29-introducing-github-copilot-ai-pair-programmer/, accessed: 27.06.2022.

[3] Chen M., Tworek J. et al., Evaluating large language models trained on code. arXiv:2107.03374, 2021, 35 p.

[4] Brown T., Mann B. et al., Language models are few-shot learners. Advances in Neural Information Processing Systems, vol. 33, 2020, pp. 1877-1901.

[5] Black S., Gao L. et al. GPT-Neo: Large scale autoregressive language modeling with meshtensorflow, 2021. Available at: https://zenodo.org/record/5551208, accessed: 17.03.2022.
[6] Wang B., Komatsuzaki A. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model, 2021. Available at: https://github.com/kingoflolz/mesh-transformer-jax, accessed: 27.06.2022.
[7] Black S., Biderman S. et al. GPT-NeoX-20B: An Open-Source Autoregressive Language Model. arXiv:2204.06745, 2022, 42 p.
[8] Hendrycks D., Basart S. et al., Measuring coding challenge competence with APPS. arXiv:2105.09938, 2021, 22 p.
[9] Li Y., Choi D. et al., Competition-Level Code Generation with AlphaCode. arXiv:2203.07814, 2022, 74 p.
[10] Shoeybi M., Patwary M. et al. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. arXiv:1909.08053, 2020, 15 p.
[11] Harlap A., Narayanan D. et al. PipeDream: Fast and Efficient Pipeline Parallel DNN Training. arXiv:1806.03377, 2018, 14 p.
[12] Yao Z., Weld D.S. et al. StaQC: A Systematically Mined Question-Code Dataset from Stack Overflow, In Proc. of the 2018 Conference on World Wide Web, 2018, pp. 1693-1703.
[13] Yao Z., Weld D.S. et al. StackOverflow-Question-Code-Dataset, 2022. Available at: https://github.com/LittleYUYU/StackOverflow-Question-Code-Dataset, accessed: 17.06.2022.
[14] Yin P., Deng B. et al., Learning to mine aligned code and natural language pairs from stack overflow, In Proc. of the 15th International Conference on Mining Software Repositories, 2018, pp. 476-486.
[15] Caballero E., Sutskever I. Description2Code Dataset, 2016. Available at: https://github.com/ethancaballero/description2code, accessed: 17.06.2022.
[16] Puri R., Kung D.S. et al. CodeNet: A Large-Scale AI for Code Dataset for Learning a Diversity of Coding Tasks. c, 2021, 22 p.
[17] IBM, Project CodeNet, 2022. Available at: https://github.com/IBM/Project_CodeNet, accessed: 17.06.2022.
[18] code-docstring-corpus, 2022. Available at: https://github.com/EdinburghNLP/code-docstring-corpus, accessed: 17.06.2022.
[19] Barone A.V.M., Sennrich R. A parallel corpus of python functions and documentation strings for automated code documentation and code generation. arXiv: 1707.02275, 2017, 5 p.
[20] Nijkamp E.б Pang B. et al. A Conversational Paradigm for Program Synthesis. arXiv:2203.13474, 2022, 22 p.
[21] Hindle A., Barr E.T. et al. On the naturalness of software. Communications of the ACM, vol. 59, issue 5, 2016, pp. 122-131.
[22] Gao L., Biderman S. et al. The Pile: An 800GB Dataset of Diverse Text for Language Modeling. arXiv:2101.00027, 2020, 39 p.
[23] Wang Z., Cuenca G. et al., MCoNaLa: A Benchmark for Code Generation from Multiple Natural Languages. arXiv:2203.08388, 2022, 11 p.
[24] Kim J., Nguyen T.D. et al. Pure Transformers are Powerful Graph Learners. arXiv:2207.02505. 2022, 28 p.
[25] Kreuzer D., Beaini D. et al., Rethinking Graph Transformers with Spectral Attention. arXiv:2106.03893, 2021, 18 p.
[26] Dwivedi V.P., Bresson X. A Generalization of Transformer Networks to Graphs. arXiv:2012.09699, 2021, 8 p.
[27] Dwivedi V.P., Bresson X. Graph Transformer Architecture, 2022. Available at: https://github.com/graphdeeplearning/graphtransformer, accessed: 17.06.2022.

Информация об авторах / Information about authors

German Arsenovich ARUTYUNOV – Master's student at the Faculty of Computer Science at HSE University. Research interests include programming language generation and programming language understanding using machine learning and deep neural networks.

Герман Аренович АРУТЮНОВ – студент магистратуры факультета компьютерных наук НИУ ВШЭ. Сфера научных интересов: генерация и анализ языков программирования посредством машинного обучения и глубоких нейронных сетей.

Sergey Mikchailovitch AVDOSHIN – Candidate of Technical Science, Professor of the School of Computer Engineering at Tikhonov Moscow Institute of Electronics and Mathematics HSE

University. Research interests include design and analysis of computer algorithms, simulation and modeling, parallel and distributed processing, machine learning.

Сергей Михайлович АВДОШИН – кандидат технических наук, профессор департамента компьютерной инженерии Московского института электроники и математики им. А.Н. Тихонова НИУ ВШЭ. Сфера научных интересов: разработка и анализ компьютерных алгоритмов, имитация и моделирование, параллельные и распределенные процессы, машинное обучение.

DOI: 10.15514/ISPRAS-2022-34(4)-7

Data distribution and parallel code generation for heterogeneous computational clusters

N.A. Kataev, ORCID: 0000-0002-7603-4026 <kataev_nik@mail.ru>
A.S. Kolganov, ORCID: 0000-0002-1384-7484 <alexander.k.s@mail.ru>
Keldysh Institute of Applied Mathematics of Russian Academy of Sciences,
4, Miusskaya sq., Moscow, 125047, Russia

Abstract. We present new techniques for compilation of sequential programs for almost affine accesses in loop nests for distributed-memory parallel architectures. Our approach is implemented as a source-to-source automatic parallelizing compiler that expresses parallelism with the DVMH directive-based programming model. Compared to all previous approaches ours addresses all three main sub-problems of the problem of distributed memory parallelization: data and computation distribution and communication optimization. Parallelization of sequential programs with structured grid computations is considered. In this paper, we use the NAS Parallel Benchmarks to evaluate the performance of generated programs and provide experimental results on up to 9 nodes of a computational cluster with two 8-core processors in a node.

Keywords: compilers; automatic parallelization; code generation; distributed memory systems

For citation: Kataev N.A., Kolganov A.S. Data distribution and parallel code generation for heterogenious computational clusters. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 4, 2022. pp. 89-100. DOI: 10.15514/ISPRAS-2022-34(4)-7

Построение распределения данных и генерация кода при распараллеливании на гетерогенный вычислительный кластер

Н.А. Катаев, ORCID: 0000-0002-7603-4026 <kataev_nik@mail.ru>
А.С. Колганов, ORCID: 0000-0002-1384-7484 <alexander.k.s@mail.ru>
ИПМ им. М.В. Келдыша РАН,
125047, Россия, Москва, Миусская пл., д.4

Аннотация. В данной статей рассматривается новый подход к компиляции последовательных программ для их последующего выполнения на вычислительных системах с распределенной памятью. Предложенный подход был реализован в виде автоматически распараллеливающего компилятора для программ на языках Си и Фортран. Для описания параллелизма, обнаруженного в программе, используется директивная модель параллельного программирования DVMH. Таким образом, реализованный компилятор выполняет преобразование программ на уровне исходного кода, добавляя в них высокоуровневые спецификации параллелизма в терминах DVMH модели. Распараллеливание основано на анализе гнезд циклов программы, содержащих обращения к многомерным массивам, для которых большинство индексных выражений линейно зависит от индуктивных переменных циклов гнезда. Основной областью применения предложенного подхода являются программы научно-технических расчетов, реализующие вычисления на структурированных сетках. В отличие от подходов к распараллеливанию программ, предложенных в других работах, наш подход охватывает решение всех трех основных задач, возникающих при распараллеливании для систем с распределенной памятью: распределение данных, распределение вычислений и оптимизация коммуникационных обменов между узлами вычислительной системы. Для оценки эффективности получаемых параллельных программ, мы использовали некоторых приложения из набора NAS Parallel Benchmarks. В статье приведены

результаты экспериментов, в которых были задействованы до 9 узлов вычислительного кластера, каждый из которых содержал два 8-ядерных процессора.

Ключевые слова: компилятор; автоматическое распараллеливание; генерация кода; системы с распределенной памятью.

Для цитирования: Катаев Н.А., Колганов А.С. Построение распределения данных и генерация кода при распараллеливании на гетерогенный вычислительный кластер. Труды ИСП РАН, том 34, вып. 4, 2022 г., стр. 89-100. 10.15514/ISPRAS-2022-34(4)-7

1. Introduction

Most of current high performance computing (HPC) systems tend to be heterogeneous and may comprise diverse compute devices. However, from the prospective of obtaining greater processing power on the way to exascale computing, the common feature of these systems is the distributed memory allocation. Such systems consist of multiple compute nodes which are connected with a high performance interconnect and each node operates with its own data. The only way to distribute computations across different nodes is sending and receiving messages over the interconnect.

Distributed memory drastically complicates parallel programming. The programmer has to take into account not only distribution of computation but also distribution of data and cost of data movement. To minimize communication overhead the programmer has to ensure data locality. Another goal is even data distribution in order to balance computations.

Unlike incremental parallelization applicable for shared memory, distributed memory requires global decision making since individual parts of a program may impose conflicting requirements. These conflicts will ultimately lead to additional communications aimed at data redistribution.

One of the parallel programming models, widely used to develop compute-intensive applications on distributed-memory clusters, is the Message Passing Interface (MPI). However, its low-level forces the programmer to manage distribution of data and computation manually, as well as communications. This means that parallelization for distributed memory even of a simple program can be very error-prone and time-consuming. Therefore, automation of parallel programming for distributed-memory systems becomes very much desirable. Especially while the complexity and size of systems is growing from year to year.

In this paper, we propose a technique for automatic translation of sequential programs of scientific-technical calculations to parallel ones suitable for execution on heterogeneous computational clusters. Our technique aims at parallel execution of structured grid computations and allows processing sequences of arbitrary nested loops with almost affine accesses.

The problem of distributed memory parallelization requires a solution to three main sub-problems. An automation tool has to distribute data as well as computation and has to manage communications: typically accesses to remote data and data redistribution. The proposed technique overcomes all these problems. However, in the paper we pay the most attention to data distribution because yet we do not find any common solution to this sub-problem, which is suitable for large compute applications, in the current literature.

We implemented our technique as an automatic parallelizing compiler which generates a parallel version of a sequential C or Fortran program with parallelism specifications expressed with DVMH [1][2] directive-base programming model. To extract properties of an original program which are necessary for its parallelization the compiler relies on static and dynamic analysis techniques. We also follow an implicit parallel programming methodology [3][4] which implies that the sequential program must be well-formed. Thus, a preliminary sequential program transformation may be helpful. The user may also assert some high-level program properties which are essential for parallelization. Nevertheless, automatic parallelization ensures that the programmer does not write parallel code directly.

This paper makes the following contributions:

- a novel approach to automatic program parallelization for distributed memory systems which covers distribution of data and computations as well as access to remote data and data redistribution;
- a parallelizing compiler for code generation which produces parallel versions of C and Fortran programs suitable for execution on heterogeneous computational clusters;
- experimental evaluation of the presented approach on some programs from the NAS Parallel Benchmarks [5].

The rest of the paper is organized as follows. Section 2 presents our program execution model and proposes a solution to the problem of distributed memory parallelization focusing on the data distribution sub-problem. Section 3 summarizes implementation details and briefly describes frameworks that we used to analyze sequential programs and to implement code generation. Section 4 provides experimental results. Section 5 discusses the related work and, finally, section 6 concludes this paper.

2. Distributed memory parallelization

2.1 Execution model

In this section, we highlight the main details of our abstract execution model representing a distributed memory system.

Our parallelization technique aims at processing sequential programs with structured grid computations. Hence we assume that a distributed memory system is formed by a multidimensional grid of a virtual nodes and each node executes operations on a part of the computational grid. The owner-computes rule [6] is applied to determine a node to execute each assignment statement. Thus, each node has its own (i.e. local) data which are allocated in its own memory space, which is not visible to other nodes. It also may access remote data which are allocated on other nodes by sending and receiving messages over the interconnect.

We also assume that the main source of parallelism is nested loops and that entire iteration of a loop can be executed by a single node only. These loops produce computations on multidimensional arrays which are the main source of data to be partitioned between virtual compute nodes. A data distribution rule divides array dimensions into almost equal blocks and implies that each node has its own sub-array with the same number of dimensions but of smaller sizes.

If the number of dimensions of a grid of nodes is less than the number of array dimensions, some array dimensions are not partitioned. Otherwise, if the number of grid dimensions is greater than the number of array dimensions, some array dimensions are replicated between nodes.

If we consider two arrays accessed in the same loop there is some kind of intuitive relation between elements of different arrays. To reduce the communication overhead, we have to distribute elements accessed at the same loop iteration to the same node. Thus, an alignment of an array A with a distributed array B is an accordance between an element of the array A and an element or a sub-array of the array B. This accordance aims to reduce the cost of data movement. To specify alignment we use affine expressions in a form $a * i + b$. Therefore, a distribution rule for the array B defines the distribution rule for the array A, i.e. if an element i of B is allocated on a compute node, the corresponding element $a * i + b$ of A is allocated on the same node.

If there is no distribution or alignment rule for a variable, we replicate it between all compute nodes. The replicated variable must have the same value in each node except reduction and private variables.

Different array accesses may produce different alignments, however, we choose only one of them to generate parallel version of a program. Other violated alignments lead to data movement. Moreover, non-affine alignments cannot be established and always lead to data movement. Our execution model supports two kinds of communications. Firstly, violated affine alignments allow us

to implement shadow edges [7]. For structured grid applications, it is useful to extend each sub-array with a shadow area to overlap with its neighbors. Shadow edges have to be updated only if their values have been altered on the owner node. Secondly, all other communications are implemented as an immediate access to remote data when corresponding data are required.

2.2 Data Distribution

In this section, we present our technique aimed to solve the data distribution sub-problem. This technique relies on a graph-based representation of an array alignment. We introduce a graph of arrays which depicts arrays accessed in loop nests. A vertex of the graph is a dimension of an array and an edge connects dimensions that should be aligned. Note that the alignment relation is transitive. Thus, two dimensions are aligned if there is a path between corresponding vertexes in a graph.

A pair of array accesses in a loop produces edges in a graph. Coefficients in affine subscript expressions which calculate offset from the beginning of array dimensions are attached to the edge and allow us to infer alignment expression. Only affine subscript expressions, which depend on the same single counter of a loop, produce edges. Note, that values of distinct induction variables of the same loop can be computed through the loop counter. Therefore, corresponding subscript expressions also produce edges in a graph.

If a pair of array accesses produces an edge in a graph, a kind and a weight of alignment are also attached to the edge. Three kinds of alignments are possible:

- WW if both accesses write into memory,
- WR if one of accesses writes into memory and another access reads from memory,
- RR if both accesses read from memory.

A weight of alignment equals to $Loop_w * Transfer_w$. $Transfe_w$ estimates a number of bytes to be send or received if the alignment is violated. Computation of $Transfer_w$ is based on sizes of arrays which are known from static and dynamic analysis. $Loop_w$ estimates a corresponding loop weight and shows how often the loop is executed. Static and dynamic analysis techniques allow us to compute loop weights. If an edge with the same subscript coefficients and the same kind already exists in a graph its weight is updated.

It is necessary to know a kind of alignment to compare different edges according to memory access types. A priority of WW edge is higher than a priority of WR edge which, in its turn, is higher than a priority of RR. The owner-computes rule establishes this priority because it is forbidden to write to non-local memory.

In the first step, loops in a sequential program are analyzed to determine all possible alignments which depend on array accesses at the same loop iteration. Procedure $buildGraph$, shown in Listing 1, builds a corresponding graph of arrays. It calls the following procedures when necessary:

- $dimension(Expr)$ to determine a dimension of an array which this subscript expression corresponds to,
- $kind(Acc_1, Acc_2)$ to determine a kind of alignment which these array accesses produce,
- $array(Acc)$ to determine a top level declaration of an accessed array, i.e. it analyzes a call graph and establishes correspondence between formal and actual parameters which have an array type.

This procedure produces a graph of arrays which may be ambiguous or may have conflicting edges. Two cases are possible:

- there is a cycle in a graph that implies two different affine expressions to specify an alignment of the same array dimension,
- two different dimensions of an array are aligned on each other, i.e. two different dimensions of

the same array should be mapped to the same node..

```
procedure buildGraph
  for L ∈ Loops of a program
   for Acc1 ∈ Array accesses in L
    for Expr1 ∈ Subscripts in Acc1
     if Expr1 is a1 * i + b1 and i is loop counter of L
      for Acc2 ∈ Array accesses in L
       for Expr2 ∈ Subscripts in Acc2
        if (Acc1 ≠ Acc2 or dimension(Expr1) ≠ dimension(Expr2)) and
           Expr2 is a2 * i + b2 and i is loop counter of L
          Add vertexes V_{array(Acc1)}^{dimension(Expr1)}, V_{array(Acc2)}^{dimension(Expr2)}, if they do not exist.
          Add an edge E_{a1,b1,a2,b2}^{kind(Acc1,Acc2)} (V_{array(Acc1).}^{dimension(Expr1)}, V_{array(Acc2).}^{dimension(Expr2)}), if it does not exist.
          Update weight of the edge.
```

Listing 1. Algorithm to build a graph of arrays

In the second step, an original graph of arrays is reduced to disambiguate these conflicts. The goal is to minimize total weight of edges to be removed from the graph. Fig. 1 shows a graph of arrays for a fragment of a source code in Listing 2.

```
integer A(40,50), B(40,50)
do i = 1, 30
  z = A(i, 1)
  do k = 5, 30
    a(i, k) = b(i, k) + b(k, k) / z
  enddo
enddo
do i = 1, 30
  z = a(i, 2)
  do k = 5, 30
    a(i, k) = b(i, k + 1) + z
  enddo
enddo
```

Listing 2. Example of a Fortran program which is used to build a graph of arrays in Fig. 1

Different weight models can be used to calculate w_1, w_2, w_3, w_4 weights, so we do not specify exact weights in the example. Assuming $w_1 > w_2$, $w_3 > w_2$ and $w_3 > w_4$, the solid lines determine a possible graph after all conflicts are disambiguated.

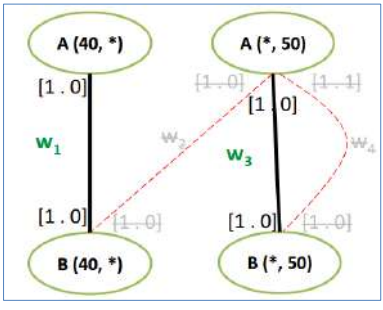


Fig. 1. Graph of arrays for the program in Listing 1

We adapted Prim's algorithm for finding a minimum spanning tree (MST). A minimum spanning tree is a subset of the edges of a connected edge-weighted undirected graph that connects all the vertices together. It has the minimum possible total edge weight and does not have cycles. If the

source graph is connected, then a spanning tree is built, but if there are several disconnected components in the source graph, then the result is a spanning forest. Our goal is to find the set of edges with the highest weight. Thus, we can use an algorithm similar to Prim's algorithm to find a maximum spanning tree, i.e. a tree which has the maximum possible total edge weight.

To resolve conflicts of a second type, we add temporary edges between all different dimensions of the same array. A weight of these edges must be greater than the sum of all weights in the graph. Hence, these temporary edges are never removed and a spanning tree does not contain edges that imply alignment of two dimensions of the same array on each other.

This algorithm has linear complexity depending on the number of vertices in the graph. However, the found solution is not always optimal, but the search of optimal solution has exponential complexity and it is not applicable in practice if the total number of dimensions of arrays in a program is significantly greater than 10. On the other hand, the algorithm for finding the maximum spanning tree has not only linear complexity but can be also performed in parallel, so it is applicable to any program with any number of arrays.

3. Code generation

We implemented our technique as automatic parallelizing Fortran and C compiler. We adapted the compiler from the System FOR Automated Parallelization (SAPFOR) [8], which is a software development suit that is focused on cost reduction of manual program parallelization, to implement our parallelization technique. To express parallelism specifications in a code the DVMH [1][2] directive-based programming model is used.

2.3 The DVMH programming model

DVMH was designed to create parallel programs of scientific technical calculations for heterogeneous computational clusters. This model provides the programmer with high-level specifications of parallelism. Thus, it introduces directives to specify data and computation distribution, to manage accesses to remote data and to highlight the computational regions which can be executed on accelerators and multiprocessors.

We choose the DVMH model because it satisfies constraints which our execution model exposes. The DVMH languages allow us to generate distribution and alignment rules in a simple way, as well as to implement shadow edges and other accesses to remote data. Moreover, the DVMH compilers and run-time system implement various optimizations: data transformation at run-time to choose the right memory access pattern, dynamic CUDA handler compilation during the program run-time, parallel execution of loops with regular loop carried dependencies on GPU and others. Therefore, the presence of these optimizations significantly simplifies the implementation of the automatic compilers.

Listing 3 shows the distribution (the *distribute* specification) and alignment (the *align* specification) rules for some arrays in the BT application from the NAS Parallel Benchmarks. The *shadow* specification determines sizes of shadow edges for the array *u*.

```
#pragma dvm array distribute [block][block][block]
double us[KMAX/2*2+1][JMAX/2*2+1][IMAX/2*2+1];

#pragma dvm array align ([k][j][i][] with us[k][j][i]),\
                  shadow[2:2][2:2][2:2][2:2]
double u[(KMAX+1)/2*2+1][(JMAX+1)/2*2+1][(IMAX+1)/2*2+1][5];
```

Listing 3. Example of data distribution in the CDVMH language.

Listing 4 shows the computation decomposition (the *parallel* specification) for a loop nest in the BT application. The computation decomposition depends on the data distribution. Hence the definition

of a parallel loop must include the *on* specification which associates loops in the perfect loop nest with dimensions of a distributed array.

The mentioned specifications imply a relation between loop iterations and array elements that enables compile-time and runtime optimizations. For example, knowing how arrays are associated with each other and knowing the loop mapping rules, the DVMH runtime system determines the optimal representation of arrays in the device memory for a given parallel loop, and subsequently, it performs the dynamic transformation of arrays before the loop execution. As a result, on GPUs all accesses to global memory, performed by CUDA threads of each warp, will be combined. Adjacent threads of the CUDA-block will access neighboring cells of the GPU's global memory and the loop can be performed up to 10 times faster [9].

The hyperplane method (the *across* specification) is implemented in the DVMH model to execute loops with regular loop carried dependencies (some kind of flow dependencies with distances limited by a constant) in parallel. The per-diagonal transformation is implemented in the DVMH runtime system to efficiently execute these kinds of loops on GPUs [9].

To update shadow edges the *shadow_renew* specification is used. To specify all other communications the DVMH model provides the *remote_access* specification.

```
#pragma dvm region
{
#pragma dvm parallel ([k][j][i] on u[k][j][i][])\
            reduction(sum(r1),sum(r2),sum(r3),sum(r4),sum(r5)),\
            private(u_exact,xi,eta,zeta,m,add)
  for(k = 0; k <= PROBLEM_SIZE - 1; ++k)
    for(j = 0; j <= PROBLEM_SIZE - 1; ++j)
      for(i = 0; i <= PROBLEM_SIZE - 1; ++i) {
      ...
        for (m = 1; m <=5; ++m)
          u_exact[m-1] = ...
               add = u[k][j][i][0] - u_exact[0];
               r1 = r1 + add * add;
               ...
      }
}
```

Listing 4. Example of computation distribution in the CDVMH language

The *region* specification forms the computational region, which can be executed on different computational devices (multi-core CPU and accelerators). The region is a fragment of a source code with one entrance and one exit. Each region may enclose one or more parallel loops. The iterations of parallel loops inside the region are partitioned between the devices selected for the region execution according to the *parallel* specification.

Code fragments outside the regions are always executed on CPU. The DVMH model introduces actualization directives (the *actual* and *get_actual* specifications) to manage data transfer between random access memory of CPU and memories of accelerators. These specifications must be placed outside computational regions. The deferred semantic of actualization directives allows DVMH runtime system to avoid redundant data transfer. Moreover, there is no difference whether all regions are executed on GPU or some part of them is targeted to the CPU-only execution. Actualization directives affect only transfer between regions and code fragments outside them. The DVMH runtime system will manage the necessary data transfer in an automatic way.

2.4 SAPFOR

SAPFOR includes an automatic parallelizing compiler that relies on static [10] and dynamic [11] analysis techniques. However, unlike conventional automatic parallelizing compilers, which may

suffer from a lack of user participation, SAPFOR relies on an implicitly parallel programming model and involves the user in the parallelization process [12].

SAPFOR has already implemented semi-automatic parallelization approach to exploit loop-level parallelism for multi-core processors and accelerators. It uses the DVMH model to express parallelism specifications. SAPFOR inserts three kinds of annotations: specification of parallel loops, specification of compute regions and specification of data transfer between a memory of CPU and memory of accelerator. It searches for the outermost perfect loop nests to execute them in parallel and then it examines whether some properties of the loop nest prevent its parallel execution (safety of control flow and memory accesses, canonical loop form, etc.). To insert actualization directives SAPFOR examines the direction of data usage in every loop. It also applies alias analysis to determine indirect memory accesses in C programs. SAPFOR relies on the optimization of data transfer at run-time that DVMH run-time library makes. So, it marks each assignment to the variable outside a compute region with the *actual* specification and places the *get_actual* specification before each statement which uses the variable changed in any DVMH region.

In our work we extend SAPFOR capabilities to produce parallel versions of C and Fortran programs suitable for execution on distributed memory systems. Our implementation allows the compiler to insert:

- *distritbute* and *align* specifications to partition array elements between compute nodes,
- *parallel* and *on* specifications to partition computations according data distribution,
- *shadow*, *shadow_renew* and *remote_access* specifications to express necessary data transfer between compute nodes.

We implemented our parallelization technique as a separate library. This library provides interface to build the graph of arrays and to get data and computation distribution in the language independent way. It automatically resolves possible conflicts in the graph and rejects arrays that cannot be partitioned across multiple computational nodes.

The core of the SAPFOR compiler architecture is the SAPFOR pass framework. Passes perform analysis and transformation of the program. New passes can be constructed to implement new capabilities. We added new SAPFOR passes to collect and fill data necessary to build the graph of arrays. To place necessary DVMH specifications in a source code we also extended the code generation passes for both C and Fortran programming languages. We use passes available in SAPFOR to determine loop-carried and spurious data dependencies, to find parallelizable loops and to find computational regions suitable for execution on accelerators and multiprocessors.

4. Experimental results

This section presents performance results from 3 applications from the NAS Parallel Benchmarks [5]: BT (Block Tri-diagonal solver), CG (Conjugate Gradient), EP (Embarrassingly Parallel) that we parallelized using out technique. Fig. 2 and fig. 3 show the execution time of the generated DVMH programs in comparison with the origin MPI programs written by the developers of the NAS Parallel Benchmarks.

We conducted experiments on the K10 [11] cluster of NUMA nodes. Each node contains two 8-core Intel Xeon E5-2660 processors and three GPUs NVIDIA Tesla M2090. All codes were compiled with Intel C/C++ and Fortran compilers version 14.0.1 with option -O2. DVMH compiler was preliminary used to translate programs with DVMH specifications to MPI+OpenMP+CUDA code. For all experiments, we use the total power of one, four and nine nodes. We did not use GPUs to achieve the fair comparison of our solution to distributed-memory parallelization problem with a manual parallelization approach using MPI.

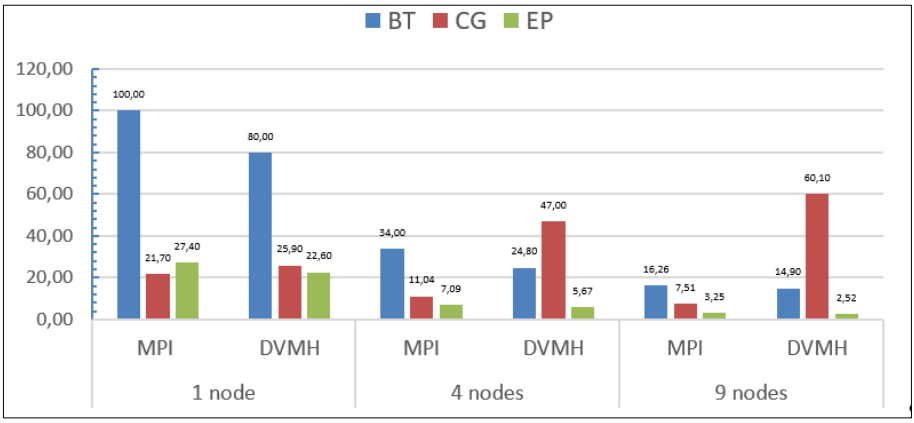


Fig. 2. Times in seconds of Fortran programs, NPB 3.3 class C

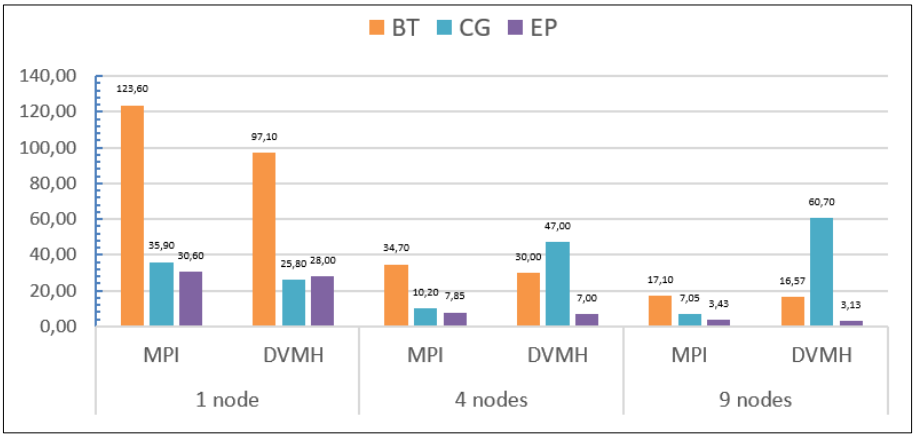


Fig. 3. Times in seconds of C programs, NPB 3.3 class C

Unfortunately, in the suite, there are no built-in C versions of considered applications, so we translated Fortran to C manually. To obtain well-formed versions of original programs, which can be converted to DVMH programs fully automatically, we applied some source-to-source transformations implemented in SAPFOR (functions inlining, loop fusion, loop distribution, and etc.). Also, we manually did some transformations which have not been implemented in SAPFOR yet.

Table 1 and Table 2 show speedup of parallel DVMH programs relative to their well-formed sequential versions. The first group of columns represents execution time of sequential programs before and after source-to-source transformations.

Number of transformations implemented in SAPFOR must be extended to obtain well-formed versions of other applications from the suite. We plan to review these transformations in future works and figure out basic source-to-source transformations that can be done automatically. Their implementation as separate passes in the SAPFOR pass framework will allow us to parallelize other applications.

Table 1. Speedup of Fortran-DVMH programs relative to well-formed sequential versions

	Serial, time (s.)		1 node (16 MPI)		4 node (64 MPI)		9 node (144 MPI)	
	Origin	Well-formed	Time (s.)	Speedup	Time (s.)	Speedup	Time (s.)	Speedup
BT	925,77	934,14	80,00	11,7	24,80	37,7	14,90	62,7
CG	282,99	297,67	25,90	11,5	47,00	6,3	60,10	4,95

EP	341,53	381,73	22,60	16,9	5,67	67,3	2,52	151,5

Table 2. Speedup of CDVMH programs relative to well-formed sequential versions

	Serial, time (s.)		1 node (16 MPI)		4 node (64 MPI)		9 node (144 MPI)	
	Origin	Well-formed	Time (s.)	Speedup	Time (s.)	Speedup	Time (s.)	Speedup
BT	955,04	816,38	97,10	8,4	30,00	27,2	16,57	49,3
CG	314,43	295,74	25,80	11,5	47,00	6,3	60,70	4,9
EP	431,95	408,89	28,00	14,6	7,00	58,4	3,13	130,6

The experiments show that automatically generated versions of BT and EP applications have the similar performance to the manually parallelized ones.

However, the MPI version of CG program running on more than one node significantly outperforms the automatically generated DVMH version. Detailed examination of the benchmark shows that most of time is spent in the multiplication of a sparse matrix by a vector. This operation leads to indirect array accesses which cannot be efficiently processed by our parallelization technique that aims at processing sequential programs with structured grid computations. The current implementation produces at each iteration a collective operation to broadcast remote data between all processes. The cost of this data movement degrades parallel program performance if data are transferred through the interconnect between different nodes. If GPUs are used, then 16 processes (1 node) will be enough to achieve high performance.

The available extension of the DVMH model for irregular grids may be helpful to increase parallelization performance, so we plan to address this problem in future works and to improve our parallelization technique.

5. Related works

Various approaches exist to simplify parallelization for distributed-memory systems. However, most of approaches do not overcome all three main sub-problems.

For example, [6] proposes an approach to derive the computation decomposition from predetermined data distribution. The polyhedral approach is used to develop a mathematical model for code generation and to optimize the communications. These optimizations include eliminating redundant messages, aggregating messages, and hiding the communication latency by overlapping the communication with computation [6].

The predefined data distribution was also used in the [14] tool. This tool provided the user with an interactive subsystem to manage the parallelization process, to define data distribution and to choose program transformations. In both works the owner-computes rule is applied to derive the computation decomposition. However, [6] makes it possible to relax owner-computes rule. Thus, locations written to can be replicated or mapped to different nodes.

The Molly [15] tool extends the capabilities of the Polly [16] compiler for shared memory systems, built on top of LLVM [17]. Molly also relies on the polyhedral model. It introduces a special data type to define distributed arrays which allows the compiler to control the absence of pointer arithmetic operations applied to distributed data. The tool uses a fixed block-distribution while communications are generated in an automatic way. The sizes of blocks are equal for every node. There is no way for the programmer to specify alignment of data on each other. To derive computation distribution the own-computes rule is applied. It is assumed that in the future the user will be able to define an arbitrary mapping of data and computation by putting appropriate directives in a source code. Molly also imposes additional restrictions on the well-structured code fragments (SCoPs), does not deal with reduction operations and supports distribution of global data only. Hence an accurate interprocedural analysis is not required.

An automatic parallelizing compiler presented in [18] also uses the polyhedral compiler framework to translate sequential C programs to parallel ones which are suitable for execution on distributed-memory systems. This work extends the capabilities of the Pluto [19] polyhedral compiler for shared memory systems. The data distribution is not required, and the computation partitioning and data dependencies determine the owner of data at a particular moment. Therefore, this approach does not support the global decision-making in data distribution and finally may lead to an increase in the frequency and the volume of communications.

The problem of data distribution was addressed in the Paradigm [20] tool. The research aims at parallelizing sequential programs in the Fortran 77 language. It also addresses other parallelization problems along with data distribution: communication optimization, support for irregular computations, multi-threaded execution and pipeline execution of loops with cross-iteration dependencies. However, only simple inter-dimensional alignment is performed and nor offset, nor stride within a given pair of dimensions cannot be used. The practical applicability of the tool is not clear because the experimental results are only given for small computing kernels.

6. Conclusion

In this paper, we introduce the technique for automatic translation of sequential programs of scientific-technical calculations to parallel ones suitable for execution on heterogeneous computational clusters. We emphasize in the paper that a problem of distributed memory parallelization requires a solution to three main sub-problems. These sub-problems include data and computation distribution as well as communication optimization and our technique addresses all of them. In the paper, we focus more on data distribution sub-problem because we do not find yet any common solution to this sub-problem, which is suitable for a large compute applications, in the current literature.

We present a novel data structure, called the graph of arrays, and use it to derive data decomposition from affine array accesses in loop nests. We also determine an alignment of arrays with each other to reduce the frequency and the volume of data movement. To choose between possible alignments, we estimate communication costs if one of them is violated.

We implemented our technique as an automatic parallelizing compiler which generates a parallel version of a sequential C or Fortran program with parallelism specifications expressed with DVMH directive-base programming model. Conducted experiments show that in conjunction with implicit parallel programming methodology the generated code is capable of matching hand-coded MPI versions of programs with structured grid computations.

References / Список литературы

[1]. Коновалов Н.А., Крюков В.А. и др. Fortran DVM - язык для разработки мобильных параллельных программ. Программирование, том: 21, вып. 1, 1995 г., стр. 49-54 / Konovalov N.A., Krukov V.A. et al. Fortan DVM: a Language for Portable Parallel Program Development. Programming and Computer Software, vol. 21, no. 1, 1995, pp. 35-38.

[2]. Bakhtin V.A., Klinov M.S. et al. Extension of the DVM-model of parallel programming for clusters with heterogeneous nodes. Bulletin of South Ural State University, series: Mathematical Modeling, Programming & Computer Software, no. 18 (277), issue 12, 2012, pp. 82-92 (in Russian) / Бахтин В.А., Клинов М.С. и др. Расширение DVM-модели параллельного программирования для кластеров с гетерогенными узлами. Вестник ЮУрГУ. Серия: Математическое моделирование и программирование, № 18 (277), вып. 12, 2012 г., стр. 82-92.

[3]. Hwu W.-m., Ryoo S. et al. Implicitly parallel programming models for thousand-core microprocessor, In Proc. of the 44th annual Design Automation Conference (DAC '07), 2007, pp. 754-759.

[4]. Vandierendonck H., Rul S., De Bosschere K. The Paralax infrastructure: automatic parallelization with a helping hand. In Proc. of 2010 19th International Conference on Parallel Architectures and Compilation Techniques (PACT), 2010, pp. 389-400.

[5]. NAS Parallel Benchmarks. Available at https://www.nas.nasa.gov/publications/npb.html, accessed 16.09.2022.

[6]. Amarasingh S.P., Lam M.S. Communication Optimization and Code Generation for Distributed Memory Machines. In Proc. of the ACM SIGPLAN 1993 Conference on Programming Language Design and Implementation, 1993, pp. 126-138.
[7]. Thomas B. HPF Library, Language and Compiler Support for Shadow Edges in Data Parallel Irregular Computations. In Proc. of the 8th International Workshop on Compilers for Parallel Computers, 2000, pp. 21-34.
[8]. Klinov M.S., Krukov V.A. Automatic parallelization of Fortran programs. Mapping to cluster. Vestnik of Lobachevsky University of Nizhni Novgorod, no. 3, 2009, pp. 128-134 (in Russian) / Клинов М.С., Крюков В.А. Автоматическое распараллеливание Фортран-программ. Отображение на кластер. Вестник ННГУ, no. 2, 2009 г., стр. 128-134.
[9]. Bakhtin V.A., Kolganov A.S. et al. Methods of dynamic tuning of DVMH programs on clusters with accelerators. In Proc. of the International conference «Russian Supercomputing Days», 2015, pp. 257–268 (in Russian) / Бахтин В.А., Колганов А.С. и др. Методы динамической настройки DVMH-программ на кластеры с ускорителями. Труды международной конференции «Суперкомпьютерные дни в России», 2015, стр. 257-268.
[10]. Kataev N.A. Application of the LLVM compiler infrastructure to the program analysis in SAPFOR. Communications in Computer and Information Science, vol. 965, 2018, pp. 487-499.
[11]. Kataev N. A., Smirnov A.A., Zhukov A.D. Dynamic data-dependence analysis in SAPFOR. CEUR Workshop Proceedings, vol. 2543, 2020, pp. 199-208.
[12]. Kataev N.A., Kolganov A.S. Additional parallelization of existing MPI programs using SAPFOR. Lecture Notes in Computer Science, vol. 12942, 2021, pp. 41-52.
[13]. K10 Heterogeneous cluster K10. Available at https://www.kiam.ru/MVS/resourses/k10.html, accessed 16.09.2022.
[14]. Zima H., Bast H., Gerndt M. SUPERB: A tool for semi-automatic MIMD/SIMD parallelization. Parallel Computing, vol. 6, issue 1, 1988, pp. 1-18.
[15]. Kruse M. Introducing Molly: distributed memory parallelization with LLVM. arXiv.1409.2088, 2014, 9 p.
[16]. Grosser T., Groesslinger A., Lengauer C. Polly - performing polyhedral optimizations on a low-level intermediate representation. Parallel Processing Letters, vol. 22, issue 04, 2012, 27 p.
[17]. Lattner C., Adve V.: LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In Proc. of the 2004 International Symposium on Code Generation and Optimization (CGO'04). 2004, pp. 75-86.
[18]. Bondhugula U. Compiling affine loop nests for distributed-memory parallel architectures. In Proc. of the International Conference on High Performance Computing, Networking, Storage and Analysis, 2013, pp. 1-12.
[19]. Bondhugula U., Hartono A. et al. A practical automatic polyhedral parallelizer and locality optimizer. SIGPLAN Notices, 43, issue 6, 2008, pp. 101-113.
[20]. Banerjee P., Chandy J.A. et al. The Paradigm compiler for distributed-memory multicomputers. Computer, vol. 28, issue 10, 1995, p. 37-47.

Information about authors / Информация об авторах

Никита Андреевич КАТАЕВ – научный сотрудник. Сфера научных интересов: компиляторные технологии, параллельные вычисления, оптимизация кода.

Nikita Andreevich KATAEV – research scientist. Research interests: compilers, parallel programming, program optimization.

Александр Сергеевич КОЛГАНОВ – кандидат физико-математических наук, научный сотрудник. Сфера научных интересов: параллельные вычисления, оптимизация кода, графические процессоры.

Alexander Sergeevich KOLGANOV– PhD, research scientist. Research interests: parallel programming, program optimization, GPUs.

DOI: 10.15514/ISPRAS-2022-34(4)-8

Настройка критериев планировщика СУБД с учётом динамической компиляции

[1,2] *Е.В. Долгодворов, ORCID: 0000-0001-6962-6448 <krym4s@ispras.ru>*
[1] *Р.А. Бучацкий, ORCID: 0000-0001-8522-1811 <ruben@ispras.ru>*
[1] *М.В. Пантилимонов, ORCID: 0000-0003-2277-7155 <pantlimon@ispras.ru>*
[1] *Д.М. Мельник,* ORCID: *0000-0002-0177-1558 <dm@ispras.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН,*
109004, Россия, г. Москва, ул. А. Солженицына, д. 25
[2] *Московский физико-технический институт*
141701, Московская область, г. Долгопрудный, Институтский переулок, д.9

Аннотация. С разработкой динамического компилятора запросов актуальной становится задача настройки критериев оптимизатора и планировщика СУБД для выбора оптимального, с точки зрения динамической компиляции, плана запроса. Необходимость настройки критериев оптимизатора обуславливается тем, что свойства различных моделей выполнения накладывают ограничения на эффективность выполнения плана запроса с использованием определённых узлов-операторов. Так, например, используемая в динамическом компиляторе push-модель даёт преимущество при выполнении запросов с использованием последовательного сканирования. При использовании динамической компиляции сканирование индекса может выполняться значительно менее эффективно, чем последовательное сканирование. Использование в плане вершин сканирования индекса уменьшает ценность метода динамической компиляции. Для преодоления указанных проблем предлагается выполнить настройку оптимизатора СУБД таким образом, чтобы тот оценивал и учитывал эффективность использования некоторых типов узлов при построении плана запроса с его последующей динамической компиляцией. В данной работе рассматривается модификация планировщика PostgreSQL для выбора наиболее эффективного пути выполнения запроса с учётом аппаратных характеристик и различий между интерпретируемой и компилируемой моделями выполнения узлов-операторов.

Ключевые слова: динамическая компиляция; JIT-компиляция; СУБД; PostgreSQL; LLVM; языки запросов; планировщик

Для цитирования: Долгодворов Е.В., Бучацкий Р.А., Пантилимонов М.В., Мельник Д.М. Настройка критериев планировщика СУБД с учётом динамической компиляции. Труды ИСП РАН, том 34, вып. 4, 2022 г., стр. 101-116. DOI: 10.15514/ISPRAS-2022-34(4)-8

Благодарности: Работа выполнена при поддержке Российского фонда фундаментальных исследований, проект № 20-07-00877 А.

JIT-aware DBMS planner configuration

[1,2] *E.V. Dolgodvorov, ORCID: 0000-0001-6962-6448 <krym4s@ispras.ru>*
[1] *R.A. Buchatskiy, ORCID: 0000-0001-8522-1811 <ruben@ispras.ru>*
[1] *M.V. Pantilimonov, ORCID: 0000-0003-2277-7155 <pantlimon@ispras.ru>*
[1] *D.M. Melnik, ORCID: 0000-0002-0177-1558 <dm@ispras.ru>*

[1] *Ivannikov Institute for System Programming of the RAS,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russian Federation
[2] *Moscow Institute of Physics and Technology*
9 Institutskiy per., Dolgoprudny, Moscow Region, 141701, Russian Federation

Abstract. Dynamic compilation of certain operator compositions might have a drastic impact on overall query performance, but not be considered during optimal query plan selection by DBMS planner due to lack of knowledge. To tackle this problem, we propose to extend cost model with criteria that make dynamic compilation overhead relevant. The necessity to set up the optimizer criteria is due to the fact that the properties of various execution models impose restrictions on the efficiency of query plan execution using certain operator nodes. For example, the push-model used in the dynamic compiler is advantageous when executing queries using sequential scan. So, dynamic compilation makes sequential scan more efficient than index scan. Using index nodes in such a plan makes the value of the dynamic compilation method diminishing. To overcome these problems, it is proposed to configure the DBMS optimizer, so that it evaluates and takes into account the efficiency of using certain types of nodes when building a query plan with its subsequent dynamic compilation. This paper discusses the modification of the PostgreSQL planner to select the most efficient query execution plan based on hardware characteristics and the execution model of operator nodes with interpretation or compilation.

Keywords: dynamic compilation; JIT-compilation; RDBMS; PostgreSQL; LLVM; query languages; planner.

For citation: Dolgodvorov E. V., Buchatskiy R. A., Pantilimonov M. V., Melnik D.M. JIT-aware DBMS planner configuration. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 4, 2022. pp. 101-116 (in Russian). DOI: 10.15514/ISPRAS-2022-34(4)-8

Acknowledgements. This work was supported by the Russian Foundation for Basic Research, project № 20-07-00877 A.

1. Введение

В ходе тестирования производительности разработанного в ИСП РАН [1-3] динамического компилятора запросов для СУБД PostgreSQL [6] на тестовом наборе TPC-H [9] было обнаружено, что для некоторых запросов выбор последовательного сканирования вместо сканирования индекса дает значительное ускорение при выполнении запроса с динамической компиляцией. Качественные различия в скорости выполнения рассмотрены на примере запроса Q06 (см. листинг 1). При тестировании использовалась база данных объемом 30 Гб, сервер с архитектурой ARM. Планы выполнения запроса Q06 с последовательным и индексным сканированием представлены в табл. 1.

```
SELECT
    SUM(l_extendedprice * l_discount) AS revenue
FROM
    lineitem
WHERE
    l_shipdate >= DATE '1996-01-01'
    AND l_shipdate < DATE '1996-01-01' + INTERVAL '1' YEAR
    AND l_discount BETWEEN 0.09 - 0.01 AND 0.09 + 0.01
    AND l_quantity < 24;
```

Листинг 1. Текст SQL-запроса Q06 из тестового набора TPC-H
Listing 1. Q06 SQL-query from the TPC-H test suite

Табл. 1. Планы выполнения запроса Q06 с использованием сканирования индекса (наверху) и последовательного сканирования (внизу)
Table 1. Q06 query execution plans using an index scan (top) and a sequential scan (bottom)

```
Aggregate
  -> Bitmap Heap Scan ON lineitem
     Recheck Cond: ((l_shipdate >= '1996-01-01'::DATE)
       AND (l_shipdate < '1997-01-01'::TIMESTAMP WITHOUT TIME zone))
     FILTER: ((l_discount >= '0.08'::DOUBLE PRECISION)
       AND (l_discount <= '0.1'::DOUBLE PRECISION))
  -> Bitmap INDEX Scan ON i_l_shipdate
     INDEX Cond: ((l_shipdate >= '1996-01-01'::DATE)
       AND (l_shipdate < '1997-01-01'::TIMESTAMP WITHOUT TIME zone))
```

```
Aggregate
  -> Seq Scan ON lineitem
    FILTER: ((l_shipdate >= '1996-01-01'::DATE)
    AND (l_shipdate < '1997-01-01'::TIMESTAMP WITHOUT TIME zone)
    AND (l_discount >= '0.08'::DOUBLE PRECISION)
    AND (l_discount <= '0.1'::DOUBLE PRECISION)
    AND (l_quantity < '24'::DOUBLE PRECISION))
```

Время выполнения измерялось путём многократного выполнения запроса и подсчёта медианы полученных результатов. Результаты представлены в табл. 2.

Табл. 2. Время выполнения запроса Q06 с использованием сканирования индекса и последовательного сканирования
Table 2. Q06 query execution time using index scan and sequential scan

План запроса	Интерпретатор, сек.	Динамический компилятор, сек.
Bitmap Index Scan	21,48	18,77
Seq Scan	29,20	6,00

Стандартный планировщик выбирает сканирование индекса, не являющееся эффективным при выполнении с динамической компиляцией. Требуется разработать критерии выбора плана исполнения для PostgreSQL, учитывающие особенности динамической компиляции запросов. Такие критерии должны отдавать предпочтение планам, эффективным с точки зрения динамической компиляции, а не интерпретатора.

В данной работе предлагается настройка критериев планировщика, которая с большей точностью сопоставляет стоимость определённого плана со временем его выполнения при использовании динамического компилятора запросов.

2. Динамический компилятор запросов СУБД PostgreSQL

Разработанный в ИСП РАН динамический компилятор запросов [1–3] представляет собой расширение к СУБД PostgreSQL, генерирующий коды алгоритмов узлов-операторов на LLVM IR [4] с изменённой моделью выполнения. Эти изменения увеличивают эффективность выполнения узлов-операторов, что покрывает расходы на динамическую компиляцию.

2.1 Push-модель выполнения запросов

В динамическом компиляторе запросов мы переходим переход от используемой во многих современных СУБД *Volcano*-модели [8] исполнения запросов к модели явных циклов. План запроса в PostgreSQL представляет собой дерево, в котором листовые вершины – узлы сканирования, а их родительские вершины – узлы соединения и агрегации.

Volcano-модель, которую также называют *pull*-моделью, реализует каждый оператор с помощью итераторов, поддерживающих интерфейс *open(), next(), close()*. Первая и последняя

функции отвечают соответственно за инициализацию и освобождение ресурсов, нужных для работы узла-оператора. В этой модели вызов узлов выполняется от родителей к детям с помощью функции *next()*, после отработки узла оператор возвращает кортеж родительской вершине. Недостатки этой модели проявляются в том, что *next()* реализован через неявный вызов функции, что, как правило, вызывает ошибочное прогнозирование перехода (*branch misprediction*). Кроме того, *next()* для последовательного сканирования требует выгрузки переменных состояния, отвечающих за номер страницы и кортежа из памяти. Это может привести к большим накладным расходам. Схема вызова операторов в *pull*-модели показана на рис. 1.

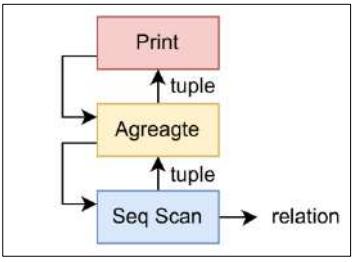


Рис. 1. Volcano-модель для запроса Q06
Fig. 1. Volcano model for Q06 query

Модель явных циклов, которую также называют *push*-моделью, предполагает прямой вызов операторов, а не рекурсивный. Выполнение всего запроса представляет собой проход по вложенным циклам, внешний из которых является циклом сканирования. В динамическом компиляторе запросов эта модель реализована с помощью функций *consume()* и *finalize()* для каждого оператора, где *consume()* вызывается для каждого произведенного кортежа и содержит часть алгоритма оператора, отвечающую за логику обработки получаемого кортежа, а *finalize()* выхывается для последнего кортежа и содержит части алгоритма оператора, отвечающего за постобработку данных. Итоговый код запроса представляется в виде последовательности вложенных циклов, в которых узлы-операторы вызываются явно. Кроме того, вызов узлов-операторов в циклах не требует сохранения и выгрузки контекста. Эти оптимизации позволяют получить ускорение до 5 раз на тестах из набора TPC-H по сравнению с выполнением запроса интерпретатором. Схема вызова узлов-операторов в *push*-модели показана на рис. 2.

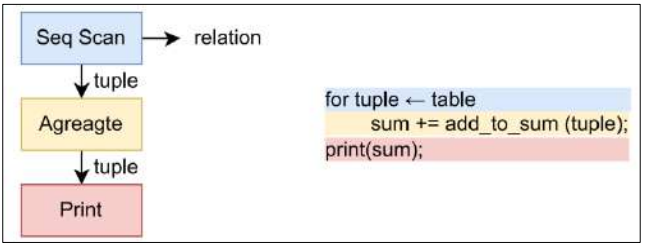


Рис. 2. Push-модель для запроса Q06
Fig. 2. Push-model for Q06 query

2.2 Динамическая компиляция выражения

В PostgreSQL обработка выражений представляет собой интерпретацию дерева выражений. Каждое такое дерево содержит в себе операторы и функции в качестве вершин и константы или переменные в качестве листовых вершин. Интерпретатор PostgreSQL осуществляет вызов функций и операторов неявно, что не даёт возможности для оптимизаций. Динамический компилятор запросов решает эту проблему, обходя это дерево рекурсивно в обратном порядке и генерируя явные вызовы функций в вышестоящих вершинах.

Таким образом, динамический компилятор запросов вносит значительные изменения в работу СУБД PostgreSQL на этапе выполнения плана, поэтому появляется необходимость выбора плана исполнения с учётом всех оптимизаций.

3. Планировщик СУБД PostgreSQL

SQL-запрос может быть выполнен разными способами, основываясь на плане запроса; при этом результат выполнения запроса будет одинаков вне зависимости от выбранного плана. Цель планировщика – построить наиболее эффективный план запроса. Планировщик PostgreSQL работает со структурами данных, называемыми путями. Каждый путь содержит информацию о сложности выполнения узла-оператора, сопоставляя ей стоимость. Собственная стоимость пути определяется суммарной стоимостью всех операций, которые выполняет узел.

При конструировании плана строится дерево путей, где стоимость узла задаётся суммой стоимости дочерних вершин и собственной стоимостью узла. Стоимость всего плана определяется стоимостью корневой вершины. Планировщик выбирает путь с наименьшей стоимостью. Стоимость операций в PostgreSQL задаётся константами, отношение которых даёт оценку отношения времени исполнения операций. Эта оценка не учитывает оптимизации при динамической компиляции, что ведёт к потере производительности. Так, последовательное чтение в случае динамической компиляции может оказаться более эффективным, нежели чтение по индексу, однако планировщик будет отдавать предпочтение второму.

3.1 Команда EXPLAIN ANALYZE

СУБД PostgreSQL предоставляет возможность профилирования плана запроса с помощью команды EXPLAIN ANALYZE [7]. Эта команда позволяет сравнить то, что предсказал планировщик, с реальными данными, полученными во время выполнения запроса. Планы запроса с EXPLAIN ANALYZE на примере запроса из листинга 1 представлены в табл. 3. Значение *cost* – это стоимость вершины, разбитая на стоимость получения первого кортежа и общую стоимость. Поле *actual time* хранит время выполнения вершины, оно разбито на два значения: время получения первого кортежа и общее время работы узла. EXPLAIN ANALYZE также выводит информацию о количестве кортежей, произведённых узлом и информацию, специфичную для каждого отдельного узла.

Табл. 3. Вывод команды EXPLAIN ANALYZE для запроса Q06 при использовании последовательного сканирования.
Table 3. Output of EXPLAIN ANALYZE command for Q06 query when using sequential scan.

```
Aggregate (cost=4455709.02..4455709.03 ROWS=1 width=8)
      (actual TIME=12463.297..12463.297 ROWS=1 loops=1)
  -> Seq Scan ON lineitem (cost=0.00..4437900.54 ROWS=3561696 width=16)
    (actual TIME=0.023..12233.865 ROWS=3573063 loops=1)
     FILTER: ((l_shipdate >= '1993-01-01'::DATE)
     AND (l_shipdate < '1994-01-01'::TIMESTAMP WITHOUT TIME zone)
     AND (l_discount >= '0.08'::DOUBLE PRECISION)
     AND (l_discount <= '0.1'::DOUBLE PRECISION)
     AND (l_quantity < '25'::DOUBLE PRECISION))
    ROWS Removed BY FILTER: 176425309
Planning TIME: 0.862 ms
Execution TIME: 12913.875 ms
```

4. Система для расчёта критериев планировщика с учётом динамической компиляции

Настройка планировщика PostgreSQL, учитывающая влияние динамической компиляции, проводится в два этапа: создание критериев и расчёт стоимости путей выполнения на основе этих критериев. Критерии, которые будут созданы на первом этапе, представляют собой набор констант, корректирующих формулы расчёта стоимости путей в планировщике СУБД с учётом динамической компиляции запросов. Константы вычисляются на основе информации о стоимости путей и времени их выполнения для тестовых запросов.

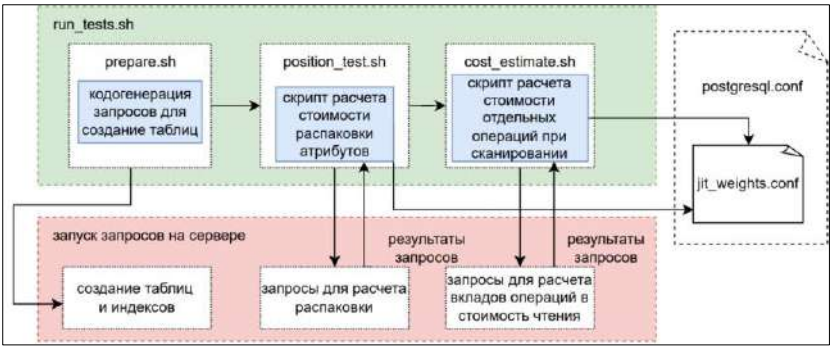


Рис. 3. Схема тестовой системы для расчёта критериев планировщика
Fig. 3. Scheme of testing system for criteria estimation of planner

Первый этап реализован в качестве тестовой системы, работа которой организована скриптом *run_tests.sh*. Изначально выполняется генерация и создание тестовых таблиц. Далее выполняется многократный запуск тестовых запросов с командой EXPLAIN ANALYZE с использованием интерпретатора СУБД PostgreSQL и динамического компилятора запросов. Результаты выполнения запросов используются для расчёта критериев для планировщика, учитывающие динамическую компиляцию запросов, после чего критерии заносятся в файл *jit_weights.conf*. Тестовые запросы сформированы на основе запросов из тестового набора TPC-H. Процесс расчёта критериев рассмотрен в подразделе 4.4. Для использования критериев необходимо записать содержимое *jit_weights.conf* в конец конфигурационного файла *postgresql.conf*. Схема работы тестовой системы представлена на рис. 3.

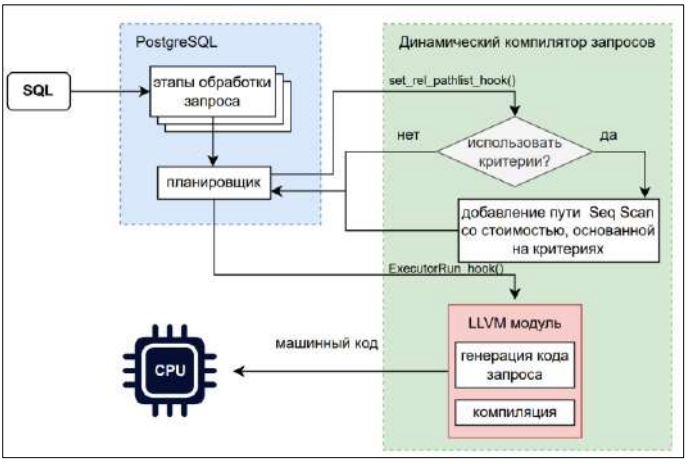


Рис. 4. Учет рассчитанных критериев планировщика при выполнении запроса динамическим компилятором
Fig. 4.Handling of estimated criteria for planner during execution with JIT-compilation

Второй этап выполняется непосредственно во время работы PostgreSQL. При запуске СУБД считывает конфигурационный файл *postgresql.conf* с настройками, инициализируя критерии для планировщика. Далее функция-перехватчик *set_rel_pathlist_hook()* на этапе планирования в зависимости от значения флага *use_jit_costs* добавляет для каждой таблицы, используемой в запросе, путь последовательного сканирования, стоимость которого была вычислена с учётом критериев. Итоговый план строится на основе путей, добавленных планировщиком PostgreSQL и путей, добавленных в функции-перехватчике. Далее происходит кодогенерация и динамическая компиляция запроса. Схема выполнения запроса с учётом рассчитанных критериев представлена на рис. 4.

4.1 Поддержка EXPLAIN ANALYZE в динамическом компиляторе запросов

В PostgreSQL для сбора статистики с помощью команды EXPLAIN ANALYZE [7] используются функции-счетчики: счётчики времени и количества кортежей, производимых узлом-оператором, счётчики количества кортежей, которые были удалены фильтром, и счётчики количества кортежей, видимость которых необходимо проверить в таблице, так как видимость не отображена в таблице индексов. Поддержка этих счётчиков динамическим компилятором запросов необходима для расчёта стоимости путей. Стоимость путей и время выполнения узлов будет использоваться при расчёте критериев для планировщика.

Счётчики времени представляют собой пару функций: *InstrStartNode()* и *InstrStopNode()*, они соответственно начинают и заканчивают отсчёт времени работы узла-оператора, а также подсчитывают число кортежей, им производимых. Эти счётчики должны поддерживаться каждой вершиной в плане выполнения запроса. Для их реализации в динамическом компиляторе запросов над каждым оператором была построена обёртка, как на листинге 2.

```
int consume_wrapper(node)
{
    InstrStartNode (node_instrument);
    ret = Call (node);
    InstrStopNode (node_instrument, 1);
    return ret;
}
```

Листинг 2. Реализация обёртки над оператором в динамическом компиляторе запросов
Listing 2. Node wrapper implementation in query JIT-compil.

В *push*-модели, используемой в динамическом компиляторе, каждому узлу-оператору соответствуют функции *consume* и *finalize*. В модели явных циклов после того, как оператор сформировал кортеж, он вызывает *consume* функцию родительского узла. Во время её исполнения счётчики не должны увеличивать время работы вершины и считать кортежи. Таким образом, вызов *consume* функции родительского узла оборачивается зеркальным кодом, что показано на листинге 3.

```
int ExecNode()
{
    /*...code...*/
    InstrStopNode(node_instrument, 1);
    parent_consume();
    InstrStartNode(node_instrument);
    /*...code...*/
}
```

Листинг 3. Реализация обертки над родительской consume() функцией в динамическом компиляторе запросов
Listing 3. Parent's consume() function wrapper implementation

Рассмотрим реализацию счётчиков на примере оператора Seq Scan, сгенерированный код которого во внутреннем представлении LLVM IR представлен на листинге 3. В

интерпретаторе СУБД PostgreSQL каждый узел оформлен в отдельную функцию, которая вызывается в обёртке *ExecProcNode()*, выставляющей счетчики *InstrStartNode()* до вызова узла и *InstrStopNode()* сразу по завершении его работы. Интерпретатор не содержит в себе обёрток наподобие той, что представлена в листинге 3, так как вышестоящие узлы не вызываются нижестоящими, лишь получая от них кортежи.

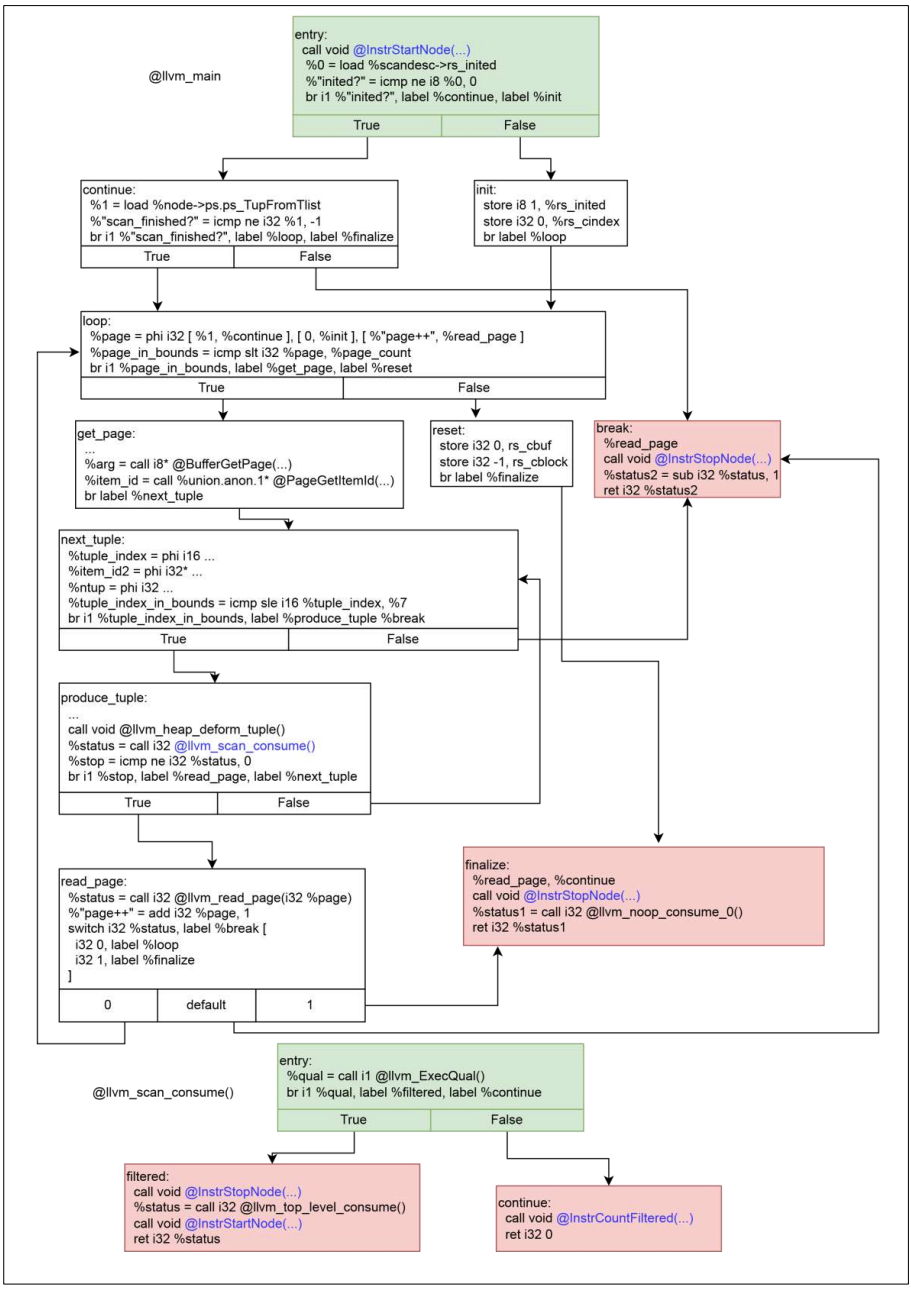


Рис. 5. Реализация оператора Seq Scan на LLVM IR
Fig. 5. LLVM Seq Scan implementation in JIT-compiler

На примере Seq Scan также рассмотрим добавление поддержки счётчика кортежей, удалённых фильтром. Поддержка этого счётчика осуществляется с помощью функци*и*

InstrCountFiltered(). В EXPLAIN ANALYZE от него зависит поле *Rows removed by Filter* некоторых путей. Счётчик добавляется в обработку выражения. Реализация поддержки счётчиков в динамическом компиляторе запросов представлена на рис. 5.

Таким образом была реализована поддержка счётчиков в динамическом компиляторе запросов для основных узлов-операторов СУБД PostgreSQL.

4.2 Влияние распаковки атрибутов на скорость работы узла Seq Scan

Планировщик СУБД PostgreSQL не учитывает в стоимости вершин распаковку атрибутов. Доступ к элементу кортежа происходит в цикле, который суммирует смещения каждого предыдущего элемента относительно друг друга, чтобы получить итоговое смещение относительно начала кортежа. Для таблиц с большим числом столбцов время распаковки элементов оказывается значительным по отношению ко времени последовательного сканирования. Интерпретатор PostgreSQL совершает эти действия в функции *slot_deform_tuple()*. Подробно алгоритм распаковки атрибутов рассмотрен в статье [11]. Пример распаковки атрибутов в СУБД PostgreSQL представлен на рис. 6. В примере рассмотрена распаковка 3 и 8 атрибутов. Распаковка происходит в два этапа: с 1 по 3 атрибуты при первом вызове и с 4 по 8 атрибуты при втором вызове. В динамическом компиляторе запросов распаковка аргументов выполняется функцией *heap_deform_tuple()*, которая распаковывает все атрибуты вплоть до последнего необходимого за один вызов.

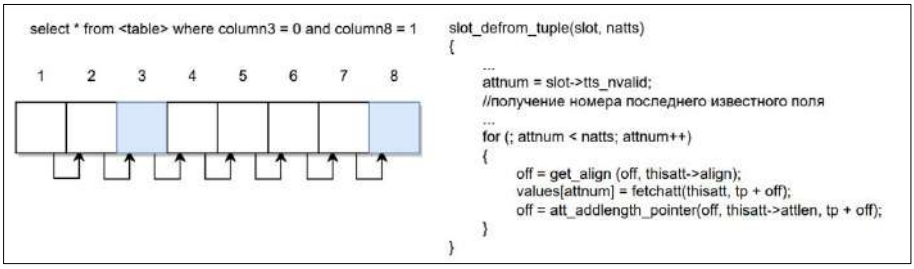


Рис. 6. Процесс распаковки атрибутов выражения в PostgreSQL
Fig. 6. Tuple deforming in PostgreSQL

При достаточно большом количестве столбцов может наблюдаться снижение эффективности последовательного сканирования в сравнении со сканированием индекса. Последнее происходит по специальной структуре, не требуя накладных расходов на распаковку.

Для демонстрации этого эффекта используется таблица с атрибутами типа *integer*. Каждая колонка этой таблицы содержит идентичные данные, чтобы обеспечить одинаковость времени выполнения при индексном сканировании. Пример построения таблицы и запросов, используемых для анализа, представлен в табл. 4. Скорость работы узлов Bitmap Index Scan и Index Scan зависит от *кардинальности запроса*, то есть количества кортежей, произведённых вершиной в ходе выполнения запроса. При увеличении кардинальности узел Bitmap Index Scan более эффективен, чем Index Scan, а Seq Scan более эффективен, чем Bitmap Index Scan. *cardinality_coef* – коэффициент, регулирующий кардинальность запроса.

Табл. 4. Запрос для анализа распаковки атрибута
Table 4. Query that is used for tuple deforming analysis

Таблица	Запрос
`CREATE TABLE deformingTest ( column_0 INT, column_1 INT, ... column_n INT );`	`EXPLAIN ANALYZE SELECT * FROM deformingTest WHERE column_[index] < [cardinality_coef];`

```
CREATE TABLE tmp
(
  index INT,
  rand DOUBLE PRECISION
);

INSERT INTO tmp (SELECT
code,
  1000 * random() FROM

generate_series(1,1000000)
  AS code);

INSERT INTO deformingTest(
  SELECT rand,
  ...
  rand FROM tmp);
```

Тестирование проводилось для 40 атрибутов. Для данной конфигурации таблицы и равномерно распределенных данных в столбцах 0, 19 и 39 время выполнения узлов Seq Scan и Bitmap Index Scan представлено в таблице 5. Время рассчитывалось как медиана времён при многократном запуске.

Табл. 5. Результаты тестирования распаковки
Table 5. Tuple deforming test results

Колонка №	Время Seq Scan, сек.	Время Bitmap Index Scan, сек.
0	15,78	15,92
19	17,75	16,03
39	20,15	16,19

Так как план не учитывает стоимость распаковки атрибутов, планировщик отдаёт предпочтение последовательному сканированию, которое не является эффективным во втором и третьем случаях.

В динамическом компиляторе запросов реализован учёт накладных расходов на распаковку атрибутов для увеличения точности оценки стоимости.

4.3 Влияние изменения модели выполнения на скорость работы узла Seq Scan

Динамический компилятор запросов, описанный в статьях [1-3], позволяет ускорить время выполнения запросов, содержащих узел Seq Scan, за счёт изменения модели выполнения с *Volcano* на модель явных циклов, что приводит к необходимости пересчёта стоимости этого пути, увеличивая его релевантность. Для этой цели формируются критерии.

В статье [12] подробно рассматривается оценка времени выполнения запроса на основе стоимости его выполнения, а также приводятся формулы, по которым происходит расчёт стоимости узлов-операторов. Так, стоимость вершины Seq Scan определяется планировщиком PostgreSQL следующей формулой:

$$total\_cost = startup\_cost + cpu\_run\_cost + disk\_run\_cost,$$

где:

$$cpu\_run\_cost = (cpu\_tuple\_cost + qpqual\_cost) \cdot rows.$$

Здесь:

$startup\_cost$ – стоимость получения первого кортежа, 0 для Seq Scan,
cpu_run_cost – стоимость обработки всех кортежей,
$disk\_run\_cost$ – стоимость чтения страниц памяти,

cpu_tuple_cost – стоимость обработки одного кортежа,
$qpqual\_cost$ – стоимость обработки выражения фильтра,
$rows$ – количество возвращённых кортежей.

Чтобы планировщик мог учитывать наличие динамической компиляции, введём корректирующий коэффициент ss_guc, обозначающий ускорение Seq Scan при выполнении запроса динамическим компилятором. В формуле его нужно учитывать именно перед ускорением обработки одного конкретного кортежа, так как это ускорение связано с использованием другой модели обработки запроса.

$$total\_cost = (\boldsymbol{ss\_guc} \cdot cpu\_tuple\_cost + qpqual\_cost) \cdot rows + disk\_run\_cost.$$

Необходимо учесть ускорение обработки выражения, для этого введём корректировочный коэффициент $qual\_guc$:

$$total\_cost = (ss\_guc \cdot cpu\_tuple\_cost + \boldsymbol{qual\_guc} \cdot qpqual\_cost) \cdot rows + disk\_run\_cost.$$

Ускорение чтения с диска никак не изменяется при динамической компиляции, поэтому нет необходимости вводить отдельные критерии. Как упоминалось в разд. 3, отношение констант, описывающих стоимость операций последовательного чтения, отличается для каждой конкретной машины, поэтому введём коэффициенты, корректирующие отношение этих констант. Сумма введённых коэффициентов равна 3, что соответствует 100%.

$$total\_cost = (\boldsymbol{cpu\_impact} \cdot ss\_guc \cdot cpu\_tuple\_cost + \boldsymbol{qual\_impact} \cdot qual\_guc \cdot qpqual\_cost) \cdot rows + \boldsymbol{disk\_impact} \cdot disk\_run\_cost.$$

В предыдущем разделе мы также показали, что необходимо учитывать стоимость распаковки атрибута. Таким образом, итоговая формула расчета стоимости узла Seq Scan представима как:

$$total\_cost = (cpu\_impact \cdot ss\_guc \cdot cpu\_tuple\_cost + qual\_impact \cdot qual\_guc \cdot (qpqual\_cost + \boldsymbol{deform\_cost})) \cdot rows + disk\_impact \cdot disk\_run\_cost \quad (1)$$

4.4 Критерии для планировщика PostgreSQL с динамическим компилятором запросов

Модификации стоимостной системы планировщика для исполнения запросов с учётом динамической компиляции, рассмотренные в предыдущих разделах, были реализованы с помощью учёта критериев на этапе планирования. Расчёт критериев производится на основе профиля исполнения запросов, сформированных на базе набора TPC-H. При расчёте стоимости вершин, критерии принимают значения, которые не изменяют стоимости вершины, по отношению к стоимости, рассчитанной оригинальным планировщиком, их значения представлены в листинге 4. Запросы выполняются повторно, чтобы обеспечить попадание данных в кеш. Результаты расчёта критериев заносятся в конфигурационный файл *jit_weights.conf*.

```
SET llvm_jit.jit_ss_cpu_tuple TO 1;
SET llvm_jit. ss_qual_cost TO 1;

SET llvm_jit.jit_qual_impact TO 1;
SET llvm_jit.jit_cpu_impact_nq TO 1;
SET llvm_jit.jit_cpu_impact_q TO 1;
SET llvm_jit.jit_disk_impact_nq TO 1;
SET llvm_jit.jit_disk_impact_q TO 1;
```

Листинг 4. Стартовые значения критериев при тестировании.
Listing 4. Starting values of criteria during testing.

Система рассматривает два типа запросов: содержащие выражения и не содержащие их. Во втором случае константа *qual_impact* считается равной единице. *Impact*-критерии вычисляются как корни системы уравнений на основе выражения 1:

$$\begin{cases} cpu\_impact\_nq \cdot ss\_guc \cdot cpu\_cost + disk\_impact\_nq \cdot disk\_cost = total\_cost \\ cpu\_impact\_nq + disk\_impact\_nq = 2 \end{cases}, \quad (2)$$

$$\begin{cases} cpu\_impact\_q \cdot ss\_guc \cdot cpu\_cost + qual\_impact \cdot qual\_guc \cdot qual\_cost \\ \quad + disk\_impact\_q \cdot disk\_cost = total\_cost \\ cpu\_impact\_q + disk\_impact\_q + qual\_impact = 3 \\ disk\_cpu\_prop = \dfrac{disk\_impact\_nq}{cpu\_impact\_nq} = \dfrac{disk\_impact\_q}{cpu\_impact\_q} \end{cases}, \quad (3)$$

где:

$$cpu\_cost = cpu\_tuple\_cost \cdot rows$$
$$disk\_cost = disk\_run\_cost$$
$$qual\_cost = (qpqual\_cost + deform\_cost) \cdot rows$$

В этих системах постфикс q обозначает критерии, рассчитанные для запросов с выражениями, nq – без выражений, $total\_cost$ – стоимость индексного сканирования, именно такую стоимость должен иметь узел Seq Scan, чтобы быть выбранным планировщиком. Третье уравнение системы 3 следует из того факта, что отношение скорости последовательного сканирования таблицы и скорости чтения данных с носителя остаётся постоянным на одной аппаратуре. Другие критерии и необходимые коэффициенты для системы уравнений рассчитываются на основе профиля выполнения тестовых запросов и результатов работы команды EXPLAIN ANALYZE. Примеры запросов, которые были использованы для расчета критериев, представлены в листинге 5.

```
EXPLAIN (analyze, format json) SELECT COUNT(*) FROM lineitem;
EXPLAIN (analyze, format json) SELECT COUNT(*)
    FROM lineitem WHERE l_shipdate >= DATE '1993-01-01';
```

Листинг 5. Примеры тестовых запросов для расчёта критериев
Listing 5. Test queries for calculating criteria

Критерий ss_guc вычисляется как отношение времени выполнения запроса без каких-либо выражений с использованием динамической компиляции и без неё:

$$ss\_guc = \frac{t_{\text{JIT}}}{t_{\text{INT}}}.$$

Критерий $qual\_guc$ вычисляется как отношение времени обработки выражения при динамической компиляции и без неё. Время обработки выражения вычисляется вычитанием времени выполнения запроса без выражений из времени выполнения запроса, содержащего выражения.

$$qual\_guc = \frac{t\_qual_{\text{JIT}}}{t\_qual_{\text{INT}}}.$$

Вычисление составляющих $cpu\_cost, disk\_cost$ и $qual\_cost$ стоимости узла Seq Scan производится на основе результата работы команды EXPLAIN ANALYZE. Стоимость каждой составляющей можно получить, выставив значение *impact*-переменной, являющиеся множителем перед составляющей, интересующей нас, в единицу и занулив все остальные *impact*-переменные.

Составляющая $qual\_cost$ уточняется стоимостью распаковки атрибутов. Для расчёта критерия $deform\_cost$ используется табл. 4. Стоимость распаковки определяется отношением времени распаковки одной колонки ко времени выполнения вершины помноженным на стоимость выполнения вершины:

$$t_e = \frac{t_n - t_1}{n-1}, deform\_cost = \frac{t_e \cdot total\_cost}{t_1},$$

где

t_e – время распаковки одного столбца,

t_1 – время выполнения запроса, обращающегося к 1 колонке,

t_n – время выполнения запроса, обращающегося к колонке номер n,

$total\_cost$ – стоимость выполнения запроса, обращающегося к 1 колонке.

Необходимая информация вычисляется на основе профилирования запросов из листинга 6.

```
EXPLAIN (analyze, format json)
SELECT COUNT(*) FROM deformingTest WHERE column_0 == 0;

EXPLAIN (analyze, format json)
SELECT COUNT(*) FROM deformingTest WHERE column_n == 0;
```

Листинг 6. Тестовые запросы для расчета критерия распаковки атрибутов
Listing 6. Test queries to estimate deforming of attributes

Таким образом, вычисляются все критерии, которые будут использованы планировщиком при расчёте плана для выполнения с динамической компиляцией.

4.5 Модификация динамического компилятора запросов

Для построения плана планировщик СУБД PostgreSQL составляет список отношений (таблиц), участвующих в запросе. Для каждого отношения планировщик строит список путей — возможных узлов-операторов для выполнения плана. Пересчёт стоимости для выполнения с учётом динамической компиляции происходит за счёт добавления нового пути Seq Scan, стоимость которого рассчитана с учётом критериев (см. подразделы 4.3 и 4.4), в каждое отношение до момента выбора наилучшего пути. Введённые критерии считываются из конфигурационного файла во время старта сервера СУБД, и изменяют соответствующие константы. Расчёт стоимости пути с учётом введённых критериев производится по формуле 1. Добавление пути проходит в функции-перехватчике *set_rel_pathlist_hook()* (рис. 4). Последовательное сканирование является базовым методом доступа к данным, соответственно нет необходимости выполнять какие-либо проверки, для того, чтобы добавить этот путь. Далее планировщик выбирает наиболее эффективный путь выполнения запроса, основываясь на стоимости.

5. AQO как альтернативные критерии для планировщика СУБД PostgreSQL

При работе с динамическим компилятором запросов было интересно узнать, как на оптимизацию плана может повлиять расширение AQO. Adaptive Query Optimization [5] — расширение для СУБД PostgreSQL, которое методами машинного обучения улучшает предсказание кардинальности. Эта информация увеличивает точность, с которой происходит расчёт стоимости вершин сканирования индекса. Для сбора статистики AQO использует информацию, полученную с помощью команды EXPLAIN ANALYZE. Добавление поддержки EXPLAIN ANALYZE в динамическом компиляторе запросов позволило использовать это расширение совместно с динамическим компилятором.

На первом этапе AQO разделяет запросы на группы, основываясь на их структуре. Запросы попадают в одну группу, если тексты запросов отличаются только константами.

AQO собирает информацию о реальной кардинальности запросов после их выполнения для каждого типа запросов при значении поля *aqo.mode 'learn'* или *'intelligent'*. Запуск запросов при значении поля *aqo.mode = 'intelligent'* или *'controlled'* вычисляет для данного типа запроса и констант предположительную кардинальность вершин, основываясь на методе *Gradient K Nearest Neighbours*, рассмотренном в статье [10]. Эта информация используется планировщиком, чтобы более точно оценить стоимость JOIN-узлов. Таким образом, AQO применим для редко обновляющихся таблиц.

Выбирая план, AQO использует стоимостную систему, изменяя лишь стоимость узлов, зависящих от селективности запроса. AQO не будет конфликтовать с рассмотренным в этой статье динамическим компилятором запросов, так как Seq Scan не относится к такому типу вершин.

Применение AQO совместно с динамическим компилятором запросов позволило планировщику выбирать более эффективные планы для запросов с JOIN-узлами, позволяя выбрать альтернативный метод объединения из-за выбора узла сканирования, отличного от выбранного планировщиком PostgreSQL.

6. Результаты

Оценка работы планировщика, учитывающего динамическую компиляцию, проводилось на запросах из тестового набора TPC-H для базы данных объемом 30 Гб на сервере с архитектурой ARM. При тестировании запросы выполнялись многократно, бралось медианное значение времени.

В табл. 6 отражены результаты тестирования запросов из TPC-H, план которых был изменён при расчёте стоимости на основе критериев, рассчитанных тестовой системой. Ускорение вычислялось как отношение времени выполнения запросов с помощью динамической компиляции без критериев ко времени выполнения запросов с использованием динамической компиляции с критериями.

Табл. 6. Результаты тестирования динамического компилятора запросов при использовании критериев
Table 6. JIT-compiler test results using criteria

TPC-H 30 ГБ	**без критериев**		**используя критерии**		**Ускорение, раз** $\mathbf{JIT}^{\frac{\text{без критериев}}{\text{с критериями}}}$	**Ускорение, раз** $\frac{\boldsymbol{PG}\text{ без критериев}}{\boldsymbol{JIT}\text{ с критериями}}$
	PG, сек.	**JIT, сек.**	**PG, сек.**	**JIT, сек.**		
Q06	22,52	20,63	32,30	8,76	2,35	2,57
Q12	53,40	48,60	73,42	43,48	1,11	1,22
Q18	95,90	58,78	102,52	59,05	0,99	1,62

Для некоторых запросов время исполнения не изменилось, даже с учётом того, что поменялся план, что связано с близкой стоимостью узлов-операторов. На примере запроса Q06 наблюдается ускорение в 2,5 раза относительно интерпретатора и в 2,35 по сравнению со старым планом, использующим динамическую компиляцию. Для запроса Q12 наблюдается ускорение в 1,11 раза.

В табл. 7 представлены результаты тестирования с использованием AQO для запросов из таблицы выше. Ускорение вычислялось, как отношение времени выполнения запросов с динамической компиляцией без AQO ко времени выполнения запросов с использованием динамической компиляции с AQO.

Табл. 7. Результаты тестирования динамического компилятора запросов с использованием AQO
Table 7. JIT-compiler test results using AQO

TPC-H 30 ГБ	**не используя AQO**		**используя AQO**		**Ускорение, раз** $\mathbf{JIT}^{\frac{\text{без } \boldsymbol{AQO}}{\text{с } \boldsymbol{AQO}}}$	**Ускорение, раз** $\frac{\boldsymbol{PG}\text{ без }\boldsymbol{AQO}}{\boldsymbol{JIT}\text{ с }\boldsymbol{AQO}}$
	PG, сек.	**JIT, сек.**	**PG, сек.**	**JIT, сек.**		
Q06	22,52	20,63	22,52	20,56	1,00	1,09
Q12	53,40	48,60	53,45	47,47	1,02	1,12
Q18	95,90	58,78	65,15	44,49	1,32	2,15

Для запроса Q18 было получено ускорение в 1.32 раз при использовании AQO совместно с динамическим компилятором запросов относительно динамического компилятора без AQO.

Это объясняется тем, что в плане запроса последовательное сканирование таблицы было заменено индексным, что позволило ускорить соединение таблиц. Выбор плана выполнения запроса, зависящий от распаковки атрибутов, рассмотрен на примере выполнения запроса из листинга 7.

```
EXPLAIN (analyze) SELECT COUNT(*) FROM deformingTest WHERE COLUMN[index] < 18;
```

Листинг 7. Тестовый запрос оценки работы планировщика в зависимости от распаковки атрибуто.

Listing 7. Test query that examines planner choice depending on attribute extraction

Для каждого столбца таблицы *deformingTest* был создан индекс, что позволило планировщику выбирать между сканированием по индексу и последовательным сканированием. Тестирование было выполнено при использовании динамической компиляции с учётом критериев. Время выполнения запросов и выбранный планировщиком путь представлены в табл. 8.

Табл. 8. Результаты тестирования времени распаковки динамическим компилятором с использованием критериев

Table 8. Deforming test results using JIT-complier

index	JIT без критериев, сек.	JIT с критериями, сек.
0	10,75 (Bitmap Index Scan)	9,97 (Seq Scan)
39	10,72 (Bitmap Index Scan)	10,74 (Bitmap Index Scan)

Планировщик, не использующий критерии, выбирает путь Bitmap Index Scan для этого запроса вне зависимости от номера атрибута в выражении, что приводит к потере эффективности при обращении к нескольким первым столбцам. Выбор узла Seq Scan приводит к потере эффективности при обращении к атрибутам с номером больше некоторого порогового значения. Таким образом, планировщик, учитывающий сформированные критерии, выбирает оптимальный по времени план, основываясь, в том числе и на стоимости распаковки атрибутов.

7. Заключение

В ходе данной работы были рассмотрены эвристические методы выбора плана для СУБД PostgreSQL, учитывающие исполнение запроса с помощью динамической компиляции. Была добавлена поддержка команды EXPLAIN с параметром ANALYZE для динамического компилятора запросов для СУБД PostgreSQL, использованная для профилирования выполнения запросов. На основе профиля выполнения запроса были сформированы критерии оптимизации выполнения, которые уточняли стоимость узла-оператора Seq Scan при выполнении запросов с динамической компиляцией. В будущем планируется обобщение алгоритмов для расчёта критериев для различных узлов-операторов.

Планировщик PostgreSQL использует стоимостную модель, которая была адаптирована для расчёта эффективных планов для выполнения запросов с динамической компиляцией. Был сформирован набор запросов на основе TPC-H, позволяющих уточнить стоимость узлов-операторов, основываясь на результатах профилирования. При использовании критериев для выполнения планов из набора TPC-H на сервере с архитектурой ARM, выбирались планы более эффективные с точки зрения динамической компиляции, что позволило уменьшить время выполнения вплоть до двух раз.

В ходе работы были также рассмотрены альтернативные критерии для выбора оптимальных планов, например, использование расширения AQO. Разработанный эвристический метод позволяет также использовать модели совместно с ним и иные алгоритмы. Так, например, планировщик PostgreSQL, используя расширение AQO с введёнными критериями, уменьшает время выполнения вплоть до 1,32 раз за счёт изменения плана.

Список литературы / References

[1] Бучацкий Р.А., Шарыгин Е.Ю. и др. Динамическая компиляция SQL-запросов для СУБД PostgreSQL. Труды ИСП РАН, том 28, вып. 6, 2016, стр. 37-48 / Buchatskiy R.A., Sharygin E.Y. et al. Dynamic compilation of SQL queries for PostgreSQL. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 6, 2016, pp. 37-48 (in Russian). DOI: 10.15514/ISPRAS-2016-28(6)-3.

[2] Шарыгин Е.Ю., Бучацкий Р.А. и др. Динамическая компиляция выражений в SQL-запросах для СУБД PostgreSQL. Труды ИСП РАН, том 28, вып. 4, 2016 г., стр. 217-240 / Sharygin E.Y., Buchatskiy R.A. et al. Dynamic compilation of expressions in SQL queries for PostgreSQL. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 4, 2016. pp. 217-240 (in Russian). DOI: 10.15514/ISPRAS-2016-28(4)- 13.

[3] Sharygin E., Buchatskiy R., et al. Runtime specialization of PostgreSQL query executor. Lecture Notes in Computer Science, vol. 10742, 2018, pp. 375-386.

[4] The LLVM Compiler Infrastructure. Available at: https://llvm.org/docs/, accessed 13.07.2022

[5] Adaptive Query Optimization. Available at: https://github.com/postgrespro/aqo, accessed 13.07.2022

[6] PostgreSQL 9.6.24 Documentation. Available at: https://postgrespro.com/docs/postgresql/9.6/index, accessed 13.07.2022

[7] Using Explain Analyze in PostgreSQL. Available at: https://postgrespro.ru/docs/enterprise/9.6/using-explain\#using-explain-analyze, accessed 13.07.2022.

[8] Graefe G. Volcano – an extensible and parallel query evaluation system. IEEE Transactions on Knowledge and Data Engineering, vol. 6, issue 1, 1994, pp. 120-135.

[9] TPC-H, an ad-hoc, decision support benchmark. Available at: http://www.tpc.org/tpch/, accessed 13.07.2022.

[10] Ivanov O., Bartunov S. Adaptive Cardinality Estimation. arXiv:1711.08330, 2017, 12 p.

[11] Zhang R., Debray S., Snodgrass R.T. Micro-Specialization: Dynamic Code Specialization of Database Management Systems. In Proc. of the Tenth International Symposium on Code Generation and Optimization, 2012, pp. 63-73.

[12] Wu W., Chi Y. et al. Predicting query execution time: Are optimizer cost models really unusable? In Proc. of the IEEE 29th International Conference on Data Engineering (ICDE), 2013, pp. 1081-1092.

Информация об авторах / Information about authors

Егор Викторович ДОЛГОДВОРОВ – студент МФТИ, лаборант отдела компиляторных технологий ИСП РАН. Научные интересы: компиляторные технологии, оптимизации.

Egor Viktorovich DOLGODVOROV – A Student at MIPT, Laboratory Assistant in Compiler Technology department at ISP RAS. Research interests: compiler technologies, optimizations.

Рубен Артурович БУЧАЦКИЙ –научный сотрудник отдела компиляторных технологий. Научные интересы: компиляторные технологии, оптимизации.

Ruben Arturovich BUCHATSKIY – PhD, Researcher in Compiler Technology department. Research interests: compiler technologies, optimizations.

Михаил Вячеславович ПАНТИЛИМОНОВ – стажер-исследователь отдела компиляторных технологий. Научные интересы: компиляторные технологии, СУБД.

Michael Vyacheslavovich PANTILIMONOV – Researcher in Compiler Technology department. Research interests: compiler technologies, DBMS.

Дмитрий Михайлович МЕЛЬНИК – старший научный сотрудник отдела компиляторных технологий. Научные интересы: компиляторные технологии, оптимизации.

Dmitry Mikhailovich MELNIK – Senior Researcher in Compiler Technology department. Research interests: compiler technologies, optimizations.

DOI: 10.15514/ISPRAS-2022-34(4)-9

Реализация функции долговременного хранения научных данных большого объема в вычислительном центре

Д.В. Иванков, ORCID: 0000-0003-4254-0104 <d.v.ivankov@yandex.ru>

Всероссийский НИИ технической физики имени академика Е.И. Забабахина, 456770, Россия, г. Снежинск, Челябинская область, ул. Васильева, 13

Аннотация. Длительное и целостное хранение объемных научных данных является одной из важных задач, стоящих перед многими вычислительными центрами. В целях снижения стоимости хранения информации, в некоторых решениях используется технология магнитно-ленточной памяти, а также специализированное программное обеспечение для управления носителями и данными. Ввиду инфраструктурной специфики и особенностей сложившихся техпроцессов генерации и обработки данных в научных лабораториях такие программно-аппаратные комплексы создаются и поддерживаются преимущественно собственными силами этих организаций. Разработка такой системы становится еще более востребованной в условиях стремления к обретению технологического суверенитета. В работе рассматриваются вопросы организации долговременного хранения цифровых научных данных в вычислительном центре ФГУП РФЯЦ-ВНИИТФ, полученных в ходе проведения расчетов задач математического моделирования. Приводится описание архитектуры и функционального состава разработанной архивной системы хранения данных. Описывается используемая модель данных и форматы группировки и записи. Рассматриваются предпринятые меры по обеспечению целостности архивных объектов, методы управления архивными носителями и вопросы технической организации архивного фонда. Приводится схема расчета аппаратной конфигурации типовой площадки архивной системы хранения данных, достаточной для обслуживания существующих потоков архивирования данных в вычислительном центре.

Ключевые слова: архивная система хранения; долговременное хранение; магнитные ленты; ленточные библиотеки; целостность информации

Для цитирования: Иванков Д.В. Реализация функции долговременного хранения научных данных большого объема в вычислительном центре. Труды ИСП РАН, том 34, вып. 4, 2022 г., стр. 117-134. DOI: 10.15514/ISPRAS-2022-34(4)-9

Large-scale scientific data and long-term data storage function in a computing center

D.V. Ivankov, ORCID: 0000-0003-4254-0104 <d.v.ivankov@yandex.ru>

E.I. Zababakhin All-Russian Scientific Research Institute of Technical Physics, 13, Vasilieva street, Chelyabinsk region, Snezhinsk, 456770, Russia

Abstract. Long-term data storing is an important task for many modern scientific laboratories and datacenters. In order to reduce cost of digital information ownership, some solutions use magnetic tape technology and special software to control medium and data. Considering the on-site infrastructure specifics and well-established workflows of data processing, these organizations build and support such systems mainly by their own efforts, what becomes an important task in seeking to acquire the technological sovereignty. This paper describes long-term data storage issues in the computing center of the Zababakhin All-Russia Research Institute of Technical Physics where mathematical modeling computations generate vast amount of scientific data. The

architecture and functional composition of the developed Archive Data Storage System are given as well as its internal data model, the chunk grouping rules, and the low-level tape format used. The measures taken to ensure an archived data consistency, methods of storage media management and issues of archival fund maintenance, are also considered. The calculation scheme of a typical archive system site's hardware configuration, sufficient to process archiving data flows existing in datacenter, is given.

Keywords: digital archive; long-term data storage; magnetic tapes; tape libraries; data consistency

For citation: Ivankov D.V. Large-scale scientific data and long-term data storage function in a computing center. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 4, 2022. pp. 117-134 (in Russian). DOI: 10.15514/ISPRAS-2022-34(4)-9

1. Введение

При проведении расчетов задач математического моделирования на высокопроизводительных вычислительных системах (далее по тексту ВВС) вычислительные процессы задействуют функции ввода-вывода главным образом для сохранения и считывания состояния своей оперативной памяти. Со стороны системы хранения (далее по тексту СХ) ВВС эта роль возложена на оперативные файловые системы (далее по тексту ФС), обладающие (по возможности) максимальной производительностью ввода-вывода и, вследствие их высокой стоимости, ограниченными объемами ресурсов хранения. Для непрерывного потока расчетов на ВВС необходимо поддержание достаточного объема свободных ресурсов оперативных ФС, что обычно достигается путем своевременного удаления «ненужных» данных и регулярного вытеснения устаревших, но не потерявших актуальности данных на специально выделенный сегмент СХ, предназначенный для долговременного хранения информации.

Многолетняя эксплуатация ВВС в ФГУП РФЯЦ-ВНИИТФ им. академ. Е.И. Забабахина (далее по тексту РФЯЦ-ВНИИТФ) показала важность управления расчетными данными в течение их жизненного цикла и необходимость организации системы долговременного хранения данных, способствующую объединению и эффективной эксплуатации информационно-вычислительных ресурсов института. Для решения этой задачи в РФЯЦ-ВНИИТФ была создана «архивная система хранения данных» (далее по тексту АСХД), опирающаяся на технологию магнитно-ленточной памяти. Целями ее создания были обозначены: создание единого ресурса долговременного хранения расчетных данных, сгенерированных в различных ВВС; обеспечение целостности и сохранности информации в течение длительного времени; а также снижение зависимости от внешних факторов и сторонних компонент, способных повлиять на возможность восстановления информации. Для управления этой системой было разработано одноименное программное обеспечение (далее по тексту ПО), которое затем вошло в состав операционной системы «СПО Супер-ЭВМ» [1].

Среди наиболее близких к АСХД по своему назначению программных систем следует выделить коммерческий продукт IBM HPSS [2], ставший стандартом де-факто во многих научных лабораториях и университетах благодаря своей зрелости и глобальной поддержке производителя. Некоторые центры обработки данных разработали и эксплуатируют собственное ПО для управления системами архивного хранения данных, например, Enstore [3] в Fermilab (США) и CTA [4] в европейском CERN.

2. АСХД

АСХД представляет собой аппаратно-программный комплекс, выполняющий загрузку, хранение, поиск и восстановление цифровой пользовательской информации. Высокая степень сохранности обрабатываемых данных обеспечивается благодаря широкому применению средств контроля целостности, встроенной репликации, использованию технологии магнитных лент, изоляции заполненных архивных носителей и самих архивных

объектов. Способность хранения значительного объема информации в течение длительного времени обеспечивается как за счет применения автоматизированных ленточных библиотек различных технологических поколений, так и благодаря разработанному регламенту обмена архивными носителями между библиотеками и стеллажным хранилищем.

В качестве источников информации для АСХД выступают стандартные файловые ресурсы хранения данных.

Программная часть системы реализована с применением клиент-серверного подхода и реляционной системы управления базами данных. АСХД обладает пользовательским и административным web-интерфейсом и интерфейсом командной строки.

3. Структура АСХД

Структура АСХД включает в себя следующий набор взаимодействующих компонентов (см. рис. 1):

- метасервер;
- сервер управления базой данных (СУБД);
- медиасервер;
- транспортный агент;
- консольный клиент (CLI);
- web-клиент;
- web-сервер;
- консоль администратора;
- автоматизированное рабочее место оператора хранилища (АРМ), оснащенное мобильным терминалом.

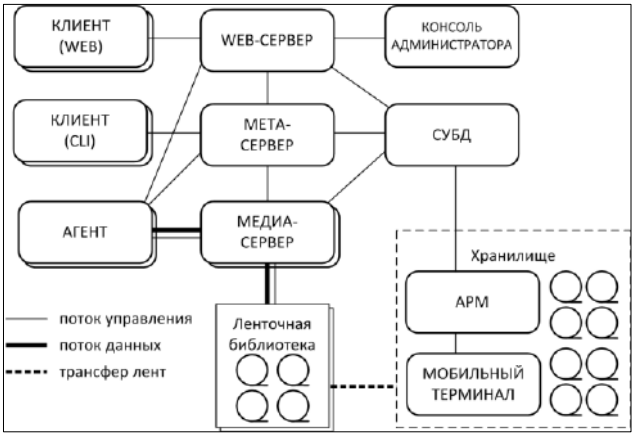


Рис. 1. Структурная схема АСХД
Fig. 1. Structural scheme of the Archive Data Storage System

3.1 Метасервер

Данный компонент является координирующим центром большей части процессов, циркулирующих в АСХД, и выполняет следующие основные функции:

- управление процессами загрузки/выгрузки архивных объектов;
- обработка (хранение/поиск/выдача) метаданных архивных объектов;
- управление очередью заданий;
- авторизация доступа к архивным объектам.

Программная реализация метасервера выполнена в виде многопоточной службы, взаимодействующей с остальными компонентами системы, а также с СУБД, обеспечивающей хранение всей метаинформации.

3.2 Медиасервер

Медиасервер является основным исполнительным компонентом АСХД, функцией которого является запись/чтение архивных объектов на архивные носители.

Программная реализация выполнена в виде многопоточной службы, в составе которой функционируют несколько компонентов:

- генератор планов записи;
- «финишер»;
- менеджер томов;
- главная коммуникационная нить;
- встроенный web-сервер;
- менеджеры ленточных накопителей.

В АСХД могут одновременно работать несколько медиасерверов, к каждому из которых подключаются одна или несколько ленточных библиотек. Медиасерверы конфигурируются таким образом, чтобы обеспечить параллельную работу множества ленточных накопителей. Каждый медиасервер эксклюзивно управляет накопителями и роботом ленточной библиотеки, поэтому текущая версия АСХД в большей степени ориентирована на использование библиотек midrange-класса. Медиасервер АСХД взаимодействует со SCSI устройствами (накопителями и библиотечными роботами), используя стандартные для Unix-подобных операционных систем средства [5, 6].

В связи с большой продолжительностью периодов записи/чтения ленточных накопителей взаимодействие метасервера с медиасервером реализовано в асинхронном режиме. Медиасервер функционирует в виде многопоточной службы, получающей задания от метасервера и обменивающейся данными с агентами.

Помимо ресурсов ленточной памяти каждый медиасервер оснащается дисковой памятью большого объёма, роль которой заключается в промежуточном хранении архивных объектов и обеспечении стабильности входного потока записываемой на ленточные накопители информации. Медиасервер хранит в этом дисковом кэше каждый полученный от агента архивный объект до тех пор, пока тот не будет полностью записан на ленточные носители. Аналогично, при восстановлении архивного объекта медиасервер сохраняет считываемый с ленточных носителей архивный объект в дисковом кэше и после успешного завершения этой операции отправляет восстановленный объект агенту.

3.3 Агент

Роль агента заключается в транспортировании пользовательских данных между ресурсами-источниками и АСХД, а также в поставке метаинформации web-клиентам. Разработанный для операционной системы Linux агент функционирует в виде системной службы, которая запущена на серверах, имеющих непосредственный доступ к ресурсам-источникам.

Являясь транспортным компонентом, агент обменивается архивируемыми (и восстанавливаемыми из архива) данными с медиасерверами под управлением метасервера.

3.4 Клиенты

К функциям клиента относятся:

- управление пользовательскими заявками (постановка в очередь, контроль, отмена) на архивирование/восстановление данных;
- назначение описательных атрибутов создаваемого архивного объекта;

- задание критериев поиска архивных объектов;
- удаление архивных объектов.

В настоящее время разработаны два вида клиента: консольный (для операционной системы Linux) и межплатформенный web-клиент.

На web-сервер АСХД возложена задача обслуживания как пользовательских запросов от web-клиента, так и выполнения административных запросов, поступающих от web-консоли администратора.

АСХД обладает собственным программным интерфейсом (REST API), позволяющим добавлять функции архивирования и восстановления данных в прикладные приложения.

3.5 АРМ оператора хранилища

К функциям АРМа оператора хранилища относятся:

- обслуживание заявок АСХД по набору и замене архивных носителей (картриджей) в библиотеках;
- инвентаризация хранилища ленточных картриджей;
- регистрация и последующее сопровождение ленточных картриджей.

АРМ оператора взаимодействует исключительно с базой данных метаинформации АСХД.

4. Модель данных

Типовая площадка АСХД может удовлетворять потребности пользователей нескольких ВВС, ресурсы хранения которых распределены между пользовательскими группами, сформированными по проектному принципу. Все существующие файловые ресурсы хранения, как и все проекты и пользователи, должны быть предварительно зарегистрированы в АСХД.

Операция архивирования данных инициируется постановкой соответствующего задания (заявки) в очередь АСХД. В результате успешной обработки задания метасервером формируется один или несколько архивных объектов, в которых инкапсулированы пользовательские данные, указанные в задании. В базу данных АСХД помещается метаинформация о каждом архивном объекте, которая включает в себя: числовой идентификатор; размер в байтах; информацию о владельце и времени создания; список доступа; текстовое описание и специфичный для организации, эксплуатирующей АСХД, набор определяемых пользователем тематических атрибутов, характеризующих содержимое архивного объекта.

Максимальный размер архивного объекта (параметр MAX_OBJECT_SIZE) ограничен объемом дискового кэша медиасервера, минимальный размер – 1КБ. В ходе выполнения архивирования объекты могут быть разрезаны на сегменты, назовем их чанками (от англ. chunk).

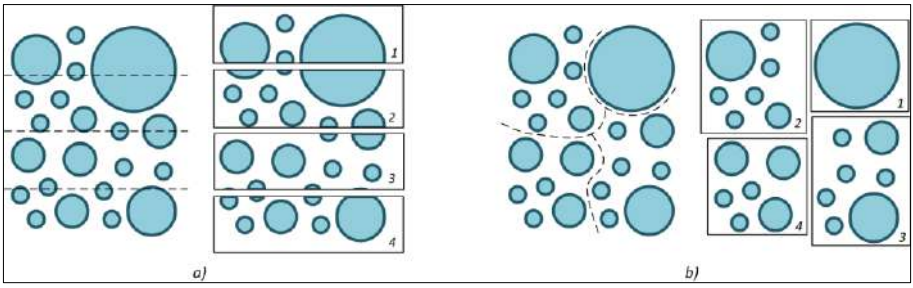


Рис. 2. Формирование чанков – (a) «распил» по размеру, (b) «распил» по границам
Fig. 2. Chunk generation – (a) splitting by size, (b) splitting along borders

Параметр CHUNK_SIZE определяет максимальный размер чанка. Если суммарный размер указанных в пользовательском задании данных больше CHUNK_SIZE, то соответствующий архивный объект разрезается на множество чанков одним из двух способов (см. рис. 2), иначе он состоит из одного чанка.

Каждый архивный объект обладает уникальным целочисленным идентификатором. Чанки архивного объекта также нумеруются по порядку. Под номером 0 регистрируется специальный чанк-дескриптор, хранящий подробную пользовательскую, взятую из заявки, и системную метаинформацию, описывающую архивный объект, а также контрольные суммы всех чанков данного архивного объекта. Чанки некоторых архивных объектов могут реплицироваться (см. п.5.3). Поэтому в ходе планирования операций записи/чтения медиасервер оперирует универсальными именами чанков, построенными по схеме:

<№ объекта> . <№ чанка> . <№ реплики>.

Например, в состав реплицируемого объекта 122 входят чанки «122.1.0», «122.2.0», «122.3.0», «122.4.0», «122.1.1», «122.2.1», «122.3.1», «122.4.1», а также два дескриптора «122.0.0» и «122.0.1» (см. рис. 3).

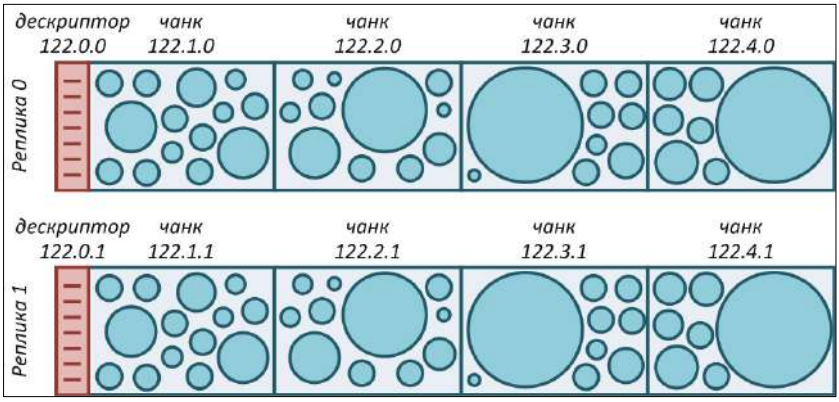


Рис. 3. Схема именования чанков
Fig. 3. Chunk naming scheme

Характерной проблемой систем хранения, использующих технологию магнитных лент, является достижение максимальной производительности ленточных накопителей, чаще всего связанное с недостаточностью входного потока информации. В этих условиях двигатель ленточного накопителя часто останавливается. Для продолжения записи он сначала перематывает ленту немного назад и затем, читая на низкой скорости записанные ранее данные, пытается найти метку конца предыдущей записи, чтобы продолжить запись вновь поступивших данных, постепенно увеличивая скорость протяжения ленты и, как следствие, производительность накопителя. К неприятным последствиям недостаточности входного потока также относится повышенный износ как самого двигателя, так и магнитных головок (эффект *shoe-shining*).

Поэтому для сокращения количества старт-стопных циклов накопителей разработчики систем хранения прибегают к обязательному предварительному кэшированию, контейнеризации предназначенных для записи данных и прочим организационно-техническим приемам.

Кэш медиасервера АСХД организован в виде локальной файловой системы достаточного объема, соизмеримого с максимальным объемом используемых типов архивных носителей. Для повышения согласованности реплик архивных объектов и для экономии потребляемых дисковых ресурсов репликация чанков реализована на этой файловой системе в виде *hard link*'ов.

Контейнеризация осуществляется путем группирования записываемых чанков в файле формата POSIX tar[7]. Тем самым медиасервер записывает группу чанков одним потоком, останавливая серводвигатель накопителя лишь один раз либо по причине выполнения работы (все чанки группы успешно записаны), либо при обнаружении «конца ленты» (носитель заполнен), либо в случае сбоя работы накопителя. В первом случае тот чанк, который оказался частично записанным, исключается из метаинформации текущей группы чанков и помещается в новую группу. Необходимым условием для старта записи группы является достаточный объем накопленных в кэше чанков, определяемый параметром MIN_DATA_SIZE_TO_WRITE.

В текущей версии АСХД используются три вида групп чанков (см. рис. 4). «Моно-группа» содержит чанки одного архивного объекта, размер которого превышает CHUNK_SIZE. Для записи множества небольших по размеру архивных объектов используется «ассорти-группа». В тех случаях, когда объем накопленных в кэше чанков недостаточен для формирования «ассорти-группы», а время ожидания соответствующих пользовательских заданий уже превышает некоторый лимит (параметр SMALL_TASK_WAITING), медиасервер использует «гибридную группу», в которую помещает наряду с накопленными чанками малоразмерных архивных объектов и один полноразмерный (CHUNK_SIZE) чанк какого-либо кэшированного объекта большего размера.

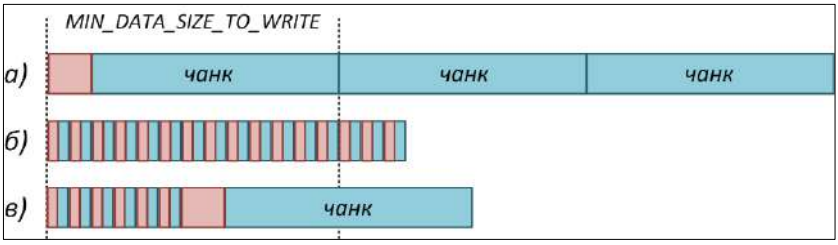


Рис. 4. Примеры трех видов групп чанков – (а) моно, (б) ассорти, (в) гибрид
Fig. 4. Chunk group samples – (а) mono group, (б) assorti group, (в) hybrid group

С целью повышения вероятности последующего восстановления архивного объекта разные реплики одного чанка никогда не помещаются в одну группу и на один носитель.

Первичная инициализация архивных носителей производится в момент их регистрации в АСХД, что снижает вероятность негативных последствий человеческих ошибок при работе с ленточными картриджами и облегчает сопровождение архивного фонда в долгосрочной перспективе. Низкоуровневый формат записи данных на архивный носитель (см. рис. 5) разработан с использованием стандартных меток [8].

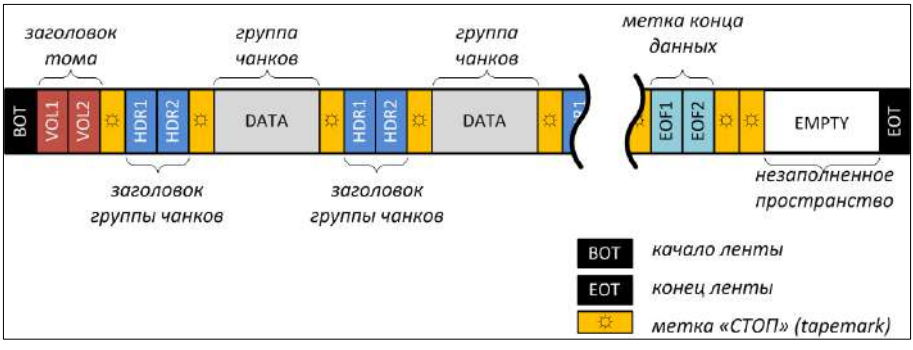


Рис. 5. Низкоуровневый формат тома
Fig. 5. Low-level volume format

5. Обеспечение целостности

Обеспечение целостности хранящихся архивных объектов в течение их срока жизни является первичной задачей АСХД. Она решается применением трех методов – использованием средств контроля целостности (контрольных сумм), введением избыточности (репликацией чанков), а также изоляцией архивных объектов.

5.1 Контроль целостности данных

Одной из болевых точек систем хранения, состоящих из большого количества компонентов, считается появление «скрытого повреждения данных», в борьбу с которым включены как производители дисков, дисковых подсистем, сетевого оборудования, так и разработчики промышленных файловых систем. Во многом эти усилия сосредоточены на внедрении механизмов сквозного контроля целостности по всей глубине стека блочного ввода/вывода – от серверного буфер-кэша до секторов на дисковых пластинах. Вместе с тем, контроль целостности информации в системах ленточной памяти по-прежнему лежит на плечах разработчиков прикладного ПО. В рамках АСХД контроль целостности данных осуществляется благодаря использованию контрольных сумм чанков на нескольких этапах их траектории движения (см. рис. 6). Согласно разработанному сетевому протоколу транспортной сессии, контрольная сумма передается непосредственно перед соответствующим чанком, что позволяет осуществлять однопроходной контроль целостности данных на приемной стороне.

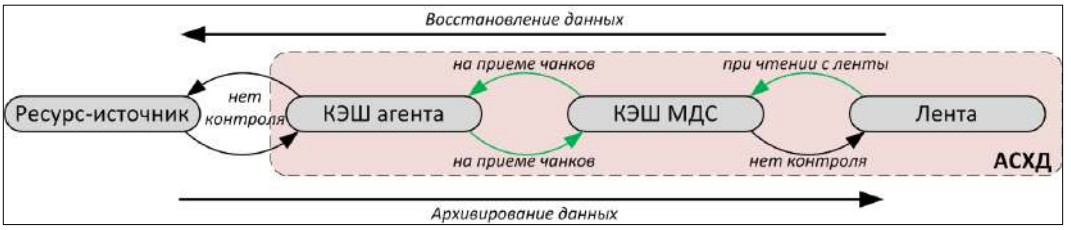


Рис. 6. Контроль целостности данных вдоль траектории их движения
Fig. 6. Data consistency control along the data path

Генерация контрольной суммы производится агентом на этапе считывания чанка из ресурса-источника, тогда как ее верификация производится:

- медиасервером при приеме архивируемых данных от агента;
- медиасервером при считывании данных с лент (в ходе выполнения операции восстановления или проверки архивных объектов);
- агентом при приеме восстанавливаемых данных от медиасервера.

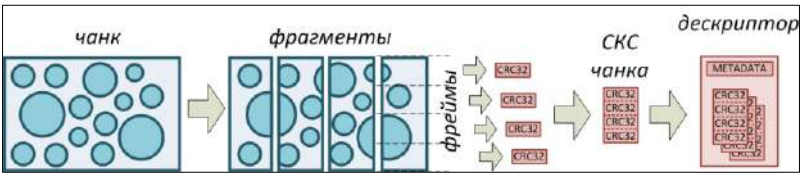


Рис. 7. Схема формирования «склеенной» контрольной суммы фрагмента
Fig. 7. Compound checksumming for a chunk fragment

Для обеспечения высокой скорости потокового контроля целостности в АСХД реализована идея «склеенных» контрольных сумм (далее по тексту СКС) чанков, заимствованная из проекта Apache HDFS [9]. В рамках транспортной сессии отправитель посылает очередной чанк в виде группы фрагментов фиксированного размера (параметр T_SESSION_BUFFER_SIZE). Непосредственно перед отправкой фрагмента рассчитывается его СКС как конкатенация 32-битных контрольных сумм более мелких порций данных (фреймов) размером T_SESSION_FRAME_SIZE (см. рис. 7).

Приемная сторона проводит однопроходной контроль целостности полученных фрагментов путем повторного расчета контрольных сумм фреймов и сравнения их с СКС, сформированной на стороне отправителя. По успешному завершению приема всех фрагментов чанка приемная сторона сохраняет все полученные СКС в виде цельной контрольной суммы чанка. После приема последнего байта архивного объекта, все контрольные суммы его чанков, наряду с подробной метаинформацией этого объекта, записываются в чанк-дескриптор, который затем помещается в «голове» каждой группы чанков на ленте.

На этапе восстановления архивного объекта с ленты считывание дескриптора (т.е. и всех контрольных сумм) непосредственно перед считыванием чанков данного архивного объекта позволяет таким же образом произвести однопроходной потоковый контроль целостности восстанавливаемых данных. Эта же схема применяется и далее, в ходе транспортирования данных от медиасервера к агенту.

5.2 Изоляция архивных данных

Для предотвращения случайного или намеренного повреждения архивных данных они должны быть максимально отдалены (изолированы) от наиболее вероятных источников проблем – от человеческой ошибки и от сбоев в программной или аппаратной части системы.

Изоляция архивного объекта в АСХД рассматривается и реализуется в нескольких аспектах:

- инкапсуляция архивного объекта – пользовательские данные преобразуются при архивировании в один из внутренних форматов АСХД и теряют связь с ресурсом-источником;
- ограничение пользовательского интерфейса – прямой доступ к архивному объекту запрещен, взаимодействие с АСХД осуществляется посредством заданий (заявок);
- организация оборота архивных носителей – заполненные пользовательскими данными ленты извлекаются из накопителей, а спустя определенный период времени они могут быть перемещены в хранилище.

Принятые меры по изоляции архивных данных позволяют, прямо или косвенно, снизить вероятность их повреждения со стороны возможных ошибочных действий пользователей, обсуживающего персонала и сбоев самой системы.

5.3 Опциональная репликация

Для улучшения сохранности информации в системах хранения широко применяется метод введения избыточности либо путем n-кратной ее репликации, либо добавлением кодов коррекции. В каждом отдельном случае выбор вида избыточности определяется, главным образом, скоростью кодирования, зависящей от сложности соответствующего алгоритма и имеющихся вычислительных ресурсов, а также объемом накладных расходов, т.е. объемом добавленной (избыточной) информации. Так как ленточная память обладает минимальной (по индустрии) удельной стоимостью гигабайта, а требования к обеспечению пропускной способности АСХД не предъявляются, то в этой системе применяется метод репликации.

Избыточность реализована на нижнем логическом уровне – на уровне чанков. Медиасервер компонует группы чанков таким образом, чтобы в рамках одной группы не было разных номеров реплик чанков одного архивного объекта. Тем самым гарантируется, что при включенной избыточности у архивного объекта повреждение одной группы его чанков не понизит шансы на целостное восстановление этого архивного объекта из другой группы.

Другая «линия обороны», повышающая вероятность восстановления архивированных данных в первоначальном состоянии, заключается в записи групп чанков, имеющих разные идентификаторы реплик, на разные носители. В случае потери или выхода из строя

отдельного носителя, на котором были сохранены чанки реплицируемого архивного объекта, медиасервер попробует восстановить недостающие чанки с другого носителя и собрать этот архивный объект.

Наконец идеальным вариантом является одновременное использование нескольких ленточных накопителей на этапе записи реплицируемого архивного объекта, когда подготовленные группы чанков, имеющих разные идентификаторы реплик, записываются параллельно. Тем самым, помимо сокращения времени записи, устраняется еще один фактор, потенциально влияющий на целостность записываемой информации – снижение качества работы отдельного накопителя, вызванное износом или загрязнением его магнитных головок.

6. Классы сервиса

Несмотря на достоинства, которые дает введение избыточности, для хранения некоторых типов данных применение репликации не всегда оправдано. Например, регулярное резервное копирование данных некоторой активной информационной системы порождает в СХ множество объектов хранения, в которых зафиксированы различные состояния одной и той же информационной системы в разные моменты времени. Если в период времени между двумя операциями резервного копирования состояние этой информационной системы изменяется лишь частично, то избыточность, в какой-то степени, появляется уже вследствие регулярности таких операций. Множественность таких сохраненных состояний информационной системы вскоре делает неактуальными наиболее старые из них, поэтому обычно они автоматически удаляются.

Так как потребность в резервном копировании возникает даже чаще, чем потребность в архивировании, а низкая удельная стоимость единицы хранения востребована в обоих этих случаях, и, учитывая, что в техническом плане эти функции не отличаются друг от друга, обе они были реализованы в рамках АСХД в виде так называемых классов сервиса.

Под классом сервиса здесь понимается именованная совокупность параметров сервиса АСХД по обработке пользовательской заявки, которую назначает ее владелец в момент постановки в очередь. В этот набор параметров входят:

- степень репликации порождаемого архивного объекта, т.е. количество реплик у каждого чанка;
- период времени, после которого архивный объект считается устаревшим и требующим удаления либо продления жизни;
- минимальный и максимальный объем объекта;
- размер чанка;
- приоритетность в обработке и пр.

Состав и значения параметров различных классов сервиса определяются техническими характеристиками площадки АСХД и особенностями технологических процессов обработки информации, существующих в организации, в которой установлена система. Настройку классов сервиса осуществляет администратор площадки АСХД.

Например, для сохранения данных в АСХД пользователю предлагаются два класса сервиса – A и B. Назначая класс A в качестве атрибута задания на архивирование, пользователь «говорит» архивной системе, чтобы сформированный на основе этой заявки архивный объект хранился в течение установленного в организации срока в нескольких репликах. Тогда как архивный объект, порожденный заданием класса B, будет храниться без репликации не более полугода, после чего будет автоматически удален системой.

Опциональная возможность включения репликации архивного объекта и задание других его свойств посредством назначения нужного класса сервиса позволяет пользователю выбирать в каждом отдельном случае наиболее подходящие способы обработки и хранения различных видов данных.

Другой важной особенностью классов сервиса является предоставляемая ими возможность реализации новых функций АСХД без необходимости изменять уже существующие отлаженные функции. Возможность добавления новых классов сервиса создает основание для дальнейшего развития системы при сохранении целевых свойств АСХД в отношении как новых, так и ранее заархивированных данных.

7. Типовой процесс

Для того чтобы заархивировать часть данных, расположенных на одном из ресурсов-источников, разработчик/пользователь соответствующего тематического проекта, зарегистрированный в АСХД, должен сформировать задание (заявку), включающее ресурс-источник, список архивируемых данных, их описание и прочую метаинформацию (см. рис. 8). Приняв в обработку это задание, АСХД считывает указанные в нем пользовательские данные, преобразует их и создает на их основе один или несколько архивных объектов, которые спустя некоторое время сохраняются на архивных носителях. В случае успешного окончания этой операции пользовательская заявка считается выполненной.

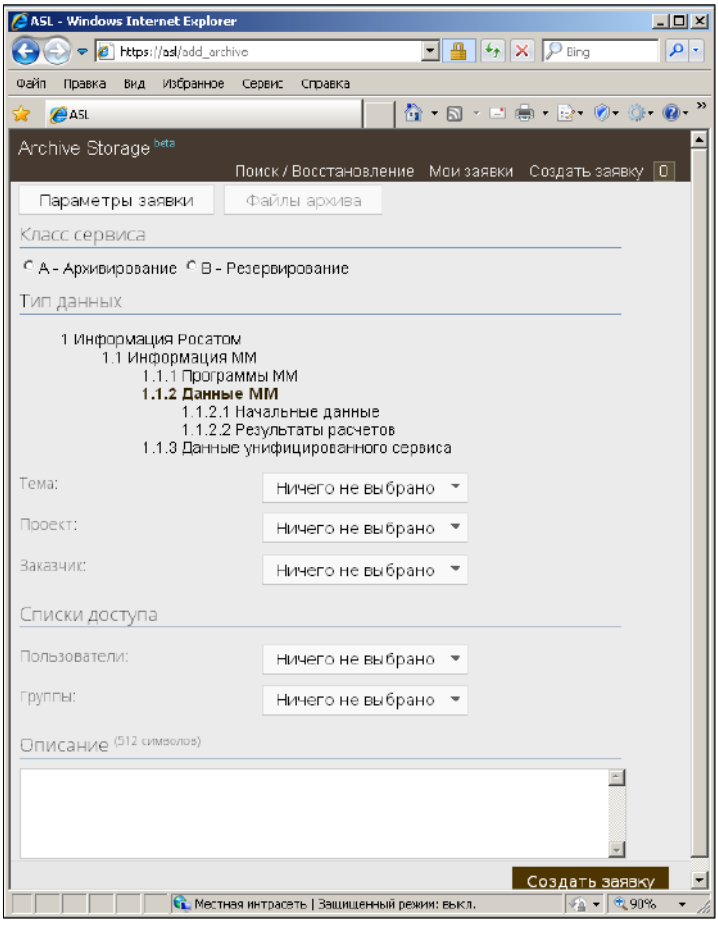


Рис. 8. Веб-форма создания пользовательской заявки
Fig. 8. Archival task creation web-form

Пользователь архивной системы может контролировать ход выполнения своих заданий и имеет доступ ко всей истории собственных заявок.

Найти необходимые архивные объекты в АСХД можно путем фильтрации всего списка архивных объектов по набору описательных атрибутов либо с помощью встроенной системы полнотекстового поиска, которая индексирует неформализованные текстовые описания каждой пользовательской заявки (см. рис. 9). Фильтрация списка объектов доступна как в консоли, так и в web-интерфейсе, а полнотекстовый поиск – только в web-интерфейсе.

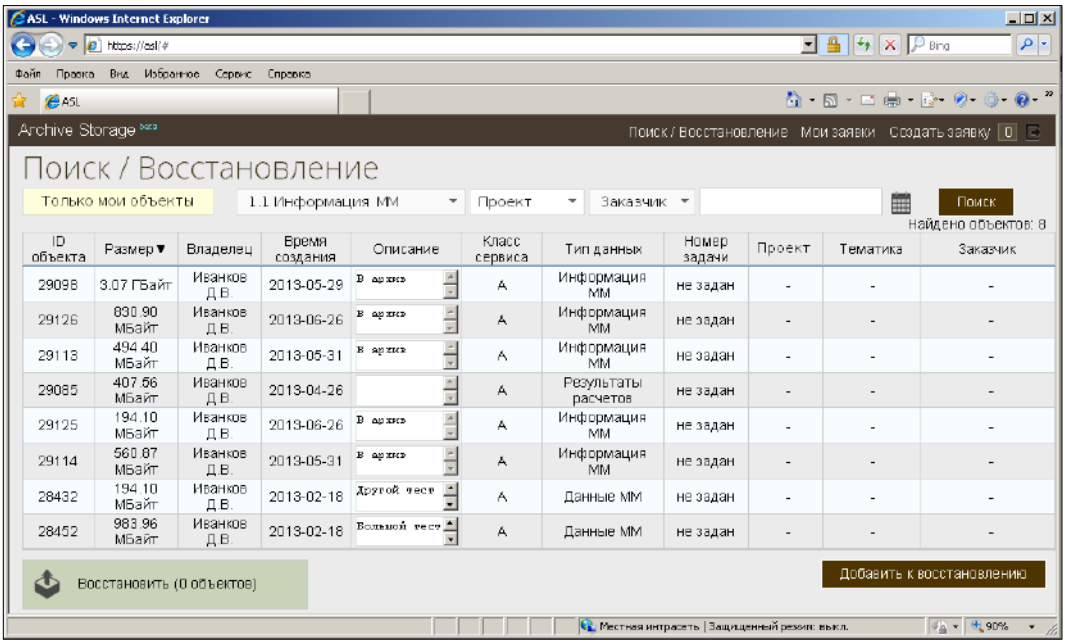


Рис. 9. Веб-форма поиска архивных объектов
Fig. 9. Archival object searching web-form

Для восстановления данных из АСХД пользователь должен знать идентификаторы соответствующих архивных объектов, которые он получает в результате выполнения операции поиска. Выбрав из результатов поиска релевантные архивные объекты, пользователь формирует задание, указывая в нем, помимо списка идентификаторов восстанавливаемых архивных объектов, ресурс-приемник, куда система должна будет поместить запрошенные пользователем данные. Система принимает это задание в обработку при условии, если (а) данный пользователь является владельцем запрошенных архивных объектов, или (б) он входит в списки доступа к этим объектам, или (в) он входит в группу «суперпользователей», имеющих доступ к любому архивному объекту.

Перед началом выполнения этого задания АСХД определяет набор архивных носителей, содержащих восстанавливаемые архивные объекты. В случае если соответствующие носители ранее были вытеснены в хранилище, АСХД формирует запрос оператору хранилища, который может либо самостоятельно (на АРМ оператора хранилища) считать запрошенные данные с носителей, либо инициировать их транспортировку и загрузку в ленточные библиотеки для последующей автоматической обработки. Считанные с архивных носителей пользовательские данные система помещает в указанное в заявке место на ресурсе-приемнике, преобразуя их в первоначальное, предшествующее архивированию, состояние.

8. Расчет площадки АСХД

При проектировании площадки АСХД для вычислительного центра необходимо провести расчет аппаратной составляющей системы, в частности определить количество и конфигурацию медиасерверов, тип, количество и конфигурацию автоматизированных

ленточных библиотек. Для этого построим простую линейную модель отдельного медиасервера, а на ее основе смоделируем работу всей АСХД (см. рис. 10).

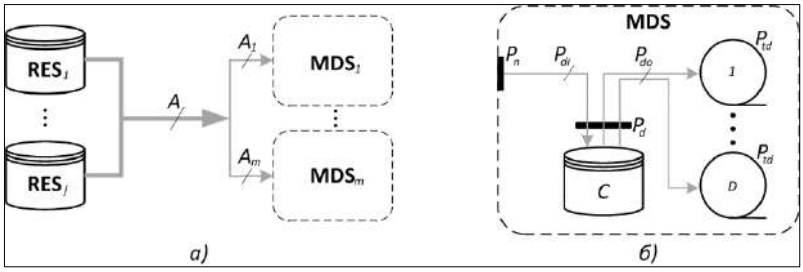


Рис. 10. (а) Модель архивной системы хранения и (б) модель медиасервера
Fig. 10. (a) The archive data system model and (б) the mediaserver model

Приведем основные положения этой модели.

- суммарный входной поток в АСХД (*A*) формируется из *J* ресурсов-источников данных (RES_j на рис. 10а), принадлежащих одной или нескольким ВВС;
- входной поток *A* равномерно распределяется между *M* однотипными медиасерверами (MDS_m на рис. 10а);
- считаем режим работы АСХД устоявшимся в течение определенного интервала времени, на протяжении которого суммарный объем регулярных пользовательских заявок на архивирование преобладает над объемом эпизодических заявок на восстановление данных; для удобства расчета длительность этого интервала выбираем равной 24 часам;
- рассматриваем медиасервер в виде накопительного устройства (дисковый кэш) и группы исполнительных устройств – ленточных драйвов (см. рис. 10б); поступающие на вход медиасервера данные сохраняются в кэше (*C* на рисунке 10б) и, по мере накопления достаточного суммарного объема, содержимое кэша записывается на ленточный носитель; для простоты считаем этот процесс линейным и бессбойным.

Параметрами модели медиасервера являются:

- пропускная способность сетевого адаптера (P_n), ограничивающая поток записи в кэш (P_{di});
- тип архивных (ленточных) носителей, определяющий в свою очередь ряд его технических характеристик: максимальную ёмкость носителя (V_t), максимальную производительность накопителя (P_t) и коэффициент аппаратного сжатия накопителя (K_z).

Также в модель вводятся коэффициенты, которые учитывают явления, имеющие место в работе реального медиасервера АСХД.

Реализация транспортного протокола (между агентом и медиасервером) вносит издержки, сужающие поток записи в кэш. Введем коэффициент снижения интенсивности сетевого потока K_n ($0 \le K_n \le 1$)

$$P_{di} = P_n \times K_n.$$

Производительность дискового контроллера в устоявшемся режиме обычно меньше пиковых «мгновенных» показателей, что связано с особенностями реализации буферизированного ввода/вывода операционной системы. Кроме того, полоса пропускания дискового контроллера может частично расходоваться и другими приложениями, запущенными на медиасервере. Обозначим возможное снижение КПД дискового контроллера через корректирующий коэффициент K_d ($0 \le K_d \le 1$). Будем считать, что в устоявшемся режиме работы медиасервера потоки записи в кэш (P_{di}) и чтения из кэша (P_{do}) должны быть одинаковыми. Для выполнения этого условия достаточно оснастить медиасервер дисковым контроллером, имеющим производительность не менее P_d.

$$P_d = P_{di} * 2/K_d.$$

Ленточный накопитель достигает наибольшей производительности (P_t) только в условиях достаточного объема записываемых или считываемых данных, когда сервопривод работает с максимальной скоростью. В реальности эти условия не всегда достижимы ввиду как различий в размерах архивных объектов, так и особенности группировки чанков. Поэтому в модель медиасервера вводится корректирующий коэффициент K_t $(0 \leq K_t \leq 1)$, который отражает снижение производительности накопителя, вызванное разными скоростными режимами сервопривода и периодами вынужденного ожидания поступления данных в кэш.

Процесс записи накопленных в дисковом кэше данных на современные ленточные носители может длиться продолжительное время. Не редкой является ситуация, когда на фоне полной занятости накопителей (D_w) операциями записи на медиасервер поступает запрос на восстановление данных из архива. Для своевременной обработки таких запросов в модель медиасервера вводится параметр D_r, который определяет количество накопителей в медиасервере, зарезервированных исключительно под операции чтения. Таким образом, необходимое количество ленточных накопителей в медиасервере можно представить суммой:

$$D_m = D_w + D_r = \frac{P_d}{P_t \times K_t} + D_r$$

Периоды записи и чтения данных, выполняемые реальным ленточным накопителем, прерываются служебными операциями над картриджами – перемотка, загрузка/выгрузка, перемещение между накопителем и слотами хранения. Поэтому для расчета производительности медиасервера в устоявшемся режиме (P_m) вводится корректирующий коэффициент K_m $(0 \leq K_m \leq 1)$, отражающий влияние периодических остановок процесса записи данных.

$$P_m = P_t \times K_t \times D_w \times K_m$$

Непрерывный режим одновременной работы нескольких ленточных накопителей может быть обеспечен при наличии достаточного объема дискового кэша (C_m) в медиасервере. Для его оценки сверху учтем максимальную степень аппаратного сжатия, достижимую на выбранном типе драйва.

$$C_m = \frac{D \times V_t}{K_z}$$

Размещающиеся в голове каждой группы чанков контрольные суммы и метаинформацию соответствующего архивного объекта (см.п.4) следует отнести к накладным расходам емкости архивного носителя, объем которых описывается корректирующим коэффициентом Q (обычно $Q \leq 0.01$).

Коэффициент компрессии (Z), описывающий степень сжатия записываемых данных, сильно зависит от самих этих данных. На практике он не превышает соответствующую паспортную характеристику накопителя $(1 \leq Z \leq K_z)$.

АСХД разрешает введение избыточности при записи архивируемых данных (см. разд. 6), поэтому при расчете потребления архивных носителей в устоявшемся режиме работы медиасервера добавим в модель коэффициент репликации $(1 \leq R \leq 2)$, описывающий среднюю степень избыточности среди множества пользовательских заявок.

Таким образом, количество медиасерверов (M), необходимое для обслуживания входного потока пользовательских данных в АСХД (A), можно оценить, отталкиваясь от вычисленной производительности отдельного медиасервера.

$$M = A/P_m$$

Количество носителей, которое потребляет АСХД для сохранения входного потока пользовательских данных, рассчитывается по следующей формуле.

$$N = \sum_{m=1}^{M} \frac{A_m \times R}{V_t \times (1 - Q) \times Z}$$

На основании вычисленного количества медиасерверов и ленточных накопителей можно спроектировать структуру аппаратной части площадки АСХД, а именно выбрать модель, конфигурацию и количество ленточных библиотек. Вычисленное количество потребляемых носителей за период позволяет определить общее количество слотов, которым должны обладать выбранные библиотеки.

Наличие долгосрочного контракта на поддержку ленточных библиотек со стороны производителя или интегратора создает наилучшие условия для сопровождения площадки АСХД. Если же такая поддержка недоступна, то в расчет аппаратной конфигурации площадки необходимо заложить некоторую степень избыточности.

Рассмотренный линейный подход к расчету аппаратной конфигурации площадки АСХД, безусловно, не претендует на полноту и точность, но позволяет провести быструю оценку объема необходимого оборудования для реализации функции архивного хранения данных в вычислительном центре.

9. Управление архивными носителями

Управление архивными носителями в АСХД может осуществляться в автоматическом и полуавтоматическом режиме.

Автоматический режим управления архивными носителями применим для площадки АСХД, весь архивный фонд которой помещается в слотах активных ленточных библиотек. Полуавтоматический режим актуален для организаций, чей архивный фонд превышает объем доступных ресурсов (количество слотов) активных библиотек. В таких ситуациях возникает необходимость организации off-site хранилища архивных носителей и разработки техпроцессов учета, розыска и транспортирования носителей между библиотеками и хранилищем. Участие в этих техпроцессах персонала (операторов хранилища) должно регламентироваться специальной политикой управления архивным фондом и разработанным регламентом обращения с архивными носителями.

Рассмотрим вопросы организации хранилища архивных носителей на примере типовой площадки АСХД, функционирующей в полуавтоматическом режиме.

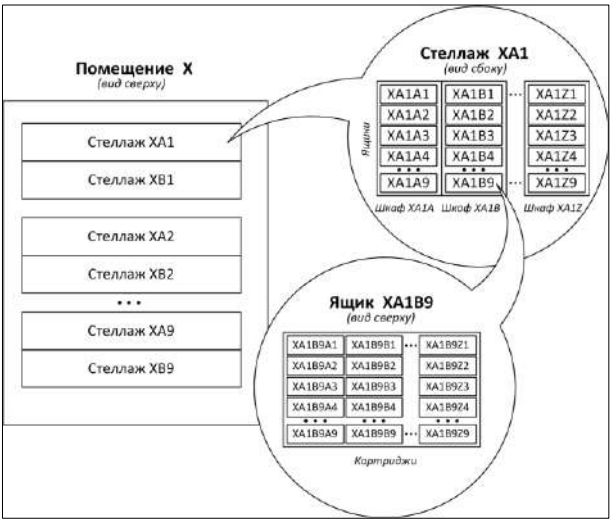


Рис. 11. Структура хранилища архивных носителей
Fig. 11. Structure of the vault

Структура хранилища (см. рис. 11) разработана так, чтобы облегчить оператору задачу поиска отдельных картриджей и упростить ориентирование в хранилище:

- хранилище может состоять из нескольких помещений;
- в каждом помещении хранилища устанавливаются стеллажи, состоящие из нескольких рядов соединенных друг с другом шкафов;
- в каждый шкаф входит несколько ящиков, в которых размещаются картриджи.

Картриджи уникально идентифицируются с помощью нанесенного на корпус штрих-кода. Каждый картридж поставляется и хранится в индивидуальной защитной пластиковой коробке. При регистрации нового картриджа в базе данных АСХД производится присвоение ему «адреса» в хранилище (см. рис. 12), который наносится на его коробку в символьной форме и в виде штрих-кода. Когда востребованные картриджи перемещаются в ленточные библиотеки соответствующие коробки остаются в архивном хранилище и никогда его не покидают. Транспортировка картриджей производится в специальных защитных контейнерах, обеспечивающих высокий уровень безопасности носителей. Задачи транспортирования носителей, поддержания порядка в хранилище и обеспечения соответствия метаинформации АСХД реальному состоянию архивного фонда возложены на службу операторов хранилища.

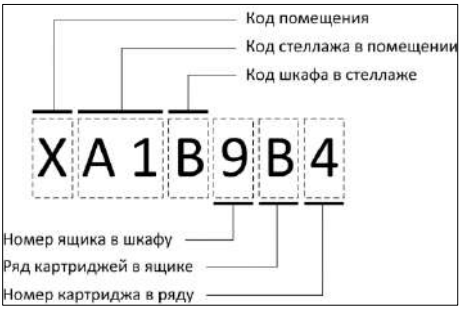


Рис. 12. Схема формирования «адреса» зарегистрированного картриджа
Fig. 12. «Address» of registered archived cartridge in the vault.

Работа оператора хранилища архивных носителей, вмещающего большое количество картриджей, построена вокруг АРМ оператора, который оснащен сканерами штрих-кодов и набором ленточных накопителей (всех используемых в АСХД типов носителей) в настольном исполнении. АРМ оператора управляется входящим в состав АСХД приложением «АРМ оператора хранилища» (см. рис. 13).

Использование сканеров штрих-кодов сокращает человеческие ошибки ввода информации и значительно облегчает работу оператора с множеством картриджей и коробок. Наличие ленточных накопителей в составе АРМ оператора позволяет оперативно выполнять пользовательские задания на восстановление отдельных архивных объектов, полностью или частично размещенных на носителях, которые были перемещены в хранилище, без необходимости транспортировки и загрузки этих носителей обратно в библиотеки. Тем самым время выполнения таких «точечных» заявок может быть сокращено.

В рамках АРМ регламентированы и реализованы следующие техпроцессы обращения с архивными носителями:

- передача носителей из хранилища для загрузки в ленточные библиотеки;
- приём в хранилище носителей, выгруженных из ленточных библиотек;
- инвентаризация хранилища архивных носителей;
- прием на учёт новых архивных носителей в установленном порядке и регистрация их на АРМ оператора хранилища АСХД.

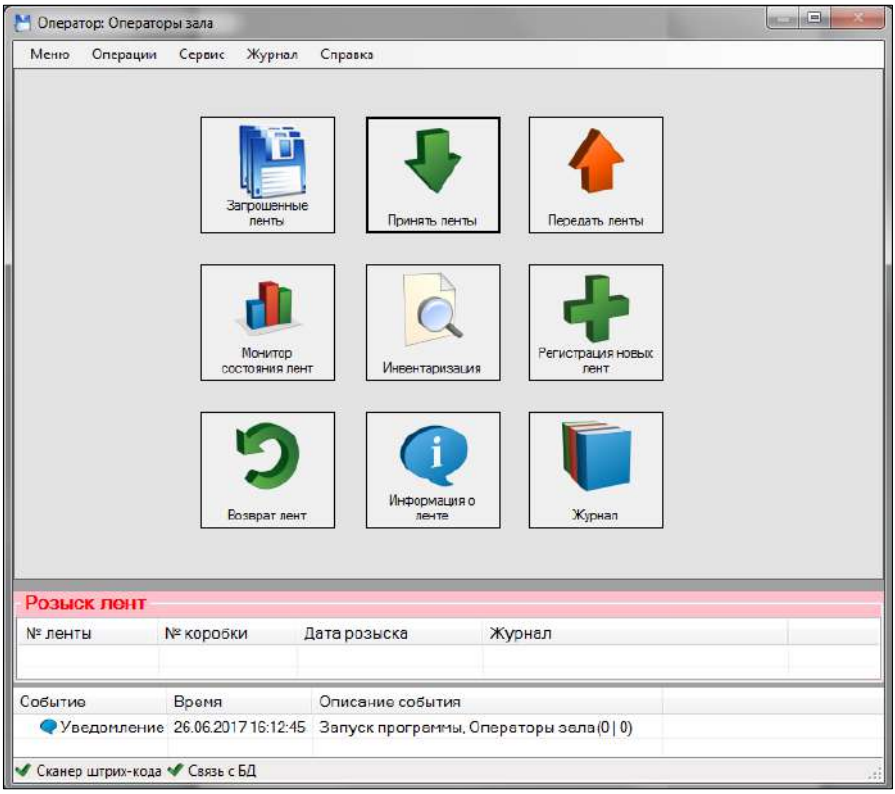


Рис. 13. Главное окно АРМа оператора хранилища
Fig. 13. Screenshot of the vault operator's desktop application

Для автоматизации техпроцесса инвентаризации хранилища используется мобильный сканер штрих-кодов с разработанным в рамках АСХД мобильным приложением, поддерживающим синхронизацию с АРМ и базой данных АСХД. Благодаря тому, что защитные коробки картриджей сделаны из полупрозрачного пластика, сканирование идентификаторов не требует изъятия картриджа из коробки, что существенно сокращает время инвентаризации хранилища.

11. Заключение

Использование архивной системы хранения данных в РФЯЦ-ВНИИТФ позволило решить проблему долговременного целостного хранения расчетных данных. За время ее эксплуатации были успешно обработаны десятки тысяч пользовательских заявок. Благодаря АСХД удалось реализовать методы управления расчетными данными в течение их жизненного цикла и тем самым повысить эффективность использования оперативных файловых систем ВВС.

Автор выражает надежду на то, что изложенный в данной статье подход мог бы быть полезен и другим специалистам, занимающимся организацией ресурсов долговременного хранения данных.

Список литературы / References

[1] СПО Супер-ЭВМ. Available at: http://vniitf.ru/article/spo-super-evm, accessed 28.08.2022 (in Russian).
[2] IBM HPSS. Available at: https://www.hpss-collaboration.org, accessed 28.08.2022.
[3] Enstore. Available at: https://www-stken.fnal.gov/enstore, accessed 28.08.2022.
[4] CERN Tape Archive. Available at: https://cta.web.cern.ch/cta, accessed 28.08.2022.

[5] Tape control program. Available at: https://github.com/iustin/mt-st, accessed 28.08.2022.
[6] Single or multi-drive SCSI media changer program. Available at: https://sourceforge.net/projects/mtx, accessed 28.08.2022.
[7] The Single UNIX Specification Version 3. Available at: https://unix.org/version3, accessed 28.08.2022.
[8] ANSI X3.27-1978. Available at: https://nulpubs.nist.gov/nistpub/Legacy/FIPS/fipspub79.pdf, accessed 28.08.2022.
[9] Apache Hadoop. Available at: https://hadoop.apache.org, accessed 28.08.2022.

Информация об авторе / Information about author

Дмитрий Владимирович ИВАНКОВ – начальник лаборатории. Сфера научных интересов: проектирование многоуровневых систем хранения данных, разработка высокопроизводительных систем хранения данных, исследования методов управления данными.

Dmitry Vladimirovich IVANKOV – Head of the Laboratory. Research interests: design of tiered data storage systems, development of high performance storage systems, research in data management methods.

DOI: 10.15514/ISPRAS-2022-34(4)-10

Метод улучшения качества речи с использованием модифицированного кодирующего-декодирующего пирамидального трансформера

А.А. Лепендин, ORCID: 0000-0001-5097-5023 <andrey.lependin@gmail.com>
Р.С. Насретдинов, ORCID: 0000-0003-3368-523X <uniform97@gmail.com>
И.Д. Ильяшенко, ORCID: 0000-0001-5119-3832 <ilya-ilyash@yandex.ru>
Алтайский государственный университет,
656049, Россия, г. Барнаул, пр. Ленина, д. 61

Аннотация. Развитие новых технологий голосового общения привело к необходимости совершенствования методов улучшения качества речи. Современные пользователи информационных систем предъявляют высокие требования как к разборчивости голосового сигнала, так и к его субъективно воспринимаемому качеству. Данная работа посвящена развитию нового подхода к решению актуальной задачи улучшения качества речи. Для этого было предложено использовать модифицированную нейронную сеть пирамидального трансформера, использующую двухкомпонентную структуру «кодер-декодер». Кодирующая компонента сети осуществляла сжатие спектра голосового сигнала в пирамидальную серию внутренних представлений. Декодирующая компонента, используя преобразования самовнимания, восстанавливала маску комплексного отношения очищенного и искаженного сигналов на основе вычисленных кодером внутренних представлений. Были рассмотрены две возможные функции потерь для обучения предложенной нейросетевой модели. Показано, что использование частотного кодирования, подмешиваемого к входным данным, позволило улучшить качество работы предложенного подхода. Реализованная на языке Python и библиотеке глубокого обучения PyTorch нейронная сеть обучалась и тестировалась на наборе данных DNS Challenge 2021. Она продемонстрировала высокое качество работы по сравнению с другими современными методами улучшения качества речи. В работе был проведен качественный анализ процесса обучения реализованной нейронной сети, который показал, что предлагаемая нейросетевая модель постепенно переходила от простого маскирования шума на ранних эпохах обучения к восстановлению пропущенных формантных компонент голоса говорящего на более поздних эпохах. Это приводило к высоким значениям численных метрик качества работы предложенного подхода и высокому субъективному качеству речи.

Ключевые слова: улучшение качества речи; очистка от шума; маскирование шума; глубокая нейронная сеть; глубокое обучение; архитектура кодер-декодер; пирамидальный трансформер; самовнимание.

Для цитирования: Лепендин А.А., Насретдинов Р.С., Ильяшенко И.Д. Метод улучшения качества речи с использованием модифицированного кодирующего-декодирующего пирамидального трансформера. Труды ИСП РАН, том 34, вып. 4, 2022 г., стр. 135-152.. DOI: 10.15514/ISPRAS-2022-34(4)-10

Благодарности: Исследование выполнено за счет гранта Российского научного фонда № 22–21–00199, https://rscf.ru/project/22-21-00199/.

Speech Enhancement Method Based on Modified Encoder-Decoder Pyramid Transformer

A.A. Lependin, ORCID: 0000-0001-5097-5023 <andrey.lependin@gmail.com>
R.S. Nasretdinov, ORCID: 0000-0003-3368-523X <uniform97@gmail.com>
I.D. Ilyashenko, ORCID: 0000-0001-5119-3832 <ilya-ilyash@yandex.ru>
Altai State University,
61, Lenin st., Barnaul, 656049, Russia

Abstract. The development of new technologies for voice communication has led to the need of improvement of speech enhancement methods. Modern users of information systems place high demands on both the intelligibility of the voice signal and its perceptual quality. In this work we propose a new approach to solving the problem of speech enhancement. For this, a modified pyramidal transformer neural network with an encoder-decoder structure was developed. The encoder compressed the spectrum of the voice signal into a pyramidal series of internal embeddings. The decoder with self-attention transformations reconstructed the mask of the complex ratio of the cleaned and noisy signals based on the embeddings calculated by the encoder. Two possible loss functions were considered for training the proposed neural network model. It was shown that the use of frequency encoding mixed with the input data has improved the performance of the proposed approach. The neural network was trained and tested on the DNS Challenge 2021 dataset. It showed high performance compared to modern speech enhancement methods. We provide a qualitative analysis of the training process of the implemented neural network. It showed that the network gradually moved from simple noise masking in the early training epochs to restoring the missing formant components of the speaker's voice in later epochs. This led to high performance metrics and subjective quality of enhanced speech.

Keywords: speech enhancement; noise reduction; noise masking; deep neural network; deep learning; encoder-decoder architecture; pyramid transformer; self-attention.

For citation: Lependin A.A., Nasretdinov R.S., Ilyashenko I.D. Speech Enhancement Method Based on Modified Encoder-Decoder Pyramid Transformer. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 4, 2022, pp. 135-152 (in Russian). DOI: 10.15514/ISPRAS-2022-34(4)-10

Acknowledgements. This work was supported by the grant from the Russian Science Foundation, project no. 22-21-00199, https://rscf.ru/en/project/22-21-00199/.

1. Введение

В настоящее время технологии цифровой обработки и передачи речи получили широкое распространение. Растет частота использования голосовых помощников в мобильных устройствах и интеллектуальных колонках. Множество мобильных устройств, планшетов и персональных компьютеров используется для аудио- и видео-звонков, телеконференций. Естественным образом повышаются требования пользователей к качеству передаваемого речевого сигнала. В связи с этим в последние годы увеличилось число практико-ориентированных исследовательских работ и проектов, ориентированных на разработку новых методов повышения качества речи (speech enhancement) [1]. К числу требований, предъявляемым к новым разработкам следует отнести следующее.

- Эффективное подавление фоновых шумов в аудиосигнале, соответствующее сценариям повседневного использования. Современные пользователи часто работают в открытых общественных пространствах, на улице, в шумных помещениях, транспорте. Из-за этого сигнал содержит фрагменты речи окружающих данного пользователя людей, крики детей, природные шумы, звуки уличного транспорта. Большинство подобных шумов является нестационарными, их мощность может быть сравнима и даже превосходить мощность полезного речевого сигнала. Их спектральные характеристики также имеют широкий «разбег»: широкополосные сигналы уличного трафика, узкополосные фоновые шумы или сигналы со спектром сложной формы в случае речевых или речеподобных шумов;
- Улучшение качества должно не только повышать разборчивости речи. Это минимально

необходимое, но не достаточное условие. Особую важность представляет высокое субъективное качество улучшенной речи. Оно предполагает помимо разборчивости сохранение характерных интонационных и тембральных особенностей голоса.

В появлении новых, более эффективных методов заинтересованы крупные IT-компании, так как это становится еще одним «продающим» фактором для их программных продуктов. В частности, это подтверждается активностью компании Microsoft, уже на протяжении нескольких лет проводящей открытые соревнований в области алгоритмов улучшения качества речи – Deep Noise Suppression (DNS) Challenge [2, 3]. Крупнейшие научные конференции Interspeech и ICASSP, посвященные методам обработки речи, предоставляют свои площадки для обсуждения новых подходов к развитию методов улучшения качества речи.

Основной вектор развития новых подходов к улучшению качества речи связан в настоящее время с развитием современных методов глубокого обучения. Большинство работ последних лет предполагают использование глубоких нейросетевых моделей. Появляются представительные открытые наборы голосовых записей, образцов шумовых сигналов реального мира, модельных и измеренных импульсных характеристик помещений. Все это способствует быстрому прогрессу как в разработке новых нейросетевых моделей для обработки искаженной речи, так и в появлении новых специализированных методов их обучения.

В данной работе предложен новый подход к решению задачи улучшения качества зашумленной речи, основанный на применении глубокой нейронной сети типа трансформер.

2. Задача улучшения качества речи

Улучшение качества речи является одним из важных элементов современных телекоммуникационных систем. В зависимости от условий применения методы улучшения качества могут быть разделены на несколько типов. Выделяют адаптивные и неадаптивные подходы в зависимости от того, подстраиваются ли параметры используемых при улучшении речи методов под не стационарно меняющиеся полезный и шумовой компонент сигнала. В зависимости от числа микрофонов и их расположения выделяют одноканальное и многоканальное улучшение речи. Существующие многоканальные методы демонстрируют лучшее качество, однако в реальных сценариях использования число микрофонов не превосходит двух, расположены они относительно близко друг к другу, так что фактически их можно рассматривать как один канал. В работе решалась задача адаптивного одноканального улучшения качества речи.

2.1 Постановка задачи

При разработке предложенного метода предполагалось, что искажение, вносимое в одноканальный речевой сигнал, можно разделить на две части [1]. Первая, аддитивная, отвечала за присутствующие в записи фоновые шумы. Вторая, мультипликативная, определялась наличием реверберации. Искаженный голосовой сигнал представлялся следующей моделью:

$$Y(t) = S(t) * H(t) + N(t) = X(t) + N(t), \quad (1)$$

где $S(t)$ – чистый сигнал без искажений, $H(t)$ – импульсная характеристика помещения, применяемая для моделирования реверберации, $X(t)$ – сигнал с реверберацией, $N(t)$ – аддитивный шум, $*$ – операция свертки. После разбиения сигнала на перекрывающиеся временные окна и быстрого преобразования Фурье, получившееся представление сигнала можно записать следующим образом:

$$Y(f,t) = S(f,t) \cdot H(f,t) + N(f,t) = X(f,t) + N\,(f,t), \quad (2)$$

где $Y(f,t)$, $S(f,t)$, $H(f,t)$, $X(f,t)$ – комплексное представление сигналов в частотно-временной области, индекс $f = 0, \dots, F-1$ отвечает за номер соответствующей частотной полосы, $t = 0, \dots, T-1$ – за номер временного окна сигнала.

В данной работе решалась исключительно задача восстановления сигнала X, который мог представлять собой как реверберированную версию чистого сигнала S, так и сам чистый сигнал S. Набор данных включал в себя как сигналы без реверберации, так и с ней. Все оценки качества работы также проводились двух версий сигнала x.

2.2 Основные методы улучшения качества речи

Одноканальное улучшение речи согласно [1] может производиться следующими классическими методами: спектральным вычитанием, статистическим моделированием, алгоритмами выделения подпространств и методами маскирования. Каждая из этих групп обладает своими достоинствами и недостатками, связанными как с ограничениями на типы подавляемого шума, так и с требуемой вычислительной мощностью. Основной объединяющей чертой этих методов в классической постановке является опора на явно или не явно используемые статистические модели аддитивного шума, что ограничивает их применение. Еще одним препятствием для использования подобных подходов является низкое субъективное качество результатов их работы.

В силу указанных ограничений в последние годы широкое распространение стали получать методы улучшения качества, основанные на глубоких нейронных сетях. Они в той или иной степени стали развитием перечисленных выше классических методов. Появились нейросетевые аналоги методов статистического моделирования [4], адаптивной фильтрации [5]. Однако наибольшее число работ [6-13] было связано с развитием методов маскирования шума. Этот подход в классической постановке [1] связан с разделением всех элементов частотно-временного представления зашумленного сигнала (2) на два вида – относящиеся к речи и относящиеся к шуму. Метод шумоочистки должен вычислить так называемую идеальную бинарную маску (ideal binary mask) вида [1]:

$$\mathrm{IBM}(f,t) = \begin{cases} 0, & Y(f,t) \text{ содержит аддитивный шум,} \\ 1, & Y(f,t) \text{ содержит полезный сигнал.} \end{cases} \quad (3)$$

Подобный подход получил дальнейшее развитие в виде нескольких модификаций методов маскирования. В работе [11] было предложено вычислять идеальные маски вещественного отношения амплитуд сигнала и шума (IRM), позволявшие учитывать относительный вклад шумовой добавки в отдельных участках спектра голосового сигнала и, более того, к возможности частичного восстановления полезной компоненты из шумового фона. Далее, метод идеальной маски комплексного отношения (CRM) [12] позволил учесть фазовую составляющую сигнала, что повысило не только разборчивость речи, но и субъективное качество речи.

Для вычисления масок в работах [6, 12, 13] применялся подход на основе кодирующей-декодирующей архитектуры. Кодирующая компонента нейронной сети сжимала представление сигнала, по этому представлению декодером вычислялась требуемая маска. В [13] для кодирования и декодирования применялись комбинации сверточных и рекурсивных слоев. Слои свертки позволяли эффективно сжимать частотно-временное представление обрабатываемого сигнала, а рекурсивные слои осуществляли моделирование временных зависимостей в нем. Недостатком такого подхода являлось следующее. Рецептивное поле преобразования свертки является хорошо локализованным, поэтому при вычислении сжатого представления спектра сигнала могли теряться связи между различными участками спектра сигнала. Рекурсивные слои, в свою очередь, обладают эффектом «забывания» внутренних

состояний [14]. В целом это приводило к неточному восстановлению голосовых формант и потере качества.

Новым подходом, позволяющим учитывать нелокальные связи между удаленными по частотной или временной оси участками голосового сигнала, явились архитектуры типа трансформер (transformer neural network), использующие механизм самовнимания (self-attention) [15]. Подобные нейронные сети нашли применение во всех основных задачах глубокого обучения – от обработки естественных языков [15] до автоматической сегментации изображений [16, 17], постепенно вытесняя предыдущие поколения глубоких нейронных сетей, основанные на преобразованиях свертки и рекуррентных слоях. В задачах обработки речи преобразование самовнимания применялось для автоматического распознавания голоса [18], оценки эмоциональной окраски [19] и синтеза речи [20]. В задаче улучшения качества речи данный механизм использовался как замена части рекуррентных и сверточных преобразований [9]. Принципиально новым в данной работе является использование механизма самовнимания на всех этапах обработки искаженного голосового сигнала.

3. Предлагаемый метод

Предложенная нейросетевая модель была основана на модификации сети пирамидального трансформера PVT (Pyramid Vision Transformer) [16]. На вход сети подавалась спектрограмма обрабатываемого сигнала $\mathrm{Y(f,t)}$. Сеть имела архитектуру «кодер-декодер» и позволяла предсказывать комплексную маску отношения очищенного и зашумленного сигнала $\mathrm{M(f,t)}$. С помощью вычисленной маски восстанавливалось комплексное частотно-временное представление очищенного сигнала $\widehat{\mathrm{X}}(\mathrm{f,t})$:

$$\widehat{\mathrm{X}} = (\mathrm{Y_r} \odot \mathrm{M_r} - \mathrm{Y_i} \odot \mathrm{M_i}) + \mathrm{j}(\mathrm{Y_r} \odot \mathrm{M_i} + \mathrm{Y_i} \odot \mathrm{M_r}), \quad (4)$$

где индексами r и i обозначены вещественная и мнимые части комплексных значений, j – комплексная единица, $\odot$ - операция поэлементного умножения.

Таким образом, для зашумленного сигнала Y, процесс улучшения качества записывался следующим образом:

$$\widehat{\mathrm{X}} = \mathrm{ISTFT}\left(\mathrm{STFT(Y)} \odot \mathrm{M}\big(\mathrm{STFT(Y)}\big)\right), \quad (5)$$

где STFT(.) и ISTFT(.) представляли собой прямое и обратное оконное преобразование Фурье, а M(.) – представляло собой преобразование, вычисляемое предлагаемой нейронной сетью.

3.1 Сеть пирамидального визуального трансформера PVT

Сеть PVT являлась альтернативой кодирующей компоненты сверточных сетей типа U-Net [21], позволяющей решать задачи классификации изображений, детектирования объектов и семантической сегментации. Она обеспечивала последовательное вычисление нескольких промежуточных представлений входного изображения. При этом использовалась прогрессивная стратегия сжатия представлений [16]. Они образовывали «пирамиду» представлений, содержащую все более общие признаки входного изображения.

Главной отличительной особенностью сети PVT по сравнению с U-Net-подобными сверточными сетями было использование для понижения размерности признакового представления преобразования нового типа. В этом преобразовании сначала проводилось разбиение входного представления F размера $\mathrm{H \times W}$ с числом каналов C на $\mathrm{HW/P^2}$ непересекающихся участков (патчей) за счет применения свертки с совпадающими по величине размером фильтра ($\mathrm{P \times P}$) и величиной сдвига (stride=P). Тем самым вычислялись закодированные представления патчей (patch embeddings) размера $(\mathrm{HW/P^2}) \times \mathrm{C'}$, где $\mathrm{C'}$

обозначено новое число каналов. После к этим представлениям для учета взаимного расположения патчей друг относительно друга добавлялись обучаемые вектора позиционного кодирования. Далее применялся трансформер [15, 16], содержащий L последовательных преобразований самовнимания (self-attention). Выход трансформера представлялся в виде нового сжатого признакового представления F' размера $\frac{\mathrm{H}}{\mathrm{P}} \times \frac{\mathrm{W}}{\mathrm{P}} \times \mathrm{C}'$.

Сеть PVT содержала 4 последовательных блока преобразований, описанных выше. На вход этой цепочки преобразований подавалось цветное трехканальное изображение. В результате вычислялось пирамидальное представление изображения $\{\mathrm{F}_1, \mathrm{F}_2, \mathrm{F}_3, \mathrm{F}_4\}$. Именно элементы этого представления, все или по отдельности, использовались для решения перечисленных выше задач классификации, детектирования и сегментации.

3.2 Модификация сети PVT для задачи улучшения качества речи

Концептуально задачи шумоочистки и сегментации изображений похожи, так как размеры входа и выхода нейронной сети должны совпадать. Содержательное отличие, которое следует учитывать, заключалось в том, что в изображении обе размерности (по ширине и по высоте изображения) равноправны, в то время как в частотно-временном представлении они имеют разный смысл (номер частотной полосы и номер временного окна). Учитывая указанные соображения, в архитектуру сети PVT были внесены некоторые изменения.

3.2.1 Частотное кодирование

Входом модифицированной сети являлась комплексная спектрограмма $\mathrm{Y}(\mathrm{f}, \mathrm{t})$, размера $F \times T \times 2$, где два канала отвечали за вещественную и мнимую компоненты. Перед вычислением пирамидального представления она конкатенировалась с матрицей частотного позиционного кодирования. Тем самым ко входному сигналу подмешивалась информация о том, какой частотной полосе соответствуют тот или иной участок спектрограммы. В качестве гипотезы предполагалось, что это могло помочь нейросетевой модели эффективно сопоставлять различные области спектра и поддерживать непрерывность голосовых формант при восстановлении речи.

Использовались K-мерные вектора кодирования частоты вида [6]:

$$\mathrm{p}(\mathrm{f}, \mathrm{k}) = \left[\cos\left(\frac{\pi \mathrm{f}}{\mathrm{F}}\right), \cos\left(\frac{2\pi \mathrm{f}}{\mathrm{F}}\right), \ldots, \cos\left(\frac{2^{K-1}\pi \mathrm{f}}{\mathrm{F}}\right)\right], \tag{6}$$

где F – это максимальная частота и $\mathrm{k} = 0, \ldots, K-1$ – индекс вектора кодирования. Они не зависели от времени и не являлись обучаемыми параметрам сети. Пример частотного кодирования размера K=10 для F=256 частотных полос представлен на рис. 1.

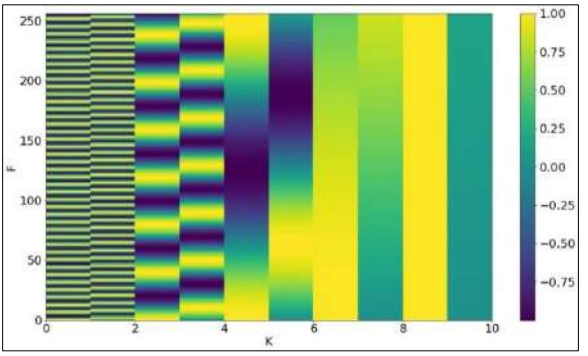


Рис. 1. Пример частотного кодирования с векторами размером K=10 для F=256 частотных полос
Fig. 1. An example of positional encoding with vectors of size K=10 for F=256 frequency bands

3.2.2 Разбиение на патчи и декодирование комплексной маски

На рис. 2 представлена схема предложенной нейронной сети с архитектурой «кодер-декодер». После применения входной свертки к спектрограммам с частотным кодированием они подавались на цепочку преобразований сжатия, обозначенных как B_1-B_4, являющихся по сути нейронной сетью PVT (выделены на рис. 2 зеленым). Были проведены несколько модификаций этих преобразований.

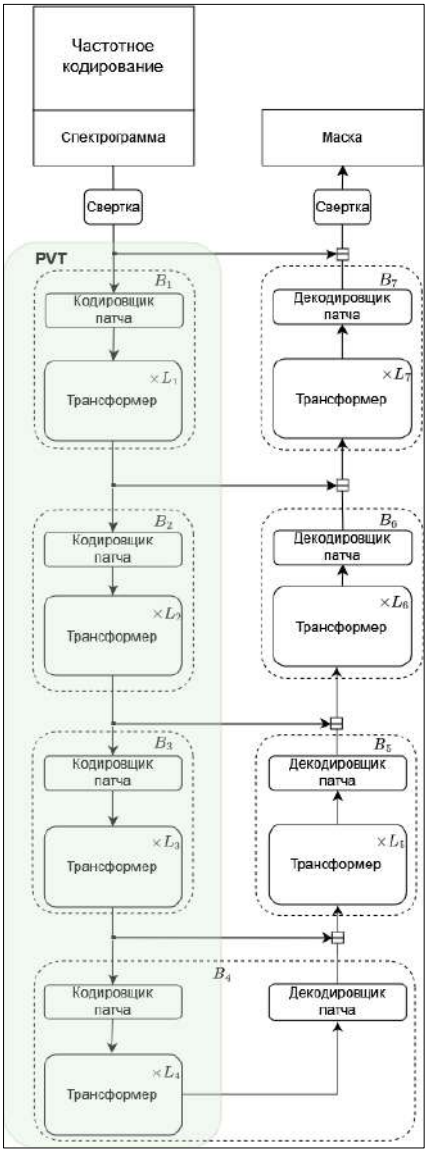


Рис. 2. Схема модифицированной нейронной сети для улучшения качества речи; зеленым выделена часть сети аналогичная пирамидальному кодировщику сети PVT.
Fig. 2. Scheme of the modified neural network for speech enhancement; the part of the network similar to the PVT network pyramidal encoder is highlighted in green.

1. Разбиение проводилось на прямоугольные патчи различных размеров. Требовалось учитывать больше информации о соседних частотных полосах сигнала и одновременно обеспечить достаточное сжатие входного частотно-временного представления для

ограничения числа параметров модели (размеры используемых патчей приведены ниже в табл. 1).

2. Последнее из преобразований сети PVT (B_4) было модифицировано. К нему был добавлен декодировщик патчей, который представлял собой транспонированную свертку, с последующей прибавкой обучаемого вектора позиционного кодирования. Следует отметить, что все вектора позиционного кодирования как кодировщике, так и в декодировщике были не зависимы друг от друга при обучении сети.

3. В отличие от задачи сегментации, был заменен используемый декодер. В работе [22] для развертывания пирамидального представления применялся декодер FPN, использующий последовательные сверточные слои. В данной работе был разработан оригинальный декодер, основанный на трансформер-преобразованиях, аналогичных применяемым в PVT. Для увеличения размера последовательных результатов в блоках B_5-B_7 использовалась упомянутая выше связка транспонированной свертки и позиционного кодирования. При этом на вход блоков B_5-B_7 подавался не только выход предыдущего блока, но его конкатенация с выходом соответствующего блока пирамидального преобразования PVT. Этот способ передачи дополнительной информации о представлениях сигнала разного масштаба был концептуально похож на применяемому в других кодирующих-декодирующих сетях.

3.3 Функции потерь

Обучение нейронных сетей для улучшения качества речи потребовало и подбора наиболее подходящей функции потерь. Входными данными для них являлись образцовый сигнал $\mathrm{X(t)}$ и улучшенный сигнал $\widehat{\mathrm{X}}(t)$, вычисленный согласно (4). Использовались отрицания нескольких существующих метрик оценки качества очистки от шума, так как минимизация значений этих метрик при обучении нейронной сети приводила бы к высокому достигнутому качеству работы. В численных экспериментах были апробированы следующие варианты функций потерь:

– отрицание отношения сигнал-шум (SNR):

$$\mathcal{L}(\widehat{\mathrm{X}},\mathrm{X}) = -\mathrm{SNR}(\widehat{\mathrm{X}},\mathrm{X}) = -10\log_{10}\left(\frac{\|\mathrm{X}\|^2}{\left\|\widehat{\mathrm{X}}-\mathrm{X}\right\|^2}\right); \quad (7)$$

– отрицание масштабно-инвариантного отношения сигнал-возмущение (SI-SDR) [23]:

$$\mathcal{L}(\widehat{\mathrm{X}},\mathrm{X}) = -\mathrm{SI\text{-}SDR}(\widehat{\mathrm{X}},\mathrm{X}) = -10\log_{10}\left(\frac{\|\alpha\mathrm{X}\|^2}{\left\|\widehat{\mathrm{X}}-\alpha\mathrm{X}\right\|^2}\right), \quad (8)$$

где $\alpha = \frac{\widehat{\mathrm{X}}^T X}{\|\mathrm{X}\|}$ – коэффициент масштабирования, корректирующий амплитуду чистого сигнала, для того чтобы откорректировать завышенную оценку отношения сигнал-шум в случае не ортогональных векторов X и $\widehat{\mathrm{X}} - \mathrm{X}$.

4. Набор данных

Для апробации предложенного подхода и предварительной оценки его эффективности использовался обучающий набор, построенный по методике соревнований DNS Challenge 2021 года [2] с образцами с частотой дискретизации 16 кГц. Образцы из соревнований DNS Challenge 2022 года [3] не использовались в данной работе по следующим причинам. Во-первых, в DNS Challenge 2022 были представлены существенно большие по размерам образцы с частотой дискретизации 48 кГц, что при доступных авторам вычислительных мощностях не позволяло эффективно проводить вычислительные эксперименты. Во-вторых, с точки зрения практического применения частотный диапазон 20 Гц-8 кГц,

соответствующий частоте дискретизации 16 кГц, достаточен для представления человеческой речи в ситуации обычного диалога. За пределы данной полосы частот голос может выходить только в отдельных специфических ситуациях пения или крика.

При построении обучающей выборки согласно [2] использовались три набора данных.

1) Образцы неискаженной речи из нескольких открытых наборов данных (LibriVox, M-AILABS и др.). Следует отметить, что часть англоязычных образцов была отсеяна из-за низкого качества записи. Для отбора образцов применялась методика ITU-T P.808 [24] субъективной оценки качества. Отсеивались образцы со средней субъективной оценкой MOS ниже 4,3 по 5-бальной шкале.
2) Образцы шума из подмножества, полученного на основе двух наборов данных Audioset и Freesound. Подмножество состояло из 60000 образцов, относящихся к 150 классам шумов. Каждый класс шумовых сигналов содержал не менее 500 образцов для балансировки выборки.
3) Набор, состоящий из 3076 измеренных и более 110 тысяч синтезированных импульсных характеристик помещений, выбранных из наборов данных openSLR26 и openSLR28.

Синтез зашумленных примеров обучающей выборки осуществлялся следующим образом. Случайно выбиралась неискаженная запись голоса и набор из нескольких случайно выбранных образцов шума. Шумовые записи масштабировались по амплитуде так, чтобы SNR суммы чистой и шумовой компонент соответствовал выбранному случайно значению, которые случайно выбирались из равномерного распределения на отрезке [0 дБ, 40 дБ].

Тестовая выборка состояла из набора образцов с соотношением сигнал-шум распределённом на отрезке от 0 дБ до 20 дБ. Эти образцы являлись частью набора данных DNS Challenge 2021 [2] и были синтезированы независимо организаторами этих соревнований. Все приведенные ниже оценки качества работы, в том числе сравнение с существующими альтернативными решениями, вычислялись на данном фиксированном наборе образцов.

5. Обучение модели

5.1 Особенности реализации

Предложенная нейронная сеть была реализована на языке программирования Python с использованием фреймворка глубокого обучения PyTorch. Обучение проводилась на рабочей станции с двумя видеокартах Nvidia GeForce 1080Ti. Общее время обучения лучшей модели составило 45 дней. Модель обучалась методом оптимизации Adam со скоростью обучения 1e-4 для всех проведённых экспериментов.

5.2 Гиперпараметры модифицированной нейронной сети

Длительность каждого зашумленного образца составляла 2 с, частота дискретизации равнялась 16 кГц. Каждый образец разбивался на временные окна 32 мс с перекрытием 16 мс. Для вычисления частотно-временного представления использовалось быстрое преобразование Фурье с окном Хэмминга. Размер входного тензора составлял $256 \times 128 \times 2$ (F=256, T=128). Для частотного кодирования использовался вектор длины K=10. Размер ядра начальной и конечной сверток составлял 3×5 со сдвигом 1.

В табл. 1 приведены значения всех гиперпараметров основной части нейронной сети. Использовались следующие обозначения (аналогичные обозначениям сети PVT [16]):

- P_i – размер патча в блоке кодирования B_i;
- $\widehat{P}_i$ – размер патча в блоке декодирования B_i;
- C_i – количество каналов на выходе блока B_i;
- L_i – количество слоев трансформера слоев в блоке B_i;

- R_i – коэффициент сжатия в слоях трансформера блока B_i;
- H_i – количество голов в слоях трансформера в блоке B_i;
- E_i – коэффициент расширения в трансформере в блоке B_i.

Табл. 1. Гиперпараметры кодирующей и декодирующей компонент модифицированной сети PVT
Table. 1. Hyperparameters of the encoder and the decoder of the modified PVT network

Стадия	Номер блока	Размер выхода	Тип слоя (преобразование)	Гиперпараметры
Кодирование	1	$\frac{F}{8}\times\frac{T}{2}$	Кодировщик патча	$P_1 = (8,2);\ C_1 = 64$
			Трансформер	$E_1 = 8,\ H_1 = 1;\ R_1 = 8,\ L_1 = 2$
	2	$\frac{F}{16}\times\frac{T}{4}$	Кодировщик патча	$P_2 = (2,2);\ C_2 = 128$
			Трансформер	$E_2 = 8,\ H_2 = 2;\ R_2 = 4,\ L_2 = 2$
	3	$\frac{F}{32}\times\frac{T}{4}$	Кодировщик патча	$P_3 = (2,1);\ C_3 = 320$
			Трансформер	$E_3 = 4,\ H_3 = 5;\ R_3 = 2,\ L_3 = 2$
	4	$\frac{F}{32}\times\frac{T}{4}$	Кодировщик патча	$P_4 = (1,1);\ C_4 = 512$
			Трансформер	$E_4 = 4,\ H_4 = 8;\ R_4 = 1, L_4 = 2$
			Декодировщик патча	$\hat{P}_4 = (1,1);\ C_4 = 320$
Декодирование	5	$\frac{F}{16}\times\frac{T}{4}$	Трансформер	$E_5 = 4,\ H_5 = 4;\ R_5 = 2,\ L_5 = 2$
			Декодировщик патча	$\hat{P}_5 = (2,1);\ C_5 = 128$
	6	$\frac{F}{8}\times\frac{T}{2}$	Трансформер	$E_6 = 8,\ H_6 = 2;\ R_6 = 4,\ L_6 = 2$
			Декодировщик патча	$\hat{P}_6 = (2,2);\ C_6 = 64$
	7	$F \times T$	Трансформер	$E_7 = 8,\ H_7 = 1;\ R_7 = 8,\ L_7 = 2$
			Декодировщик патча	$\hat{P}_7 = (8,2);\ C_7 = 16$

6. Результаты и обсуждение

6.1 Используемые метрики оценки качества

В большинстве современных работ [6-13] для оценки качества методов улучшения речи не применялось отношение сигнал-шум, так как оно не отражает ни реальную разборчивость, ни воспринимаемое слушателем субъективное качество восстановленной речи. Поэтому в данной работе использовались следующие методы оценки качества.

1) Perceptual Evaluation of Speech Quality (PESQ) [24] – алгоритм автоматической оценки качества речевых записей, моделирующий субъективную оценку MOS по непрерывной шкале от -0,5 (наихудшее качество) до 4,5 (наилучшее качество). Выделялись два основных вида PESQ-метрики: узкополосная NB-PESQ для сигналов с частотой дискретизации 8 кГц, и широкополосная WB-PESQ для сигналов с частотой дискретизации 16 кГц;
2) Short-Time Objective Intelligibility Measure (STOI) [25] – алгоритм автоматической оценки разборчивости речи в диапазоне от 0 (наихудшее качество) до 1 (наилучшее качество). Оценка разборчивости в нем производилась с использованием сравнения спектральных характеристик чистого и очищенного голосовых сигналов;

3) Scale-Invariant Signal Distortion Rate (SI-SDR) [23] – устойчивая к различиям в громкости речевых сигналов модификация соотношения сигнал-шум.

Все эти методы в качестве входных данных требовали пары речевых сигналов, состоящие из искаженных и соответствующих очищенных образцов.

6.2 Выбор функции потерь

Для выбора лучшей функции потерь (раздел 3.3) использовалась специально синтезированная подвыборка набора данных DNS Challenge 2021 [2]. Она состояла из 125 часов чистой речи, добавление шума проводилось аналогично полной выборке данных. Количество эпох обучения равнялось 12. Результаты представлены в табл. 2. Видно, что самый стабильный результат на всех метриках показала модель, которая обучалась с функцией потерь -SI-SDR. Использование SNR в качестве функции потерь существенно снижает качество работы как для чистых образцов, так и для образцов с реверберацией, причем для последних относительный разрыв больше, что, вероятно, связано с наличием в образцовых «чистых» сигналах существенных искажений.

Табл. 2. Сравнение качества работы для двух функций потерь
Table. 2. Performance comparison for two loss functions

Функция потерь	**С реверберацией**				**Без реверберации**			
	STOI	**SI-SDR**	**NB-PESQ**	**WB-PESQ**	**STOI**	**SI-SDR**	**NB-PESQ**	**WB-PESQ**
-SI-SDR	0,9295	13,43	2,764	2,028	0,8892	13,1	3,014	2,333
-SNR	0,9188	10,73	2,554	1,887	0,873	10,43	2,777	2,171

6.3 Использование частотного кодирования

Экспериментально была проверена необходимость использования частого кодирования (раздел 3.2.2). Для проверки был использован полный набор DNS Challenge 2021. В модели в качестве функции потерь было применено отрицание SI-SDR. Количество эпох обучения равнялось 53. Как видно из табл. 3, использование частотного кодирования существенно улучшает метрики качества.

Табл. 3. Сравнение качества работы при включении/выключении частотного кодирования
Table. 3 Performance comparison of models with and without frequency encoding.

Использование частотного кодирования	**С реверберацией**				**Без реверберации**			
	STOI	**SI-SDR**	**NB-PESQ**	**WB-PESQ**	**STOI**	**SI-SDR**	**NB-PESQ**	**WB-PESQ**
Нет	0,9305	17,05	3,489	2,992	0,9618	18,52	3,252	2,73
Да	0,9362	17,51	3,561	3,11	0,9658	19,06	3,347	2,811

6.4 Сравнение с существующими решениями

Лучшей нейросетевой моделью в серии экспериментов оказалась версия модифицированной нейронной сети с включенным частотным кодированием. Использовалась функция потерь – SI-SDR. Обучение лучшей модели осуществлялось в течение 121 эпохи на полном наборе данных DNS Challenge 2021.

Предложенная модель тестировалась и сравнивалась на тестовой выборке синтетического набора данных соревнования DNS Challenge 2021 (табл. 4). В первой строке табл. 4 приведены значения метрик для необработанных зашумленных образцов тестовой выборки. Ниже приведены оценки метрик качества сетей PoCoNet [6], FullSubNet [7], предыдущей модели TS-LSTM [26], разработанной авторами данной работы, и предложенной модели. Модель PoCoNet использовала смешанную архитектуру кодировщик-декодировщик,

сочетающую сверточные слои с блоками самовнимания. Сети FullSubNet и TS-LSTM использовали рекуррентные слои. Показано, что предложенная модель превосходит существующие аналоги по всем метрикам.

Табл. 4. Сравнение качества работы предложенной модели с альтернативными решениями
Table. 4 Performance comparison of the proposed model with alternative methods.

Модель	С реверберацией				Без реверберации			
	STOI	SI-SDR	NB-PESQ	WB-PESQ	STOI	SI-SDR	NB-PESQ	WB-PESQ
Без обработки	0,8662	9,030	2,753	16822	0,9152	9,07	2,454	1,582
PoCoNet	-	-	-	2,832	-	-	-	2,748
FullSubNet	0,9262	15,750	3,473	2,969	0,9611	17,29	3,305	2,777
TS-LSTM	0,8600	9,376	3,050	2,220	0,9605	17,58	3,338	2,832
Предложенная модель	**0,9402**	**17,67**	**3,592**	**3,176**	**0,9692**	**19,53**	**3,398**	**2,899**

6.5 Особенности процесса обучения предложенной нейронной сети

При разработке новых нейросетевых методов интерес представляет интерпретация того, как идет обучение модели, на каких этапах этого процесса происходит качественное изменение результатов работы нейронной сети. Это имеет как научную, так и практическую значимость, так как может позволить лучше понять каким образом строить процесс обучения нейронных сетей для улучшения речи и, в более широком смысле, может помочь при создании новых подходов к обработке речевых сигналов.

В табл. 5 вынесены усредненные метрики качества работы обучаемой нейросети на тестовой выборке на 4, 21, 53 и 121 эпохах. Видно, что количественные оценки качества меняются все медленнее по мере того, как идет процесс обучения, и, как показывают приведенные ниже примеры, это не связано напрямую с качественным изменением поведения обучаемой сети.

Табл. 5. Сравнение качества работы предложенной модели на промежуточных эпохах обучения
Table. 5 Performance comparison of the proposed model at intermediate training epochs

Номер эпохи	С реверберацией				Без реверберации			
	STOI	SI-SDR	NB-PESQ	WB-PESQ	STOI	SI-SDR	NB-PESQ	WB-PESQ
4	0,913	15,48	3,294	2,687	0,9492	16,65	3,062	2,399
21	0,9281	15,48	3,464	2,964	0,9661	18,9	3,334	2,806
53	0,9362	17,51	3,561	3,11	0,9658	19,06	3,347	2,811
121	0,9402	17,67	3,592	3,176	0,9692	19,53	3,398	2,899

На рис. 3 представлены спектрограммы для речевого сигнала с шумом из класса «пение птиц». Участки спектра непосредственно со звуками птиц выделены белыми прямоугольниками. Также в паузах видны добавки широкополосного шума (в диапазоне частот до 4,5-5 кГц). Уже после 4 эпохи обучения (рис. 3в) они были почти полностью вычищены сетью, однако искажения от пения птиц остались. После 21 эпохи (рис. 3г) эти искажения практически полностью исчезли, а на 54 эпохе (рис. 3д) сеть научилась достраивать на этом месте форманты полезного речевого сигнала, формируя их из шума.

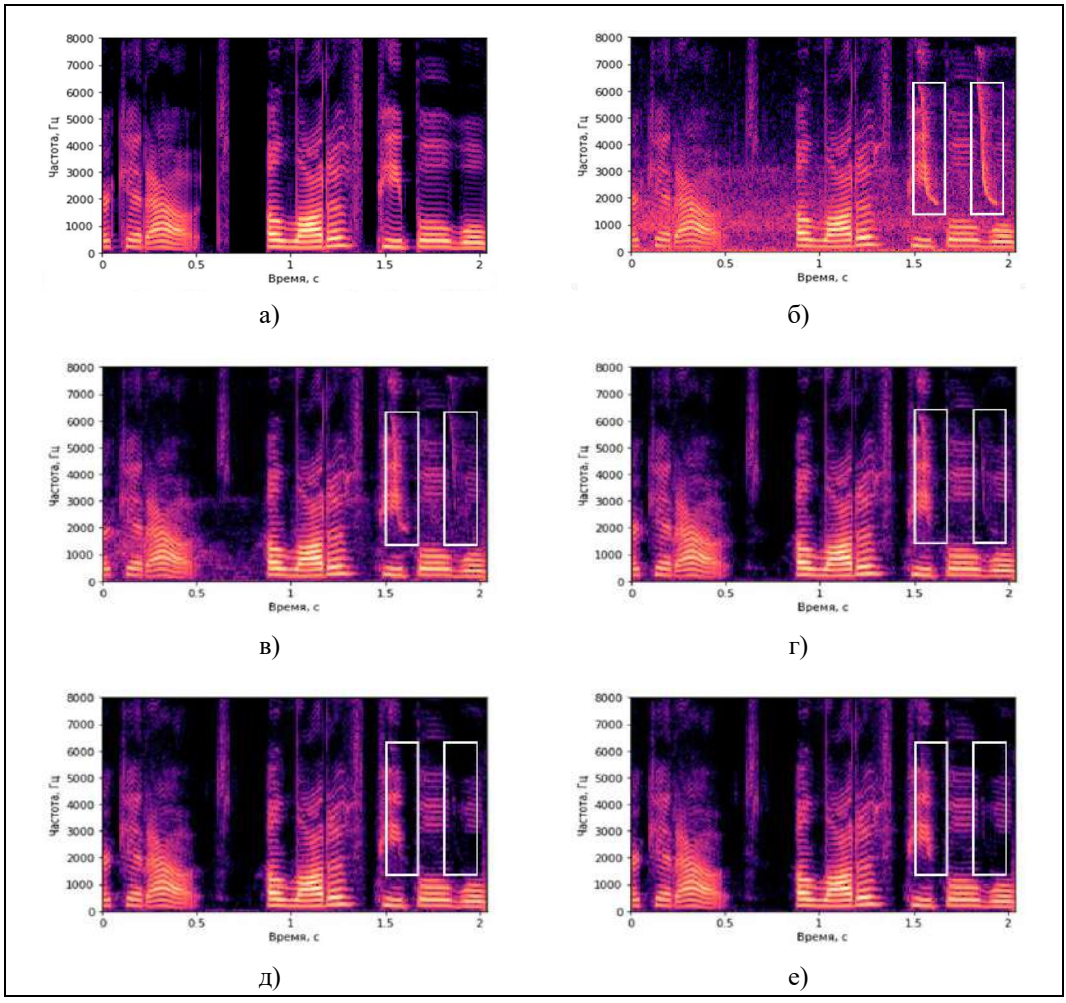


*Рис. 3. Пример спектрограмм чистого речевого сигнала **без реверберации** (а), с широкополосным шумом из класса **«пение птиц»** (б) и результатов работы модели после 4 (в), 21 (г), 53 (д) и 121 (е) эпох обучения*

*Fig. 3. Example of spectrograms of a clean speech signal **without reverberation** (a), with wideband noise from the "**birdsong**" class (b) and the results of the model after 4 (c), 21 (d), 53 (e) and 121 (f) epochs learning*

В случае того же исходного чистого сигнала, но с накладываемой реверберацией (рис. 4), видно, что сеть убирала искажения как от широкополосного шума, так и от птичьего пения уже с 4 эпохи (рис. 4в).

Дальнейшее обучение сети (рис. 4г-е) не существенно повлияло на вид спектра восстанавливаемого речевого сигнала. Это легко объясняется тем, что спектр реверберированной речи имеет меньше мелких деталей, формантные полосы более размыты.

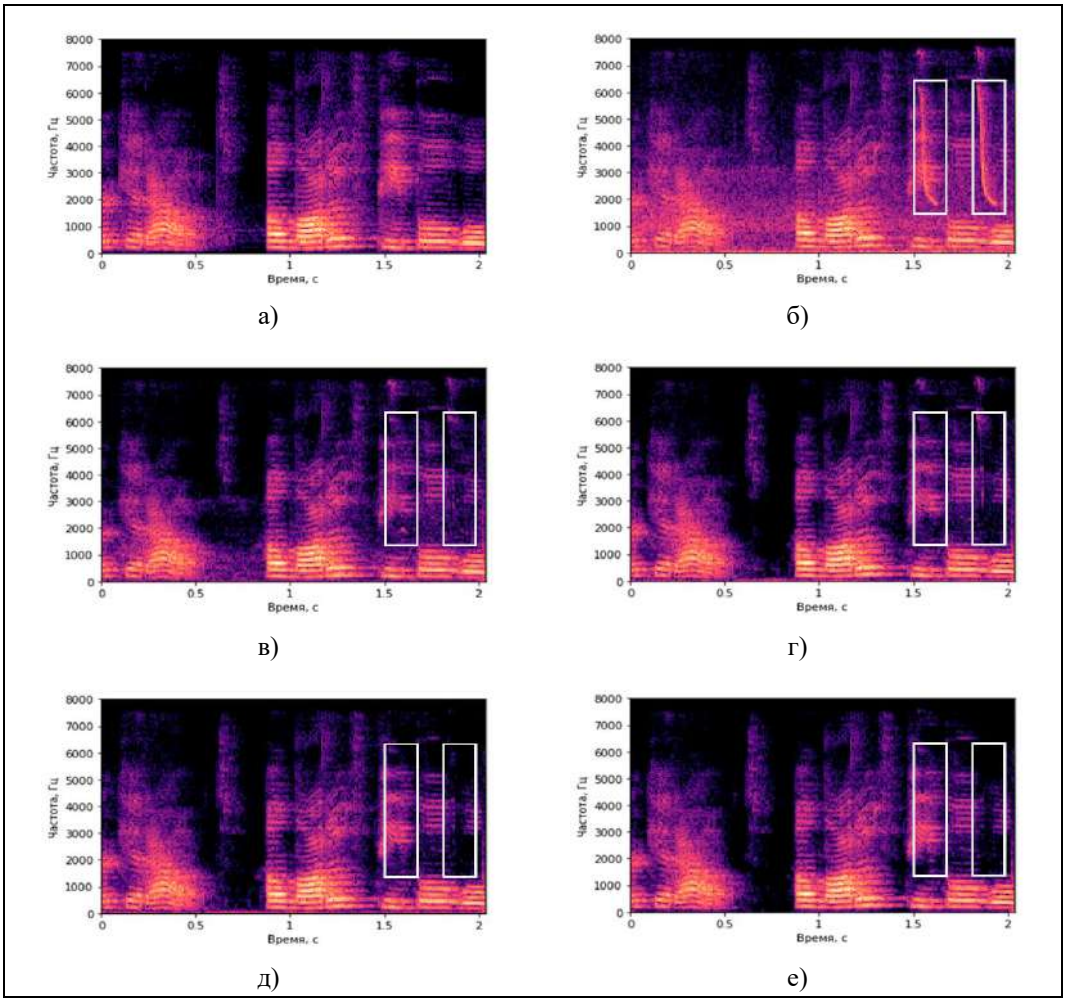


Рис. 4. Пример спектрограмм чистого речевого сигнала ***с реверберацией*** *(а), с широкополосным шумом из класса* ***«пение птиц»*** *(б) и результатов работы модели после 4 (в), 21 (г), 53 (д) и 121 (е) эпох обучения*

Fig. 4. Example of spectrograms of a clean speech signal ***with reverberation*** *(a), with wideband noise from the "**birdsong**" class (b) and the results of the model after 4 (c), 21 (d), 53 (e) and 121 (f) epochs learning*

На рис. 5 представлены спектрограммы сигнала без реверберации с шумом класса «музыка» (звучащий на фоне музыкальный инструмент типа синтезатора). Шумовое искажение сосредоточено в узкой полосе частот, выделенной на рис. 5 белым прямоугольником. Начиная с 4 эпохи обучения (рис. 5в) сеть устранила значительную часть искажения, а в течение последующих эпох (рис. 5г-е), училась восстанавливать недостающие части спектральных компонент. Интересно отметить, что в области, выделенной зеленым прямоугольником, сеть сначала (рис. 5в) «воспринимала» часть шума на 1,5 кГц за форманту речевого сигнала и пыталась продолжить её, однако после следующих эпох обучения она устранила этот артефакт.

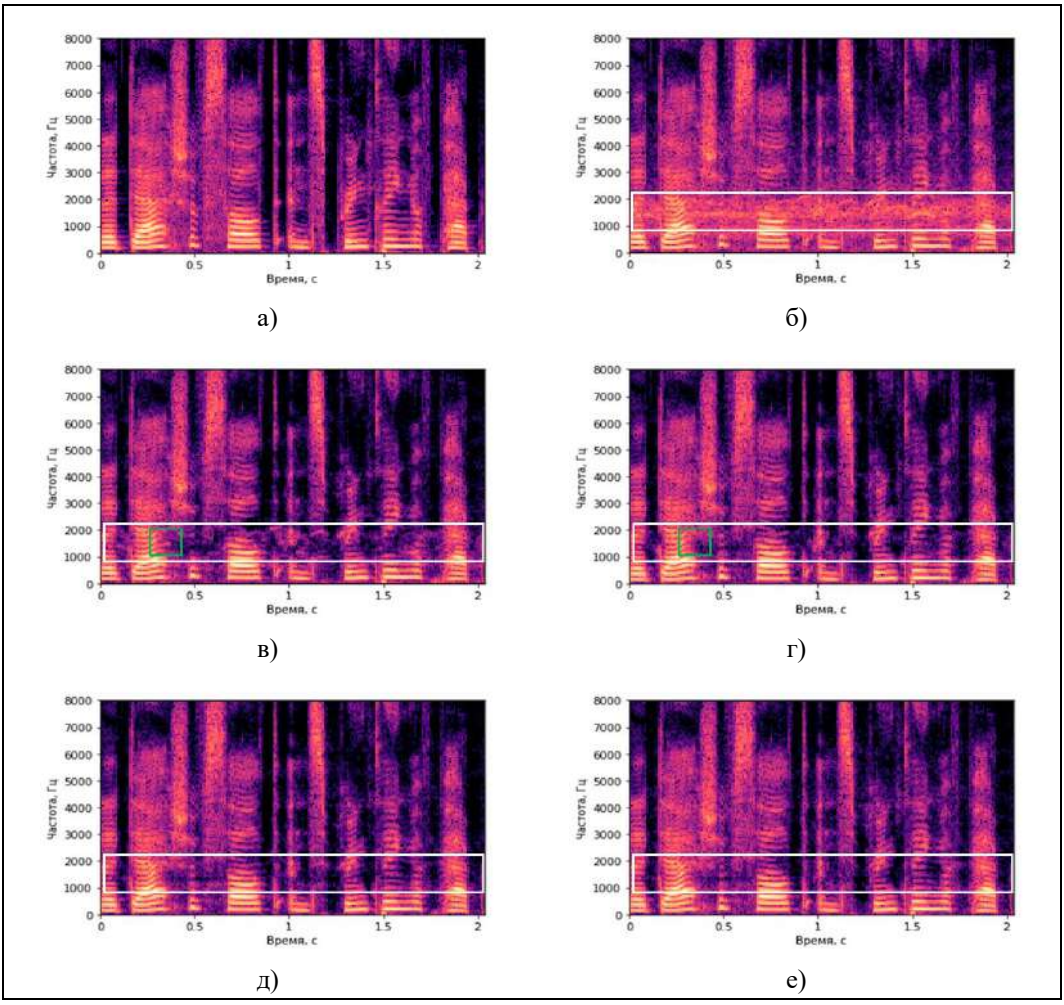


*Рис. 5. Пример спектрограмм чистого речевого сигнала **без реверберации** (а), с узкополосным шумом из класса **«музыка»** (б) и результатов работы модели после 4 (в), 21 (г), 53 (д) и 121 (е) эпох обучения*
*Fig. 5. Example of spectrograms of a clean speech signal **without reverberation** (a), with narrowband noise from the **"music"** class (b) and the results of the model after 4 (c), 21 (d), 53 (e) and 121 (f) training epochs*

На рис. 6 представлены спектрограммы для сигнала с искажением из класса «шум моря». В данном сигнале присутствовало широкополосное зашумление и звуки морских птиц, выделенные белым прямоугольником. Сигнал в процессе обучения последовательно улучшался без резких изменений до 53 эпохи (рис. 6д). В отличие от предыдущих примеров, более существенные изменения в работе обучаемой сети, пришлись как раз на поздний период между 53 (рис. 6д) и 121 (рис. 6е) эпохами, что совершенно не нашло отражения в средних количественных оценках качества из табл. 5.

Таким образом, в процессе обучения нейронная сеть довольно быстро (за несколько первых эпох) училась выделению и удалению шумовых искажений, протяженных как в частотной (шириной порядка килогерц), так и во временной (от десятых долей секунды и больше) областях. При этом восстановление деталей полезного сигнала было не точным (рис. 5в). Дальнейшее обучение привело к тому, что сеть стала не только удалять шумы, но и правильно достраивать отдельные форманты восстанавливаемой речи. Чем выше мощность или

сложнее структура аддитивного шума, тем больше эпох обучения требовалось для эффективной работы.

Слабое относительное изменение усредненных метрик (табл. 5) на поздних эпохах без приведенного выше анализа спектрограмм могло интерпретироваться как достижение сходимости параметров сети, что привело бы к ранней остановке процесса обучения, и, как следствие, недостаточному качеству восстановления речевых сигналов со «сложной» шумовой добавкой (рис. 6). Поэтому возможной рекомендацией при обучении нейросетевых методов шумоочистки является необходимость дополнительного выборочного контроля работы обучаемой нейронной сети на отдельных образцах с качественно разными классами шумов.

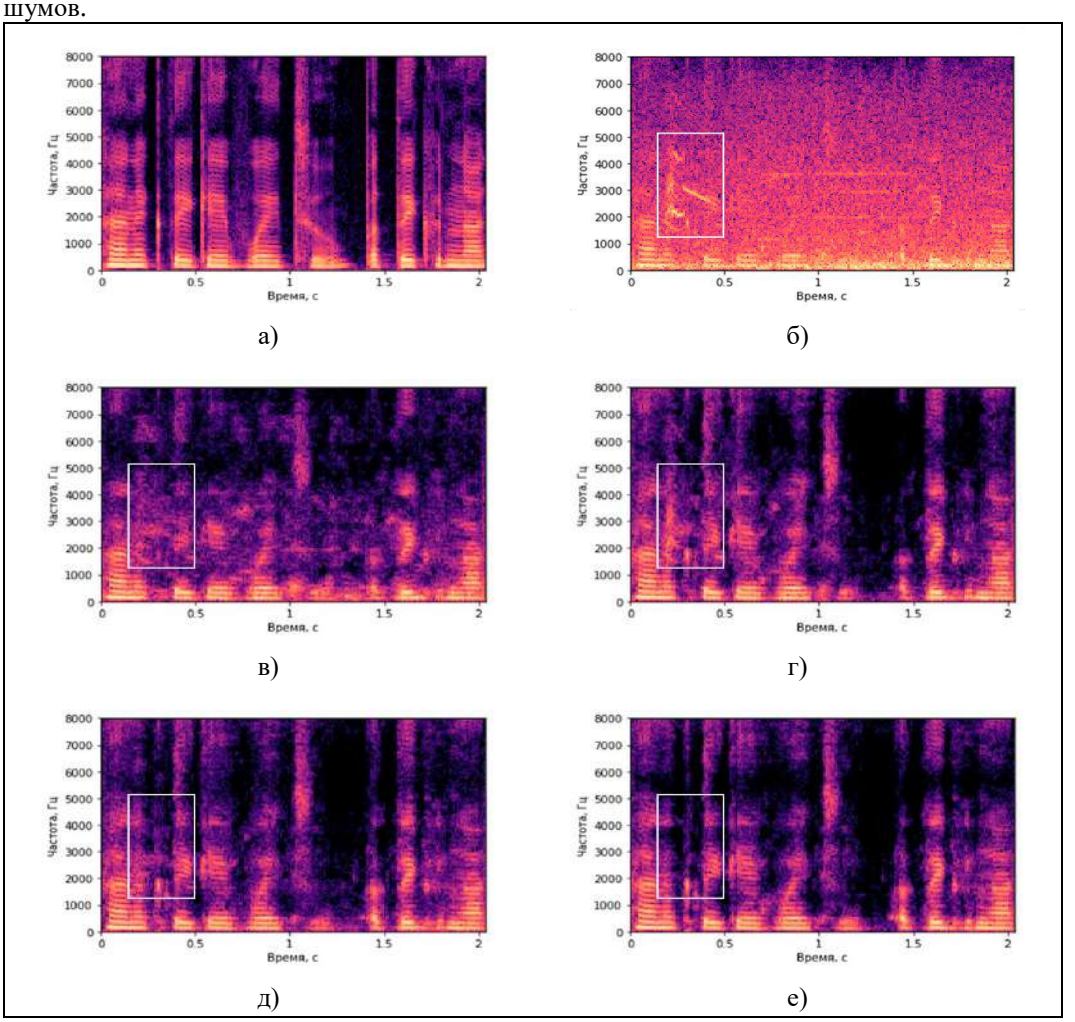


*Рис. 6. Пример спектрограмм чистого речевого сигнала **без реверберации** (а), с широкополосным шумом из класса **«шум моря»** (б) и результатов работы модели после 4 (в), 21 (г), 53 (д) и 121 (е) эпох обучения*

*Fig. 6. Example of spectrograms of a clean speech signal **without reverberation** (a), with wideband noise from the " sound of the sea" class (b) and the results of the model after 4 (c), 21 (d), 53 (e) and 121 (f) training epochs*

7. Заключение

В данной работе разработан новый тип кодирующей-декодирующей нейронной сети для решения задачи улучшения качества речи. Он основан на использовании модифицированной сети пирамидального трансформера и оригинальном методе декодирования иерархического представления обрабатываемого аудиосигнала.

Продемонстрирована важность добавления векторов частотного кодирования при формировании входных данных и проведен выбор наиболее подходящей функции потерь для наилучшего достижимого качества работы предложенного подхода.

Сравнение по метрикам качества показало, что разработанная нейронная сеть превосходит существующие современные решения. Анализ процесса обучения сети продемонстрировал постепенное совершенствование используемого сетью способа улучшения речевых сигналов – от маскирования шумовых областей в спектрограмме до восстановления искаженных формантных полос.

Разработанная нейросетевая модель может найти свое применение как непосредственно в прикладных приложениях, так и в качестве «учителя» при использовании метода дистилляции для обучения быстрых нейросетевых моделей-«учеников».

Список литературы / References

[1] Loizou P. Speech Enhancement. Theory and Practice, 2nd Edition. CRC Press, 2017. 711 p.
[2] Reddy C., Dubey H., Koishida K. Interspeech 2021 Deep Noise Suppression Challenge. arXiv preprint arXiv: 2101.01902, 2021, 5 p.
[3] Dubey H., Gopal V., Cutler R. et al. ICASSP 2022 Deep Noise Suppression Challenge. DNS C, 2022. arXiv preprint arXiv: 2202.13288, 2022. 5 p.
[4] Borgström B.J., Brandstein M.S., Dunn R.B. Improving Statistical Model-Based Speech Enhancement with Deep Neural Networks. In Proc. of the 16th International Workshop on Acoustic Signal Enhancement (IWAENC), 2018, pp. 471-475.
[5] Zheng C., Liu W. et al. Low-latency Monaural Speech Enhancement with Deep Filter-bank Equalizer. The Journal of the Acoustical Society of America, vol. 151, issue 5, 2021, article no. 3291.
[6] Umut I., Giri R. et al. PoCoNet: better speech enhancement with frequency-positional embeddings, semi-supervised conversational data, and biased loss. In Proc. of the INTERSPEECH 2020, 2020, pp. 2487-2491.
[7] Hao X., Su X. et al. FullSubNet: a full-band and sub-band fusion model for real-time single-channel speech enhancement. In Proc. of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2021, pp. 6633-6637.
[8] Xu, R., Wu, R. et al. Listening to sounds of silence for speech denoising. In Proc. of the Conference on Neural Information Processing Systems (NeurIPS 2020), 2020, pp. 9633-9648.
[9] Zheng C., Peng X., Zhang Y. Interactive Speech and Noise Modeling for Speech Enhancement. arXiv preprint arXiv: 2105.05537, 2020, 9 p.
[10] Luo, Y., Mesgarani, N. Conv-TasNet: surpassing ideal time-frequency magnitude masking for speech separation. IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 27, issue 8, 2019, pp. 1256-1266.
[11] Liu Y., Zhang H. et al. Supervised speech enhancement with real spectrum approximation. In Proc. of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2019, pp. 5746-5750.
[12] Williamson D.S., Wang Y., Wang D. Complex ratio masking for monaural speech separation. IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 24, issue 3, 2016, pp. 483-492.
[13] Tan K., Wang D.A. Convolutional recurrent neural network for real-time speech enhancement. 2018, 2018, pp. 3229-3233.
[14] Tallec C., Ollivier Y. Can recurrent neural networks warp time? In Proc. of the International Conference on Learning Representation, 2018, pp. 1-13.
[15] Vaswani A., Shazeer N. et al. Attention is all you need. In Proc. of the Conference on Neural Information Processing Systems (NIPS 2017), 2017, 11 p.

[16] Wang W., Xie E. et al. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In Proc. of the IEEE/CVF International Conference on Computer Vision (ICCV), 2021, , pp. 548-558.
[17] Cao H., Wang Y. et al. Swin-unet: Unet-like pure transformer for medical image segmentation. arXiv preprint arXiv: 2105.05537, 2020. 14 c. DOI: 10.48550/arXiv.2105.05537
[18] Kun Wei, K., Guo P., Jiang N. Improving Transformer-based Conversational ASR by Inter-Sentential Attention Mechanism, arXiv preprint arXiv: 2207.00883, 2022, 5 p.
[19] Wang Y., Shen G. et al. Learning Mutual Correlation in Multimodal Transformer for Speech Emotion Recognition. In Proc. of the INTERSPEECH 2021, 2021, pp. 4518-4522.
[20] Wu, C., Xiu, Z. et al. Transformer-based acoustic modeling for streaming speech synthesis. c INTERSPEECH 2021, 2021, pp. 146-150.
[21] Ronneberger O., Fischer P., Brox T. U-Net: convolutional networks for biomedical image segmentation. Lecture Notes in Computer Science, vol. 9351, 2015, pp. 234-241.
[22] Lin T, Dollár P., Girshick R. Feature Pyramid Networks for Object Detection. In Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 936-944.
[23] Roux J.L., Wisdom S. et al. SDR – Half-baked or well done? In Proc. of the IEEE International Conference on Acoustics, Speech and Signal Processing, 2019, pp. 626-630.
[24] Naderi B., Cutler R. An opensource implementation of ITU-T recommendation P.808 with validation. In Proc. of the INTERSPEECH 2020, 2020, pp. 2862-2866.
[25] Taal C.H., Hendriks R.C. et al. A short-time objective intelligibility measure for time-frequency weighted noisy speech. In Proc. of the IIEEE International Conference on Acoustics, Speech and Signal Processing, 2010, pp. 4214-4217.
[26] Nasretdinov R., Ilyashenko I., Lependin A. Two-Stage Method of Speech Denoising by Long Short-Term Memory Neural Network. Communications in Computer and Information Science, vol 1526, 2021, pp. 86-97.

Информация об авторах / Information about authors

Андрей Александрович ЛЕПЕНДИН – кандидат физико-математических наук, доцент кафедры информационной безопасности Института цифровых технологий, электроники и физики Алтайского государственного университета. Сфера научных интересов: цифровая обработка сигналов, машинное обучение, глубокие нейронные сети, голосовая биометрия, улучшение качества речи.

Andrey Aleksandrovich LEPENDIN – candidate of science in physics and mathematics, associate professor of the Department of information security of the Institute of Digital Technology, Electronics and Physics of Altai State University. Research interests: digital signal processing, machine learning, deep neural nets, voice biometry, speech enhancement.

Рауф Салаватович НАСРЕТДИНОВ – аспирант кафедры информационной безопасности Института цифровых технологий, электроники и физики Алтайского государственного университета. Сфера научных интересов: машинное обучение, глубокие нейронные сети, автоматическое распознавание речи, улучшение качества речи.

Rauf Salavatovich NASRETDINOV – post-graduate student of the Department of information security of the Institute of Digital Technology, Electronics and Physics of Altai State University. Research interests: machine learning, deep neural nets, automatic speech recognition, speech enhancement.

Илья Дмитриевич ИЛЬЯШЕНКО – аспирант кафедры информационной безопасности Института цифровых технологий, электроники и физики Алтайского государственного университета. Сфера научных интересов: машинное обучение, глубокие нейронные сети, биометрические методы верификации, улучшение качества речи.

Ilya Dmitrievich ILYASHENKO – post-graduate student of the Department of information security of the Institute of Digital Technology, Electronics and Physics of Altai State University. Research interests: machine learning, deep neural nets, biometric verification, speech enhancement.

DOI: 10.15514/ISPRAS-2022-34(4)-11

Экспериментальная оценка алгоритма маркирования текстовых документов на основе изменения интервалов между словами

[1] *А.В. Козачок, ORCID: 0000-0002-6501-2008 <a.kozachok@academ.msk.rsnet.ru>*
[2] *В.И. Козачок, ORCID: 0000-0001-5384-2269 <v.kozachok@academ.msk.rsnet.ru>*
[2] *С.А. Копылов, ORCID: 0000-0003-2841-5243 <gremlin.kop@mail.ru>*
[2] *П.Н. Горбачев, ORCID: 0000-0002-4511-0348 <png@academ.msk.rsnet.ru>*
[1] *Ю.В. Маркин, ORCID: 0000-0003-1145-5118 <ustas@ispras.ru>*
[1] *Д.О. Обыденков, ORCID: 0000-0002-9296-6333 <obydenkov@ispras.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Академия Федеральной службы охраны Российской Федерации, 302015, Россия, г. Орёл, ул. Приборостроительная, д. 35*

Аннотация. В статье представлены результаты экспериментальной оценки параметров алгоритма маркирования электронных документов, основанного на изменении интервалов между словами. Разработанный алгоритм маркирования предназначен для повышения защищенности электронных документов, содержащих текстовую информацию, от утечки по каналам, обусловленным печатью, сканированием или фотографированием и последующей отправкой сформированного изображения. В качестве анализируемых параметров алгоритма выступают такие характеристики как емкость встраивания, невидимость, необнаруживаемость, извлекаемость и робастность. В ходе оценки емкости встраивания разработанного алгоритма приведены аналитические выражения, позволяющие рассчитать величину предельно достижимой емкости встраивания. Полученные количественные оценки и проведенные эксперименты позволили обосновать выбор допустимых значений встраиваемого маркера. Для определения невидимости встраиваемой информации в исходный документ осуществлена оценка невидимости и необнаруживаемости встроенного маркера. В ходе проведения экспертной оценки обоснована невидимость разработанного алгоритма к визуальному анализу, а также отсутствие значительных статистических отклонений в распределении анализируемых параметров в процессе оценки стойкости разработанного алгоритма маркирования к потенциально наилучшему методу стеганографического анализа. Получены количественные значения извлекаемости разработанного алгоритма маркирования посредством оценки точности извлечения. Проведенный анализ показал высокие значения точности извлечения маркера из отсканированных изображений, позволяющие гарантированно извлекать встроенные данные, а также направления совершенствования извлекаемости маркера из сфотографированных изображений. В процессе оценки устойчивости разработанного алгоритма маркирования к осуществлению преобразований и внесению искажений определены основные параметры робастности разработанного алгоритма маркирования к процессам печати, сканирования и фотографирования. Сформулированы выводы о возможности применения разработанного алгоритма маркирования и направления дальнейших исследований.

Ключевые слова: защита от утечки информации; маркирование; распознавание образов; обработка изображений; стеганографический анализ

Для цитирования: Козачок А.В., Козачок В.И., Копылов С.А., Горбачев П.Н., Маркин Ю.В., Обыденков Д.О. Экспериментальная оценка алгоритма маркирования текстовых документов на основе изменении интервала между словами. Труды ИСП РАН, том 34, вып. 4, 2022 г., стр. 153-172. 10.15514/ISPRAS-2022-34(4)-11

Experimental evaluation of the text documents marking algorithm based on interword distances shifting

[1] *A.V. Kozachok, ORCID: 0000-0002-6501-2008<a.kozachok@academ.msk.rsnet.ru>*
[2] *V.I. Kozachok, ORCID: 0000-0001-5384-2269 <v.kozachok@academ.msk.rsnet.ru>*
[2] *S.A. Kopylov, ORCID: 0000-0003-2841-5243 <gremlin.kop@mail.ru>*
[2] *P.N. Gorbachev, ORCID: 0000-0002-4511-0348 <png@academ.msk.rsnet.ru>*
[1] *Y.V. Markin, ORCID: 0000-0003-1145-5118 <ustas@ispras.ru>*
[1] *D.O. Obydenkov, ORCID: 0000-0002-9296-6333 <obydenkov@ispras.ru>*

[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia
[2] *Academy of Federal Guard Service,*
35, Priborostroitel'naya st., Orel, 302015, Russia

Abstract. The article presents the experimental parameter evaluation results of the electronic documents marking algorithm, based on interword distances shifting. The developed marking algorithm is designed to increase the security of electronic documents containing textual information from leakage through channels caused by printing, scanning or photographing, followed by sending the generated image. The algorithm analyzed parameters are such characteristics as embedding capacity, invisibility, undetectability, extractability and robustness. In the course of embedding capacity estimation of the developed algorithm, analytical expressions are given that make it possible to calculate the maximum achievable embedding capacity value. The obtained quantitative estimates and the experiments carried out made it possible to substantiate the admissible values choice of the embedded marker. To determine the embedded information invisibility in the source document, an invisibility and undetectability assessment of the embedded marker was carried out. During the expert evaluation, the developed algorithm invisibility to visual analysis was substantiated, as well as the absence of significant statistical deviations in the distribution of the analyzed parameters in the process of assessing the resistance of the developed marking algorithm to the potentially best steganographic analysis method. The quantitative extractability of the developed marking algorithm was carried out by assessing the extraction accuracy. The analysis performed showed accuracy high values of marker extraction from scanned images, which makes it possible to reliably extract embedded data, as well as determine directions for improving the extraction accuracy from photographed images. In the assessing process the stability of the developed marking algorithm to the transformations implementation and distortions introduction, the main robustness parameters of the developed marking algorithm to the printing, scanning and photographing processes are determined. Conclusions are formulated on the using possibility the developed marking algorithm and directions for further researches are identified.

Keywords: information leakage protection; marking; pattern recognition; image processing; steganographic analysis

For citation: Kozachok A.V., Kozachok V.I., Kopylov S.A., Gorbachev P.N., Markin Y.V., Obydenkov D.O. Experimental evaluation of the text documents marking algorithm based on interword distances shifting. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 4, 2022. pp. 153-172 (in Russian). DOI: 10.15514/ISPRAS-2022-34(4)-11

1. Введение

В последнее десятилетие задача по обеспечению безопасности конфиденциальной информации и данных стала одной из наиболее актуальных проблем в области информационной безопасности. С ростом количества информации и данных, обрабатываемых и циркулирующих в информационно-телекоммуникационных сетях, возросло и количество инцидентов информационной безопасности. Основной вектор нарушений в области информационной безопасности приходится на действия внешних нарушителей. Так, в 2021 году злоумышленниками осуществлено 2418 компьютерных атак на информационные ресурсы частных лиц (349 случаев) и информационно-коммуникационные сети госучреждений (322 случая) [1]. Несмотря на значительное

превосходство по количеству (1729 против 2418 случаев), утечки конфиденциальной информации, реализованные внутренними нарушителями, привели к компрометации 8,42 млрд. записей конфиденциальной информации и данных [2]. Высокие значения скомпрометированной информации позволяют отнести задачу по обеспечению защиты информации от утечки к наиболее актуальным направлениям исследований.

В результате анализа каналов утечки установлено, что наибольшее число случаев нарушений информационной безопасности связано с отправкой конфиденциальной информации по сети и электронной почте (89,6 % случаев от общего числа). При этом основным типом данных подвергшимся компрометации являются изображений печатных текстовых документов, полученных посредством сканирования или фотографирования. Наличие указанного канала утечки обусловлено отсутствием механизмов маркирования и обнаружения печатных документов в современных DLP-системах [3–6].

В ходе решения задачи по повышению защищенности конфиденциальной информации от утечки по каналу, обусловленному сканированием или фотографированием напечатанных документов с последующей отправкой по сети, разработан алгоритм маркирования электронных документов, основанный на внедрении маркера за счет горизонтального сдвига слов (символов) [7–8].

Алгоритм маркирования электронных текстовых документов, выводимых на печать, основан на применении технологии выделения строк текста из изображения, с последующим выделением слов (символов) внутри строк, формирования из последних областей встраивания и встраивания в сформированные области метки за счет изменения интервалов между словами в каждой строке. Встраивание информации осуществляется за счет внедрения в строку, содержащую более трех слов, пробела установленной длины (удлиненного пробела) с компенсацией (увеличением или уменьшением) остальных величин интервалов между словами в строке.

Практическая реализация прототипа алгоритма маркирования позволила оценить применимость разработанного алгоритма к маркированию различных типов данных. Так разработанный прототип реализует маркирование не только электронных текстовых документов формата: .doc, .docx, .rtf, .pdf, odt, .fodt но и электронных изображений, содержащих текстовые области, вне зависимости от используемого формата и алгоритма сжатия. Реализованный прототип позволяет провести экспериментальную оценку разработанного алгоритма маркирования и сделать вывод о возможности повышения защищенности электронных документов от утечки по каналу, обусловленному сканированием или фотографированием напечатанных на бумаге документов.

2. Экспериментальная оценка параметров алгоритма маркирования электронных документов

Экспериментальная оценка разработанного алгоритма осуществлялась на основе количественной и качественной оценки основных параметров, характеризующих встраиваемый маркер (алгоритм маркирования). К таким параметрам относятся [9–11]:

- емкость встраивания (полезная нагрузка);
- невидимость (перцептивная прозрачность);
- необнаруживаемость (сложность обнаружения);
- извлекаемость;
- робастность.

2.1 Емкость встраивания (полезная нагрузка)

Емкость встраивания (полезная нагрузка) – количество информации, которое может быть встроено (внедрено) в исходный документ [12, 13]. Основной величиной, характеризующей максимально возможное количество информации, которое может быть встроено в исходный документ, является предельно достижимая емкость встраивания. Предельно достижимая емкость встраивания η разработанного алгоритма маркирования рассчитывается по формуле:

$$\eta = \sum_{l=1}^{N} |DS[l]|, \quad (1)$$

где N – количество строк электронного документа, l – строка текста ($l = 1, 2, \ldots N$), DS – блок, состоящий из двух последовательно следующих пробелов (интервалов между словами).

Количество строк текстового документа полностью заполненного текстом N может быть рассчитано посредством выражения [14]:

$$N = \left\lfloor \frac{H - (m_t - m_b}{\gamma + \beta \cdot \gamma} \right\rfloor, \quad (2)$$

где H – высота текстового документа, m_t, m_b – размер верхнего и нижнего поля текстового документа соответственно, γ – размер кегля шрифта, β – величина множителя межстрочного интервала.

Для электронных документов, оформленных в соответствии с требования ГОСТ Р 7.0.97–2016 [15], значения высоты текстового документа H, размеров верхнего и нижнего поля текстового документа m_t, m_b представляют фиксированные величины со следующими значениями:

- высота текстового документа: формат А4 – 297 мм., А5 – 210 мм.;
- поля листа документа: верхнее – 20 мм., нижнее – 20 мм.;
- размер кегля шрифта: 12, 13 и 14 пт;
- величина множителя межстрочного интервала: 1–1,5.

Исходя из представленных значений, рассчитаны предельные значения количества строк страницы текстового документа полностью заполненной текстом:

- формат А4: от 52 (межстрочный интервал 1, кегль шрифта 12 пт) до 30 строк (межстрочный интервал 1,5, кегль шрифта 14 пт).;
- формат А5: от 34 (межстрочный интервал 1, кегль шрифта 12 пт) до 19 строк (межстрочный интервал 1,5, кегль шрифта 14 пт).

Помимо количества строк, приходящихся на страницу электронного документа, полностью заполненного текстом, осуществлен расчет среднего числа пробелов в строке. В результате проведенного анализа получены следующие значения:

- шрифт с кеглем 12 пт – 9 пробелов (А4), 6 пробелов (А5);
- шрифт с кеглем 13 пт – 8 пробелов (А4), 5 пробелов (А5);
- шрифт с кеглем 14 пт – 7 пробелов (А4), 4 пробела (А5).

Полученные оценки количества строк страницы текстового документа полностью заполненной текстом N, а также среднего числа пробелов в строке позволяют рассчитать значения предельно достижимой емкости встраивания η в зависимости от параметров оформления текстового документа. Полученные значения представлены в таблицах 1 и 2.

Табл. 1. Величина предельно достижимой емкости встраивания страницы текста формата A4
Table 1. The maximum achievable capacity value for embedding of an A4 text page

Кегль шрифта γ (пт)	**Величина межстрочного интервала β (множитель)**	**Количество строк N**	**Предельно достижимая емкость встраивания η**
12	1	52	208
	1,25	42	168
	1,5	35	140
13	1	48	192
	1,25	38	152
	1,5	32	128
14	1	45	135
	1,25	36	108
	1,5	30	90

Табл. 2. Величина предельно достижимой емкости встраивания страницы текста формата A5
Table 2. The maximum achievable capacity value for embedding of an A5 text page

Кегль шрифта γ (пт)	**Величина межстрочного интервала β (множитель)**	**Количество строк N**	**Предельно достижимая емкость встраивания η**
12	1	34	102
	1,25	27	81
	1,5	23	69
13	1	32	64
	1,25	25	50
	1,5	21	42
14	1	29	58
	1,25	23	46
	1,5	19	38

Анализ полученных результатов позволяет сделать вывод о том, что предельно достижимая емкость встраивания разработанного алгоритма маркирования составляет 208 бит для листа формата А4 и 102 для формата А5. При этом указанные значения достигаются посредством полного заполнения текстом страницы электронного документа. Полученные значения емкости встраивания разработанного алгоритма позволяют встраивать маркер размером 64 бита для документов формата А4, и 32 бита – формата А5. Для обоснования оптимального размера маркера необходимо оценить невидимость и необнаруживаемость встроенных данных (маркера) разработанного алгоритма маркирования.

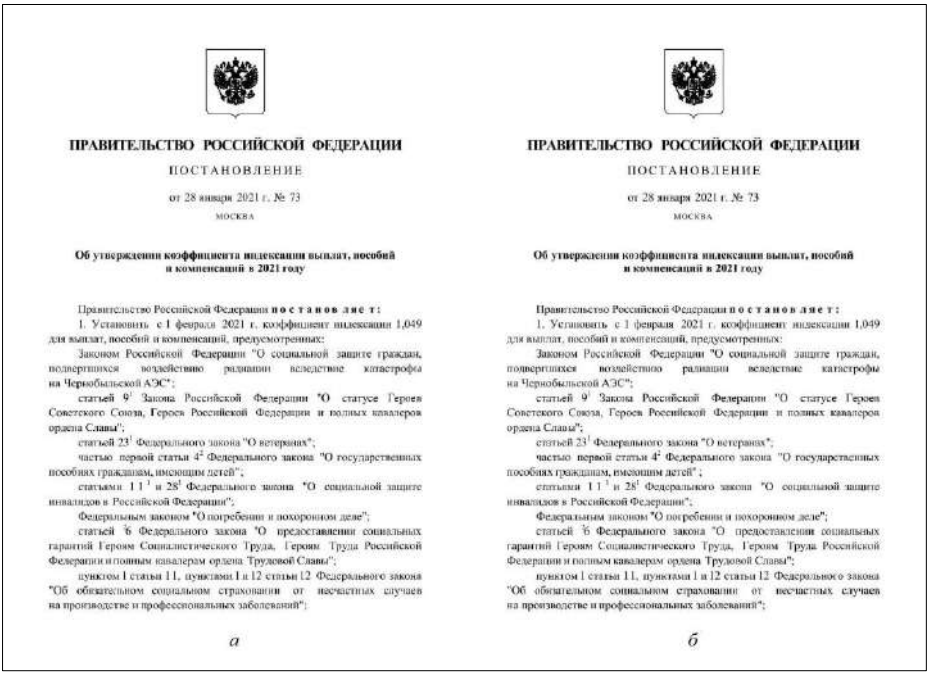
2.2 Невидимость (перцептивная прозрачность)

Невидимость (перцепционная прозрачность) встроенной информации (маркера) – качественная характеристика, отражающая степень искажения контейнера встраиваемыми данными. Данная характеристика основана на перцептивном восприятии человека и может быть оценена посредством проведения визуального анализа наблюдателем [16, 17].

В ходе оценки невидимости разработанного алгоритма маркирования проведены три группы исследований, направленных на оценку зависимости невидимости встроенного маркера от величины удлиненного пробела посредством визуального анализа. Визуальный анализ проводился посредством экспертного зрительного анализа подписанных электронных

документов, содержащих текст, представленных как в напечатанном, так и в электронном виде (в формате отсканированных изображений).

Внедрение маркера осуществлялось в электронные изображения, содержащие помимо текстовой информации различные графические объекты, таблицы, а также элементы рукописных подписей и печатей. Пример оригинального и соответствующего ему маркированного изображения представлен на рис. 1.



ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

ПОСТАНОВЛЕНИЕ

от 28 января 2021 г. № 73

МОСКВА

Об утверждении коэффициента индексации выплат, пособий и компенсаций в 2021 году

Правительство Российской Федерации **п о с т а н о в л я е т:**

1. Установить с 1 февраля 2021 г. коэффициент индексации 1,049 для выплат, пособий и компенсаций, предусмотренных:

Законом Российской Федерации "О социальной защите граждан, подвергшихся воздействию радиации вследствие катастрофы на Чернобыльской АЭС";

статьей 9^{1} Закона Российской Федерации "О статусе Героев Советского Союза, Героев Российской Федерации и полных кавалеров ордена Славы";

статьей 23^{1} Федерального закона "О ветеранах";

частью первой статьи 4^{2} Федерального закона "О государственных пособиях гражданам, имеющим детей";

статьями 11^{1} и 28^{1} Федерального закона "О социальной защите инвалидов в Российской Федерации";

Федеральным законом "О погребении и похоронном деле";

статьей 6 Федерального закона "О предоставлении социальных гарантий Героям Социалистического Труда, Героям Труда Российской Федерации и полным кавалерам ордена Трудовой Славы";

пунктом 1 статьи 11, пунктами 1 и 12 статьи 12 Федерального закона "Об обязательном социальном страховании от несчастных случаев на производстве и профессиональных заболеваний";

а

ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

ПОСТАНОВЛЕНИЕ

от 28 января 2021 г. № 73

МОСКВА

Об утверждении коэффициента индексации выплат, пособий и компенсаций в 2021 году

Правительство Российской Федерации **п о с т а н о в л я е т:**

1. Установить с 1 февраля 2021 г. коэффициент индексации 1,049 для выплат, пособий и компенсаций, предусмотренных:

Законом Российской Федерации "О социальной защите граждан, подвергшихся воздействию радиации вследствие катастрофы на Чернобыльской АЭС";

статьей 9^{1} Закона Российской Федерации "О статусе Героев Советского Союза, Героев Российской Федерации и полных кавалеров ордена Славы";

статьей 23^{1} Федерального закона "О ветеранах";

частью первой статьи 4^{2} Федерального закона "О государственных пособиях гражданам, имеющим детей";

статьями 11^{1} и 28^{1} Федерального закона "О социальной защите инвалидов в Российской Федерации";

Федеральным законом "О погребении и похоронном деле";

статьей 6 Федерального закона "О предоставлении социальных гарантий Героям Социалистического Труда, Героям Труда Российской Федерации и полным кавалерам ордена Трудовой Славы";

пунктом 1 статьи 11, пунктами 1 и 12 статьи 12 Федерального закона "Об обязательном социальном страховании от несчастных случаев на производстве и профессиональных заболеваний";

б

Рис. 1. Изображение: а) не содержащее встроенные данные; б) содержащее маркер
Fig. 1. Image: a) not containing embedded data; b) containing a marker

Первая группа исследований направлена на определение факта наличия встроенной информации, содержащейся в анализируемых подписанных изображениях (изображениях, содержащих встроенный маркер). В данной группе исследований аналитики (эксперты по стеганографическому анализу) не обладали информацией о содержании в анализируемых изображениях встроенных данных. В качестве анализируемых изображений выступали изображения, в которых осуществлено внедрение маркера посредством изменения величины удлиненного пробела разработанным алгоритмом маркирования. В результате анализа подписанных изображений не определен факт наличия встроенных данных.

При проведении второй группы исследований аналитики обладали информацией о факте наличия в анализируемых изображениях встроенных данных. При этом алгоритм внедрения информации оставался неизвестным. В результате проведенного визуального анализа не были обнаружены аномалии в структуре текста подписанных изображений, что позволило сделать вывод о невидимости встроенные данных к данному виду анализа.

В ходе проведения третьей группы исследований аналитики обладали информацией об используемом алгоритме внедрения информации, при этом место встраивания (строки текста) для аналитиков оставалось неизвестным. В результате визуального анализа аналитиками обнаружена каждая третья строка текста, содержащая удлиненный пробел. В остальных случаях позиции удлиненного пробела определены ошибочно или не определены совсем.

Результаты визуального анализа позволяют сделать вывод о стойкости разработанного подхода маркирования текстовых документов к обнаружению посредством визуального анализа встроенного маркера. Для определения статистических отклонений в сформированных в результате маркирования изображениях необходимо оценить необнаруживаемость (сложность обнаружения) разработанного алгоритма маркирования.

2.3 Необнаруживаемость (сложность обнаружения)

Под необнаруживаемостью (сложностью обнаружения) понимается количественная характеристика, отражающая степень искажения статистических характеристик контейнера, не связанных с перцептивным восприятием человека. Необнаруживаемость может быть оценена как стойкость разработанного алгоритма маркирования к потенциально наилучшему методу стеганографического анализа [18, 19].

Стеганографический анализ (стегоанализ, стеганоанализ) – наука об обнаружении факта присутствия (наличия) скрываемой информации в анализируемых контейнерах (объектах) [20, 21]. В качестве контейнера (объекта) могут выступать данные мультимедиа: изображения, аудио и видео данные. Помимо обнаружения факта сокрытия данных внутри анализируемых объектов дополнительной целью стегоанализа может выступать извлечение встроенной информации. Исходя из особенностей разработанного алгоритма маркирования, в качестве анализируемого контейнера выступают изображения, полученные посредством сканирования или фотографирования напечатанных на бумаге текстовых документов. Контейнеры отличные от указанных в работе не рассматриваются.

Существующие методы стеганографического анализа изображений можно разделить на специальные и общие (слепой стегоанализ) [22]. Обнаружение встроенной информации специальными методами осуществляется только при наличии информации (в том числе априорной) об используемом алгоритме стеганографического внедрения информации. К специальным методам относятся сигнатурные и схемные. Достоинством методов данной группы является возможность извлечения встроенной информации из подписанного изображения. При этом, методы данной группы малоэффективны против новых или неизвестных алгоритмов стеганографического внедрения информации. Сигнатурные методы стегоанализа основаны на поиске в анализируемом изображении сигнатуры (шаблона), характеризующего конкретный алгоритм или программное средство, осуществляющее стеганографическое внедрение информации. Методы сигнатурного стегоанализа позволяют однозначно определить факт встраивания и идентифицировать используемый стеганографический алгоритм (программное средство). При этом количество программных средств внедрения информации, имеющих собственные сигнатуры, исчисляется несколькими десятками, что не позволяет использовать данный подход для обнаружения новых (неизвестных) алгоритмов (программных средств).

Анализ существующих подходов к стеганографическому анализу позволяет обосновать выбор в качестве потенциально наилучших методов стеганографического анализа следующие подходы: визуальный стегоанализ и метод статистического стегоанализа в пространственной области, основанный на анализе гистограмм, построенных по частотам пробелов изображения. Основываясь на полученных результатах оценки невидимости (перцепционной прозрачности), сделан вывод о стойкости разработанного алгоритма маркирования к проведению визуального стегоанализа.

В ходе экспериментальной оценки стойкости разработанного подхода к методу статистического стегоанализа в пространственной области, основанного на анализе гистограмм, построенных по частотам пробелов изображения (потенциально наилучший метод стегоанализа), осуществлено извлечение информации как из неподписанных изображений (не содержащих встроенные данные), так и из изображений, содержащих встроенный маркер.

На рис. 2 представлен пример маркированных текстовых электронных документов (где 2*а* – сканированная версия документа, 2*б* – конвертированное изображение), которые совместно с оригинальными документами подвергались статистическому стегоанализу в пространственной области, основанному на анализе гистограмм.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
(МИНОБРНАУКИ РОССИИ)

РАСПОРЯЖЕНИЕ

Москва

Об утверждении кандидатур на должность руководителей научных организаций, подведомственных Министерству науки и высшего образования Российской Федерации

На основании пункта 9 Положения о порядке и сроках согласования и утверждения кандидатур на должность руководителя научной организации, находящейся в ведении Федерального агентства научных организаций, утвержденного постановлением Правительства Российской Федерации от 5 июня 2014 г. № 521, с учетом постановления президиума федерального государственного бюджетного учреждения «Российская академия наук» от 24 ноября 2020 г. № 180 «О согласовании кандидатур на должности руководителей научных организаций» и протокола заседания комиссии по кадровым вопросам Совета при Президенте Российской Федерации по науке и образованию от 22 декабря 2020 г. № 34:

1. Утвердить кандидатуры на должность руководителей научных организаций, подведомственных Министерству науки и высшего образования Российской Федерации, согласно приложению к настоящему распоряжению.

а)

РАСПОРЯЖЕНИЕ

ПРЕЗИДЕНТА РОССИЙСКОЙ ФЕДЕРАЦИИ

О заключении международного договора о продлении Договора между Российской Федерацией и Соединенными Штатами Америки о мерах по дальнейшему сокращению и ограничению стратегических наступательных вооружений от 8 апреля 2010 г.

Принять предложение Правительства Российской Федерации о заключении международного договора о продлении Договора между Российской Федерацией и Соединенными Штатами Америки о мерах по дальнейшему сокращению и ограничению стратегических наступательных вооружений от 8 апреля 2010 г.

Поручить МИДу России провести переговоры с Американской Стороной о заключении указанного международного договора, содержащего основные положения, одобренные Правительством Российской Федерации, и по достижении договоренности обеспечить его заключение.

КАНЦЕЛЯРИЯ

Президент
Российской Федерации В.Путин

25 января 2021 года
№ 25-рп

б)

Рис. 2. Примеры немаркированных документов
Fig. 2. Examples of unmarked documents

Результаты извлечения величин пробелов (интервалов между словами) из оригинальных изображений и маркированных копий (рис. 2), представлены в таблице 3.

Табл. 3. Результат извлечения интервалов между словами
Table 3. Extracting intervals between words result

Изображение	Извлеченные значения интервалов между словами (в пикселях)
Рис. 2*а* (оригинальный)	27,2; 16,3; 16,3; 16,3; 26,8; 18,1; 18,1; 18,1; 18,1; 26,8; 20,1; 20,1; 26,5; 20,1; 20,1; 16,2; 26,8; 21,8; 21,8; 28,8 21,8; 21,8; 28,8; 21,8; 21,8; 21,8; 79,8; 62,1; 62,1; 62,1; 79,8; 62,1; 29,9; 29,9; 29,9; 40,1; 29,9; 40,1; 41,3; 31,9; 31,9; 31,9; 51,34; 67,8; 51,3; 51,3; 51,3; 51,3; 51,3; 67,8; 67,8...
Рис. 2*а* (маркированный)	19; 18; 18; 21; 19; 23; 22; 18; 21; 23; 20; 20; 21; 22; 24; 21; 22; 23; 24; 22; 23; 24; 24; 24; 24; 22; 68; 68; 69; 65; 68; 70; 34; 34; 33; 34; 33; 32; 34; 36; 34; 33; 57; 55; 58; 55; 57; 55; 57; 57; 55; 57; 56; 56; 56; 54; 74; 74; 75; 72; 74; 74; 74; 74; 79; 52; 52; 52; 48; 51; 53; 39; 39; 35; ...
Рис. 2*б* (оригинальный)	47,5; 26,5; 26,8; 20,5; 20,5; 20,5; 20,5; 21,7; 29,2; 21,7; 21,7; 21,7; 19,1; 19,1; 26,8; 19,1; 19,1; 26,8; 20,4; 20,4; 27,8; 20,4; 19,8; 28,5; 19,8; 19,8; 38,9; 30,3; 30,3; 30,3; 30,3; 24,5; 33,5; 24,5; 24,5; 24,5; 33,5; 23,4; 23,4; 32,5; 23,4; 32,5; 23,4; 23,4; 67,9; 82,5; 67,9; 67,9; 67,9; 18,6; 18,6; ...
Рис. 2*б* (маркированный)	36; 38; 21; 24; 19; 22; 23; 24; 20; 24; 24; 24; 22; 23; 22; 23; 19; 21; 23; 22; 22; 22; 19; 23; 23; 23; 33; 31; 29; 34; 33; 30; 26; 26; 29; 27; 27; 27; 27; 27; 24; 26; 27; 24; 69; 70; 71; 73; 71; 21; 23; 22; 22; 22; 22; 29; 28; 32; 33; 31; 30; 61; 60; 61; 61; 58; 60; 58; 59; 57; 28; 27; 28; 24; ...

Анализ статистических значений полученных величин интервалов между словами позволяет сделать вывод об отсутствии статистической зависимости в распределении интервалов между словами как в оригинальных (немаркированных), так и в маркированных изображениях. На основе полученных массивов величин интервалов между словами осуществлено построение гистограмм (функции плотности распределения вероятностей). Результат построения гистограмм распределения интервалов между словами оригинального и маркированного изображения 2*а* и 2*б* представлены на рисунках 3 и 4 соответственно.

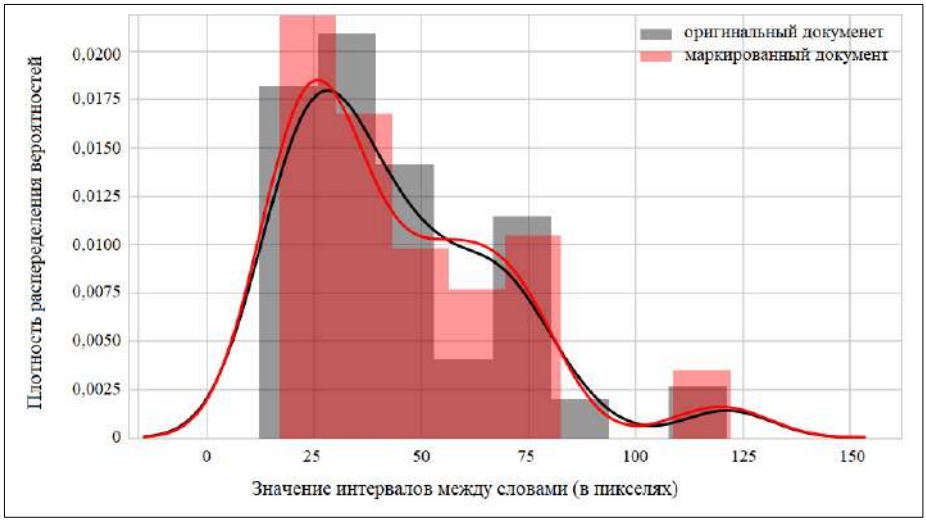


Рис. 3. Гистограмма распределения значений интервалов между словами оригинального и маркированного изображения 2а

Fig.3. Interval values distribution histogram between the words of the original and marked image 2a

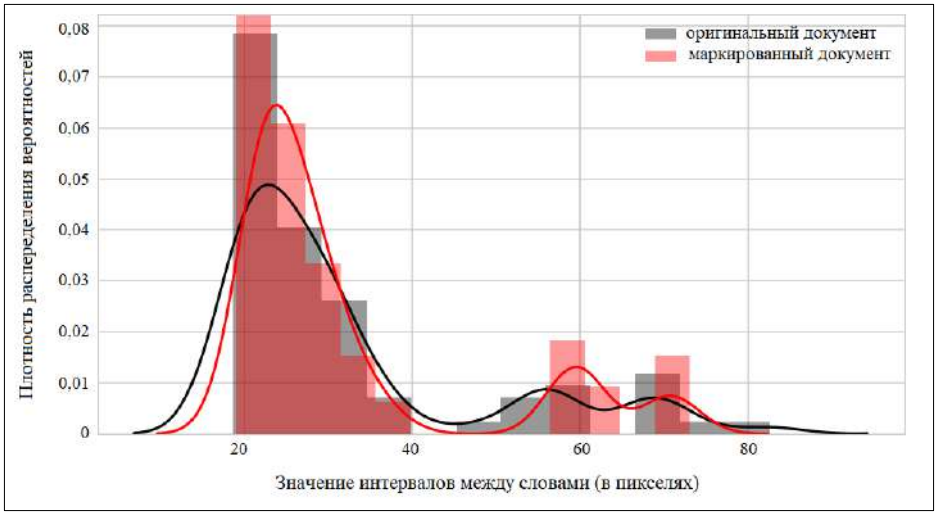


Рис. 4. Гистограмма распределения значений интервалов между словами оригинального и маркированного изображения 2б

Fig.4. Interval values distribution histogram between the words of the original and marked image 2b

Сравнительный анализ гистограмм распределения величин интервалов между словами позволяет сделать вывод и подобие (сходстве) гистограммы маркированного документа оригиналу. Стоит отметить, что имеются незначительные аномалии в сформированных гистограммах, которые обусловлены процессами конвертации, сканирования или

фотографирования изображения. Помимо аномалий указанные преобразования вносят изменения в величины извлеченных значений интервалов между словами в пикселях ввиду изменения разрешения изображения.

Результаты стойкости разработанного алгоритма маркирования электронных документов к потенциально наилучшему методу стегоанализа позволяют сделать вывод об стойкости разработанного алгоритма к указанному методу стегоанализа. Полученные значения невидимости и необнаруживаемости разработанного алгоритма маркирования позволяют перейти к оценке извлекаемости встроенной информации.

2.4 Извлекаемость

Извлекаемость – способность правильного извлечения встроенных данных из контейнера [23]. В ходе оценки извлекаемости встроенных данных из изображений было осуществлено встраивание маркера в текстовые электронные документы (формат .doc, .docx, .pdf, .rtf, .odt, .fodt) и извлечение встроенного маркера (формат .png) со следующими параметрами:

- кегль шрифта: 12 пт, 13 пт и 14 пт;
- межстрочный интервал: 1; 1,25 и 1,5.

Для экспериментальной оценки зависимости точности извлечения встроенных данных от параметров оформления электронных документов, осуществлено: встраивание маркера в подготовленные электронные документы, содержащие текст, преобразование (конвертация) электронных документов из формата .doc, .docx, .pdf, .rtf, .odt, .fodt в формат изображения PNG и извлечение встроенной информации. При преобразовании электронного документа в формат изображения использовано значение растеризации (разрешение изображения), равное 300 точек на дюйм.

Количественно извлекаемость может быть выражена посредством точности извлечения встроенных данных, которая описывается посредством методов оценки классификации, используемых в задачах теории распознавания образов [133–135]. Для количественной оценки точности извлечения рассмотрены следующие метрики точности классификатора: Accuracy и F-мера:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN};\ \ F - \text{мера} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN},$$

где TP – истинно-положительный, TN – истинно-отрицательный, FP – ложно-положительный и FN – ложно-отрицательный результат.

В процессе экспериментальной оценки было извлечено более 15 000 бит (более 350 страниц текстовой информации), что позволяет утверждать о том, что доверительный вероятность равна 0,95 при точности 0,01. Результаты зависимости точности извлечения от параметров оформления текстовых документов (размер кегля шрифта, величина множителя межстрочного интервала) представлены в табл. 4.

Табл. 4. Точность извлечения данных из изображений в зависимости от параметров оформления текстовых документов

Table 4. Data extraction accuracy from images depending on the design parameters of text documents

Кегль шрифта γ (пт)	**Величина межстрочного интервала β (множитель)**	**Точность излечения F-мера**	**Ложные срабатывания**	**Пропуск цели**
12	1	0,963	0,018	0,019
	1,25	0,961	0,015	0,024
	1,5	0,964	0,020	0,016
13	1	0,966	0,017	0,017
	1,25	0,965	0,015	0,020

	1,5	0,970	0,020	0,010
14	1	0,978	0,014	0,018
	1,25	0,972	0,014	0,014
	1,5	0,980	0,012	0,008

Полученные значения точности извлечения встроенного маркера из изображений позволяют сделать вывод об отсутствии зависимости результата извлечения встроенной информации от параметров электронного документа (размера кегля шрифта, величины межстрочного интервала). Результат извлечения встроенной информации характеризуется одиночными ошибками извлечения первого и второго рода и позволяет сделать вывод о возможности извлечения встроенной информации разработанным алгоритмом из подписанных изображений со значение точности извлечения более 0,95.

В ходе оценки зависимости точности извлечения от разрешения сформированного изображения проведены три группы исследований. В рамках первой группы осуществлена количественная оценка точности извлечения в зависимости от разрешения изображения, сформированного посредством конвертации электронного документа, содержащего встроенный маркер, в изображение. В рамках второй группы оценена зависимость точности извлечения от разрешения изображения, сформированного посредством печати и сканирования электронного документа, содержащего встроенные данные. Третья группа исследований позволила оценить зависимость точности извлечения от разрешения изображения, сформированного посредством печати и фотографирования электронного документа, содержащего встроенные данные.

В процессе количественной оценки точности использованы следующие значения разрешения изображений: 150, 200, 300, 400, 500, 600 и 1200 точек на дюйм. Результаты точности извлечения встроенной информации из изображений, содержащих текст с размером шрифта 12 пт, величиной межстрочного интервала 1, представлены в табл. 5.

Табл. 5. Результаты извлечения информации из конвертированного изображения, содержащего встроенный маркер

Table 5. Results of extracting information from a converted image containing an embedded marker

Разрешение изображения	**Показатель истинно-положительных значений**	**Показатель истинно-отрицательных значений**	**Accuracy**	**F-мера**
150	0,97	0,95	0,96	0,96
200	0,97	0,97	0,97	0,97
300	1	0,97	0,97	0,97
400	1	0,97	0,98	0,98
500	1	0,98	0,99	0,99
600	1	1	1	1
1200	1	0,98	0,99	0,99

Анализ полученных результатов точности извлечения позволяет сделать вывод о наличие незначительной зависимости значений точности извлечения от используемых параметров электронного документа, содержащего текст, и наличие зависимости от разрешения сформированного изображения. При этом стоит отметить тот факт, что показатели точности извлечения для изображений с разрешением в 150 и 200 точек на дюйм, характеризуются наличием ошибок первого и второго рода. В то время как изображения с расширением в 300 точек на дюйм и выше имеют незначительный процент ошибок первого рода, что соответствует наличию одиночных ошибок, которые могут быть устранены посредством применения помехоустойчивых кодов в процессе маркирования.

В ходе второй группы экспериментов осуществлена оценка зависимости точности извлечения встроенных данных от разрешения отсканированного изображения и параметров оформления документов. Для этого осуществлено встраивание маркера в электронные документы, содержащие текст, печать электронных документов, сканирование напечатанных документов и излечение встроенной информации. В качестве используемой метрики оценки точности извлечения информации использована F-мера. В процессе сканирования напечатанных электронных документов использовались следующие значения: 150, 200, 300, 400, 500 и 600 точек на дюйм.

Результаты точности извлечения встроенной информации из электронных изображений, полученных посредством печати и сканирования электронных документов, оформленных в соответствии с требованиями ГОСТ 7.0.97–2016, представлены на рис. 5.

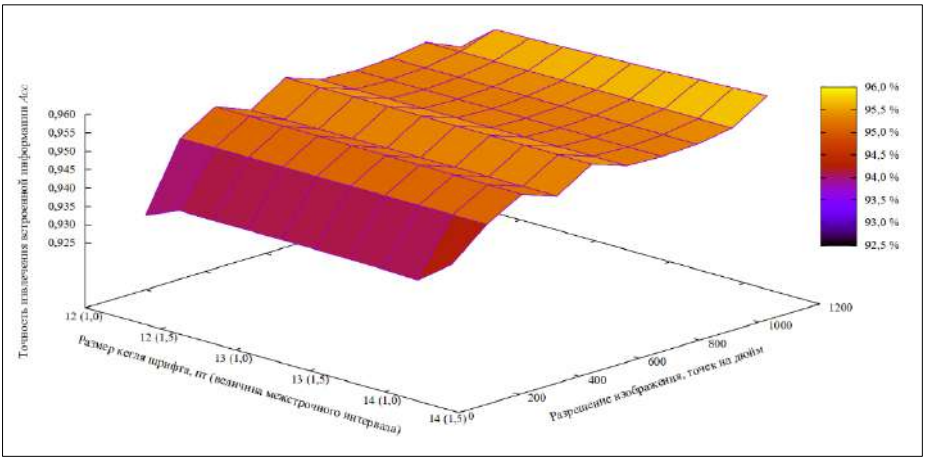


Рис. 5. Зависимость точности извлечения от параметров отсканированного изображения и оформления текстового документа, содержащего встроенный маркер
Fig.5. Extraction accuracy dependency on the parameters of the scanned image and text document design containing an embedded marker

Полученные значения точности извлечения встроенных данных из отсканированных изображений позволяет отнести разработанный алгоритм маркирования к алгоритмам с высокой точностью извлечения (близкой или равной 100%) ввиду наличия ошибок извлечения, которые могут быть исправлены в случае применения методов помехоустойчивого кодирования. Указанные значения точности извлечения достигаются для электронных документов, содержащих текст и оформленных в соответствии требований ГОСТ 7.0.97–2016, в независимости от используемой гарнитуры шрифта, напечатанных и отсканированных с разрешением сканера не менее 300 точек на дюйм. Стоит отметить, что в современных сканирующих устройствах показатель разрешения сканера в 300 точек на дюйм соответствует стандартному качеству сканирования.

В ходе проведения третьей группы исследований осуществлена оценка зависимости точности извлечения встроенной информации от величины разрешения изображения, полученного посредством фотографирования, напечатанного на бумаге электронного документа и параметров оформления электронных документов, содержащих текст. Фотографирование напечатанного электронного документа осуществлено с последующей цифровой обработкой сформированного изображения, состоящей из корректировки перспективы изображения и обрезки областей, не относящихся к исходному электронному документу. В качестве используемой метрики оценки точности извлечения информации выступает F-мера.

Результаты точности извлечения встроенной информации из электронных изображений, полученных посредством печати и фотографирования электронных документов, содержащих

текст и оформленных в соответствии с требованиями ГОСТ 7.0.97–2016, представлены на рис. 6.

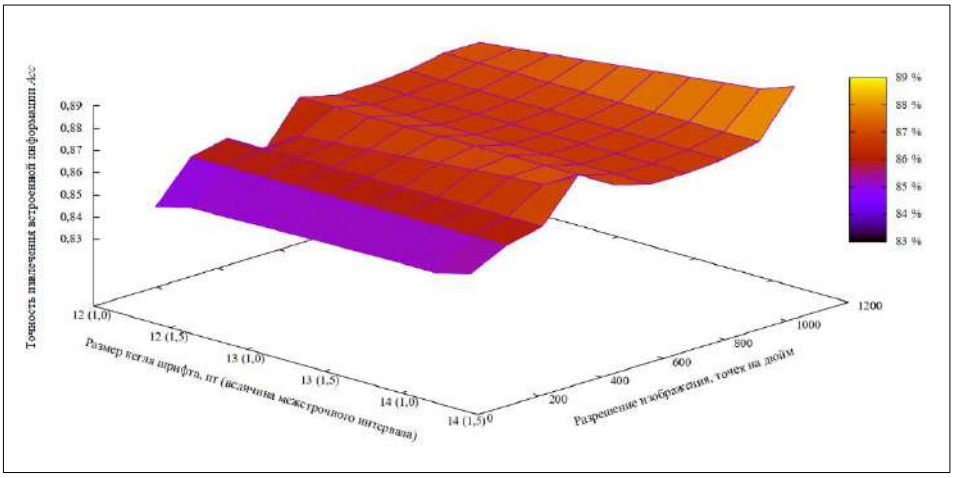


Рис. 6. Зависимость точности извлечения от параметров сфотографированного изображения и оформления текстового документа, содержащего встроенный маркер
Fig.6. Extraction accuracy dependency on the parameters of the phtted image and text document design containing an embedded marker

Полученные значения точности извлечения встроенных данных из сфотографированных изображений не позволяет отнести разработанный алгоритм маркирования к алгоритмам с высокой точностью извлечения (близкой или равной 100%) ввиду наличия большого количества ошибок. Указанная особенность не позволяют обеспечить правильное извлечение встроенной информации и точно установить источник утечки информации. Для устранения указанного недостатка может быть применена циклическая схема внедрения маркера, основанная на повторении встраиваемого маркера допустимое число раз совместно с методами помехоустойчивого кодирования.

Точность извлечения разработанного алгоритма маркирования зависит от используемых параметров оформления документов: кегль шрифта и величина межстрочного интервала, а также от разрешения изображения. Точность извлечения разработанного алгоритма маркирования превышает значение в 95 % при следующих параметрах:

- разрешение отсканированного изображения более 300 точек на дюйм, сфотографированного – более 600 точек на дюйм;
- печать и сканирования электронного документа с внедрением закодированного маркера методом помехоустойчивого кодирования;
- печать и фотографирование электронного документа с внедрением закодированного маркера методом помехоустойчивого кодирования по циклической схеме внедрения.

Полученные результаты точности извлечения встроенной информации из изображений, содержащих текстовый документ со встроенным маркером, позволяют перейти к оценке робастности разработанного алгоритма маркирования к осуществлению преобразований и внесению искажений, возникающим в процессе печати, сканирования и фотографирования исходного документа.

2.5 Робастность

Робастность – способность встроенных данных сохранять свойство инвариантности после осуществления различных преобразований над контейнером, подмены или удаления

встроенных данных [24]. Робастность является качественной характеристикой. Проведенная экспериментальная оценка извлекаемости показала, что в разработанном алгоритме маркирования обеспечиваются высокие значения точности извлечения встроенной информации после печати и сканирования исходного документа. Стоит отметить, что процесс сканирования, как и фотографирования, характеризуется наличием искажений и преобразований, которые вносят печатающее, фотосчитывающее устройство или объектив фотоаппаратуры, а также внешние факторы: освещение, вибрация и прочее. Исходя из описанных особенностей, исследование робастности разработанного алгоритма маркирования проводились по двум направлениям. В рамках первого направления исследовалась устойчивость (робастность) встроенной информации к следующим преобразованиям и искажениям, возникающим в процессе печати и сканирования электронного документа:

- поворот изображения;
- изменение соотношения сторон изображения (масштабирование);
- сжатие изображения с потерями (форматы .jpeg, .tif);
- сжатие изображения без потерь (форматы .png, .bmp, .tif, .gif);
- билатеральная фильтрация;
- гауссовская фильтрация (гауссовский фильтр размытия);
- медианная фильтрация;
- внесение фона в изображение.

Особенность разработанного алгоритма заключается в извлечении интервалов между словами, которые расположены на прямой линии относительно центра слов. В случае поворота изображения прямая линия, проходящая через центры слов, преобразуется в ломаную и значения интервалов между словами не могут быть правильно извлечены. Указанная особенность вносит ограничение не только на робастность разработанного алгоритма маркирования к указанному преобразованию, но и на возможность извлечения встроенной информации из анализируемых изображений.

Рис. 7. Извлечение данных из изображения, повернутого на 2 градуса
Fig.7. Extracting data from an image rotated by 2 degrees

В ходе экспериментальной оценки устойчивости разработанного алгоритма к повороту определены предельные значения углов, на которые может быть повернут текст. Разработанный алгоритм позволяет извлекать встроенную информацию из изображения, повернутого не более чем на ±3 градуса. Пример извлечения встроенной информации из изображения, повернутого на 2 градуса, представлен на рис. 7.

В процессе оценки устойчивости разработанного алгоритма к изменению соотношения сторон изображения (масштабирование) проведены эксперименты по растягиванию подписанного изображения как в горизонтальной или вертикальной плоскости, так и в обеих плоскостях одновременно. Диапазон исследуемых значений изменения коэффициента масштабирования составляет 0,5...2,5 (от 1/2 до 2,5 размера исходного документа).

Анализ полученных результатов показал, что разработанный алгоритм маркирования обеспечивает устойчивость встроенной информации к изменению отношения сторон изображения (масштабированию) в пропорциях (коэффициентов масштабирования), находящихся в пределах: 0,5…2,5 относительно исходного размера изображения как в горизонтальной или вертикальной плоскости, так и в обеих плоскостях одновременно.

В хода анализа стойкости разработанного алгоритма маркирования к осуществлению сжатия изображения с потерями, осуществляемое в процессе формирования изображения. В процессе экспериментальной оценки исходное изображение подвергалось сжатию по алгоритму JPEG с показателем качества 10…100 с шагом 10%. В результате анализа полученных результатов установлено, что робастность разработанного алгоритма к сжатию изображения с потерями обеспечивается в случае использования показателя качества сжатия от 30 % и более.

Стойкость разработанного алгоритма к сжатию изображения без потерь заложена в алгоритм маркирования и извлечения встроенного маркера. Так, во время внедрения и извлечения маркера сформированное изображения подвергается растеризации изображения – преобразованию изображения в формат .png, который является одним из представителей форматов изображения, использующим сжатие без потерь или полное отсутствие сжатия. В результате чего можно сделать вывод о том, что разработанный алгоритм обеспечивает устойчивость встроенного маркера к сжатию исходного изображения без потерь.

Экспериментальная оценка стойкости разработанного алгоритма к фильтрации изображения проведена для следующих фильтров: билатеральный, гауссовский (фильтр размытия) и медианный фильтр. Билатеральная фильтрация – нелинейная фильтрация, выполняющая пространственное усреднение в пределах своей маски, применяемая для удаления шума и сглаживания однородных областей.

Исходя из особенностей применения фильтра, направленных на сглаживание областей изображения, результат извлечения встроенной информации обладает устойчивостью к применению данного фильтра. Предельные значения примененного фильтра составляют: маска фильтра (диаметр соседних пикселей) – не более 10, сигма-фильтра в цветной и пространственной областях – не более 150.

Гауссовский (гауссов) фильтр – фильтр линейного сглаживания, предназначенный для удаления шума, описываемого гауссовским (нормальным) законом распределения. Как и билатеральный гауссовский фильтр применяется для уменьшения резких изменений в градациях серого изображения.

Полученные значения позволяют сделать вывод об устойчивости разработанного алгоритма маркирования к осуществлению гауссовской фильтрации изображения с предельным значением ядра свертки в (30, 30) по высоте и ширине или радиуса размытия в 8 пикселей.

Медианный фильтр представляет собой эвристический метод обработки изображения, который удаляет из сигнала (изображения) фрагменты с размерами, меньшими, чем половина размера окна фильтра, и при этом мало искажает или почти совсем не искажает остальные

участки сигнала (изображения). Оценка устойчивости к медианной фильтрации изображения, содержащего текстовые данные, производилась за счет применения медианного фильтра с шагом изменения радиуса размытия от 1 до 9 пикселей. Предельное значение радиуса размытия 9 характеризует пороговое значение робастности к медианной фильтрации разработанного алгоритма маркирования.

Помимо фильтрации формируемое изображение может подвергаться внесению дополнительных элементов, в частности изменение цвета фона изображения, вызванное износом сканирующего устройства или подкладыванием дополнительных листов бумаги, обладающих оттенками цвета отличным от белого.

Полученные результаты извлечения встроенной информации после внесения дополнительно фона изображения позволяют сделать вывод о наличии стойкости разработанного алгоритма маркирования к указанному типу искажения. Граничные значения стойкости определяются прозрачностью фона. В случае использования прозрачности фона меньше 10% исходный текст и графические изображения в документе будут нечитаемыми. Указанный факт ограничивает стойкость разработанного алгоритма маркирования предельным значением прозрачности фона изображения не менее чем 10%.

В результате проведенной экспериментальной оценки робастности разработанного алгоритма маркирования к осуществлению преобразования формата, обусловленного печатью и сканированием, получены результаты, представленные в табл. 6.

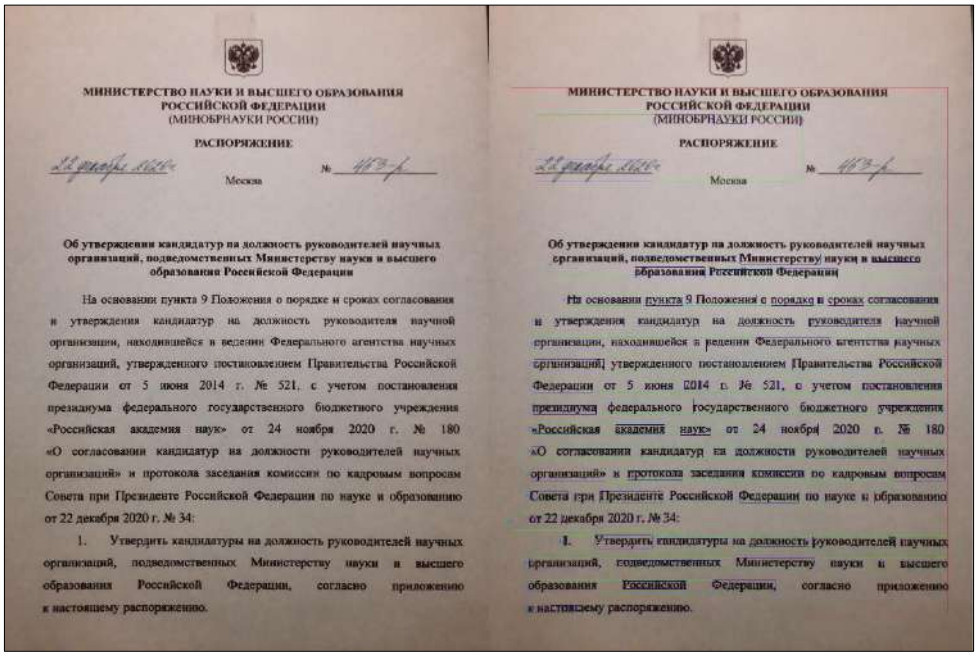
Табл. 6. Робастность разработанного алгоритма к преобразованиям и искажениям, возникающим в процессе печати и сканирования
Table 6. Robustness of the developed algorithm to transformations and distortions that occur during printing and scanning

Тип преобразования	**Параметры робастности**
Поворот изображения	±3∘ относительно горизонтально выравненного текста
Изменение соотношения сторон	Коэффициент масштабирования 0,5…2,5 относительно исходного размера
Сжатие изображения с потерями	По алгоритму JPEG с показателем качества не менее 30 %
Сжатие изображения без потерь	Алгоритмы: RLE, LZW и Deflate
Балатеральная фильтрация	Диаметр соседних пикселей не более 10, сигма-фильтра в цветной и пространственной областях не более 150
Гауссовская фильтрация	Размер ядра по высоте и ширине не более (30, 30)
Медианная фильтрация	Предел ядра свертки 9 пикселей

Результаты робастности, представленные в табл. 6, а также значения точности извлечения встроенной информации из изображений, содержащих встроенный маркер, позволяют сделать вывод о возможности гарантированного извлечения встроенного маркера (с точностью извлечения более 95%) из изображений, полученных посредством печати и сканирования электронных документов.

В рамках второго направления исследований осуществлена оценка устойчивости (робастности) встроенной информации к преобразованию электронного документа в изображения посредством печати и фотографирования, а также осуществления сопутствующих искажений. Отличительной особенностью процесса фотографирования от сканирования является наличие геометрических искажений в трех плоскостях: горизонтальный наклон, вертикальное отклонение и вращение относительно центра. Кроме того, процесс фотографирования характеризуется наличием искажений, вносимых объективом фотоаппаратуры и условиями съемки: неравномерно освещенные области, муар, виньетирование, искажение перспективы и т.д.

Указанные особенности требуют проведения дополнительного этапа предварительной обработки изображения, направленного на коррекцию перспективы и обрезку областей изображения, изначально не относившихся к исходному текстовому документу. Без проведения этапа предварительной обработки извлечение встроенной информации не может быть реализовано с требуемым значением точности извлечения. Пример извлечения информации из изображения, содержащего встроенные данные, сформированного посредством фотографирования, напечатанного на бумаге электронного документа, содержащего текст, представлен на рис. 8.


МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
(МИНОБРНАУКИ РОССИИ)

РАСПОРЯЖЕНИЕ

Москва

Об утверждении кандидатур на должность руководителей научных организаций, подведомственных Министерству науки и высшего образования Российской Федерации

На основании пункта 9 Положения о порядке и сроках согласования и утверждения кандидатур на должность руководителя научной организации, находящейся в ведении Федерального агентства научных организаций, утвержденного постановлением Правительства Российской Федерации от 5 июня 2014 г. № 521, с учетом постановления президиума федерального государственного бюджетного учреждения «Российская академия наук» от 24 ноября 2020 г. № 180 «О согласовании кандидатур на должности руководителей научных организаций» и протокола заседания комиссии по кадровым вопросам Совета при Президенте Российской Федерации по науке и образованию от 22 декабря 2020 г. № 34:

1. Утвердить кандидатуры на должность руководителей научных организаций, подведомственных Министерству науки и высшего образования Российской Федерации, согласно приложению к настоящему распоряжению.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
(МИНОБРНАУКИ РОССИИ)

РАСПОРЯЖЕНИЕ

Москва

Об утверждении кандидатур на должность руководителей научных организаций, подведомственных Министерству науки и высшего образования Российской Федерации

На основании пункта 9 Положения о порядке и сроках согласования и утверждения кандидатур на должность руководителя научной организации, находящейся в ведении Федерального агентства научных организаций, утвержденного постановлением Правительства Российской Федерации от 5 июня 2014 г. № 521, с учетом постановления президиума федерального государственного бюджетного учреждения «Российская академия наук» от 24 ноября 2020 г. № 180 «О согласовании кандидатур на должности руководителей научных организаций» и протокола заседания комиссии по кадровым вопросам Совета при Президенте Российской Федерации по науке и образованию от 22 декабря 2020 г. № 34:

1. Утвердить кандидатуры на должность руководителей научных организаций, подведомственных Министерству науки и высшего образования Российской Федерации, согласно приложению к настоящему распоряжению.

Рис. 8. Извлечение данных из изображения, сформированного посредством фотографирования
Fig.8. Extracting data from an image formed by photographing

Оценка устойчивости встроенной информации к искажениям, возникающим в процессе печати и фотографирования электронного документа, проводилась по тем же критериям, что и оценка устойчивости к искажениям, возникающим при печати и сканировании. Разработанный алгоритм показал те же значения робастности применении операций печать и фотографирование, что и значения, представленные в табл. 6. При этом точность извлечения встроенной информации не превышает значение в 89 %. Полученные значения робастности к сканированию и фотографированию позволяют описать робастность разработанного алгоритма маркирования следующим образом:

- печать и сканирование электронного документа: робастность к искажениям, представленным в табл. 6, при использовании методов помехоустойчивого кодирования в процессе маркирования;
- печать и фотографирование электронного документа: наличие дополнительного этапа обработки изображения, направленного на корректировку перспективы изображения и удаления областей, не присутствовавших в исходном текстовом документе; использование фотоаппаратуры, формирующей изображение не менее чем 4000 × 3000 (12 мегапикселей); использование циклической схемы встраивания и методов помехоустойчивого (при наличии достаточной емкости встраивания).

4. Заключение

Проведенная экспериментальная оценка основных параметров разработанного алгоритма маркирования, основанного на изменении интервалов между словами, позволяет сделать вывод о повышении защищенности электронных документов, содержащих текстовую информацию, от утечки посредством печати, сканирования или фотографирования с последующей отправкой изображения в случае внедрения предложенного подхода в компоненты DLP- или SIEM-системы. При этом наличие одиночных ошибок в процессе извлечения информации из отсканированных изображений и низкие значения точности извлечения встроенных данных из сфотографированных изображений требуют дальнейшего совершенствования предложенного подхода к маркированию электронных документов, а также проведения сравнительного анализа полученных результатов с существующими аналогами. Решение поставленных задач является направлением дальнейших исследований.

Список литературы / References

[1] Cybersecurity threatscape: Year 2021 in review. Positive Technologies, 2022, 23 p. Available at: https://www.ptsecurity.com/upload/corporate/ww-en/analytics/Cybersecurity_threatscape_2021_ENG.pdf, accessed 10.08.2022.

[2] Отчёт об исследовании утечек информации ограниченного доступа в 2021 году. InfoWatch, 2022 г., 32 стр. / Restricted Information Leakage Study Report in 2021. InfoWatch. 2021, 32 p. Available at: https://www.infowatch.ru/analytics/analitika/v-2021-stalo-bolshe-umyshlennykh-utechek, accessed 10.08.2022 (in Russian).

[3] Jain M., Lenka S.K. A Review on Data Leakage Prevention using Image Steganography. International Journal of Computer Science Engineering, vol. 5, no 2, 2016, pp 56-59.

[4] Lopez G., Richardson N., Carvajal J. Methodology for Data Loss Prevention Technology Evaluation for Protecting Sensitive Information. Revista Politecnican, vol. 36, no 3, 2015, pp. 60-69.

[5] Alneyadi S., Sithirasenan E., Muthukkumarasamy V. A survey on data leakage prevention systems. Journal of Network and Computer Applications, vol. 62, 2016, pp. 137-152.

[6] Jadhav P., Chawan P.M. Data Leak Prevention system: A Survey. International Research Journal of Engineering and Technology, vol. 6, no. 10, 2019, pp. 197-199.

[7] Козачок А.В., Копылов С.А. и др. Алгоритм маркирования текстовых документов на основе изменения интервалов между словами, обеспечивающий устойчивость к преобразованию формата. Труды ИСП РАН, том 5, вып. 5, 2021 г., стр. 131-146. DOI: 10.15514/ISPRAS-2021-33(4)-10. / Kozachok A.V., Kopylov S.A. et al. Text documents marking algorithm based on interword distances shifting invariant to format conversion. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 4, 2021, pp. 131-146 (in Russian).

[8] Kozachok A.V., Kopylov S.A. et al. Text marking approach for data leakage prevention. Journal of Computer Virology and Hacking Techniques, vol. 15, no. 3, 2019, pp. 219-232.

[9] Salomon D. Data privacy and security: encryption and information hiding. Springer Science & Business Media, New York, 2003. 469 p.

[10] Kapila B., Thind T. Review and analysis of data security using image steganography. In Proc. of the 2nd International Conference on Computation, Automation and Knowledge Management (ICCAKM), 2021, pp.227-231.

[11] Woo C.-S. Digital image watermarking methods for copyright protection and authentication. PhD Thesis. Information Security Institute, Faculty of Information Technology, Queensland University of Technology, 2007, 197 p.

[12] Mohanarathinam A., Kamalraj S. et al. Digital watermarking techniques for image security: a review. Journal of Ambient Intelligence and Humanized Computing, vol. 11, 2020, pp. 3221-3229.

[13] Khadam U., Iqbal M.M. et al. Digital Watermarking Technique for Text Document Protection Using Data Mining Analysis. IEEE Access, vol. 7, 2019, pp. 64955-64965.

[14] Kozachok A.V., Kopylov S.A. Estimation of Watermark Embedding Capacity with Line Space Shifting. In Proc. of the Ivannikov Memorial Workshop (IVMEM), 2020, pp. 29-34.

[15] Национальный стандарт Российской Федерации. Система стандартов по информации, библиотечному и издательскому делу. Организационно-распорядительная документация. Требования к оформлению документов. ГОСТ Р 7.0.97–2016, Стандартинформ, 2019 г., 32 стр. /

National standard of the Russian Federation, System of standards on information, librarianship and publishing. Organizational and administrative documentation. Requirements for presentation of recordsю GOST R 7.0.97–2016, Standartinform, 2019, 32 p. (in Russian).

[16] Zhou N.R., Hou W.M.X., Wen R.H. Imperceptible digital watermarking scheme in multiple transform domains. Multimedia Tools and Applications, vol. 77, 2018, pp. 30251–30267.

[17] Wu J.Y., Huang W.L., Xia-Hou W.M. Imperceptible digital watermarking scheme combining 4-level discrete wavelet transform with singular value decomposition. Multimedia Tools and Applications, vol. 79, 2020, pp. 22727–22747.

[18] Грибунин В.Г., Оков И.Н., Туринцев И.В. Цифровая стеганография. Москва, СОЛОН-Пресс, 2017 г., 262 стр. / Gribunin V.G., Okov I.N., Turincev I.V. Digital steganography. Moscow, SOLON-Press, 2017, 262 p. (in Russian).

[19] Коржик В.И. Цифровая стеганография и цифровые водяные знаки. Санкт-Петербург, СПбГУТ, 2017 г., 424 стр. / Korzhik V.I. Digital Steganography and Digital Watermarking. Saint-Petersburg, SPbSUT, 2016, 226 p. (in Russian)

[20] Козачок А.В., Копылов С.А., Бочков М.В. Оценка параметров необнаруживаемости разработанного подхода к маркированию текстовых электронных документов. Вопросы кибербезопасности, no. 1(35), 2020, стр. 62-73 / Kozachok A.V., Kopylov S.A., Bochkov M.V. Undetectability Parameters Estimation of the Developed Approach to Text Electron Documents Marking. no 1(35), 2020, pp. 62-73 (in Russian).

[21] Karampidis K., Kavallieratou E., Papadourakis G. A review of image steganalysis techniques for digital forensics. Journal of Information Security and Applications, vol. 40, 2018, pp. 217-235.

[22] Yang Z., Huang Y., Zhang Y.-J. A fast and efficient text steganalysis method. IEEE Signal Processing Letters, vol. 26, no. 4, 2019, pp. 627-631.

[23] Kadian P., Arora S.M., Arora N. Robust Digital Watermarking Techniques for Copyright Protection of Digital Data: A Survey. Wireless Personal Communications, vol. 118, 2021, pp. 3225-3249.

[24] Menendez-Ortiz A., Feregrino-Uribe C. et al. A Survey on Reversible Watermarking for Multimedia Content: A Robustness Overview, IEEE Access, vol. 7, 2019, pp. 132662-132681.

Информация об авторах / Information about authors

Александр Васильевич КОЗАЧОК – доктор технических наук, доцент, заведующий лабораторией безопасного программного обеспечения и анализа данных. Сфера научных интересов: методы и системы защиты информации, кибербезопасность, машинное обучение, анализ данных.

Alexander Vasilievich KOZACHOK – Doctor of Technical Sciences, associate professor, Head of the Laboratory of Secure Software and Data Analysis. Research interests: algebraic structures in the information security methods and systems, cybersecurity, machine learning, data analysis.

Василий Иванович КОЗАЧОК – доктор социологических наук, профессор, сотрудник Академии Федеральной службы охраны Российской Федерации. Его научные интересы включают: безопасность информации, защита информации от несанкционированного доступа, построение информационных систем в защищённом исполнении.

Vasilii Ivanovich KOZACHOK – Doctor of Sociological Sciences, Professor. Employer of the Academy of Federal Guard Service. His research interests include: information security, information unauthorized access protection, information systems construction in a secure design.

Сергей Александрович КОПЫЛОВ – кандидат технических наук, сотрудник Академии Федеральной службы охраны Российской Федерации. Его научные интересы включают: методы машинного обучения, обработка цифровых изображений, текстовая стеганография.

Sergey Alexandrovich KOPYLOV – PhD in Technical Sciences. Employer of the Academy of Federal Guard Service. His research interests include machine learning methods, digital image processing, text steganography.

Павел Николаевич ГОРБАЧЕВ – сотрудник Академии Федеральной службы охраны Российской Федерации. Его научные интересы включают: информационная безопасность,

методы машинного обучения, распознавание образов, текстовая стеганография, обработка изображений.

Pavel Nikolaevich GORBACHEV is employer of the Academy of Federal Guard Service. His research interests include: information security, machine learning methods, pattern recognition text steganography and image processing.

Юрий Витальевич МАРКИН – кандидат технических наук, научный сотрудник. Область научных интересов: информационная безопасность, анализ сетевого трафика, обработка изображений, алгоритмы машинного обучения.

Yury Vital'evich MARKIN – PhD in Technical Sciences. Researcher. Scientific interests: information security, network traffic analysis, image processing, machine learning algorithms.

Дмитрий Олегович ОБЫДЕНКОВ – аспирант. Его научные интересы включают методы сокрытия и защищённой передачи информации, компьютерные сети, технологии анализа сетевого трафика.

Dmitry Olegovich OBYDENKOV is a graduate student. His scientific interests include methods for information hiding and secure transmission, computer networks, technologies of network traffic analysis.

DOI: 10.15514/ISPRAS-2022-34(4)-12

Алгоритм поиска специалистов с уникальными навыками на основе цифрового следа

[1] *А.С. Леонов, ORCID: 0000-0002-7904-182X <alex.thunder@tut.by>*
[1] *А.А. Лаптев, ORCID: 0000-0003-1754-6643 <nickname.avast@gmail.com>*
[1] *А.А. Лаушкина, ORCID: 0000-0001-7887-3299 <nastasjalausckina@mail.ru>*
[1] *М.В. Синько, ORCID: 0000-0001-6075-9594 <michael.v.sinko@gmail.com>*
[1,2] *О.О. Басов, ORCID: 0000-0001-5788-4845 <oobasov@mail.ru>*

[1] *Национальный исследовательский университет ИТМО,*
197101, Россия, г. г. Санкт-Петербург, Кронверкский пр., д. 49, лит А
[2] *Институт системного программирования им. В.П. Иванникова РАН,*
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. В последние годы из-за значительных изменений на рынке труда компании стали чаще сталкиваться с различными проблемами при поиске и отборе кандидатов. Основной причиной данных проблем является то, что существующие интернет-ресурсы для поиска кандидатов не позволяют найти специалиста с требуемым набором компетенций и полноценно оценить его опыт, навыки, достижения и личностные характеристики. В результате, необходимым становится создание сервиса для поиска эксклюзивных специалистов. Большая часть подобных специалистов не имеют резюме в открытом доступе, не ищут работу, но готовы рассматривать интересные предложения. В результате, данная работа посвящена изучению возможности поиска специалистов с уникальными компетенциями в сети Интернет на основе анализа их цифрового следа. Гипотеза заключается в том, что возможно получить полный профиль уникального специалиста, если собрать, объединить и проанализировать данные из различных источников. В ходе настоящей работы были проанализированы возможности, предоставляемые открытыми источниками данных в сети Интернет, а также определены наукометрические показатели специалиста и параметры его благонадежности. Составлен алгоритм поиска требуемых специалистов на основе этих данных, спроектирована, разработана и протестирована автоматизированная система, реализующая данный поиск.

Ключевые слова: уникальный специалист; благонадежность; противоречивые данные; обобщенная линейная модель; сервис поиска

Для цитирования: Леонов А.С., Лаптев А.А., Лаушкина А.А., Синько М.В., Басов О.О. Алгоритм поиска специалистов с уникальными навыками на основе цифрового следа. Труды ИСП РАН, том 34, вып. 4, 2022 г., стр. 173-186. DOI: 10.15514/ISPRAS-2022-34(4)-12

Благодарности: Исследование выполнено за счет гранта Российского научного фонда (проект № 22-21-00604).

Algorithm for finding specialists with unique skills based on a digital footprint

[1] *A.S. Leonov, ORCID: 0000-0002-7904-182X <alex.thunder@tut.by>*
[1] *A.A. Laptev, ORCID: 0000-0003-1754-6643 <nickname.avast@gmail.com>*
[1] *A.A. Laushkina, ORCID: 0000-0001-7887-3299 <nastasjalausckina@mail.ru>*
[1] *M.V. Sinko, ORCID: 0000-0001-6075-9594 <michael.v.sinko@gmail.com>*
[1,2] *O.O. Basov, ORCID: 0000-0001-5788-4845 <oobasov@mail.ru>*

[1] *ITMO University,*
49-A, Kronverkskiy prospekt, St. Petersburg, 197101, Russia
[2] *Ivannikov Institute for System Programming of the Russian Academy of Sciences,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. In recent years, due to significant changes in the labor market, companies have become more likely to face various problems when searching and selecting candidates. The main reason for these problems is that the existing Internet resources for finding candidates do not allow you to find a specialist with the required set of competencies and fully evaluate his experience, skills, achievements and personal characteristics. As a result, it becomes necessary to create a service for finding exclusive specialists. Most of these specialists do not have a resume in the public domain, are not looking for a job, but are ready to consider interesting offers. As a result, this work is devoted to the study of the possibility of finding specialists with unique competencies on the Internet based on the analysis of their digital footprint. The hypothesis is that it is possible to get a complete profile of a unique specialist if you collect, combine and analyze data from various sources. In the course of this work, the possibilities provided by open data sources on the Internet were analyzed, as well as the scientometric indicators of a specialist and the parameters of his reliability were determined. An algorithm for searching for the required specialists based on these data has been compiled, an automated system implementing this search has been designed, developed and tested.

Keywords: unique specialist; trustworthiness; contradictory data; generalized linear model; search service

For citation: Leonov A.S., Laptev A.A., Laushkina A.A., Sinko M.V., Basov O.O. An algorithm for finding specialists with unique skills based on a digital footprint. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 4, 2022. pp. 173-186 (in Russian). DOI: 10.15514/ISPRAS-2022-34(4)-12

Acknowledgements. The research was carried out at the expense of a grant from the Russian Science Foundation (project No. 22-21-00604).

1. Введение

В последние годы рынок труда претерпел значительные изменения как со стороны работодателей, так и со стороны соискателей. Компании сталкиваются с различными проблемами при поиске и отборе кандидатов, поскольку существующие интернет-ресурсы не приносят результатов [1]. Часто кандидаты, которые соответствуют запросу работодателя, не заинтересованы в смене места работы, из-за чего их резюме отсутствует в общем доступе. Нестандартные интернет-платформы для поиска работников мало применяются организациями, поскольку данные ресурсы малоизвестны и не включают в себя необходимые требования, такие, как поиск по множеству предметных областей, а также возможность проведения анализа благонадежности кандидата. Сложности подбора специалистов усугубляются высоким процентом миграции выпускников университетов, ученых и исследователей за рубеж (см. рис.1).

Состояние современной экономики и условия трудового рынка создают несоответствие между требованиями работодателей и того, что могут предложить существующие ресурсы по поиску работников. Данные сервисы по большей части ориентированы на поиск работников, не менявших свою сферу деятельности и имеющих лишь одну специальность.

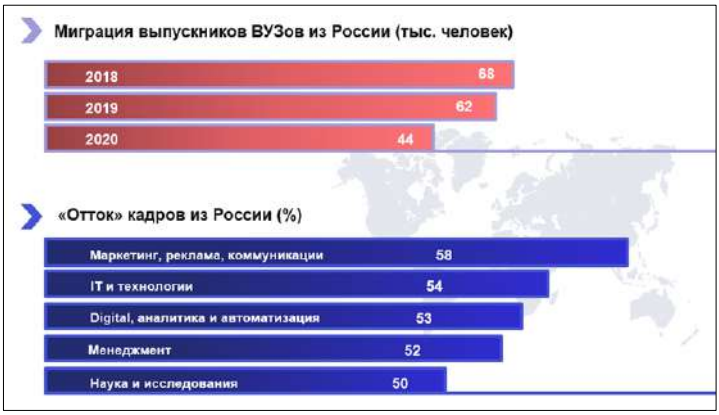


Рис. 1. Миграция выпускников ВУЗов из России и отток кадров
Fig. 1. Migration of university graduates from Russia and staff outflow

В результате наступившей пандемии COVID-19 начали появляться новые запросы и тенденции [2, 3, 4]. Согласно исследованию Boston Consulting Group (BCG), в начале 2021 года, 89% работников предпочтут удаленный формат работы. Такой формат позволяет привлекать к работе людей из разных точек мира, что делает работников, знающих иностранные языки и готовых к межкультурному сотрудничеству, более востребованными. Такие специалисты имеют уникальные междисциплинарные навыки и чаще всего их сложно найти. С внедрением новых информационно-коммуникационных технологий работодатели все чаще проводят проверку в социальных сетях, также известную как киберпроверка, как часть процесса найма [5]. Цифровая трансформация, ускоряющаяся под влиянием пандемии Covid-19, развивает и цифровые HR-технологии, использование которых необходимо в условиях усиливающейся конкуренции в борьбе за лучших сотрудников [6].

Поскольку текущие ресурсы не позволяют отбирать соответствующих запросу работодателя соискателей, поиск подобных сотрудников, которые обладают набором определенных навыков, усложняется и требует автоматизации — необходимо создать сервис по поиску эксклюзивных специалистов. Такой специалист не имеет резюме, не ищет работу, работает на стыке областей, а также имеет выраженный цифровой след. Автоматизированная система должна учитывать требования к искомому специалисту, предоставлять пользователю метрики благонадежности и общую оценку кандидата, находить с помощью различных источников, в том числе социальных сетей, недостающие данные.

Уникальный специалист – это специалист, который имеет уникальный набор знаний, компетенций и навыков в различных сферах [7].

Рассматривая уникального специалиста в научной сфере, уровень его квалификации можно определить благодаря его публикационной активности и наукометрическим показателям: цитируемость, количество соавторов, количество статей, h-индекс. Такие специалисты зачастую делятся своими работами и жизнью в социальных сетях [8]. Соответственно некоторые данные, которые невозможно узнать из базы научных статей, можно получить через цифровой след специалиста. Объединив данные из различных источников, можно получить некий описательный портрет найденного специалиста и с помощью цифрового следа дополнить недостающие данные.

Таким образом, целью работы является экспериментальное исследование возможности поиска человека с уникальными компетенциями в сети Интернет, используя ключевые слова и специально разработанный web-ресурс, а также возможность использования различных источников данных для составления цифрового портрета человека и принятия решения на основе полученных данных.

2. Объекты и методы исследования

Данные об уникальном специалисте могут быть неполными или отсутствовать, поэтому используется несколько общедоступных источников для получения информации о кандидате:

- база данных рецензируемой научной литературы Scopus [9];
- база данных рецензируемой научной литературы Web of Sciences;
- социальная сеть “ВКонтакте”;
- поиск в сети Интернет;
- Российский научный фонд (РНФ) [10];
- Федеральный институт промышленной собственности (ФИПС) [11].

С технической точки зрения характеристики уникального специалиста определяются соответствием его показателей набору ключевых слов, которые система получает как начальные данные от пользователя.

Алгоритм поиска информации о специалисте был условно разделен на первичный и вторичный поиск. Первичный – поиск основной информации. Вторичный – поиск дополнительной информации.

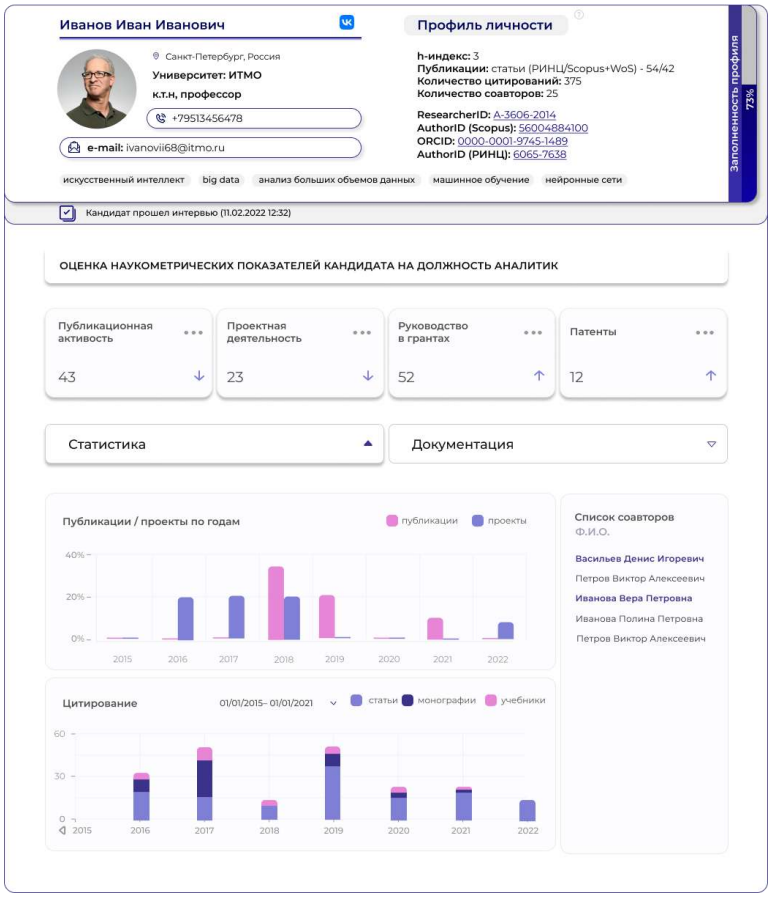


Рис. 2. Наукометрические показатели
Fig. 2. Scientometric indicators

На первичном этапе для поиска специалиста используются средство сбора данных “Модуль взаимодействия с API Scopus”. На вход подаются ключевые слова от пользователя, которые требуются для поиска специалиста. По данным ключевым словам, с помощью API

(Application Programming Interface) производится получение списка статей, по id которых загружается список авторов. Зная id автора, можно получить следующую информацию о нем: ФИО, Scopus ID, Orcid ID, места аффилиации, список публикаций за последние 10 лет, H-индекс, количество цитирований, список соавторов.

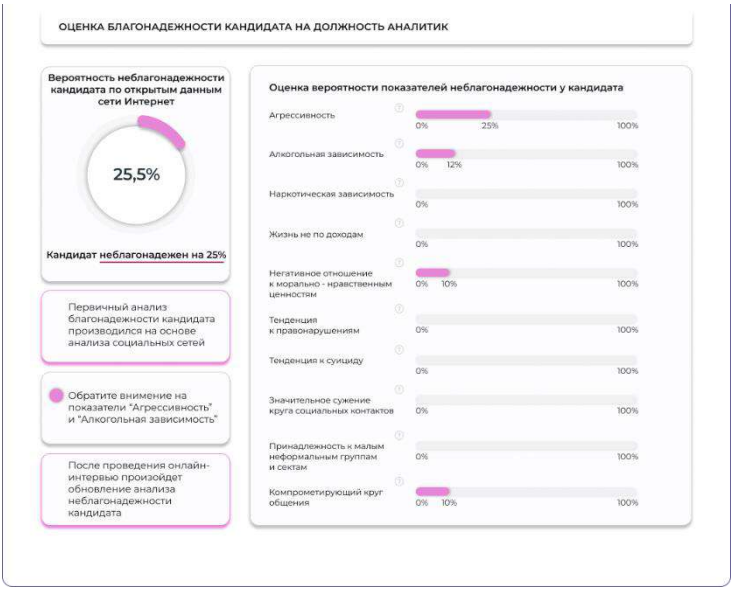


Рис. 3. Показатели благонадежности первичного анализа
Fig. 3. Reliability indicators of the primary analysis

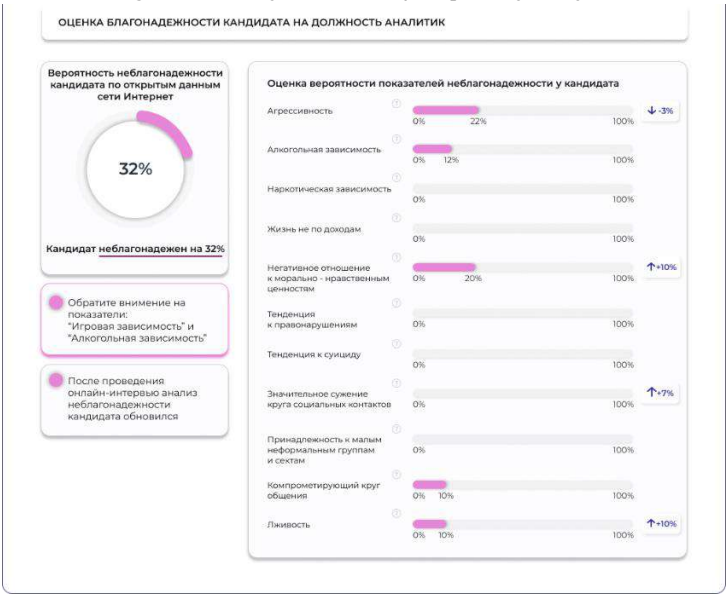


Рис. 4. Показатели благонадежности вторичного анализа
Fig. 4. Reliability indicators of secondary analysis

Алгоритм вторичного поиска информации использует следующие модули: модуль поиска информации о патентах РНФ и ФИПС, модуль поиска фото, модуль анализа страниц, модуль поиска по социальной сети ВК, модуль поиска в масс-медиа. На основе полученных данных и их анализа формируется цифровой профиль кандидата, который, помимо собранных

данных, включает в себя наукометрические показатели (см. рис. 2) и показатели неблагонадежности (см. рис. 3, 4).

Стрелки в блоках наукометрических данных кандидата показывают положение относительно других кандидатов в данной области. Графики наглядно отображают количество публикаций кандидата в научных журналах и его уровень цитируемости по введенным ключевым словам. Справа от графиков пользователь может ознакомиться со списком соавторов кандидата и при нажатии – ознакомиться с их профилем в Scopus.

2.1 Первичная оценка благонадежности кандидата

Благонадежность – соблюдение правил, законов и готовность согласиться с действующими требованиями общества/компании. Неблагонадежность – это поведение (или готовность к нему), в результате которого человек нарушает установленные нормы поведения. К показателям неблагонадежности относятся: уровень агрессивности, причастность к уголовным преступлениям, причастность к наркотикам, увлеченность азартными играми, причастность к финансовым махинациям [12, 13, 14, 15].

Показатель благонадежности кандидата определяется наличием или отсутствием факторов неблагонадежности в два этапа.

Оценка первичной благонадежности (ПБ) реализовывалась с помощью сбора данных из социальных сетей по:

1) агрессивности – на основе автоматического поиска слов ненормативной лексики;
2) алкогольной зависимости – на основе автоматического поиска слов, относящихся к категории алкоголь;
3) наркотической зависимости – по употреблению слов рядом со словами, относящихся к категории наркотики;
4) негативному отношению к морально-нравственным ценностям – анализ тональности по отношению к словам, которые могут быть употреблены в тексте постов, опубликованных на странице в социальной сети ВК;
5) игровой зависимости – по употреблению слов рядом со словами, относящихся к категории игр;
6) тенденциям к суициду – оценка депрессивного состояние по фотографиям из социальной сети Instagram;
7) наличию финансовых махинаций – наличие человека в реестре недобросовестных лиц (ФНС).

Формула подсчета оценки первичной благонадежности (1):

$$\text{ПБ} = \sum(\text{КОЭФ.ЗНАЧ}_i * \text{Б1}_i), \quad (1)$$

где Б1_i – благонадежность до интервью по параметрам i из пунктов 1-7, КОЭФ.ЗНАЧ_i – коэффициент значимости параметра i, который выбирает сам пользователь системы.

На первом этапе использовался автоматизированный сбор и анализ постов из социальной сети кандидата [16, 17]. Было предположено, что алгоритмы машинного обучения на основе различных показателей, извлеченных из активности пользователей социальных сетей, способны предсказывать типы/черты личности с высокой точностью [18, 19]. Для оценки типа личности кандидата было предложено моделировать результаты прохождения кандидатом теста Айзенка версии EPI [20]. По количеству информации, публикуемой в социальных сетях, были отобраны тренировочная и контрольная группы по 120 и 30 человек соответственно для создания обучающего набора данных и оценки эффективности алгоритмов. (см. рис. 5). В 99,3% случаях алгоритм KNN (k-nearest neighbors) верно определял тип личности.

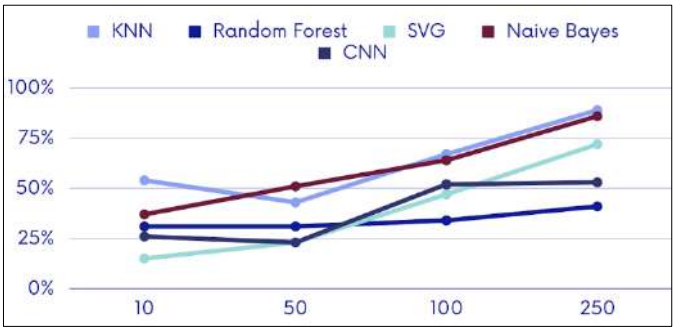


Рис. 5. Сравнение точности алгоритмов
Fig. 5. Comparison of algorithm accuracy

2.2 Вторичная оценка благонадежности кандидата

Оценка вторичной благонадежности (ВБ) реализовывалась с помощью сбора данных на основе онлайн-интервью, в процессе которого выделялись аналогичные показатели, включая оценку лживости:

1) агрессивность – на основе выделения поведенческих маркеров вербальной и невербальной агрессии, согласно классификации Басса-Дарки;
2) алкогольная зависимость – на основе оценки истинности и ложности ответов на поставленные вопросы (Как вы думаете, насколько частое употребление алкоголя и какого может являться алкогольной зависимостью? Аргументируйте это. Как часто вы употребляете алкоголь?);
3) наркотическая зависимость – на основе оценки истинности и ложности ответов на поставленные вопросы (Считаете ли вы зависимость проблемой? Есть ли в вашем окружении люди, которые употребляют или употребляли наркотические вещества? Как вы относитесь к людям, которые употребляют наркотические вещества? Аргументируйте свою позицию);
4) негативное отношение к морально – нравственным ценностям - на основе оценки истинности и ложности ответов на поставленные вопросы (Как вы относитесь к {фактор был признан негативным}?);
5) игровая зависимость – на основе оценки истинности и ложности ответов на поставленные вопросы (Считаете ли вы игровую зависимость социальной проблемой? Что по вашему мнению, является игровой зависимостью? Если ли в вашем окружении люди, о которых можно сказать, что у них есть игровая зависимость? Как вы относитесь к людям, которые играют в игры больше четырех раз в день? Аргументируйте свою позицию);
6) тенденции к суициду – на основе оценки истинности и ложности ответов на поставленные вопросы (Были ли у вас случаи глубокой депрессии? Расскажите об этом Как по вашему мнению, есть ли оправдание самоубийству? Аргументируйте свою позицию);
7) тенденции к правонарушениям – на основе оценки истинности и ложности ответов на поставленные вопросы (Есть ли у вас родственники, которые были судимы? Расскажите об этом. Бывали ли у вас случаи нарушения закона? Есть ли у вас факты административного правонарушения? Расскажите об этом.) В случае, если человек был найден в реестре недобросовестных лиц, то также задается вопрос: “Расскажите, пожалуйста, почему вы были включены в реестр дисквалифицированных лиц?”;
8) лживость – оценка истинности или ложности ответов кандидата на основе анализа вербальных и невербальных признаков онлайн-интервью.

Формула подсчета оценки вторичной благонадежности (2):

$$\text{ПБ} = \sum(\text{КОЭФ.ЗНАЧ}_i * \text{Б2}_i), \quad (2)$$

где Б2_i – благонадежность до интервью по параметрам i из пунктов 1-8, КОЭФ.ЗНАЧ_i – коэффициент значимости параметра i, который выбирает сам пользователь системы.

Оценка истинности или ложности ответов во втором этапе заключалась в устранении противоречий на основе анализа вербальных и невербальных признаков. Для исследования использовались видеоматериалы общей продолжительностью более 15 часов, которые были собраны в процессе интервьюирования 88 человек.

Для анализа видеоряда был использован вектор мимики лица с двумя компонентами. Первый компонент - это вектор выражения лица, второй компонент - это вектор, который содержит углы поворота головы. Для получения данных векторов используется сверточная нейронная сеть BlazeFace. Определение пространственных координат лицевых ориентиров вокруг осей трехмерного пространства позволяет построить карту глубины изображения для определения рельефа лица, поворота и наклона головы человека. Поток полученных карт объектов содержит информацию об изменениях выражений лица с течением времени и позволяет исследовать ее.

Для извлечения и анализа аудиоряда был разработан алгоритм, выделяющий следующие особенности речевого сигнала:

1) доля повышения/понижения громкости голоса для каждого фрагмента относительно среднего значения;
2) доля повышения/понижения тона голоса для каждого фрагмента относительно среднего значения;
3) уровень дрожания частоты (jitter);
4) значение темпа (количество произносимых слов в минуту);
5) доля сигнала, не связанного с речью (зашумленность аудио);
6) категориальный признак (молчание);
7) класс эмоций (7 классов: гнев, скука, тревога, радость, печаль, отвращение, отсутствие эмоций.

После чего была обучена модель сверточной нейронной сети с полностью подключенным классификатором (на наборе данных Emo-DB), который принимает аудио сигнал в качестве входного значения в форме MFCC для извлечения класса эмоций из спектрограммы. Класс эмоций извлекается ежесекундно, фрагменты меньшей длительности дополняются нулями до требуемого размера.

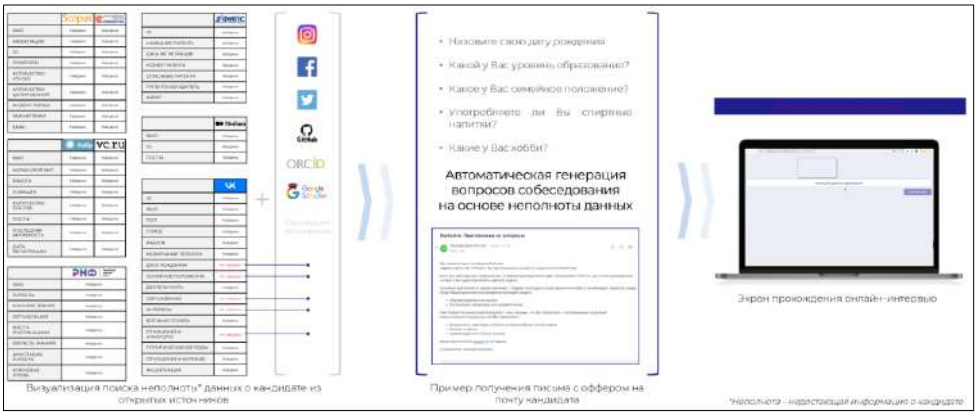


Рис. 6. Генерация персонализированных вопросов
Fig. 6. Generating personalized questions

Далее происходит конкатенация извлеченных векторов видео и аудио признаков в одну последовательность. После чего эта последовательность передается на вход нейронной сети класса “Трансформер”, которая на выходе имеет два значения: “правда” или “ложь”.

Сбор данных для второго этапа производился в несколько итераций интервьюирования. На первой – кандидату требовалось ответить на ряд общих вопросов. На основе анализа результатов, полученных после первого интервью, производилась генерация персонализированных вопросов для второй итерации (см. рис. 6).

После прохождения и анализа результатов интервьюирования, информация об уровне неблагонадежности и типе личности в его карточке обновляется и дополняется. Уведомление о прохождении опроса и результатах анализа отправляются на почту пользователю, который пригласил кандидата.

3. Разработка программного решения

В качестве архитектуры программной системы был выбран микросервисный подход на основе REST API, который заключается в том, что вся система разбивается на некоторое количество автономных изолированных программных модулей, обладающих слабым зацеплением и сильной связностью. Данный подход помогает реализовать архитектуру любой сложности в силу того, что сами микросервисы слабо зависят друг от друга в функциональном плане, важен лишь только формат данных, который они передают между собой.

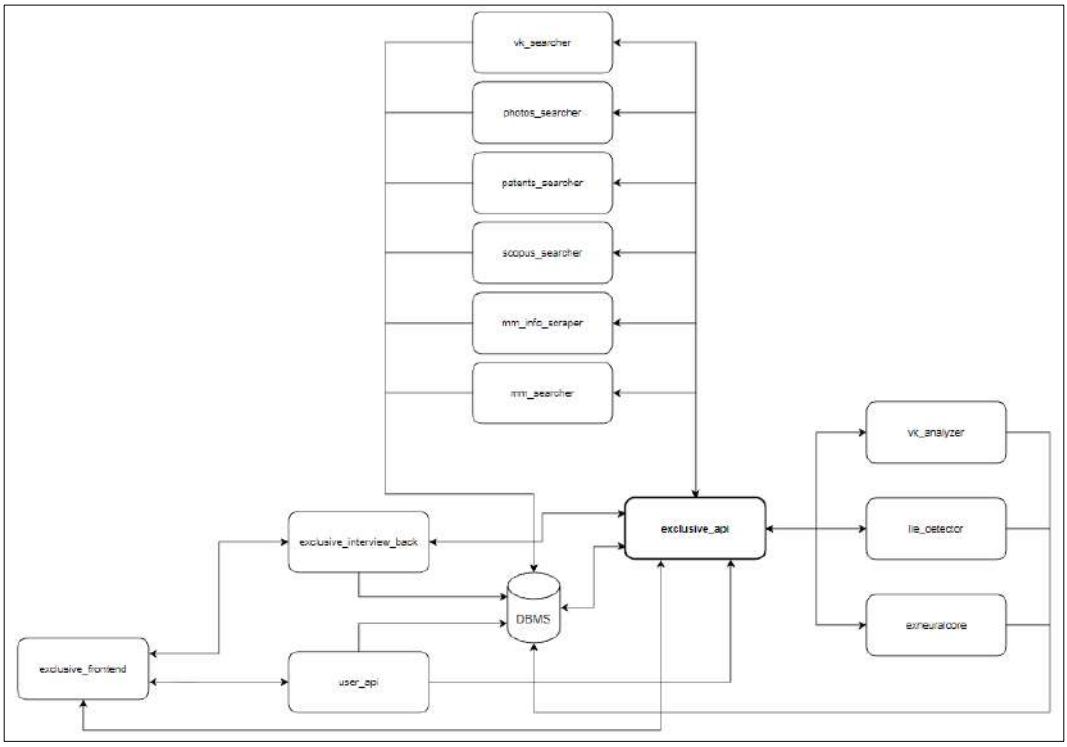


Рис 7. Архитектура программного решения
Fig. 7. Architecture of the software solution

Архитектура программного решения (см. рис. 7) представляет из себя конвейер из микросервисов и СУБД (Системы Управления Базами Данных) (DBMS). В состав системы входят следующие микросервисы [21, 22, 23]:

- exclusive_api – главный (управляющий) микросервис, который принимает запросы на поиск, запускает другие микросервисы и получает от них результат работы;
- exclusive_frontend – клиентская часть системы;
- exclusive_interview_back – микросервис, который отвечает за запись видео во время интервью и сохранение его, а также ответов кандидата, в DBMS;
- exneuralcore – клиентская часть сервиса;
- lie_detector – микросервис, который анализирует видеоответ кандидата и определяет, лгал человек или нет;
- mm_info_scraper – микросервис, извлекающий данные о человеке из кода страниц web-сайтов;
- mm_searcher – микросервис поиска информации о человеке на сайтах habr.com, vc.ru, medium.com;
- patents_searcher – микросервис поиска патентов человека на сайтах РНФ и ФИПС;
- photos_searcher – микросервис поиска фотографий кандидата на сайтах его аффиляций;
- scopus_searcher – микросервис поиска статей и авторов посредством Scopus API;
- vk_searcher – микросервис поиска профиля человека в социальной сети ВКонтакте;
- vk_analyzer – микросервис анализа профиля человека, найденного в социальной сети ВКонтакте;
- user_api – микросервис, отвечающий за пользовательскую часть сайта: регистрация, авторизация, восстановление паролей, заполнение информации о пользователе;
- DBMS – система управления баз данных, которая хранит в себе БД с информацией о кандидатах.

Разработка серверной части велась на языке программирования Python 3 и использованием фреймворка FastAPI, а также таких инструментов как git, Docker, Nginx, MySQL. Клиентская часть была написана с использованием фреймворка React.js и языка программирования JavaScript.

4. Заключение

В ходе данной работы были изучены возможности поиска специалистов с уникальными компетенциями в сети Интернет на основе цифрового следа. В основу работы была положена гипотеза о том, что специалисты с уникальными компетенциями имеют цифровой след, по которому можно собрать информацию о них, достаточную для того, чтобы предоставить работодателю для принятия решения о найме.

Для того, чтобы проверить эту гипотезу, было проведено экспериментальное исследование возможности поиска человека с уникальными компетенциями в сети Интернет на основе заданных ключевых слов, а также возможность использования различных источников данных для составления цифрового портрета человека.

В рамках достижения цели исследования было решено разделить поиск специалиста на 2 этапа, первичный и вторичный.

В ходе первичного поиска происходит поиск информации по заданным ключевым словам. В качестве источника данных для первичного поиска был выбран Scopus API, который позволяет получить информацию о научных статьях и их авторах на основе заданных ключевых слов, что полностью соответствует поставленной задаче.

В ходе вторичного поиска происходит дополнение информации, полученной в ходе первичного поиска. Для каждого найденного человека производится:

- поиск фотографии через поисковую систему Google на основе его ФИО и сайта аффиляции;
- поиск контактных данных на основе страницы, с которой была получена фотография;
- поиск патентов в системах РНФ и ФИПС;

- поиск профиля в социальной сети ВКонтакте;
- поиск упоминаний на сайтах масс-медиа, в качестве таковых были выбраны habr.com, vc.ru, medium.com.

Было реализован веб-ресурс, предоставляющий пользователю интерфейс для поиска специалистов по указанным алгоритмам и вывод найденной информации в виде карточек кандидатов. Кроме того, ресурс позволяет работодателю отправлять приглашение на онлайн-интервью заинтересовавшему его кандитату, а также принимать решение о дальнейшем найме на основе анализа вербальных и невербальных характеристик кандидата с учетом оценки его благонадежности.

На данный момент сервис позволяет находить кандидатов только из научной сферы деятельности, что позволяет оценить узкий круг специалистов. В дальнейшем полученную систему можно усовершенствовать путем добавления новых источников данных и расширения предметной области, что позволит получать еще более полные данные о людях с уникальными компетенциями.

Список литературы / References

[1]. Dekay S. Are business-oriented social networking web sites useful resources for locating passive jobseekers? Results of a recent study. Business Communication Quarterly, vol. 72, issue 1, 2009, pp. 101-105.

[2]. Романович М. А., Мохсени Х. и др. Актуальные вопросы и анализ рынка образовательных услуг в постпандемийный период. Экономика. Информатика, том 48, вып. 4, 2022 г., стр. 717-725 / Romanovich M.A., Hossein Mohseni H. et al. 2021. Current issues and analysis of the educational services market in the post-pandemic period. Economics. Information technologies, vol. 48, issue 4, 2022, pp. 717-725 (in Russian).

[3]. Jinghui Y. The Effects of COVID-19 on the Labor Market. Journal of Economics Business and Management, vol. 10, issue 1, 2022, pp. 65-71.

[4]. Ando S., Balakrishnan R. et al. European Labor Markets and the COVID-19 Pandemic: Fallout and the Path Ahead. International Monetary Fund, issue 4, 2022, 96 p.

[5]. Jacobson J., Gruzd A. Cybervetting job applicants on social media: the new normal? Ethics and Information Technology, vol. 22, issue 2, 2020, pp. 175-195.

[6]. Peshkova O.A. Digital Footprint Analysis Technology: Some Aspects of Its Application in Recruitment. Lecture Notes in Networks and Systems, vol. 397, 2022, pp. 368-375.

[7]. Marin G.D., Nilă C. Branding in social media. Using LinkedIn in personal brand communication: A study on communications/marketing and recruitment/human resources specialists perception. Social Sciences & Humanities Open, vol. 4, issue 1, 2021, article no. 100174.

[8]. Bohnert D., Ross W.H. The influence of social networking web sites on the evaluation of job candidates. Cyberpsychology, Behavior, and Social Networking, vol. 13, issue 3, 2010, pp. 341-347.

[9]. Индексы научного цитирования / Science citation indices. Available at: https://www.econ.msu.ru/elibrary/consulting/index/ (in Russian), accessed 04.05.2022.

[10]. Российский научный фонд, Создавая фундамент будущего / Russian Science Foundation. Laying Groundwork for Future. Available at: https://www.rscf.ru/, accessed 05.05 2022.

[11]. Федеральный институт промышленной собственности / Federal Institute of Industrial Property. Available at: https://www.fips.ru/, accessed 05.05.2022.

[12]. Иванова О.Ф. Проявления агрессии у представителей различных национальных культур. Вестник Евразии, no. 1, 2004 г., стр. 34-54 / Ivanova O.F. Manifestations of aggression among representatives of various national cultures. Bulletin of Eurasia, no. 1, 2004, pp. 34-54 (in Russian).

[13]. Bohnert D., Ross W.H. The influence of social networking web sites on the evaluation of job candidates. Cyberpsychology, Behavior, and Social Networking, vol. 13, issue 3, 2010, pp. 341-347.

[14]. Шумский П.П., Иванова Л.Н. Эмоции как процесс, влияющий на деятельность человека. Мозырь. 1998, 324 стр. / Shumsky P.P., Ivanova L.N. Emotions as a process that affects human activity. Mozyr. 1998, 324 p. (in Russian).

[15]. Яшенева А.С. Психолингвистические признаки агрессивной речи. Наука и образование сегодня, том 8, вып. 19, 2017, стр. 67-69 / Yasheneva A.S. Psycholinguistic signs of aggressive speech. Science and Education Today, vol. 8, issue. 19, 2017, pp. 67-69 (in Russian).

[16]. Kowsari K., Meimandi K.J. et al. Text classification algorithms: A survey. Information, vol. 10, issue 4, 2019, 68 p.
[17]. Rachman F.H., Sarno R., Fatichah C. CBE: Corpus-based of emotion for emotion detection in text document. In Proc. of the 3rd International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE), 2016, pp. 331-335.
[18]. Lima A.C., de Castro L.N. A multi-label, semi-supervised classification approach applied to personality prediction in social media. Neural Networks, vol. 58, 2014, pp. 122-130.
[19]. Pratama B. Y., Sarno R. Personality classification based on Twitter text using Naive Bayes, KNN and SVM. In Proc. of the International Conference on Data and Software Engineering (ICoDSE), 2021, pp. 170-174.
[20]. Тест-опросник Г. Айзенка epi (адаптация Г. Шмелева)/ Test questionnaire epi by G. Eysenck (adapted by G. Shmelev). Available at: http://test-metod.ru/index.php/metodiki-i-testy/1/48-test-oprosnik-g-ajzenka-epi-adaptatsiya-g-shmeleva (in Russian), accessed 04.05.2022.
[21]. Newman S. Newman S. Building Microservices: Designing Fine-Grained Systems. O'Reilly Media, 2021, 280 p.
[22]. Richardson C. Microservices patterns: with examples in Java. Manning, 2018, 540 p.
[23]. Richardson L., Amundsen M., Ruby S. RESTful Web APIs: Services for a Changing World. O'Reilly Media, 2013. 406 p.

Информация об авторах / Information about authors

Александр Сергеевич ЛЕОНОВ – магистрант 2-го года обучения факультета ИКТ Университета ИТМО, инженер исследовательского центра в сфере искусственного интеллекта "Сильный искусственный интеллект в промышленности". Сфера научных интересов: информационные и когнитивные технологии.

Alesander Sergeevich LEONOV is a 2nd year master's student at the Faculty of the Faculty of Infocommunication Technologies at ITMO University, an engineer at the research center in the field of artificial Intelligence "Strong Artificial Intelligence in Industry". Research interests: information technology.

Андрей Александрович ЛАПТЕВ – аспирант 2-го года обучения факультета ЦТ Университета ИТМО, инженер исследовательского центра в сфере искусственного интеллекта "Сильный искусственный интеллект в промышленности". Его научные интересы включают области computer science, проектирование сложных систем, построение математических моделей.

Andrey Aleksandrovich LAPTEV is a PhD student at the Faculty of Digital Transformation of ITMO University, an engineer at the research center in the field of artificial intelligence "Strong Artificial Intelligence in Industry". His research interests include computer science, design of complex systems, construction of mathematical models.

Анастасия Александровна ЛАУШКИНА – аспирант 2-го года обучения факультета ЦТ Университета ИТМО, инженер исследовательского центра в сфере искусственного интеллекта "Сильный искусственный интеллект в промышленности". Научные интересы включают когнитивную психологию, построение математических моделей.

Anastasia Alexandrovna LAUSHKINA is a PhD student at the Faculty of Digital Transformation of ITMO University, an engineer of the research center in the field of artificial intelligence "Strong Artificial Intelligence in Industry". His research interests include cognitive psychology, the construction of mathematical models.

Михаил Витальевич СИНЬКО – аспирант 2-го года обучения факультета ЦТ Университета ИТМО, инженер исследовательского центра в сфере искусственного интеллекта "Сильный искусственный интеллект в промышленности". Сфера научных интересов: компьютерное зрение, обработка аудиосигналов, генеративные состязательные сети, рекомендательные модели.

Mikhail Vitalievich SINKO is a PhD student at the at the Faculty of Digital Transformation of ITMO University, an engineer of the research center in the field of artificial intelligence "Strong Artificial Intelligence in Industry". Research interests: computer vision, audio signal processing, generative adversarial networks, recommendation models.

Олег Олегович БАСОВ – доктор технических наук, профессор факультета цифровых трансформаций Университета ИТМО, специалист, 02.1 сектор сопровождения научно-исследовательских работ и образовательной деятельности, Федеральное государственное бюджетное учреждение науки Институт системного программирования им. В.П. Иванникова Российской академии наук. Сфера научных интересов: полимодальные инфокоммуникационные системы, квантовые коммуникации, интеллектуальные транспортные системы, беспилотные летательные аппараты, робототехника.

Oleg Olegovich BASOV – Doctor of Technical Sciences, Professor of the Faculty of Digital Transformations at ITMO University, specialist, 02.1 sector of support for Research and Educational activities, Federal State Budgetary Institution of Science V.P. Ivannikov Institute of System Programming of the Russian Academy of Sciences. Research interests: polymodal infocommunication systems, quantum communications, intelligent transport systems, unmanned aerial vehicles, robotics.

DOI: 10.15514/ISPRAS-2022-34(4)-13

Методы и подходы к автоматическому связыванию сущностей на русском языке

[1,2] *А.А. Мезенцева, ORCID: 0000-0003-2159-5771 <anastmezentseva@gmail.com>*
[1,2] *Е.П. Бручес, ORCID: 0000-0002-1055-5339 <bruches@bk.ru>*
[1] *Т.В. Батура, ORCID: 0000-0003-4333-7888 <tatiana.v.batura@gmail.com>*

[1] *Институт систем информатики им. А. П. Ершова СО РАН,*
630090, Россия, г. Новосибирск, пр. Лаврентьева, д. 6
[2] *Новосибирский государственный университет,*
630090, Россия, г. Новосибирск, ул. Пирогова, д. 1

Аннотация. На сегодняшний день большое внимание уделяется решению задач обработки текстов с использованием информации об окружающем нас мире, например, в информационном поиске, построении вопросно-ответных и диалоговых систем. Поэтому важно установить соответствие между сущностями в обрабатываемом тексте и базой знаний. Данная статья посвящена автоматическому связыванию сущностей с Вики-данными. В качестве сущностей рассматриваются научные термины на русском языке. Традиционно система связывания сущностей состоит из трёх этапов: распознавание сущностей, генерация кандидатов и ранжирование кандидатов. Наша система принимает на вход текст, в котором уже выделены термины. Для генерации кандидатов мы используем построковое совпадение терминов и сущностей в базе знаний. Этап ранжирования кандидатов является наиболее сложным, так как требует использования семантической информации. Проведены эксперименты с различными подходами к решению этой задачей: с использованием косинусной близости, классическими методами машинного обучения и нейронными сетями. Также мы расширили корпус RUSERRC, добавив вручную размеченные данные для обучения моделей. Полученные результаты показали, что использование метода, основанного на косинусной близости, позволяет получить не только более высокие результаты, по сравнению с другими подходами, но и решать эту задачу без вручную размеченных данных. Набор данных и код находятся в открытом доступе и доступны для других исследователей.

Ключевые слова: связывание сущностей; база знаний; научные термины

Для цитирования: Мезенцева А.А., Бручес Е.П., Батура Т.В. Методы и подходы к автоматическому связыванию сущностей на русском языке. Труды ИСП РАН, том 34, вып. 4, 2022 г., стр. 187-200. DOI: 10.15514/ISPRAS-2022-34(4)-13

Methods and techniques to automatic entity linking in Russian

[1,2] *A.A. Mezentseva, ORCID: 0000-0003-2159-5771 <anastmezentseva@gmail.com>*
[1,2] *E.P. Bruches, ORCID: 0000-0002-1055-5339 <bruches@bk.ru>*
[1] *T.V. Batura, ORCID: 0000-0003-4333-7888 <tatiana.v.batura@gmail.com>*

[1] *A. P. Ershov Institute of Informatics Systems,*
6, Acad. Lavrentjev pr., Novosibirsk 630090, Russia
[2] *Novosibirsk State University,*
st. Pirogova, д. 1, Novosibirsk, 630090, Russia

Abstract. Nowadays, there is a growing interest in solving NLP tasks using external knowledge storage, for example, in information retrieval, question-answering systems, dialogue systems, etc. Thus it is important to establish relations between entities in the processed text and a knowledge base. This article is devoted to entity

linking, where Wikidata is used as an external knowledge base. We consider scientific terms in Russian as entities. Traditional entity linking system has three stages: entity recognition, candidates (from knowledge base) generation, and candidate ranking. Our system takes raw text with the defined terms in it as input. To generate candidates we use string match between terms in the input text and entities from Wikidata. The candidate ranking stage is the most complicated one because it requires semantic information. Several experiments for the candidate ranking stage with different models were conducted, including the approach based on cosine similarity, classical machine learning algorithms, and neural networks. Also, we extended the RUSERRC dataset, adding manually annotated data for model training. The results showed that the approach based on cosine similarity leads to better results compared to others and doesn't require manually annotated data. The dataset and system are open-sourced and available for other researchers.

Keywords: entity linking; knowledge base; scientific terms

For citation: Mezentseva A.A., Bruches E.P., Batura T.V. Methods and techniques to automatic entity linking in Russian. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 4, 2022. pp. 187-200 (in Russian). DOI: 10.15514/ISPRAS-2022-34(4)-13

1. Введение

Сегодня в области обработки естественного языка большое внимание уделяется подходам, позволяющим использовать внешние источники знаний для решения тех или иных практических задач: вопросно-ответные системы, автоматическое реферирование, машинный перевод и пр. В качестве таких источников чаще всего используются графы знаний, содержащие информацию о сущностях, а также их атрибутах и семантических отношений между сущностями. Примерами таких графов знаний являются Freebase [1], DBpedia [2], Вики-данные.

Для использования таких структур встаёт задача автоматического связывания сущностей (Entity Linking), в рамках которой нужно соотнести выделенную сущность во входном тексте с одной из сущностей в графе знаний. Сложность задачи заключается в том, что часто одна и та же сущность может быть по-разному выражена в тексте (например, "Python", "пайтон", "питон"), а также быть многозначной (например, извлечённая сущность из текста "NLP" может быть связана с сущностями из графов знаний "Natural Language Processing", "Non-linear programming", "Neuro-lingustic programming" и пр., в зависимости от контекста). Научная новизна данной работы заключается в следующем.

- Создан набор данных для решения задачи связывания сущностей, в котором термины из научных текстов на русском языке связаны с Вики-данными, если это возможно.
- Проведено сравнение нескольких подходов к связыванию сущностей, основанных на классическом машинном обучении и моделях глубокого обучения.

2. Обзор методов

При традиционном подходе решение данной задачи разбивается на два этапа: генерация множества кандидатов из сущностей из базы знаний, а затем ранжирование выбранных кандидатов [3, 4, 5].

С активным развитием систем конечного цикла (end-to-end), стали появляться решения этой задачи без разделения на два этапа. Так, в работе [6] описывается подобная система связывания сущностей.

Другим популярным направлением становится решение задачи в постановке zero-shot learning, т.е. создание системы без обучающих примеров. Такая идея описывается в работах [7, 8].

Кроме того, важным аспектом становится решение этой задачи без привязки к конкретному языку. Для подобных исследований в статье [9] предлагается корпус для оценки таких методов, содержащий данные на 100 языках, а также оценка подходов zero-shot learning и few-shot learning, применяемых ко всем языкам. В работе [10] показано, что межъязыковая

постановка задачи более сложная, чем мультиязыковая. Авторами описан опыт создания набора данных, в котором связаны статьи из Википедии с событиями из Вики-данных. Эксперименты проводились с базовым подходом для ранжирования BM25 и моделями на основе BERT.

Также появляются работы, решающие задачу связывания сущностей не только с использованием текста, но и изображений [11]. В статье описан размеченный авторами корпус новостных статей на английском языке с различными типами сущностей. Также было показано, что задача мультимодального связывания является значительно более сложной, чем связывания только текстовой или визуальной информации, к тому же, отсутствие текстового или визуального контекста приводит к снижению метрик.

3. Формальная постановка задачи

Данная работа посвящена задаче связывания сущностей, где в качестве сущностей рассматриваются термины.

Назовём *Entities* множество сущностей и *Properties* множество свойств. База знаний состоит из множества троек вида *<Subject Predicate Object>*, где *Subject* и *Object* являются элементами множества *Entities*, а *Predicate* – элементом множества *Properties*.

Назовём токеном x_i – слово или знак препинания в тексте. Рассмотрим последовательность токенов $X = \{x_1, x_2, \dots, x_n\}$. Сущностью *Ent* будет называться подпоследовательность таких токенов, которая представляет собой термин. Тогда мощность множества *E*, которое содержит в себе сущности *Ent*, всегда меньше либо равна мощности множества *X*, включая значение 0.

Задача автоматического связывания сущности состоит в построении такой функции *F*, которая для каждой сущности из множества *E* ставила бы в соответствие элемент из множества *Entities*, либо ε, где ε – отсутствие сущности в заданной базе знаний:

$$F: E \to Entities \cup \varepsilon.$$

4. Описание данных

4.1 Создание вручную размеченного корпуса

В открытом доступе существует корпус из 1680 научных статей в области информационных технологий на русском языке RuSERRC, в 80 текстах вручную размечены термины, отношения между ними, указаны ссылки на сущности из Викиданных [12]. Этот набор данных применялся только для тестирования системы и никак не использовался на этапе обучения моделей.

Табл. 1. Статистика по вручную размеченного корпуса
Table 1. Manually annotated set statistics

Параметр	Количество
Тексты	136
Токены	12809
Термины	2028
Ссылки на сущности из Вики-	938

Для обучения моделей нам потребовался дополнительный вручную размеченный корпус. Чтобы получить его, были выполнены следующие шаги:

1) выгружены тексты аннотаций научных статей по информационным технологиям на русском языке из журнала «Программные продукты и системы»;
2) произведена авторазметка имеющимся модулем [13], где ранжирование реализовано с помощью косинусного расстояния между векторами;
3) исправлены ошибки авторазметки, а именно, для каждой определенной системой из п.2 сущности определялось, подходит ли название и описание к термину, упомянутому в тексте, и если не подходит, то нужно предложить верную сущность из базы знаний.

Корпус находится в открытом доступе, его статистические параметры указаны в табл. 1.

4.2 Подготовка данных для проведения экспериментов

В данной статье задача выбора наиболее релевантной сущности рассматривается как бинарная классификация. Для обучения бинарного классификатора необходим набор данных, в котором содержались бы позитивные (1) и негативные (0) примеры. Позитивные примеры были взяты из размеченного корпуса, описанного выше. Неподходящие кандидаты (“нули”) получились в результате применения существующего модуля генерации (подробнее в разделе 4), исключением уже полученных кандидатов из корпуса. Различную статистическую информацию о полученном корпусе можно найти в табл. 2.

Табл. 2. Статистика по корпусу для обучения моделей
Table 2. Training set statistics

Параметр	**Количество**
Экземпляры класса 1	637
Экземпляры класса 0	23552
Уникальные термины	1645
Минимальная длина контекста	6
Максимальная длина контекста	21
Средняя длина контекста	13
Идентификаторы из Вики-данных	24189
Уникальные идентификаторы	6586

5. Описание системы

В ходе исследования была разработана система для связывания сущностей. В общем виде алгоритм состоит из следующих этапов.

1) На вход подается термин и его контекст (пять токенов до и после). Выполняется лемматизация без учёта контекста (все слова приводятся в нижний регистр).
2) Генерируются запросы, чтобы получить список униграмм, биграмм и триграмм из названия входного упоминания.
3) Генерируется список кандидатов для выполнения поиска по построковому совпадению с основным и альтернативными названиями сущности и в дампе Вики-данных.
4) Выполняется ранжирование кандидатов с использованием алгоритма получения коэффициентов для оценки того, насколько кандидат является подходящим. Также были проведены эксперименты с использованием дополнительных весов, для того чтобы учесть количество совпадающих токенов в кандидате и в упоминании сущности.

5) На выходе алгоритм выдает наиболее релевантный идентификатор сущности из Викиданных.

Иллюстрация взаимодействия модулей системы представлена на рис. 1.

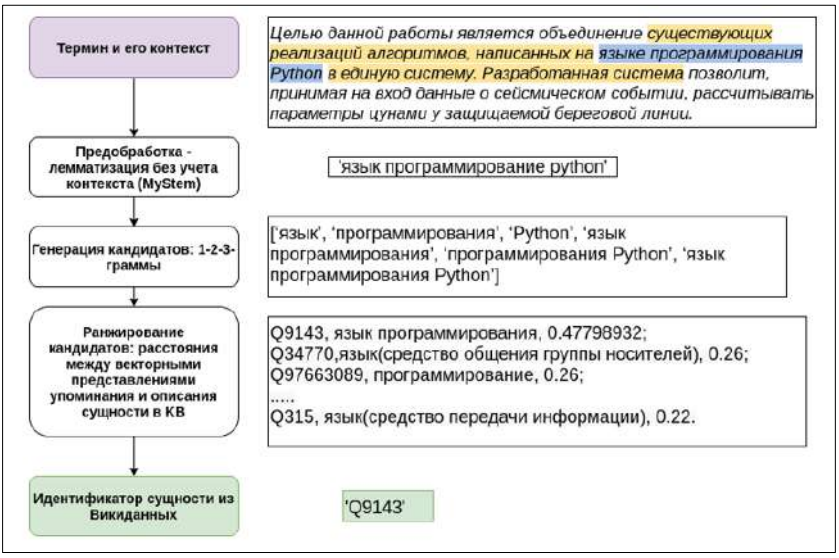


Рис. 1. Схема работы системы
Fig. 1. General architecture of the system

6. Эксперименты

6.1 Базовый подход

В качестве базового алгоритма ранжирования было использовано косинусное расстояние между векторами, полученными из предобученной модели DeepPavlov. Для кандидатов, не имеющих подходящей сущности в базе знаний, был введен порог, равный 0.35. Первый вектор – потокенно усредненный для контекста входного упоминания, состоящий из n токенов до и после, где n = 5. Второй вектор – так же усредненный для названия, описания и синонимов сущности-кандидата из базы знаний. Результаты для этого подхода представлены в первой строке табл. 5.

6.2 Генерация векторов признаков

Модели машинного обучения принимают на вход вектора признаков. Для разных моделей использовались разные способы получения этих векторов.

Для моделей классического машинного обучения и перцептрона использовалась конкатенация усреднённого вектора входного термина (и его контекста), а также названия (и описания) сущности-кандидата из базы знаний. На рис. 2 представлена общая схема получения векторов.

Пусть A – последовательность токенов;

$len(A)$ – количество элементов последовательности A;

$FastText(w)$ – вектор для токена w, взятый из предобученной модели FastText.

Тогда для последовательности A потокенно усредненный вектор будет вычисляться по формуле:

$$WordMean(A) = \sum_{i=1}^{len(A)} \frac{FastText(A[i])}{len(A)}.$$

В этом случае векторы признаков, подаваемые на вход модели машинного обучения, будут вычисляться по формуле:

$$features = WordMean(X) \cdot WordMean(Y), \text{ где}$$

$\cdot$ – операция конкатенации;

X – последовательность токенов термина и контекста;

Y – название и описание сущности-кандидата из базы знаний.

Затем к конкатенации были добавлены общий усреднённый вектор данных векторов или косинусное расстояние между данными векторами, т.е.

$$features_{mean} = features \cdot \frac{WordMean(X) + WordMean(Y)}{2};$$

$$features_{cosine} = features \cdot cosine_{distance}\big(WordMean(X), WordMean(Y)\big), \text{ где}$$

$cosine_{distance}(X, Y)$ – функция определения косинусного расстояния между векторами.

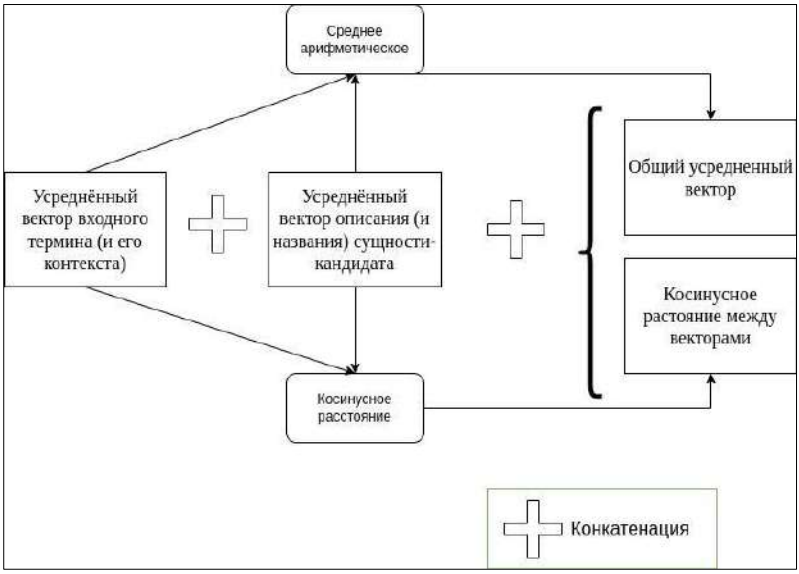


Рис. 2. Схема генерации векторов признаков
Fig. 2. Feature vector generation scheme

Для моделей со сверточными слоями на входе использовалась матрица фиксированного размера – была взята максимальная длина в 60 токенов. В векторы преобразовывалась следующая последовательность: термин, контекст, описание кандидата и альтернативные названия, т.е.

$$M = [FatText(X[1]), FastText(X[2]), \dots, FastText(Y[i-1]), FastText(Y[i])], len(M) = 60.$$

6.3 Описание процесса обучения и тестирование модели

Шаги для обучения модели выполнялись следующие.

1) Загрузка данных в следующий формат – список кортежей. Каждый кортеж состоит из двух элементов: первый – это контекст упоминания в тексте и описание сущности из базы знаний; второй – метки классов (0 или 1).
2) Получение векторов признаков одним из способов, описанных в разделе 6.2;

3) Разделение выборки на обучающую (80%) и валидационную (20%) с применением стратификации – статистический метод, который влияет на то, чтобы в каждой выборке содержался примерно одинаковый процент примеров каждого класса. Для стратификации использовался метод train_test_split (с параметром stratify) из библиотеки Scikit-Learn.
4) Классы в полученном наборе данных (который описан в разделе 4.2) несбалансированные. Поэтому в экспериментах применялись различные методы для решения этой проблемы:
   a) в случае с CatBoostClassifier указывался параметр auto_class_weights='Balanced';
   b) для моделей глубокого обучения функция class_weight из библиотеки Scikit-Learn.
5) Чтобы модели не переобучались, применялся ранний останов, т.е. максимум через 10 эпох обучение останавливалось, если значение функции потерь на валидационном множестве не уменьшалось.

Общая схема обучения модели представлена на рис. 3, а применение в модуле ранжирования – на рис. 4.

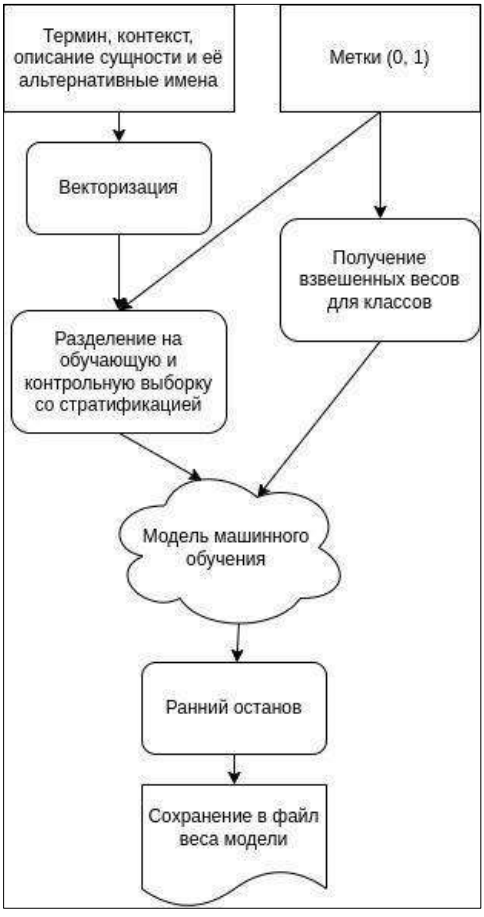


Рис. 3. Схема обучения модели
Fig. 3. Model training scheme

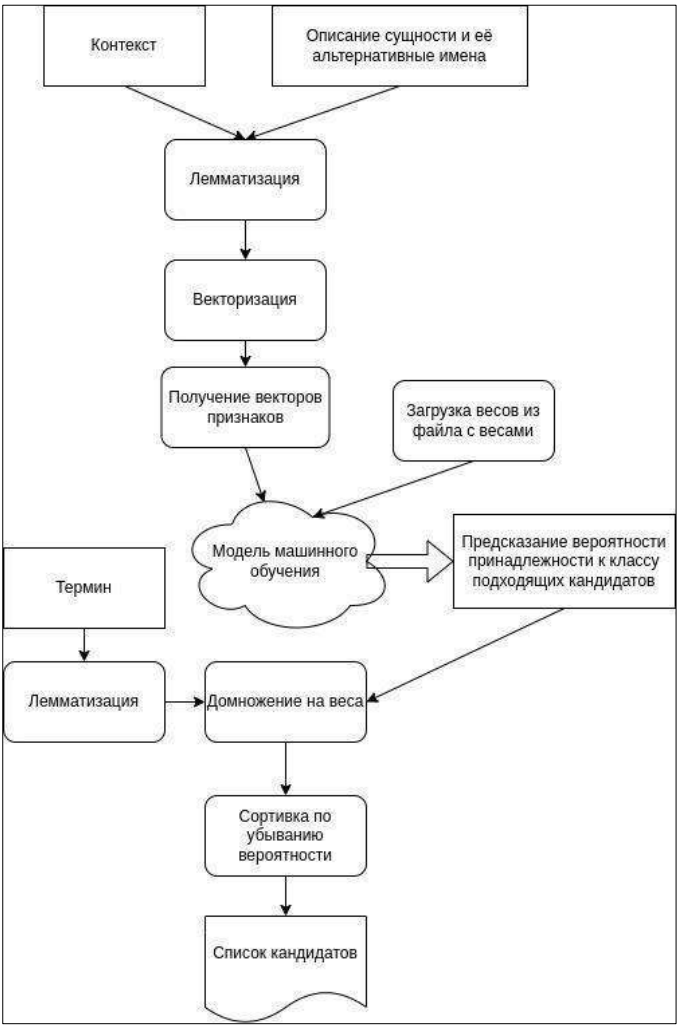


Рис. 4. Схема применения модели в модуле ранжирования кандидатов
Fig. 4. Scheme of applying the model in the candidate ranking module

6.4 Модели классического машинного обучения

Среди моделей классического машинного обучения нами были исследованы два базовых подхода: *логистическая регрессия* и *градиентный бустинг*. В качестве реализации для *логистической регрессии* использовался класс LogisticRegression из библиотеки scikit-learn. На вход модели подавались конкатенации векторов (более подробное описание приведено в разделе 6.2). К сожалению, метрика linked accuracy на тестовых данных составила чуть больше одного процента, поэтому в итоговые результаты данный эксперимент не вошёл. В качестве реализации для *градиентного бустинга* [14] был выбран CatBoostClassifier. Были проведены эксперименты с различными параметрами, но лучше всего подошли те, которые были выставлены по умолчанию. Существенные настройки следующие: ранний останов – 10, балансировка весов, валидационная выборка задана, максимальное количество деревьев – 500, глубина дерева – 6, все деревья одного размера.

6.5 Модели глубокого обучения

В качестве векторов признаков в ниже описанных моделях использовалась только конкатенация векторов без каких-либо модификаций. Для реализации использовался модуль Keras из фреймворка Tensorflow. Эксперименты проводились с несколькими архитектурами:

I. Многослойный персептрон в нескольких модификациях, гиперпараметры которых указаны в табл. 3.

Табл. 3. Эксперименты с архитектурой многослойного перцептрона
Table 3. Experiments with the architecture of a multilayer perceptron

Гиперпараметры	**Эксперимент 1**	**Эксперимент 2**	**Эксперимент 3**
Архитектура сети	два полносвязных слоя – 600 и 300; выходной – один нейрон	два полносвязных слоя – 600 и 200; выходной – два нейрона	три полносвязных слоя – 600, 400, 400; выходной – два нейрона
Функция активации	ReLu		
Функция активации на выходном слое	сигмоида	softmax	
Шаг обучения	0.00003		0.00002
Функция потерь	binary_crossentropy	categorical_crossentropy	
Размер батча	24		
patience	10		

Шаг обучения эмпирически подбирался на первом эксперименте и дальше изменялся, если модель не начинала обучаться. Для второго и третьего эксперимента метки в датасете преобразовывались следующим образом: 0 →[1,0]; 1→[0,1].

II. Сверточная нейронная сеть.

В качестве реализации использовался Conv1D слой из библиотеки Keras. Общие параметры для всех экспериментов: шаг обучения – 0.00003; размер пулинг слоя – 5; 2 нейрона на выходном слое с активационной функцией softmax.

Отличие трёх экспериментов отражено в табл. 4, а именно, количество и размерность окон для сверточных слоёв, входной размер которых (60, 300) с активационной функцией ReLu.

Табл. 4. Размерность карт признаков
Table 4. Feature maps' dimension

Параметр	**Эксперимент 1**	**Эксперимент 2**	**Эксперимент 3**
Размерность карты признаков	100, 5	100,5 - 100,5	200,5 - 200,5

III. Классификатор с двумя входами (упоминание, описание сущности).

На вход классификатору подаются два вектора: 1) упоминание и контекст; 2) описание сущности из базы знаний. Вектора признаков получены тем же способом, что и для сверточной сети. Затем каждый из векторов подается на вход модели следующей архитектуры: два сверточных слоя (карта признаков - (200,5)), после каждого из которых слой

пулинга. После этого выходы для каждого из входов конкатенируются друг с другом и подаются на вход следующей модели, которая уже состоит из двух полносвязных слоёв: 1) двадцать нейронов и relu активация, 2) один нейрон и сигмоида.

При проведении эксперимента модели подавались по 50 примеров за одну итерацию обучения.

IV. Классификатор с тремя входами (термин, контекст, описание сущности).

Основные отличия от классификатора с двумя входами в следующем:

1) вместо двух имеется три входа: 1) упоминание, 2) контекст упоминания, 3) описание сущности; вектор упоминания был вынесен в отдельный вход, исходя из предположения, что если его подавать вместе с контекстом, то он может стать менее четкой информация о том, какой именно термин необходимо связать с сущностью;
2) количество нейронов на предпоследнем слое берется равным 100.

При проведении эксперимента модели подавались по 100 примеров за одну итерацию обучения.

Для подходов, описанных выше, чтобы улучшить метрики на тестовых данных, был применен метод домножения на веса. Гипотеза состоит в том, что важно учитывать то, какое количество токенов в кандидате совпадает с теми, из которых состоит упоминание. Значения весов вычислялись по формуле:

$$weight = \frac{n_{matching}}{n_{all}},$$

$n_{matching}$ – количество совпадающих токенов;

n_{all} – количество токенов во входной сущности.

7. Результаты

Для оценки подходов к ранжированию использовалась точность, которая определяется как отношение количества верно связанных терминов ко всем терминам. Так как нам удалось связать не все термины в корпусе, информативнее будет разделить эту метрику на две: *accuracy* – принимает во внимание все сущности, и *linked_accuracy* – считается только на том наборе терминов, для которых нашлась сущность в графе знаний в корпусе. Таким образом, *accuracy* вычисляется по формуле:

$$accuracy = n\_correct\_entities / n\_all\_entities, \text{ где}$$

n_correct_entities – количество верно связанных терминов;

n_all_entities – количество всех терминов в корпусе.

Обозначим через *n_all_linked_entities* количество всех терминов в корпусе, которые имеют связь с сущностью в Викиданных. Тогда *linked_accuracy* вычисляется по формуле:

$$linked\_accuracy = n\_correct\_linked\_entities \ / \ n\_all\_linked\_entities, \text{ где}$$

n_correct_linked_entities – количество верно связанных терминов среди всех связанных терминов.

Итоговые результаты для основных подходов перечислены в табл. 5.

Значение метрик приведено в двух вариантах – с весами и без них. Не применяя подхода к взвешиванию, только CatBoostClassifier, где признаки подсчитывались по формуле для $features_{cosine}$ (см. подраздел 6.2), удалось добиться лучшего значения 0.28 метрики *linked_accuracy*. Также можно заметить, что классические подходы и нейросетевые находятся примерно в одном диапазоне значений: *accuracy* – от 15 до 17, *linked_accuracy* – от 22 до 28 процентов.

Табл. 5. Результаты тестирования методов
Table 5. Experimental results

Подход		**Метрики без весов**		**Метрики с весами**	
		accuracy	linked_accuracy	accuracy	linked_accuracy
Косинусное расстояние		0.38	0.22	0.55	0.54
Catboost + конкатенация		0.16	0.25	0.20	0.35
Catboost конкатенация косинусное		0.17	0.28	0.23	0.43
Catboost конкатенация среднее		0.15	0.23	0.20	0.36
Перцептрон	Эксперимент 1	0.14	0.20	0.16	0.24
	Эксперимент 2	0.16	0.24	0.17	0.28
	Эксперимент 3	0.15	0.22	0.16	0.25
CNN	Эксперимент 1	0.17	0.28	0.19	0.32
	Эксперимент 2	0.16	0.25	0.18	0.29
	Эксперимент 3	0.17	0.28	0.19	0.32
Классификатор с двумя входами		0.15	0.22	0.15	0.23
Классификатор с тремя входами		0.15	0.22	0.15	0.23

Домножение на веса показало себя неплохо, так как позволило увеличить значения для каждого из подходов. Но в то же время ни один из методов не смог превзойти базовый метод, основанный на косинусном расстоянии между векторами. Полученные результаты показали, что подход, учитывающий косинусное расстояние и количество совпадающих токенов, дает лучшую точность по сравнению с другими.

Мы предполагаем, что это может быть связано с несколькими причинами. Во-первых, сущности в базе знаний не всегда содержат полную и точную информацию, что может привести к неверным предсказаниям моделей. Во-вторых, в данной работе используются статические векторы слов, которые плохо справляются с проблемами омонимии, а в данной задаче эта проблема является основной. Мы предполагаем, что контекстуальные векторы (такие, как ELMo [15] или BERT [16]) справятся с данной задачей лучше. Кроме того, кажется важным включить информацию не только о локальном контексте вокруг термина, но и о более глобальном, например, общей тематике статьи и пр.

В качестве ориентира для сравнения полученных метрик применялись данные из статьи [17], где авторы рассматривают задачу связывание сущностей в мультиязычном аспекте. В частности, они связывали элементы Вики-данных, используя дополнительно имена соответствующих статей из Википедии на разных языках. На датасете Wikinews авторы получили точность, равную 0.66 для текстов с общеупотребимой лексикой на русском языке.

8. Заключение

В данной работе представлена система автоматического связывания сущностей с внешней базой знаний, где в качестве сущностей выступают научные термины, и связываются они с сущностями из Викиданных.

Особое внимание уделено этапу ранжирования как наиболее сложному во всей системе, т.к. необходимо установить семантическую связь между двумя сущностями, которые могут быть представлены в тексте в разном виде, быть синонимичными или омонимичными. Задача ранжирования решалась как задача бинарной классификации: требовалось установить соответствие между термином из текста и сущностью из базы знаний. Были проведены эксперименты с различными архитектурами машинного обучения: логистической регрессией, градиентным бустингом, разными видами нейронных сетей.

Для генерации признаков для моделей были использованы статические векторы FastText, но мы предполагаем, что для данной задачи лучше подойдут контекстные векторные представления. Эксперименты такого типа являются одним из наиболее приоритетных направлений дальнейшей работы.

Список литературы / References

[1]. Bollacker K., Evans C. et al. Freebase: A collaboratively created graph database for structuring human knowledge. In Proc. of the 2008 ACM SIGMOD International Conference on Management of Data, 2008, pp. 1247-1249.

[2]. Auer S., Bizer C. et al. DBpedia: A Nucleus for a Web of Open Data. Proceedings of the 6th International Semantic Web Conference (ISWC). Lecture Notes in Computer Science, vol. 4825, 2007, pp. 722-735.

[3]. Bunescu R., Pąsca M. Using encyclopedic knowledge for named entity disambiguation. In Proc. of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL), 2006, pp. 9-16.

[4]. Cucerzan S. Large-scale named entity disambiguation based on Wikipedia data. In Proc. of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), 2007. pp. 708-716.

[5]. Ratinov L., Roth D. et al. Local and global algorithms for disambiguation to Wikipedia. In Proc. of the 49th Annual Meeting of the Association for Computational Linguistics, 2011. pp. 1375-1384.

[6]. Kolitsas N., Ganea O., Hofmann T. End-to-end neural entity linking. In Proc. of the 22nd Conference on Computational Natural Language Learning, 2018, pp. 519-529.

[7]. Logeswaran L., Chang M. et al. Zero-Shot Entity Linking by Reading Entity Descriptions. In Proc. of the 57th Annual Meeting of the Association for Computational Linguistics, 2019, pp. 3449-3460.

[8]. Zhang S., Cheng H. et al. Knowledge-rich self-supervision for biomedical entity linking. arXiv:2112.07887, 2021, 13 p.

[9]. Botha J., Shan Z., Gillick D. Entity linking in 100 languages. In Proc. of the 2020 Conference on Empirical Methods in Natural Language Processing, 2020, pp. 7833-7845.

[10]. Pratapa A., Gupta R., Mitamura T. Multilingual event linking to wikidata. In Proc. of the Workshop on Multilingual Information Access (MIA), 2022, pp. 37-58.

[11]. Wang X., Tian J. et al. WikiDiverse: A Multimodal Entity Linking Dataset with Diversified Contextual Topics and Entity Types. In Proc. of the 60th Annual Meeting of the Association for Computational Linguistics, vol. 1, 2022, pp. 4785-4797.

[12]. Bruches E., Pauls A. et al. Entity Recognition and Relation Extraction from Scientific and Technical Texts in Russian. In Proc. of the Science and Artificial Intelligence Conference (S.A.I.ence 2020), 2020, pp. 41-45.

[13]. Bruches E., Mezentseva A., Batura T. A System for Information Extraction from Scientific Texts in Russian. Communications in Computer and Information Science, vol. 1620, 2022, pp. 234-245.

[14]. Dorogush A., Gulin A. et al. Fighting biases with dynamic boosting. arXiv:2011.09817, 2017, 5 p.

[15]. Peters M., Neumann M. et al. Deep contextualized word representations. In Proc. of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics, vol. 1, 2018, pp. 2227-2237.

[16]. Devlin J., Chang M. et al. Bert: Pre-training of deep bidirectional transformers for language understanding. In Proc. of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL), vol. 1, 2019. pp. 4171-4186.

[17]. De Cao N., Wu L. et al. Multilingual autoregressive entity linking. Transactions of the Association for Computational Linguistics (TACL), 2022. vol. 10, pp. 274-290.

Информация об авторах / Information about authors

Анастасия Алексеевна МЕЗЕНЦЕВА – студентка НГУ, программист 1-ой категории ИСИ СО РАН. Сфера научных интересов: связывание сущностей, вопросно-ответные системы.

Anastasia Alekseevna MEZENTSEVA – student, NSU, first grade programmer IIS SB RAS. Research interests: entity linking, question-answering systems.

Елена Павловна БРУЧЕС – кандидат технических наук, младший научный сотрудник ИСИ СО РАН, старший преподаватель НГУ. Сфера научных интересов: извлечение информации, распознавание именованных сущностей, создание онтологий.

Elena Pavlovna BRUCHES – PhD in Technical Sciences, Junior Researcher, IIS SB RAS, Senior Lecturer at NSU. Research interests: information extraction, named entity recognition, ontology creation.

Татьяна Викторона БАТУРА – кандидат физико-математических наук, доцент, старший научный сотрудник ИСИ СО РАН. Сфера научных интересов: компьютерная лингвистика, автореферирование, извлечение информации, пополнение графов знаний.

Tatiana Viktorovna BATURA – PhD in Physics and Mathematics, Associate Professor, Senior Researcher, IIS SB RAS. Research interests: computational linguistics, summarization, information extraction, knowledge graph completion.

DOI: 10.15514/ISPRAS-2022-34(4)-14

Theoretical Foundations of an Algorithm of Visualization of a Set of Points of a Multidimensional Space for Use in Anthropotechnical Decision Support Systems

I.A. Alexandrov, ORCID: 0000-0003-1818-5763 <alexandrov@ikti.ru>
V.Zh. Kuklin, ORCID: 0000-0003-2332-895X <vzh.kuklin@gmail.com>
A.N. Muranov, ORCID: 0000-0003-1436-2160 <muranov@ikti.ru>
A.A. Tatarkanov, ORCID: 0000-0001-7334-6318 <tatarkanov@ikti.ru>

Institute of Design-Technology Information Science RAS,
127055 build. 1a., 18, Vadkovsky av., Moscow, Russia

Abstract. Active usage of data collections by experts and decision makers tasked with preparing decision alternatives is an essential characteristic of effectiveness of an anthropotechnic system. In many cases such data analysis may require a standalone visual analysis that implies projection of a multidimensional array of data onto a lower-dimensional space. The article below presents the results of developing the theoretical foundation of such algorithm that is oriented towards an interactive analysis procedure.

Keywords: anthropotechnic systems; decision support system; multidimensional data visualization; methods of interactive information analysis

For citation: Alexandrov I.A., Kuklin V.Zh., Muranov A.N., Tatarkanov A.A. Theoretical Foundations of an Algorithm of Visualization of a Set of Points of a Multidimensional Space for Use in Anthropotechnical Decision Support Systems. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 4, 2022. pp. 201-210 (in Russian). DOI: 10.15514/ISPRAS-2022-34(4)-14

Acknowledgements. The results presented are a part of a larger work being completed in accordance with Subsidy Agreement No. 075-11-2022-029 as of April 8, 2022 on the subject: “Creation of a digital cluster for data storage and processing with consequent formation of dynamic stream using AI technologies.” with Ministry of Science and Higher Education of the Russian Federation.

Теоретические основы алгоритма визуализации множества точек многомерного пространства для использования в антропотехнических системах поддержки принятия решений

И.А. Александров, ORCID: 0000-0003-1818-5763 <alexandrov@ikti.ru>
В.Ж. Куклин, ORCID: 0000-0003-2332-895X <vzh.kuklin@gmail.com>
А.Н. Муранов, ORCID: 0000-0003-1436-2160 <muranov@ikti.ru>
А.А. Татарканов, ORCID: 0000-0001-7334-6318 <tatarkanov@ikti.ru>

Институт конструкторско-технологической информатики РАН,
127055, Россия, г. Москва, пер. Вадковский, д. 18, стр. 1а

Аннотация. Существенным фактором эффективного функционирования антропотехнической системы является активное использование при подготовке управленческих решений информационных массивов с необходимыми для подготовки вариантов решений лицами, принимающими решения, и привлеченными экспертами. Во многих случаях для анализа подобных данных, наряду с

использованием автоматизированных методов, оказывается необходимым проводить визуальный анализ, используя методы проектирования многомерных данных в пространство визуализации малой размерности. В работе представлены результаты разработки теоретического обоснования подобного алгоритма, ориентированного на интерактивную процедуру анализа.

Ключевые слова: антропотехнические системы; система поддержки принятия решений; многомерная визуализация данных; методы интерактивного анализа информации.

Для цитирования: Александров И.А., Куклин В.Ж., Муранов А.Н., Татарканов А.А. Теоретические основы алгоритма визуализации множества точек многомерного пространства для использования в антропотехнических системах поддержки принятия решений. Труды ИСП РАН, том 34, вып. 4, 2022 г., стр. 201-210. DOI: 10.15514/ISPRAS-2022-34(4)-14

Благодарности: Представленные результаты являются частью работы, выполняемой в соответствии с Соглашением о субсидии № 075-11-2022-029 от 08.04.2022 по теме: «Создание цифрового кластера хранения и обработки данных с последующим формированием динамического потока с использованием технологий искусственного интеллекта». с Министерством науки и высшего образования Российской Федерации**.**

1. Introduction

A significant factor of efficient functioning of an antropothecnical system is active usage of bulk data arrays that are utilised for preparation of candidate solutions by the involved decision making persons and experts. In the majority of cases of data analysis, in addition to automated methods, it is necessary to conduct visual data analysis by visualising the multivariable dataset within a space of lesser dimension (e.g. 3 or 4). The paper presents theoretical foundations required to develop such an algorithm for interactive data analysis.

The majority of the data visualisation procedures known nowadays have a disadvantage of being a finite image which corresponds to the minimum of error, while the exact map is unknown. This leads to certain difficulties when a need to update the source data set arises which in turn leads to the need to repeat the procedure from scratch.

A careful analysis shows that if we sacrifice some consistency of the transformation that would give us certain advantages, i.e. a visualisation mode which would allow a user to intervene in an interactive way as well as an explicit optimal projection definition (formula).

The paper considers a task of mapping an initial data set from an n-dimensional space into a (lesser) m-dimensional space (m = 3 or 4) – the visualisation (display) space with locally minimal distortion of mutual location of points in the original (data source) space.

2. Subjects, methods and results of the research paper

2.1 Source data and problem definition

Let's introduce important definitions and denotations.

Let R^k − vector space of dimension k, R^n – initial dataspace, R^m − display space ($m < n$), P – ($n \times n$) − projection matrix, $P: R^n \rightarrow R^m$, $R^{\perp}$ – (n—m) dimensional subspace R^n, that is an orthogonal complement of R^m, $X = \{x^i\}_{i=1}^N$ − initial ensemble of points in R^n, $\Sigma(R^k)$ – a group of orthogonal ($k \times k$) − matrices in R^k.

Let's define a configuration set X as a set of vectors Z,

$$Z = \{z_j \,|\, z_j = x^i - x^k, i = \overline{1, N-1}, k = i+1, N\}. \quad (1)$$

Let's consider that we met the following condition $i \neq j \Leftrightarrow x^i \neq x^j$ and lets denote a number of elements Z through J, *card* $Z = J = N(N—1)/2$.

Let's define a distortion criterion of configuration set X like

$$E(f)[X] = \sum_{i=1}^{N-1} \sum_{k=i+1}^{N} \omega_{ik} \left| \left\| x^i - x^k \right\|^2 - \left\| Pf(x^i) - Pf(x^k) \right\|^2 \right|, \quad (2)$$

and class Φ of non-linear functions vector functions, f: $R^n \to R^n$,

$$\Phi = \{ f \,/\, f = f(x) = Sx + \sum_{\mu=1}^{m} e_\mu \lambda_\mu \left\| P_\perp x \right\| \} ,$$

where $S \in \Sigma(R^n)$, $\{e_\mu\}_{\mu=1}^{n}$ − orthonormal basis in R^n,

$P_\perp$ − $(n \times n)$ − projection matrix form R^n in $R^\perp$,

$\omega_{ik} = \omega(x^i, x^k)$ − scalar nonnegative function, defined at R^n.

Let us recall, that for matrix P the following conditions are met

A task of visualization is to define the map $f \in \Phi$, for which

$$E(f)[X] = \min_{f \in \Phi} E(f)[X] . \quad (3)$$

Let us note that outlined criterion is generalization of a known continuity criterion of Shepherd ([2–4]).

2.2 Visualization problem in the linear case

For the linear visualisation the task is defined as follow: find a matrix $S \in \Sigma(R^n)$ that satisfies:

$$E(S)[X] = \min_{S \in \Sigma(R^*)} E(S)[X].$$

Under the assumption that the following theorem (theorem 1) is correct.

Theorem 1. Let $X = \{x^k\}_{k=1}^{N} \subseteq R^n$,

$$E(S)[X] = \sum_{i=1}^{N-1} \sum_{k=i+1}^{N} \omega_{ik} \left| \left\| x^i - x^k \right\|^2 - \left\| PS(x^i - x^k) \right\|^2 \right|, \quad (4)$$

where $S \in \Sigma(R^n)$, $\omega^{jk} = \omega(x^i, x^k)$ meets a condition

$\forall S \in \Sigma(R^n)\ \omega_{ik}(S_{x^i}, S_{x^k}) = \omega(x^i, x^k)$.

Then a matrix $\hat{S} \in \Sigma(R^n)$, meets a condition $E(\hat{S})[X] = \min_{S \in \Sigma(R^n)} E(S)[X]$, is defined in the following way:

$$\hat{S} = \lim_{k \to \infty} S_k \ , \ S_0 = I \ , \ \forall k \geq 0 \ \ S_{k+1} = R_k S_k \ , \quad (5)$$

where $R_k = R_k(i, j, \varphi)$ is Jacobi rotation method, $i \leq m$, $j \leq m+1$,

$$\varphi = \frac{1}{2} arctg \frac{2a_{ij}^{(k)}}{a_{ii}^{(k)} - a_{jj}^{(k)}}, \ |\varphi| \leq \frac{\pi}{4},$$

$$a_{\mu\nu}^{(k)} = \Sigma_{p<}^{N-1} \Sigma_q^N \omega_{pq} (x_\mu^{p(k)} - x_\mu^{q(k)})(x_\nu^{p(k)} - x_\nu^{q(k)}) ,$$

$$x^{p(k)} = (x_1^{p(k)}, x_2^{p(k)}, \ldots, x_n^{p(k)}) = S_k x^p , \ p = 1, N$$

at that the matrix $\hat{S}$ is defined with accuracy up to arbitrary left multiplier represented in the form of:

$$\hat{S} = \begin{pmatrix} S_m & O \\ O^{\mathrm{T}} & S_\perp \end{pmatrix}, \quad (6)$$

where O − $(m \times (n — m))$ − zero matrix, $S_m \in \Sigma(R^m)$, $S_\perp \in \Sigma(R^\perp)$.

Theorem 1 is a foundation of a multidimensional data set visualisation computation procedure with subsequent minimisation of distortion on a set of orthogonal transformations of the source data space.

Practical application of the procedure consists of iterative application of the following steps:

- calculation of a matrix $R(i, j, \varphi)$ subject to the requirement of minimisation of the distortion

(i.e. residual), transformation of matrix A and formulation of a resulting transform - matrix $\hat{S}$;

- transformation of the source data set in R^n and its visualisation (if required) by projecting it onto the visualisation (display) space, output ot the displaying device and making a decision whether a subsequent iteration is required.

The result of the procedure application is a matrix of an optimising transformation. Additionally, we know the value of the distortion criterion. The block representation of matrix *A* allows us to assess the distance between the achieved value of the distortion criterion and its global minimum. Block diagonalisation of the matrix *A* is a collateral result of the procedure. Let's emphasise that any method of block diagonalisation of matrix *A* gives us a solution to the visualisation problem. Hence, in order to solve the linear visualisation problem any diagonalisation procedure conducted according to the aforementioned description will suffice, including, but not limited to, the principal component analysis. Nevertheless, the procedure above possesses the following peculiarities:

- While the optimal visualisation is being constructed it is possible to output it while maintaining the monotonically decreasing of distortion criteria (it is known that existing PCA implementations do not have such a property) [4]. It is due to the fact that the most efficient realizations are based on matrix diagonalization, which are optimal either in terms of memory minimization or in terms of solution time [5]. At that intermediary results as a rule are not available for a researcher);
- Solution procedure does not require obligatory matrix *A* diagonalisation and is limited only by the condition of block diagonalization, that shortens the time of solution;
- Class of task to which the procedure above can be applied is not limited to the case considered but also allows customisation of the dot product as well as of the norm function $\omega(x^i, x^k)$ depending on the task specifics.

However, in the general case ($n >> m$) [6] a small value of distortion criterion cannot be achieved with the use of linear transformation. This means that the input of traits that are orthogonal to the visualisation space is significant for the distortion criterion. Consequently, we need to resort to non-linear methods in order to reduce the distortion criterion.

2.3 Visualization problem in the nonlinear case

Considering nonlinear mapping of a general form

$$f(x) = f_0 + f_1(x) + f_2(x) + \cdots,$$

where f_0 is a constant vector, which can be equal to zero without losing generality

f_1 – linear,

f_2 – quadric forms defined on the components of the vector x, let's note that using summands, corresponding to forms of higher order – a third one and further results into a significant increment of computational complexity. Let's note that the existing nowadays procedures of visualization also use the methods of minimization not higher that the second order, that is described by the example $f(x) = Ax + f_2(x)$.

For the majority of known procedures of visualization, the drawback is that the result is only a finite image (that is the final result of the procedure), where a criterion minimum was achieved. While the transforming image itself remains unknown, that results into the complexity while trying to refill the initial data set by additional data and get the result of designing without repeated procedure in general.

The analysis shows that giving up the consistency of the transformation to some extent, it is possible to get some advantages in return, that is interactive mode of visual data processing with the intervention mode of a user for him to run the analysis and optimum scale designing image in explicit form.

Let's consider that a linear optimal map for X is the identity mapping. Moreover let's define a class of allowed mappings Φ as follows:

$$\Phi = \{ f \,/\, f = f(x) = x + \sum_{\mu=1}^{n} \lambda_\mu e_\mu \left\| P_\perp x \right\| \}, \tag{7}$$

where $P_\perp$ is a projection from R^n in $R^\perp$,

λ_μ – unknown coefficients of the mapping, $\mu = \overline{1,n}$,

$\{e_\mu\}_{\mu=1}^n$ – orthonormal basis in R^n.

Then a task of visualization will be as following.

Let the set $X = \{x^i\}_{i=1}^N \subseteq R^n$, be defined, then the distortion criterion of the configuration of set X is

$$E(f)[X] = \Sigma_{i<}^{N-1} \Sigma_k^N \omega_{ik} \left| \left\| x^i - x^k \right\|^2 - \left\| Pf(x^i) - Pf(x^k) \right\|^2 \right|, \tag{8}$$

and moreover, it is assumed that

$$E(I)[X] = \min_{S \in \Sigma(R^n)} E(S)[X],$$

where I – an identity matrix.

To define mapping $\hat{f} \in \Phi$, for which

$$E(\hat{f})[X] = \min_{f \in \Phi} E(f)[X]. \tag{9}$$

Let's transform a formula (8) to a form more convenient for analysis. Omitting the intermediary results, we get

$$E(f)[X] = \Sigma_{i=1}^{N-1} \Sigma_{k=i+1}^{N} \zeta_{ik} \left| t^2 + 2t(\eta_{ik}, e)^2 - \zeta_{ik} \right|,$$

where $t = (\Sigma_{\mu=1}^n \lambda_\mu^2)^{1/2}$

$$e = \frac{1}{t}(\lambda_1, \lambda_2, \ldots, \lambda_m, 0, \ldots 0) \in R^m, \left\| e \right\| = 1$$

$$\xi_{ik} = \omega_{ik} a_{ik}^2 \geq 0,$$

$$\eta_{ik} = P(x^i - x^k) a_{ik}^{-1} \in R^m,$$

$$\zeta_{ik} = \left\| P_\perp (x^i - x^k) \right\| a_{ik}^{-1},$$

$$a_{ik} = \left\| P_\perp x^i \right\| - \left\| P_\perp x^k \right\|.$$

Let's note that ζ_{ik}, η_{ik}, ξ_{ik}, do not depend on t and e, and that is why the problem of criterion $E(f)[X]$ minimization can be considered as a problem of finding a scalar $\hat{t} \geq 0$ and a unitary vector, for which

$$h(t,e) = \min_{t \geq 0, \left\| e \right\| = 1} h(t,e), \; h(t,e) = \Sigma_{j=1}^J \alpha_j^2 \left| t^2 + 2t(r_j, e) - \beta_j^2 \right|$$

subject to $\beta_j^2 \geq 1$, $r_j \in R^m$, $e \in R^m$. Let's introduce notations

$$J_+(t) = \{ j \,/\, t^2 + 2t(r_j, e) \geq \beta_j^2 \}, \; J_-(t) = \{ j \,/\, t^2 + 2t(r_j, e) \geq \beta_j^2 \},$$

$$t_j = \sqrt{\beta_j^2 + (r_j, e)^2} - (r_j, e) \geq 0 \tag{10}$$

and dispose of the module in the formulae for $h(t, e)$,

$$h(t,e) = G_0 t^2 + G_1 t - G_2$$

$$G_0 = \sum_{J_+(t)} \alpha_j^2 - \sum_{J_-(t)} \alpha_j^2$$

$$G_1 = \sum_{J_+(t)} \alpha_j^2 (r_j, e) - \sum_{J_-(t)} \alpha_j^2 (r_j, e)$$

$$G_2 = \sum_{J_+(t)} \alpha_j^2 \beta_j^2 - \sum_{J_-(t)} \alpha_j^2 \beta_j^2$$

Without limitation of generality it is possible to consider that t_j are ordered,

$$0 = t_0 < t_1 \le t_2 \le ... \le t_j < t_{j+1} < ...$$

Let's consider a function $h''(t_k + 0, e)$. It is obvious that

$$h''(+0, e) < 0, \quad h''(t_j + 0, e) > 0$$

that is why exists k_0, of the kind that

$$\forall t \le t_{k_0} \; h''(t - 0, e) < 0, \; \forall t \ge t_{k_0} \; h''(t + 0, e) \ge 0. \tag{11}$$

And as the function $h''(t, e)$ is piecewise constant, then

$$\forall k \le k_0 \; \min_{t \in [t_{k-1}, t_k]} h(t, e) = \min\{h(t_{k-1}, e), h(t_k, e)\}$$

Then it is easy to get the following

$$h'(t_{k+1} + 0, e) = h'(t_k + 0, e) + (2t_{k+1} - t_k) h''(t_k + 0, e) + \; + 4\alpha_{k+1}^2 \sqrt{\beta_{k+1}^2 + (r_{k+1}, e)^2}$$

from which, considering (10), we get

$$h'(t_{k1} + 0, e) \ge 0, \; h''(t_{k1} + 0, e) \ge 0 \Rightarrow \forall k \ge k_1, \; h'(t_k + 0, e) \ge 0. \tag{12}$$

Following the previous considerations, it is clear that it is correct that

Lemma 1. Let's

$$h(t, e) = \Sigma_{j=1}^{J} \alpha_j^2 \left| t^2 + 2t(r_j, e) - \beta_j^2 \right|,$$

where $t \ge 0$, $\forall j \quad r_j \in R^m$, $e \in R^m$, $\|e\| = 1$, values k_0 and k_1 depend on the conditions

$$k_0 = \min\{k \,/\, h''(t_{k-1} + 0, e) \le 0, \quad h''(t_k + 0, e) \ge 0\},$$

$$k_1 = \min\{k \,/\, \sum_{j=1}^{k-1} \alpha_j^2 \sqrt{\beta_j^2 + (r_j, e)^2} - \sum_{j=k+1}^{J} \alpha_j^2 \sqrt{\beta_j^2 + (r_j, e)^2} \ge 0\}$$

then

$$\exists \hat{t} = t(e) : \forall t \ge 0 \; h(\hat{t}, e) \le h(t, e), \tag{13}$$

Such that t ∈ $\{0, t_1, \ldots, t_{k0}\} \cup [t_{k0}, t_{k1}]$, where interval is $[t_{k0}, t_{k1}]$ empty, if $k_0 \ge k_1$.

Lemma 1 gives the definition of a function $h(t, e)$ absolute minimum, where e is fixed. At the same time the procedure of finding the minimum is not iterative. Let's consider a case of arbitrate (varying e).

Let $\lambda = te = (\lambda_1, \lambda_2, \ldots, \lambda_m)$, then

$$h(\lambda) = h(t, e) - \sum_{j=1}^{J} \alpha_j^2 \left| (\lambda, \lambda) + 2(\lambda, r_j) - \beta_j^2 \right|.$$

Let's denote by e_1 the current value of e, and let's $t^{(1)}$ meets (12) at $e = e_1$. Let's consider a problem of λ defining, for which $h(\lambda) < h(\lambda^{(1)})$, where $\lambda^{(1)} = t^{(1)} e_1$. For this we will consider the following two cases – $t^{(1)} \ne t_k$ and $t^{(1)} = t_k$, where k – has a value that meets (9). Let's start with (any) $\forall k \; t^{(1)} \ne t_k$. Then at some neighbourhoods of a point $\lambda^{(1)}$ the function $h(\lambda)$ is twice continuously differentiated, consequently, antigradient v

$$\nu = -grad\ h(\lambda)\big|_{\lambda=\lambda^{(1)}} = -\left(\frac{\partial h(\lambda^{(1)})}{\partial \lambda_1}, \frac{\partial h(\lambda^{(1)})}{\partial \lambda_2}, \ldots, \frac{\partial h(\lambda^{(1)})}{\partial \lambda_m}\right) \quad (14)$$

Is either equal to zero, then $\lambda^{(1)}$ there is a point of a function local minimum $h(\lambda)$ or defines the directions of maximum decrease and then

$$\exists \theta > 0 : h(\lambda^{(1)} + \theta\nu) \le h(\lambda^{(1)}). \quad (15)$$

Let then $t^{(1)} = t_k$ for some k. Let's represent $h(\lambda)$ in the form of

$$h(\lambda) = h_0(\lambda) + h_k(\lambda)$$

where

$$h_k(\lambda) = \alpha_k^2 \left| (\lambda, \lambda) + 2(\lambda, r_k) - \beta_k^2 \right|,$$

And as $h_0(\lambda)$ in some neighbourhoods of a point $\lambda^{(1)}$ is twice continuously differentiated, let's define (it should be grad in the formulae)

$$\nu = -grad\ h(\lambda)\big|_{\lambda=\lambda^{(1)}} \quad (16)$$

If $\nu \neq 0$, let's assume that $\nu_0 = \nu\|\nu\|^{-1}$, then at $\theta > 0$, $\theta \square\ 1, \|w\| = 1$, we will get

$$(\nu_0, w) > 0 \Rightarrow h_0(\lambda^{(1)} + \theta w) - h_0(\lambda^{(1)}) = -\theta(\nu_0, w) + o(\theta)$$

and as

$$h_k(\lambda^{(1)} + \theta w) - h_k(\lambda^{(1)}) = \alpha_k^2 \mid \theta w, 2(\lambda^{(1)} + r_k) + \theta w) \mid$$

consequently,

$$h(\lambda^{(1)} + \theta w) - h(\lambda^{(1)}) = -\theta[(\nu_0, w) - \alpha_k^2 \mid w, 2(\lambda^{(1)} + r_k)) \mid] + o(\theta),$$

and if exists w such that $\|w\| = 1$ and

$$(\nu_0, w) - \alpha_k^2 \mid (w, 2(\lambda^{(1)} + r_k)) > 0 \mid, \quad (17)$$

then

$$\exists \theta > 0 : h(\lambda^{(1)}) > h(\lambda^{(1)} + \theta w)$$

Immediate calculation shows that given:

$$\nu_0 \neq (\lambda^{(1)} + r_k)\left\|\lambda^{(1)} + r_k\right\|^{-1} \quad (18)$$

Vector w, defined by the formulae

$$w = [\rho^2 \nu_0 - \sigma(\lambda^{(1)} + r_k)][\rho\sqrt{\rho^2 - \sigma^2}]^{-1}, \quad (19)$$

where

$$\rho = \left\|\lambda^{(1)} + r_k\right\|, \sigma = (\nu_0, \lambda^{(1)} + r_k)$$

meets (17). If (18) is not fulfilled but the condition is correct

$$\alpha_k^2 \left\|\lambda^{(1)} + r_k\right\| < \frac{1}{2},$$

then (17) is correct at $w = \nu_0$.

Finally we get that $\lambda^{(1)}$ is a point of a function local minimum $h(\lambda)$, if one of the conditions is met:

$$\left.\begin{array}{r} \forall k \in \{1,2,\ldots,J\}\ t^{(1)} \neq t_k,\ grad\ h(\lambda^{(1)}) = 0 \\ \exists k : t^{(1)} \neq t_{(k)},\ grad\ h_0(\lambda^{(1)}) = 0 \\ \exists k : t^{(1)} \neq t_{(k)},\ grad\ h_0(\lambda^{(1)}) = \eta(\lambda^{(1)} + r_k) \neq 0,\ \alpha_k^2 \rho \geq \dfrac{1}{2} \end{array}\right\} \quad (20)$$

If none of the conditions are met, then the function $h(\lambda)$ decreases towards one of the vectors defined by one of the formulas − (14), (16) or (19).

Let's denote v as a unit vector of the decrease of function $h(\lambda)$. Let's consider a step choice in this direction. Let's represent $h(\lambda)$ in the form of

$$h(\lambda) = h_0(t^1 + \Theta w) + h_k(\lambda), \quad (21)$$

where

$$k = \max\{j / t^1 \geq t_j, 1 \leq j \leq J\}. \quad (22)$$

Let's note that the set of admissible values θ, such as

$$\theta > 0,\ h(\lambda^{(1)} + \theta v) < h(\lambda^{(1)})$$

as nonempty by the definition of v. Now let's define θ_0,

$$\theta_0 = \min\{\theta / \exists j : h_j(\lambda^{(1)} + \theta v) = 0\},$$

Then for $\theta > \theta_0$ representation (21) is nonexistent, therefore, θ_0 is a minimal element of multiple positive roots of equation of the type

$$\theta^2 + 2\theta(\lambda^{(1)} + r_k, v) - a_j = 0, \quad (23)$$

where

$$a_j = (\lambda^{(1)}, \lambda^{(1)}) + 2(\lambda^{(1)}, r_j) - \beta_j^2.$$

As (2) $\forall j \leq k\ a_j \geq 0$, $\forall j > k\ a_j < 0$, consequently at $j > k$ equation (23) has a positive root, which is defined by a formulae

$$\theta_j = \sqrt{(v, \lambda^{(1)} + r_j)^2 - a_j} - (v, \lambda^{(1)} + r_j).$$

At $j \leq k$ the condition of having a positive root is as following,

$$(v, (\lambda^{(1)} + r_j)) < 0,\ (v, \lambda^{(1)} + r_j)^2 > a_j,$$

and then

$$\theta = |\ (v, \lambda^{(1)} + r_j)\ | - \sqrt{(\lambda^{(1)} + r_j v)^2 - a_j}.$$

Choosing θ_0 from the condition $\theta_0 = \min\limits_{1 \leq j \leq J} \theta_j$ and supposing that

$$\bar{t} = \left\| \lambda^{(1)} + \theta_0 v \right\|,$$

$$e_2 = \frac{1}{2}(\lambda^{(1)} + \theta_0 v),$$

we will get

$$h(\bar{t}, e_2) < h(t^{(1)}, e_1) = h(\lambda^{(1)}).$$

Applying lemma 1 at $e = e_2$, we will get that

$$\exists t^{(2)} \geq 0 :\ h(t^{(2)}, e_2) \leq h(\bar{t}, e_1) < h(t^{(1)}, e_1).$$

following which it is consequent that

$$h(\lambda^{(2)}) = h(t^{(2)}, e_2) < h(t^{(1)}, e_1) = h(\lambda^{(1)}).$$

From the aforementioned considerations it is consequent that the following theorem is correct

Theorem 2. Let $X = \{x^i\}_{i=1}^N \subseteq R^n$, $f \in \Phi$,

$$E(\lambda) = E(f)[X] = \sum_{i=1}^{N-1} \sum_{k=i+1}^{N} \omega_{ik} \left| \left\| x^i - x^k \right\|^2 - \left\| Pf(x^i) - Pf(x^k) \right\|^2 \right|,$$

and $\lambda^{(0)} = (\lambda_1^0, \lambda_2^0, \ldots, \lambda_m^0)$ – a predefined first approximation, where $\lambda = te = (\lambda_1, \lambda_2, \ldots, \lambda_m)$,

$$h(\lambda) = h(t, e) - \sum_{j=1}^{J} \alpha_j^2 \left| (\lambda, \lambda) + 2(\lambda, r_j) - \beta_j^2 \right|.$$

Then for sequence $\lambda^{(0)}, \lambda^{(1)}, \ldots, \lambda^{(j)}, \ldots,$.

$$\forall j \geq 0 \; E(\lambda^{(j)}) > E(\lambda^{(j+1)}), \lim_{k \to \infty} E(\lambda^{(k)}) > E(\hat{\lambda})$$

which means that for $\hat{\lambda}$ either one of the conditions (20) is met, or the function $h(\lambda)$ decreases towards

$$\nu = -grad\ h(\lambda)\big|_{\lambda = \lambda^{(1)}}.$$

The results above represent a theoretical basis for an algorithm of visualisation and can be used for the development of software that will be introduce an interactive mode of visually-aided data mining in the process of which data mining a graphical output can be displayed to the user allowing the user to assess the quality of the mapping and swiftly adjust the variables as to improve the process of visualisation. Additionally, the explicit definition of the projection can be further used for visual analysis after the source data set is updated without the need for redundant reapplication of the initial procedure of transformation calculation.

It is worth mentioning that software implementation of the algorithm assumes development of a feature that would allow choosing "visually most preferable" (according to the user) mapping that would permit choosing standalone subsets as a basis for subsequent classification and identification of key factors within those subsets.

References / Список литературы

[1] A.N. Kolmogorov, S.V. Fomin. Elements of the theory of functions and functional analysis. M., Nauka, 1981, 623 p. (in Russian) / А.Н. Колмогоров, А.В. Фомин. Элементы теории функций и функционального анализа. 5-е изд. М., Наука, 1981 г., 623 стр.

[2] V.N. Volkova,V.A. Voronkov et al. Theory of systems and methods of system analysis in control and communication. M., Radio and Svyaz, 1983, 248 p. (in Russian) / В.Н. Волкова, В.А. Воронков и др. Теория систем и методы системного анализа в управлении и связи. М., Радио и связь, 1981 г., 248 стр.

[3] A.Yu. Terekhina. Data mining by the methods of multidimentional scaling. M., Nauka, 1986, 168 p. (in Russian) / А.Ю. Терёхина. Анализ методами многомерного шкалирования. М., Наука, 1986 г., 168 стр.

[4] G. Fukunaga. Introduction to Statistical Pattern Recognition. 2nd Edition. Academic Press, 1990, 626 p.

[5] B.N. Parlett. The Symmetric Eigenvalue Problem. Society for Industrial and Applied Mathematics, 1987, 416 p.

[6] M. Kendall. The Advanced Theory of Statistics, Vol. 3: Design and Analysis and Time-Series. 2nd Esitiion. Arnold, 1968, 567 p.

Информация об авторах / Information about authors

Ислам Александрович АЛЕКСАНДРОВ – кандидат технических наук, старший научный сотрудник. Сфера научных интересов: системный анализ, автоматизация, искусственные нейронные сети.

Islam Alexandrovich ALEXANDROV – Candidate of Technical Sciences, Senior Researcher. Research interests: system analysis, automation, artificial neural networks.

Владимир Жанович КУКЛИН – доктор технических наук, доцент, ведущий научный сотрудник. Сфера научных интересов: системный анализ, информационные системы, методы экспертного анализа данных, нейрокомпьютерные технологии.

Vladimir Zhanovich KUKLIN – Doctor of Technical Sciences, Associate Professor, Leading Researcher. Research interests: system analysis, information systems, methods of expert data analysis, neurocomputer technologies.

Александр Николаевич МУРАНОВ – кандидат технических наук, старший научный сотрудник. Сфера научных интересов: оптимизация, статистическая обработка данных, автоматизация.

Alexander Nikolaevich MURANOV – Candidate of Technical Sciences, Senior Researcher. Research interests: optimization, statistical data processing, automation.

Аслан Адальбиевич ТАТАРКАНОВ – научный сотрудник. Сфера научных интересов: системный анализ, автоматизация, искусственные нейронные сети.

Aslan Adalbievich TATARKANOV – Researcher. Research interests: system analysis, automation, artificial neural networks.

DOI: 10.15514/ISPRAS-2022-34(4)-15

Построение требований и архитектуры облачного оркестратора платформенных сервисов

Н.А. Лазарев, ORCID: 0000-0002-1008-1022 <lazarevn@ispras.ru>
О.Д. Борисенко, ORCID: 0000-0001-8297-5861 <al@somestuff.ru>
Институт системного программирования им. В.П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. Облачные технологии предоставляют пользователям простое и надежное масштабирование ресурсов, за счет чего они получили широкое распространение. На сегодняшний день особенно актуальна задача управления распределенными службами в облачной среде. Для этого используются специальные программы “оркестраторы”, которые реализуют функции управления жизненным циклом приложений. Однако существующие решения имеют множество ограничений и неприменимы в общем случае. Кроме того, не существует единого стандарта или протокола для взаимодействия с такими инструментами, из-за чего требуется адаптация программ для каждого частного случая. Основными задачами этой статьи являются выявление требований к оркестратору платформенного уровня облачных вычислений (PaaS), а также предложение подходов к построению гибкой архитектуры инструментов этого класса.

Ключевые слова: облачные вычисления; оркестрация; PaaS; распределенные системы; TOSCA; OCCI

Для цитирования: Лазарев Н.А., Борисенко О.Д. Построение требований и архитектуры облачного оркестратора платформенных сервисов. Труды ИСП РАН, том 34, вып. 4, 2022 г., стр. 211-228 DOI: 10.15514/ISPRAS-2022-34(4)-15

Requirements and architecture design for cloud PaaS orchestrator

N.A. Lazarev, ORCID: 0000-0002-1008-1022 <lazarevn@ispras.ru>
O.D. Borisenko, ORCID: 0000-0001-8297-5861 <al@somestuff.ru>
Ivannikov Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. Cloud technologies provide abilities for simple and reliable scaling of resources, due to which they have become widespread. The task of managing distributed services in a cloud environment is especially relevant today. Special programs are used for that purpose named “orchestrators” which implement the functions of lifecycle management for applications. However, the existing solutions have many limitations and are not applicable in the general case. Also there is no single standard or protocol for interaction with such tools which requires adaptation of programs for each particular case. The main objectives of this paper are to identify the requirements for a platform-level cloud computing (PaaS) orchestrator, as well as to propose flexible architecture patterns for such tools.

Keywords: cloud computing; orchestration; PaaS; distributed systems; TOSCA; OCCI

For citation: Lazarev N.A., Borisenko O.D. Requirements and architecture design for cloud PaaS orchestrator. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 4, 2022. pp. 211-228 (in Russian). DOI: 10.15514/ISPRAS-2022-34(4)-15

1. Введение

Постоянное развитие технологий производства компьютерной техники приводит к её заметному удешевлению, что позволяет строить все большие вычислительные центры. В связи с этим получили толчок в развитии облачные вычисления – технология предоставления доступа к вычислительным ресурсам посредством сети Интернет, абстрагирующая пользователей от физических устройств. Согласно определению NIST [1], облачные вычисления подразделяются на три базовых уровня:

- инфраструктурный (IaaS) – предоставление виртуальных машин, блочных устройств и настройка сетевого взаимодействия;
- платформенный (PaaS) – размещение на ресурсах провайдера облачных услуг библиотек, сервисов и инструментов, поддерживаемых провайдером, для дальнейшего использования пользователем в своих программах без прямого управления нижележащей инфраструктурой;
- сервисный (SaaS) – обеспечение доступа пользователя к конечным приложениям, расположенным на ресурсах облачного провайдера.

Кроме того, облачные платформы подразделяются по признаку доступности ресурсов: публичные облака предоставляют услуги для любых пользователей, а приватные работают в рамках некоторой организации.

Для каждого уровня предоставления услуг, как правило, требуется специальное программное обеспечения для управления соответствующими ресурсами, все чаще называемое оркестраторами [2]. На сегодняшний день достаточно полно описан уровень IaaS[3]: определены предоставляемые ресурсы и способы предоставления доступа к ним. Для более высоких уровней существует множество реализаций, имеющих различные сферы применения и наборы интерфейсов, которые современные стандарты не покрывают. Эта статья посвящена проблематике разработки PaaS-оркестратора: определению актуальных задач, требуемого функционала и необходимых компонентов для применимости как для развертывания заранее подготовленных сервисов, так и описания пользователями сценариев развертывания собственного программного обеспечения, например, в процессе разработки.

Под PaaS-оркестратором здесь понимается программное обеспечение, обладающее следующей базовой функциональностью:

- развертывание ресурсов по запросу;
- абстрагирование пользователей от управления виртуальными машинами, сетями и дисками;
- возможность развертывания сервисов на виртуальных машинах без использования технологий контейнеризации;

С логической точки зрения, это программное обеспечение явным образом отделено от инфраструктурного слоя и использует доступные вызовы для получения инфраструктурных ресурсов облачного провайдера (виртуальных машин или контейнеров; программно-определяемых сетей и хранения). Сама же работа по управлению распределенными системами происходит уже не при помощи IaaS API облачного провайдера, а на уровне управления и настройки ОС и программных компонентов в экземплярах виртуальных машин или контейнеров.

Остальная статья организована следующим образом. Во втором разделе рассмотрены вычислительные задачи из различных сфер, рассмотрены основные преимущества контейнерных технологий, на основе чего сформулированы требования к PaaS-оркестратору. В третьем разделе проведен обзор существующих инструментов оркестрации, соответствующих рассматриваемой области, а также описаны применяемые технологии. Далее, в четвертом разделе предложена общая архитектура оркестратора исходя из сформулированных требований и рассмотренных решений. Наконец, в пятом разделе подведены итоги и сформулированы направления для дальнейшей работы.

2. Анализ применимости

Несмотря на наличие множества инструментов, называемых оркестраторами, на сегодняшний день не устоялось ни однозначного определения данного термина, ни даже функционала, который должен предоставлять тот или иной оркестратор. Из этого также вытекает отсутствие однозначных критериев сравнения и оценки таких инструментов. Зачастую каждый автор выбирает интересующий его набор критериев, который формируется на основе некоторых конкретных задач. Например, в сравнении TOSCA-оркестраторов [4], наиболее значимыми критериями сравнения являются поддерживаемая версия стандарта TOSCA и поддерживаемые облачные платформы, но наряду с ними оцениваются также год выпуска, поддерживаемые платформы (операционные системы), сложность установки и другие параметры, не отражающие функциональности инструментов. В другой статье [5], посвященной сравнению облачных оркестраторов, достаточно подробно рассматриваются аспекты, касающиеся работы оркестраторов в облачной среде, однако не рассматриваются способы обработки пользовательских данных и доступа к предоставляемым сервисам.

Одной из основных целей данной работы является формирование универсальных требований к инструментам оркестрации, которые позволили бы сравнивать их с точки зрения функциональности.

2.1 Актуальные задачи

На данный момент границы PaaS сильно размыты, так как набор услуг для работы конечных приложений сильно варьируется. Рассмотрим несколько сценариев использования оркестраторов в различных сферах.

С распространением удаленного обучения стремительно возрос спрос на обеспечение виртуальных учебных мест. Облачные платформы позволяют удовлетворить этот спрос различными способами: предоставляя сервисы для видеоконференций, обмена файлами, проверки заданий, либо предоставляя сами вычислительные ресурсы напрямую. Однако, существуют случаи, когда необходимо предоставить доступ конечных пользователей, например, студентов, непосредственно к некоторой платформе для получения навыков работы с ней. Примером может служить обучение работе с HPC-платформами или написание программ, ориентированных на работу с GPU. Помимо развертывания таких платформ, требуется также оркестрация инструментов взаимодействия с ними: от установки и конфигурации компиляторов до развертывания дополнительных сервисов.

Сегодня облачные вычисления применяются в естественных и гуманитарных науках[6], в том числе: физике и астрофизике, биоинформатике и медицине, социальных и других гуманитарных науках, науках о Земле и окружающей среде, химии (хемоинформатике).

В некоторых направлениях успели сложиться свои традиции и популярные наборы программных инструментов, в то время как другие начинали формироваться в последние несколько лет. Однако, в большинстве случаев справедливы следующие общие проблемы [7,8,9,10,11].

1) *Необходимость хранения и обработки больших объемов данных.* Современные задачи ориентированы на построение крупных и сложных моделей, в связи с чем возрастают требования на хранимые данные. Данные могут быть получены из открытых источников, собраны с помощью дополнительного оборудования, сгенерированы, а также получены от экспертов. Кроме того, в некоторых случаях требуется дополнительная предобработка данных: анонимизация и шифрование, устранение дубликатов и некорректных записей, а также другая подготовка к вычислениям. Таким образом, необходимо обеспечение как достаточного объема хранилища, так и возможность интеграции со сторонними программными и аппаратными средствами.
2) *Организация параллельных вычислений.* Другое следствие роста объемов данных – нелинейное повышение времени работы алгоритмов их обработки. В то время как

возможности вертикального масштабирования вычислительных ресурсов ограничены и требуют все больших затрат, современные платформы параллельной обработки данных на большом числе вычислительных узлов позволяют достигать повышения производительности программ без использования дорогостоящего оборудования. Кроме того, многие современные программные средства позволяют использовать графические ускорители для повышения эффективности программ. Хотя облачная среда позволяет автоматически создавать и настраивать виртуальные машины, особенно актуальна задача автоматизации настройки сложных распределенных систем.

3) *Массовое применение методов машинного обучения.* Наряду с задачами использования вычислительных ресурсов свои требования накладывает развитие методов обработки данных. Методы машинного обучения также являются ресурсоемкими и чувствительны к обеспечению горизонтального масштабирования. Эта область постоянно развивается, в связи с чем набор инструментов, используемых в ML-платформах постоянно растет, в связи с чем необходимо наличие механизма постоянного обновления, используемого в конкретном решении используемого инструментария. Наконец, использование машинного обучения и нейронных сетей тесно связано с построением нелинейного рабочего процесса, требующего постоянного обновления данных и взаимодействия с пользователями.
4) *Потребность в обмене и организации общего доступа к данным.* Современные задачи все чаще требуют участия множества специалистов из различных областей. В связи с этим требуются дополнительные службы, упрощающие взаимодействие между различными пользователями систем.
5) *Обработка ошибок.* Вместе с повышением сложности вычислительных систем все критичнее становится необходимость в автоматическом обнаружении и исправлении ошибок, а также сборе информации об ошибках. Для этих нужд необходима сквозная интеграция систем мониторинга и сбора логов, учитывающая специфику отдельных приложений.

Еще одной важной сферой для анализа задач оркестрации является коммерческая разработка приложений. С одной стороны, в современном бизнесе все чаще используют те же технологии, что и в науке, в том числе, машинное обучение и анализ больших данных. С другой стороны, в процессе коммерческого использования приложений критичны такие параметры, как время ответа пользователю и время недоступности сервиса, в то время как в науке и образовании этому уделяют меньше внимания. В связи с этим, например, задача масштабирования усложняется тем, что необходимо поддерживать постоянную доступность приложения для конечных пользователей. Также возникает явная необходимость обеспечения балансировки нагрузки, обеспечения корректного обновления системы.

2.2 Контейнерная оркестрация

Фактически, стандартным решением для построения коммерческих приложений стало использование контейнеров – легковесной виртуализации на уровне операционной системы и соответствующих оркестраторов, так как они обладают рядом преимуществ при необходимости обеспечения бесперебойной работы:

- легковесность контейнеров относительно виртуальных машин;
- образы контейнеров содержат все необходимое для запуска приложения;
- прозрачное версионирование приложения;
- абстракция от физических ресурсов и операционной системы при запуске;
- простое создание идентичных экземпляров приложения;
- предсказуемое поведение приложения при запуске.

Перечисленные свойства позволяют полностью изолировать процесс развертывания распределенных приложений от внутреннего устройства отдельных его компонент и крайне важны с точки зрения моделирования и оркестрации сложных систем.

Отдельно стоит отметить устоявшийся процесс разработки и развертывания прикладных систем при использовании контейнерных технологий:

1) независимая разработка отдельных компонентов системы;
2) написание файла, описывающего процесс создания образа для отдельных компонентов системы: Containerfile или Dockerfile;
3) создание и загрузка образа в приватный или публичный репозиторий контейнеров;
4) описание конфигураций запуска отдельных компонентов при развертывании всей системы, а также описание взаимодействия компонентов между собой;
5) развертывание всей системы;
6) обновление рабочей системы.

Для оркестрации контейнерных приложений существует множество инструментов, например, Docker Compose, Docker Swarm, Apache Mesos, Kubernetes. При этом выбор оркестратора зависит от масштаба системы [12].

На сегодняшний день наиболее популярным инструментом для оркестрации больших программных систем является Kubernetes. В контексте анализа задач оркестрации стоит отметить ключевой функционал данной платформы:

- **устойчивое развертывание**: формат описания сервисов в Kubernetes позволяет декларативно описывать желаемое состояние. Этот механизм позволяет контролируемо обновлять конфигурацию приложения и переводить на нее нагрузку только после того, как изменения полностью вступили в силу;
- **мониторинг**: контроль за работой сервисов как на уровне статуса всего контейнера, так и за счет выполнения указанных команд внутри контейнера;
- **масштабирование**: в процессе работы приложения Kubernetes позволяет в зависимости от актуальной нагрузки увеличивать или уменьшать количество контейнеров, соответствующих каждому сервису: автоматически или при помощи операторов;
- **конфигурация хранилища**: пользователь может настраивать как локальную файловую систему для своих сервисов, так и подключать сторонние хранилища, доступные по сети;
- **распределение нагрузки**: распределение трафика между контейнерами за счет анализа количества запросов по адресу каждого из контейнеров;
- **контроль за конфиденциальной информацией**: обеспечивается сохранность пользовательских и системных аутентификационных данных;

С другой стороны, использование контейнерной виртуализации накладывает ограничения на работу программ в части взаимодействия с аппаратной частью и хостовой операционной системой, а также имеют недостатки с точки зрения обеспечения безопасности. Кроме того, предполагаемый функционал работы PaaS-оркестратора шире, чем предполагаются при работе с контейнерными оркестраторами. Рассмотрим подробнее, какие типы платформ возникают при оркестрации уровня PaaS на основе приведенного обзора задач.

2.3 Определение функциональности оркестратора

Базовой функциональностью, предоставляемой разработчикам приложений, является среда сборки и выполнения программ. В данный уровень платформ входят компиляторы для различных языков программирования, настройка (возможно, виртуальных) окружений, обеспечение изоляции ресурсов. Отдельно стоит выделить настройку контейнерной среды исполнения, причем на данном уровне имеется в виду непосредственно инструменты для запуска контейнеров без дополнительного окружения. Используя такое окружение разработчики приложений (в том числе, ориентированных на запуск в контейнерном окружении) получают полноценную абстракцию как от физических ресурсов, так и от

операционных систем и необходимости установки дополнительных пакетов, имеют стабильное окружение для сборки и запуска собственных проектов. Аналогом данной абстракции является раздел FROM при описании образа контейнера в формате Dockerfile. Заметим, что обеспечение только этого уровня подразумевает необходимость обеспечения непосредственного доступа пользователей к виртуальным машинам с настроенным окружением посредством ssh или других способов.

Другим обязательным уровнем PaaS является обеспечение взаимодействия с различными инструментами хранения данных. Помимо подключения (виртуальных) блочных устройств к пользовательским виртуальным машинам, может потребоваться создание распределенной файловой системы или объектного хранилища, либо монтирование уже существующих хранилищ. За счет такой функциональности разработчики получают абстракцию для управления данными в различных форматах, сравнимую с подключением томов к контейнерам. При этом подразумевается лишь обеспечение доступа пользователей к данным инструментам по сети, в то время как вся логика взаимодействия с хранилищем остается вне оркестратора.

Наконец, наиболее сложным уровнем функциональности PaaS является обеспечение работы с различными службами, предоставляющими программные интерфейсы по сети. Частым примером для данного уровня является развертывание СУБД по запросу, хотя зачастую его относят к уровню хранилищ. Другими примерами для данного уровня могут являться: инструменты для сбора и обработки логов, мониторинга, веб-разработки, платформы параллельного выполнения программ и планировщики использования ресурсов, а также платформы оркестрации контейнеров. В отличие от контейнерного окружения, в зависимости от платформы, её конфигурации, а также зависимостей могут потребоваться дополнительные вычислительные ресурсы, установка дополнительных пакетов, настройка других инструментов и другие действия. Тем не менее, такой функционал обеспечивает уровень абстракции, аналогичный оркестрации контейнеров: пользователи PaaS-оркестратора получают полноценную возможность использовать сторонние платформы при разработке собственных инструментов.

При этом важно заметить, что аналогично уровню хранения данных, при развертывании высокоуровневых платформ задача оркестратора состоит в обеспечении доступа пользователя к тому или иному инструменту, в то время как любая бизнес-логика требует участия пользователя. Например, рассмотрим задачу предоставления реляционной СУБД по запросу. В простейшем случае эта задача подразумевает создание виртуальной машины достаточной мощности и подключения к ней блочного (виртуального) устройства достаточного объема, установки необходимых для запуска СУБД пакетов, настройки параметров виртуальной машины и самой СУБД, создания пользователя СУБД, а также настройки сетевого файервола для обеспечения доступа. Также, в зависимости от конфигурации всей системы, может потребоваться дополнительная настройка на уровне исполнения программ. В конечном итоге пользователь получает адрес СУБД, а также аутентификационные данные. Однако, в некоторых случаях может потребоваться создание связанных кластеров реляционных СУБД, состоящих из нескольких узлов. В этом случае необходимо обозначить границы возможностей оркестратора: любые манипуляции, для которых необходима информация о схеме базы данных может быть настроена только самим пользователем. Так, оркестратор может настроить кластер PostgreSQL с полной репликацией данных, однако для распределения данных между узлами необходимо указывать параметр распределения в каждой таблице базы данных, поэтому без участия пользователя это сделать невозможно.

Еще одним важным слоем является обеспечение масштабируемости распределенных систем. Использование распределенных систем может диктоваться как требованиями по соблюдению того или иного уровня соглашения о предоставлении услуг (SLA), требованиями к качеству и отзывчивости систем (QoS), а также тем, что для обеспечения

работоспособности программы не хватает физических ресурсов самих инфраструктурных узлов, и программа в принципе не может быть запущена на одном вычислительном узле. Все перечисленные причины к использованию распределенных систем не накладывают специфических требований к оркестратору, поскольку SLA, требования QoS относятся больше к бизнес-преимуществам компаний-провайдеров облачных услуг, однако явным образом добавляют требование на возможность настройки сложных с точки зрения сетевой топологии программных систем, которые требуют корректного распределения конфигурационных файлов, учитывающего порядок запуска каждого из компонентов и их взаимосвязи, а также контроль версий входящих в состав компонентов.

Контроль версий включает в себя множество вспомогательных задач: от учета версий пакетов, устанавливаемых на виртуальные машины, до отслеживания совместимости различных интерфейсов между собой. Так, при развертывании Apache Spark может потребоваться также настройка других инструментов, например, Apache Hadoop. При этом для Apache Spark версии 3.0.0 и выше возможна совместимость только с версиями Apache Hadoop не меньше 2.2.0. Кроме того, официальные сборки доступны только для ограниченного числа комбинаций версий данных платформ, в то время как при необходимости развертывания других потребуется дополнительно обеспечить сборку проектов. Также стоит отметить, что для работы данных платформ требуется настройка JVM определенной версии. В случае, если пользователю потребуется также JVM другой версии, необходима изоляция времени исполнения между платформой и пользовательскими программами. Таким образом, при оркестрации требуется поддерживать многоуровневый контроль совместимости различных компонентов.

Таким образом, на основе проведенного анализа, а также с учетом других работ, можно сформулировать следующий набор функциональных возможностей PaaS-оркестратора для покрытия актуальных задач:

1) механизм описания предоставляемых услуг, учитывающий уровни:
   a) предоставления среды исполнения;
   b) настройки хранилищ данных;
   c) предоставления программных интерфейсов по сети;
2) обобщение пакетных менеджеров уровня операционной системы с возможностью обновления программного обеспечения и контролем совместимости поставляемых пакетов;
3) управление конфигурационными файлами и их специализация под версии развертываемых программных систем, в том числе подстановка параметров;
4) сквозная доставка пользовательских данных;
5) интеграция систем мониторинга для обеспечения отказоустойчивости и масштабирования;
6) обеспечение доступа к развернутым сервисам посредством ssh или инструментов аутентификации в соответствии с уровнем предоставляемых услуг;
7) обеспечение балансировки нагрузки.

Кроме того, в зависимости от контекста использования оркестратора требуется обеспечение программного или пользовательского интерфейса с разграничением доступа к развернутым сервисам и действиям с ними. Также в современных оркестраторах требуется предусмотреть возможность мультиоблачного режима работы: в рамках построения гетерогенной гибридной облачной среды (при одновременном использовании публичных и приватных вычислительных ресурсов), при работе в рамках географически распределенной облачной среды, либо для унифицированного использования ресурсов независимых облачных провайдеров.

3. Обзор оркестраторов

В данном разделе рассматриваются существующие решения, относящиеся к оркестрации уровня PaaS. Так как основной целью работы является определение архитектуры и таксономии мультиоблачных оркестраторов, коммерческие разработки будут рассматриваться только с точки зрения предоставляемого функционала. В свою очередь, основной интерес для текущего исследования представляют открытые оркестраторы, позволяющие решать задачи, описанные во второй главе. Общее сравнение рассматриваемых оркестраторов приведено в табл. 1.

Табл. 1. Сравнение существующих оркестраторов
Tabl. 1. Existing orchestrators comparison

Критерий	**INDIGO-DC**	**yorc**	**Cloudify**	**Heroku**	**ElastiCluster**	**Michman**
Механизм описания платформ	TOSCA Simple Profile v1.0	TOSCA Simple Profile v1.2	DSL на основе TOSCA Simple Profile v1.3	Heroku Buildpack на основе git	Нет	Собственный язык в формате JSON
Управление пакетами	Задается в шаблонах	Задается в шаблонах	Задается в шаблонах	Обработка зависимостей	Задается в Ansible	Подключение репозиториев
Управление конфигурациями	Задается в шаблонах	Задается в шаблонах	Задается в шаблонах	Обработка переменных окружения	Задается в Ansible	Задается в описании сервисов
Доставка пользовательских данных	Задается в шаблонах	Задается в шаблонах	Задается в шаблонах	Нет	Задается в Ansible	Нет
Мониторинг	zabbix	consul	Плагины	Дополнения	Нет	consul
Доступ к ресурсам	Выделенный сервис	ssh	ssh, подключение к ldap	ssh	ssh	ssh
Балансировка нагрузки	Нет	Нет	Плагины	Встроенная маршрутизация	Нет	Нет

3.1 Формат описания предоставляемых услуг

В соответствии с предложенными критериями, в первую очередь необходимо рассмотреть возможность описания предоставляемых услуг. Хотя для конечных пользователей формат описания зачастую скрыт за счет использования какого-либо пользовательского интерфейса, используемое внутреннее представление оркестратора определяет как технологические возможности оркестратора, так и удобство его интеграции. Сегодня существует множество форматов и языков описания распределенных систем, имеющих различные области применимости, модель предоставления услуг и назначение[13].

Наибольший интерес среди них в рамках этой работы представляют независимые от облачной платформы языки и стандарты, позволяющие описать как топологию предоставляемых услуг уровня PaaS, так и процесс развертывания таких топологий. К таким можно отнести OCCI [14], mOSAIC [15], SOCCA [16], STRATOS [17], и другие, однако наиболее современным стандартом для описания топологий в облачной инфраструктуре любых уровней является TOSCA [18].

Основные сущности в данном стандарте делятся на два уровня: типы узлов и взаимоотношений, содержащие информацию о возможных компонентах топологий, а также различные шаблоны, которые конкретизируют описанные типы узлов и взаимосвязей и

описывают их объединение в топологии. TOSCA также определяет функционал инструментов, реализующих обработку документов, соответствующих стандарту:

- парсер (Parser): получает отдельные шаблоны и дополнительные “блоки” из одного или нескольких репозиториев, производит валидацию и нормализацию шаблонов;
- резолвер (Resolver): применяет входные данные для шаблонов, конвертирует нормализованные шаблоны в представление узлов, вызывает встроенные функции, а также вычисляет требования для узлов и создает граф взаимоотношений;
- оркестратор (Orchestrator): непрерывно создает или удаляет реализации для представления узлов, обновляет значения атрибутов узлов, обновляет результаты работы Resolver, а также может изменять представления узлов.

Отметим, что стандарт не ограничивает область применимости, в связи с чем не все TOSCA-оркестраторы могут быть использованы для управления ресурсами уровня PaaS. Поэтому, хотя многие инструменты заявляют о поддержке или реализации данного стандарта, в данной работе будут рассмотрены инструменты, получившие активное применение в научной и коммерческой среде.

- Indigo-DC [19] – облачная вычислительная платформа, предназначенная для научных сообществ и используемая в рамках европейского проекта Horizon 2020. В частности, в Indigo-DC реализуется управление ресурсами уровня PaaS;
- Yorc [20] – инструмент управления жизненным циклом приложений для работы в различных облачных платформах, а также в контейнерной среде Kubernetes, планировщике Slurm и при подключении к физическим серверам без виртуализации, который используется в рамках платформы Lexis [21];
- Cloudify [22] – мультиоблачная платформа оркестрации, ориентированная на автоматизацию развертывания программного обеспечения в процессе разработки. Использует собственный язык описания услуг, основанный на TOSCA.

Стандарт TOSCA позволяет связывать с различными сущностями специальные архивы, содержащие необходимые для развертывания скрипты, данные и прочие артефакты. Таким образом, настройка развертывания полностью лежит на разработчиках шаблонов топологий, в связи с чем снижается гибкость использования TOSCA-оркестраторов. Например, при необходимости добавления новой версии некоторого сервиса, пользователю необходимо либо самому добавить соответствующие артефакты к существующему описанию, либо запросить обновление у автора шаблона.

К минусам этого стандарта можно также отнести большое количество глав стандарта, находящихся в разработке, сложность использования, а также отсутствие соответствия описанных топологий конкретным ресурсам в облачной среде.

В большинстве публичных облачных платформ существуют свои внутренние форматы описания предоставляемых услуг, например, шаблоны Azure Resource Manager (ARM templates [23]), шаблоны AWS CloudFormation [24], шаблоны Google Cloud [25]. Ключевым недостатком такого подхода является непереносимость описаний между различными облачными платформами. С другой стороны, такие форматы ориентированы на конкретные возможности облачных платформ и позволяют более точно описывать детали развертывания требуемых услуг. Такой подход используют также некоторые открытые инструменты оркестрации: в то время как стандарт TOSCA ориентирован на описание наиболее широкого набора облачных услуг, в существующих инструментах может предоставляться ограниченный функционал, за счет чего снижается сложность описания услуг и использования оркестратора. Среди таких оркестраторов можно выделить следующие.

- Heroku [26] – публичная PaaS платформа с открытым исходным кодом, ориентированная на настройку среды разработки в мультиоблачной среде на основе контейнерной виртуализации. Позволяет предоставлять сценарии развертывания в любых форматах с использованием репозиториев git;

- ElastiCluster [27] – инструмент командной строки для автоматизации управления вычислительными кластерами в облачной среде, используемый в проекте ATLAS [28]. В качестве средства описания развертывания систем используется Ansible[29];
- Michman [30] – PaaS оркестратор с открытым исходным, ориентированный на предоставление вычислительных платформ для научных исследований. На данный момент используется собственный формат описания предоставляемых услуг в формате JSON в совокупности с ролями Ansible.

3.2 Управление пакетами

В некоторых случаях может быть необходимо использовать специфичные пакеты при развертывании пользовательских платформ. Например, это может потребоваться при использовании модифицированных в целях безопасности или изменения функционала библиотек. Также, в рамках приватных облачных платформ может возникнуть необходимость установки пакетов без доступа в интернет – для этого необходимо подключать локальные для этой платформы репозитории, либо зеркала. Также как при оркестрации контейнерных окружений, в рамках PaaS-оркестратора необходимо обеспечить подключение пользовательских репозиториев.

Коммерческие публичные решения зачастую скрывают детали развертывания предоставляемых услуг, либо используют публичные репозитории без возможности подключения собственных репозиториев. С другой стороны, при определении собственного шаблона в каждой платформе возможно определение скриптов установки и настройки системы, что позволяет при необходимости указать необходимый способ установки пакетов при развертывании услуги из этого шаблона. В этом случае пользовательские репозитории также должны быть публичными.

В облачных оркестраторах, основанных на стандарте TOSCA, управление установкой пакетов зачастую реализуется разработчиками шаблонов топологий предоставляемых услуг. В случае использования оркестратора в рамках приватной облачной платформы, в шаблонах могут использоваться репозитории, развернутые в этой платформе. Недостатком такого подхода является недостаточный контроль за конфликтами между пакетами. Например, при установке двух платформ, использующих JVM различных версий, может потребоваться дополнительная настройка среды исполнения, что может приводить к непредвиденным ошибкам на этапе развертывания.

В мультиоблачных PaaS-оркестраторах, использующих собственный формат описания, зачастую применяется один из описанных выше подходов в зависимости от того, кем описывается процесс развертывания.

3.3 Управление конфигурационными файлами

Для большинства служб уровня PaaS требуется указание пользователями параметров, применяемых в процессе разворачивания, например, подключаемые плагины, квоты на использование вычислительных ресурсов, параметры авторизации и другие. В зависимости от используемого представления и пользовательских интерфейсов, оркестраторы могут предоставлять гибкую обработку любых параметров, описанных для каждой услуги, либо обеспечивать обработку только для заранее определенных параметров.

3.4 Доставка пользовательских данных

При разработке приложений, ориентированных на обработку больших объемов данных, требуется подключать сторонние системы хранения данных помимо настройки среды исполнения. В то время как публичные облачные платформы обеспечивают сквозную интеграцию с собственными объектными хранилищами, при работе в приватных облаках

могут использоваться открытые и закрытые объектные хранилища, подключаемые блочные хранилища, а также внешние распределенные файловые системы.

На текущий момент реализации стандарта TOSCA позволяют явно описывать только подключение блочных устройств к пользовательским виртуальным машинам. Альтернативно, разработчики шаблонов могут использовать фиктивное определение служб для реализации подключения к другим типам хранилищ, однако в этом случае теряется основное использование стандарта – независимость от облачных платформ и других инструментов.

В свою очередь, инструменты оркестрации, использующие другие форматы описания ресурсов, могут предоставлять сквозную интеграцию с системами хранения данных и монтирование внешних файловых систем.

3.5 Обеспечение отказоустойчивости и масштабирования

В отличие от контейнерной среды, при работе с виртуальными машинами оркестратор не всегда имеет прямой доступ к управляемым ресурсам и требует настройки дополнительных систем мониторинга. В зависимости от целей, мониторинг может осуществляться как на уровне операционной системы – например, через сбор метрик использования ресурсов системы, так и через специализированные запросы к предоставляемым сервисам – через сетевые запросы к ним.

Таким образом, требование получения информации о способах проверки жизнеспособности и загруженности ресурсов подразумевают явное или неявное использование систем мониторинга, а также способы описания проверок в формате описания.

В оркестраторах также могут использоваться встроенные системы анализа состояния управляемыми ресурсами.

3.6 Доступ к ресурсам

Важным аспектом предоставления ресурсов по сети является обеспечение безопасного доступа к ним. В случае предоставления платформы для разработки ПО основным способом доступа является подключение к виртуальным машинам посредством SSH, поэтому зачастую задача сводится к добавлению пользовательских публичных ключей на виртуальные машины.

С другой стороны, при настройке более высокоуровневых платформ требуется обеспечить безопасную доставку паролей при инициализации платформ, что особенно критично при работе в публичных облачных платформах.

3.7 Балансировка нагрузки

Основным преимуществом облачных платформ является гибкость в настройке вычислительных ресурсов. Однако, при горизонтальном масштабировании методом создания идентичных экземпляров служб возникает задача корректного распределения нагрузки между экземплярами. Такая функциональность может решаться различными способами: за счет использования балансировки на инфраструктурном уровне, с использованием специальных встроенных плагинов, а также за счет подключения дополнительных служб в разворачиваемых платформах.

4. Основные компоненты оркестратора

В этом разделе инструменты оркестрации подробнее рассматриваются с точки зрения входящих в их состав служб для выявления общих составляющих, необходимых для работы оркестратора. Также здесь приводится обобщение входящих в оркестраторы компонент и предлагается унифицированная архитектура.

4.1 Архитектура существующих оркестраторов

4.1.1 Indigo-DC

Архитектура Indigo-DC основывается на концепции микросервисов. В состав оркестратора входят следующие компоненты:

- сервис оркестрации (Orchestrator Service) – центральный компонент системы, обеспечивающий развертывание и управление ресурсами;
- сервис аутентификации (IAM Service) – обеспечивает проверку прав пользователей;
- графический интерфейс (GUI) – обеспечивает доступ пользователей к имеющимся шаблонам TOSCA и сервису оркестрации;
- репозиторий шаблонов TOSCA (Repository) – хранит шаблоны TOSCA;
- сервис оркестрации (Orchestrator Service) – центральный компонент системы, обеспечивающий развертывание и управление ресурсами;
- инфраструктурный менеджер (Infrastructure Manager) – разворачивает инфраструктурные ресурсы в различных облаках;
- сервисы управления данными (Data Management Services) – набор сервисов, предоставляющих доступ к хранилищам данных в унифицированном виде.
- служба мониторинга (Monitoring) – обеспечивает сбор метрик из различных облаков и передает в унифицированном виде в сервис оркестрации;
- служба обработки политик (Brokering/Policy Service) – обеспечивает контроль за политиками использования ресурсов;
- служба управления QoS и SLA (QoS/SLA Service) – обеспечивает контроль за QoS и SLA как для отдельных пользователей, так и для всей системы в целом.

4.1.2 Yorc

Оркестратор Yorc основан на модели приложения без сохранения состояния с целью упрощения горизонтального масштабирования. Основной функционал оркестратора реализован в виде одного сервиса, который управляет ходом развертывания ресурсов. К нему также быть подключены дополнительные плагины. Для взаимодействия с оркестратором реализованы сервис, предоставляющий REST API, и интерфейс командной строки. Также для работы с оркестратором может быть использован инструмент Alien4Cloud[31].

Кроме того, в Yorc используются сторонние сервисы: Consul[32] для управления сервисами и мониторинга и Vault[33] для хранения конфиденциальных данных.

4.1.3 Cloudify

Cloudify реализует клиент-серверный подход: основным компонентом является Cloudify Manager, который взаимодействует с агентами на управляемых виртуальных машинах. В свою очередь, Cloudify Manager включает следующие сервисы:

- REST API;
- база данных;
- брокер сообщений;
- сервис управления развертыванием.

Помимо использования REST API напрямую, в Cloudify также реализованы графический интерфейс и интерфейс командной строки. Также предоставляется доступ к логам и мониторингу.

4.1.4 Heroku

В основе работы Heroku используется принцип подготовки образов, готовых к запуску. Для взаимодействия с инструментом предоставляется API, CLI, а также могут использоваться

файлы настройки непосредственно в публичных репозиториях. Кроме того, в состав оркестратора входят службы для маршрутизации трафика и других сетевых настроек, логирования и мониторинга разворачиваемых сервисов, а также системы аутентификации и контроля потребления ресурсов.

Система развертывания, встроенная в Heroku основана на технологии контейнерной виртуализации, однако подразумевает сборку образа самим оркестратором из базовых образов операционных систем. Также для сборки и развертывания сервисов в Heroku используются службы для:

- учета пакетных зависимостей;
- правил сборки программ;
- настройки переменных окружения;
- подключения внешних служб;
- контроля версий сервисов.

4.1.5 ElastiCluster

ElastiCluster является инструментом командной строки, упрощающий использование Ansible для развертывания распределенных систем в облачной среде. Разработчиками предоставляются готовые сценарии для создания кластеров Slurm, Spark и Hadoop. Инструмент позволяет настраивать подключение к различным облачным провайдерам уровня IaaS, отслеживать текущее состояние и масштабировать развернутые кластеры, подключать и настраивать различные распределенные файловые системы.

4.1.6 Michman

Michman также использует микросервисную архитектуру, основными компонентами которой являются сервисы REST и LAUNCHER. REST обеспечивает доступ пользователей к описаниям поддерживаемых оркестратором сервсисов, а также к управлению разворачиваемым кластерам. В LAUNCHER реализована основная логика управления развертывания кластеров. Michman использует Consul для мониторинга, Vault для хранения конфиденциальных данных и LogStash для доступа к логам запуска кластеров. Также в оркестраторе реализовано подключение пакетных репозиториев для обеспечения режима работы без доступа в Интернет.

4.2 Анализ и обобщение рассмотренных оркестраторов

Рассмотренные выше оркестраторы используют различные подходы в организации работы, так как рассчитаны на разные области применимости: от упрощения ручной настройки распределенных вычислительных кластеров до использования в публичных облаках. Оркестраторы, основанные на микросервисной архитектуре, могут быть явным образом разделены на компоненты, имеющие четкие границы применимости. Для монолитных инструментов выделение компонентов предполагает лишь логическое разделение. При этом некоторые компоненты находят реализацию в каждом инструменте в том или ином виде. Основные компоненты и их взаимодействие представлены на рис.1.

В первую очередь, в каждом рассмотренном инструменте используется некоторая абстракция управления вычислительными ресурсами инфраструктурного уровня, в связи с чем может быть выделен соответствующий компонент – IaaS manager. В качестве этого компонента может использоваться интерфейс некоторой облачной платформы, сторонний IaaS-оркестратор, либо встроенный программный компонент. Также в некоторых случаях в качестве альтернативы инфраструктурному уровню может использоваться контейнерная среда, либо физические узлы без виртуализации.

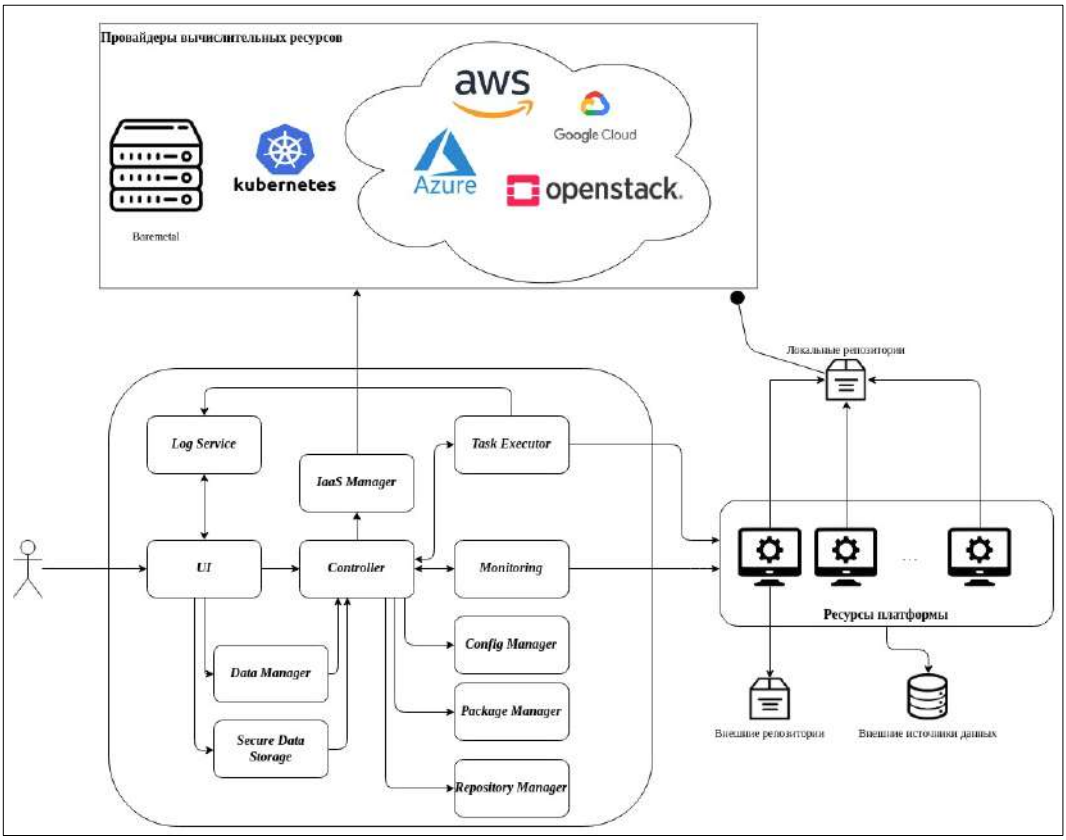


Рис. 1. Унифицированная архитектура PaaS оркестратора
Fig. 1. Unified PaaS orchestrator architecture

В инструментах, основанных на TOSCA, инфраструктурные ресурсы описываются наравне с остальными компонентами топологии платформы: для них существуют общие “нормативные” типы, которые могут быть уточнены для конкретного облачного провайдера, либо могут быть использованы собственные типы. Однако на сегодняшний день во всех рассматриваемых оркестраторах процесс создания экземпляров таких узлов реализован при помощи дополнительных типов узлов. Рассмотрим данный процесс на примере создания виртуальной машины, для которой в стандарте существует нормативный тип tosca.Nodes.Compute. В каждом оркестраторе созданы собственные типы узлов, выведенные из нормативного: “yorc.nodes.openstack.Compute”, “tosca.nodes.indigo.Compute”, “cloudify.nodes.Compute”. Далее в каждом из инструментов реализованы обработчики этих типов в виде функций или подключаемых плагинов для каждого из поддерживаемых облачных сред.

Оркестратор Heroku в большей степени ориентирован на работу в контейнерной среде, однако поддерживает работу с образами операционных систем для дальнейшей настройки на них платформ. ElastiCluster позволяет настраивать подключение к различным провайдерам, указывать образы и размеры виртуальных машин, которые будут в дальнейшем использоваться. Наконец, в Michman на текущий момент также настраивается подключение к облачной платформе при конфигурации оркестратора и выставляются параметры для создания виртуальных машин при создании кластеров. Далее в каждом оркестраторе реализовано создание по заданным параметрам вычислительных ресурсов, однако для пользователя эта информация скрыта, вместо этого пользователь оперирует конечными платформами.

Для обеспечения работы пользователя в рассмотренных инструментах предлагаются различные пользовательские интерфейсы – *UI*. В зависимости от области применимости оркестратора могут быть реализованы: графический интерфейс (GUI), интерфейс командной строки (CLI), а также может быть предоставлен REST API для программного использования. Кроме того, в ходе работы с UI в некоторых инструментах подразумевается также обеспечение аутентификации и доступа к описаниям поддерживаемых сервисов.

Также в большинстве инструментов в том или ином виде реализован доступ пользователей к службам мониторинга, необходимым для масштабирования и отказоустойчивости разворачиваемых сервисов. В общем случае для этого может быть выделен компонент *Monitoring*.

Для отслеживания процесса запуска разворачиваемых сервисов и получения дополнительной информации в случаях ошибок в оркестраторах обеспечивается доступ к логам. В большинстве рассмотренных инструментов хранение логов разделяется на два случая:

1) для доступа к логам развернутых сервисов предполагается развертывание дополнительных служб, либо предоставляется доступ к виртуальным машинам с сервисами.
2) информация о работе самого оркестратора и процессе запуска сервисов записывается локально, либо для этого явным образом выделен отдельный компонент – *Log Service*.

При развертывании приложений требуется настройка пакетов уровня операционной системы. В общем случае это может подразумевать две задачи:

1) Обобщение различных пакетных менеджеров (apt, yum, pip и другие) и разрешение конфликтов при установке пакетов, необходимых для различных сервисов;
2) Подключение к локальным пакетным репозиториям для установки пакетов без доступа в Интернет, а также для использования приватных пакетов.

В большинстве рассмотренных оркестраторов решение этих задач полностью контролируется разработчиками шаблонов и ролей разворачиваемых сервисов. С другой стороны, Heroku для подготовки образа к развертыванию описываются используемые в сервисе пакетные зависимости, доступные из публичных репозиториев. В оркестраторе Michman доступно подключение дополнительных либо локальных репозиториев. В общем случае оркестраторе для этих целей могут быть выделены компоненты для контроля пакетных зависимостей – *Package Manager*, и контроля репозиториев – *Repository Manager*.

Для запуска пользовательских сервисов необходима настройка конфигурационных файлов и переменных окружения. На данный момент явным образом этот уровень не выделен ни в одном инструменте оркестрации, хотя в Heroku используется механизм описания переменных окружения, доступных при развертывании приложений, а в Michman предложен механизм параметризации сервисов. В оркестраторах, основанных на TOSCA, обработка параметров запуска сервисов также возможна при создании шаблонов. Для подстановки пользовательских и системных значений в конфигурационные файлы может использоваться прямая модификация файлов, а также шаблоны файлов, например, в формате Jinja[34]. Заметим, что при совместном использовании нескольких сервисов могут быть модифицированы одни и те же конфигурационные файлы, что может приводить к конфликтам. Для корректной обработки пользовательских параметров и изоляции окружения различных сервисов предлагается выделить дополнительного компонента оркестратора – *Config Manager*.

Следующим важным аспектом работы оркестратора является хранение и обработка пользовательских данных: как необходимых для работы платформы, так и ключей и паролей для доступа к ресурсам. В связи с этим возникает необходимость добавления соответствующих компонентов: *Data Manager* и *Secure Data Storage*. В то время как основная задача Secure Data Storage – обеспечение безопасного хранения небольших объемов данных, Data Manager необходим для обеспечения доставки пользовательских архивов и

подключения внешних хранилищ и файловых систем. Для подключения внешних источников данных в Indigo-DC явным образом выделена служба обработки данных, в ElastiCluster реализовано подключение распределенных файловых систем, а в Heroku доступны дополнения. В свою очередь, в оркестраторах Yorc и Michman используется Vault – инструмент для безопасного хранения паролей, сертификатов, ключей доступа и других конфиденциальных данных.

Наконец, с повышением сложности управляемых оркестраторами сервисов между ними возникают зависимости разного рода. Кроме того, отдельные операции в процессе развертывания платформы требуют значительного времени. В связи с этим в большинстве оркестраторов явным образом выделены управляющие компоненты, которые могут быть обозначены в общем случае *Controller* и *Task Executor*. В то время как Controller необходим для координации всех компонентов оркестратора и обновления текущего статуса всех развернутых сервисов, Task Executor выполняет функции непосредственного выполнения отдельных команд на виртуальных машинах. При этом выполнение команд может выполняться в асинхронном режиме.

Отметим, что выделенные компоненты являются логическими составляющими, а конечная реализация может иметь как монолитную архитектуру, так и распределенную. Кроме того, предложенная архитектура может иметь дополнения в зависимости от формата использования. Например, в коммерческом использовании необходимо добавление компонентов, отвечающих за поддержку SLA, аудит пользовательских действий, а при работе в мультиоблачном режиме оркестратор должен поддерживать технологию единого входа (SSO). Однако в текущей работе основное внимание уделяется специфике организации развертывания распределенных платформ, в то время как такие дополнительные компоненты нацелены на обеспечение работы с пользователями.

5. Заключение

Современные учебные, научные и коммерческие задачи требуют не только больших вычислительных мощностей, но и сложного распределенного программного обеспечения. Платформы обработки больших данных могут требовать до тысяч вычислительных узлов, что делает невозможным ручное управление такими системами. В статье рассмотрены актуальные задачи, для которых используются облачные вычисления и современные способы описания распределенных платформенных сервисов. Исходя из этих задач, сформулирован набор требований к функциональности PaaS-оркетраторов.

На основе проведенного обзора существующих оркестраторов, предложена обобщенная архитектурная модель PaaS-оркестратора. Описанная архитектура позволяет реализовать развертывание распределенных вычислительных систем в устоявшемся при использовании контейнерной виртуализации формате:

- разработчики отдельных сервисов аналогично написанию Containerfile предоставляют информацию о том, как развертывать их программное обеспечение, включающую зависимости, требования к ресурсам, предоставляемые интерфейсы и скрипты развертывания;
- разработчики распределенных систем могут объединять доступные сервисы в распределенные топологии, предоставляя готовые к запуску шаблоны, аналогичные, например, декларативным описаниям сервисов в Kubernetes;
- конечные пользователи могут использовать готовые шаблоны, уточнять параметры в шаблонах, настраивать параметры масштабируемости отдельных компонент, а также управлять развернутыми сервисами в процессе работы.

Дальнейшая работа в этой области требует развития предложенной модели и построения полноценной таксономии облачных вычислений уровня PaaS. Также планируется применение предложенной архитектуры в оркестраторе Michman.

Кроме того, помимо предложенного набора требований к PaaS-оркестратору требуется разработка модельной задачи, которая могла бы позволить объективно оценивать такие инструменты.

Список литературы / References

[1] Mell P. and Grance T. The NIST definition of cloud computing, 2011. Available at: https://csrc.nist.gov/publications/detail/sp/800-145/final, accessed: 26.08.2022.

[2] Tomarchio O., Calcaterra D. & Modica G.D. Cloud resource orchestration in the multi-cloud landscape: a systematic review of existing frameworks. Journal of Cloud Computing, vol. 9, 2020, article no. 49, 24 p.

[3] Dukaric R. and Juric M.B. Towards a unified taxonomy and architecture of cloud frameworks. Future Generation Computer Systems, vol. 29, issue 5, 2013, pp. 1196-1210.

[4] Luzar A., Stanovnik S., and Cankar M. Examination and comparison of TOSCA orchestration tools. Communications in Computer and Information Science, vol. 1269, 2020, pp. 247-259.

[5] Baur D., Seybold D. et al. Cloud orchestration features: Are tools fit for purpose? In Proc. of the IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC), 2015, pp. 95-101.

[6] Ramon-Cortes C., Alvarez P. et al. A survey on the Distributed Computing stack. Computer Science Review vol. 42, 2021, article no. 100422, 22 p.

[7] Netto M.A.S., Calheiros R.N. et al. HPC cloud for scientific and business applications: taxonomy, vision, and research challenges. ACM Computing Surveys (CSUR), vol. 51, issue 1, 2018, article no. 8, pp. 1-29.

[8] Parodi A., Danovaro E. et al. LEXIS weather and climate large-scale pilot. Advances in Intelligent Systems and Computing, vol. 1194, 2020, pp. 267-277.

[9] Caballer M., Zala S. et al. Orchestrating complex application architectures in heterogeneous clouds. Journal of Grid Computing, vol. 16, issue 1, 2018, pp. 3-18.

[10] Yang X., Wallom D. et al. Cloud computing in e-Science: research challenges and opportunities. The Journal of Supercomputing, vol. 70, issue 1, 2014, pp. 408-464.

[11] Mathews S.M. Explainable artificial intelligence applications in NLP, biomedical, and malware classification: a literature review. Advances in Intelligent Systems and Computing, vol. 998, 2019, pp. 1269-1292.

[12] Mercl L. and Jakub P. The comparison of container orchestrators. Advances in Intelligent Systems and Computing, vol. 797, 2019, pp. 677-685.

[13] Nawaz F., Ahmad M., and Naeem K.J. Service description languages in cloud computing: state-of-the-art and research issues. Service Oriented Computing and Applications, vol. 13, issue 2, 2019, pp. 109-125.

[14] Metsch T., Papaspyrou A. et al. Open cloud computing interface–core. Open Grid Forum, OCCI-WG, Specification Document. Available at: https://ogf.org/documents/GFD.221.pdf, accessed: 26.08.2022.

[15] Moscato F., Aversa R. et al. An analysis of mOSAIC ontology for Cloud resources annotation. In Proc. of the Federated Conference on Computer Science and Information Systems (FedCSIS), 2011, pp. 973-980.

[16] Tsai W.-T., Sun X., and Balasooriya J. Service-Oriented Cloud Computing Architecture. In Proc. of the Seventh International Conference on Information Technology: New Generations, 2010, pp. 684-689.

[17] Pawluk P., Simmons B. et al. Introducing STRATOS: A Cloud Broker Service. In Proc. of the IEEE Fifth International Conference on Cloud Computing, 2012, pp. 891-898.

[18] TOSCA Version 2.0 Committee Specification Draft 04. Available at: http://docs.oasis-open.org/tosca/TOSCA/v2.0/TOSCA-v2.0.pdf, accessed: 26.08.2022.

[19] Salomoni D., Campos I. et al. INDIGO-DataCloud: A platform to facilitate seamless access to e-infrastructures. Journal of Grid Computing, vol. 16, issue 3, 2018, pp. 381-408.

[20] Ystia Suite. Available at: https://ystia.github.io/, accessed: 26.08.2022.

[21] Scionti A., Martinovic J. et al. HPC, Cloud and Big-Data Convergent Architectures: The LEXIS Approach. Advances in Intelligent Systems and Computing, vol 993, 2019, pp. 200-212.

[22] Cloudify web-site. Available at: https://cloudify.co/, accessed: 26.08.2022.

[23] Microsoft Azure documentation. Available at: https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/, accessed: 26.08.2022.

[24] AWS CloudFormation documentation. Available at: https://aws.amazon.com/cloudformation/resources/templates/, accessed: 26.08.2022.

[25] Google Cloud documentation. Available at: https://cloud.google.com/deployment-manager/docs/fundamentals, accessed: 26.08.2022.

[26] Heroku web-site. Available at: https://www.heroku.com/, accessed: 26.08.2022.

[27] ElastiCluster Repository. Available at: https://github.com/elasticluster/elasticluster, accessed: 26.08.2022.

[28] Haug S., Sciacca F.G. ATLAS computing on Swiss cloud SWITCHengines. Journal of Physics: Conference Series, vol. 898, issue. 5, 2017.
[29] Ansible web-site. Available at: https://www.ansible.com/, accessed: 26.08.2022
[30] Aksenova E., Lazarev N. et al. Michman: an Orchestrator to deploy distributed services in cloud environments. In Proc. of the 2020 Ivannikov ISPRAS Open Conference (ISPRAS), 2020, pp. 57-63.
[31] Alien4Cloud web-site. Available at: https://alien4cloud.github.io/, accessed: 26.08.2022.
[32] Consul web-site. Available at: https://www.consul.io/, accessed: 26.08.2022.
[33] Vault web-site. Available at: https://www.vaultproject.io/, accessed: 26.08.2022.
[34] Jinja web-site. Available at: https://palletsprojects.com/p/jinja/, accessed: 26.08.2022.

Информация об авторах / Information about authors

Никита Алексеевич ЛАЗАРЕВ – аспирант и стажёр-исследователь. Сфера научных интересов: облачные технологии, распределенные системы, разработка ПО.

Nikita LAZAREV is a postgraduate student and research intern. Research interests: cloud technologies, distributed systems and software development.

Олег Дмитриевич БОРИСЕНКО – научный сотрудник отдела информационных систем и руководитель команды облачных технологий. Сфера научных интересов: облачные технологии и разработка ПО.

Oleg BORISENKO is a specialist and team leader at the Department of Information Systems. Research interests: cloud technologies and software development.

DOI: 10.15514/ISPRAS-2022-34(4)-16

Методы определения элементов PQRST-комплекса электрокардиограммы

[1,3] *О.А. Машкова, ORCID: 0000-0002-4916-1660 <mashkova@ispras.ru>*
[2] *В.В. Шаклеин, ORCID: 0000-0002-4239-0807 <shaklein@ispras.ru>*
[1,4] *Ю.В. Маркин, ORCID: 0000-0003-1145-5118 <ustas@ispras.ru>*
[1] *Е.А. Карпулевич, ORCID: 0000-0002-6771-2163 <karpulevich@ispras.ru>*
[1,2] *В.В. Ананьев, ORCID: 0000-0002-5070-8117 <novisp53@ispras.ru>*
[1,5] *А.А. Асатрян, ORCID: 0000-0002-2529-0169 <arianasat@ispras.ru>*
[1,5] *Ш.Т. Тигранян, ORCID: 0000-0003-1536-9954 <shtigranyan@ispras.ru>*
[1,4] *С.Н. Скорик, ORCID: 0000-0002-8316-7302 <skorik@ispras.ru>*
[1,3] *Д.Ю. Турдаков, ORCID: 0000-0001-8745-0984 <turdakov@ispras.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Новгородский государственный университет им. Ярослава Мудрого, 173003, Россия, г. Великий Новгород, ул. Большая Санкт-Петербургская, д. 41*
[3] *Московский государственный университет имени М.В. Ломоносова, 119991, Россия, Москва, Ленинские горы, д. 1*
[4] *Московский физико-технический институт, 141700, Россия, Московская область, г. Долгопрудный, Институтский пер., 9*
[5] *Российско-Армянский университет, ул. Овсепа Эмина 123, Ереван, 119991 РА*

Аннотация. Электрокардиограмма (ЭКГ) является одним из наиболее распространенных медицинских исследований. Качественная расшифровка 12-канальной электрокардиограммы важна для последующей постановки диагноза и назначения лечения. Один из важных шагов при расшифровке ЭКГ – определение границ элементов PQRST-комплекса. В статье рассматриваются математические методы определения границ волн P, T и комплекса QRS, а также пиков R, P и T, приводятся недостатки математических методов определения элементов PQRST-комплекса. Кроме того, приводятся значения метрик, полученных в результате обучения нейросетевой модели сегментации PQRST-комплекса. Проведенные эксперименты показывают актуальность использования нейросетевых и комбинированных подходов к анализу комплекса PQRST.

Ключевые слова: классификация ЭКГ; сверточная нейронная сеть; PQRST-комплекс; вейвлет-преобразование

Для цитирования: Машкова О.А., Шаклеин В.В., Маркин Ю.В., Карпулевич Е.А., Ананьев В.В., Асатрян А.А., Тигранян Ш.Т., Скорик С.Н., Турдаков Д.Ю. Методы определения элементов PQRST-комплекса электрокардиограммы. Труды ИСП РАН, том 34, вып. 4, 2022 г., стр. 229-240. DOI: 10.15514/ISPRAS-2022-34(4)-16

Благодарности: Исследование выполнено при финансовой поддержке РФФИ и МНТИ в рамках научного проекта № 19-51-06001

Methods for determining the elements of the PQRST-complex of the electrocardiogram

[1,3] *O.A. Mashkova, ORCID: 0000-0002-4916-1660 <mashkova@ispras.ru>*
[2] *V.V. Shaklein, ORCID: 0000-0002-4239-0807 <shaklein@ispras.ru>*
[1,4] *Yu.V. Markin, ORCID: 0000-0003-1145-5118 <ustas@ispras.ru>*
[1] *E.A. Karpulevich, ORCID: 0000-0002-6771-2163 <karpulevich@ispras.ru>*
[1,2] *V.V. Ananev, ORCID: 0000-0002-5070-8117 <novisp53@ispras.ru>*
[1,5] *A.A. Asatryan, ORCID: 0000-0002-2529-0169 <arianasat@ispras.ru>*
[1,5] *S.T. Tigranyan, ORCID: 0000-0003-1536-9954 <shtigranyan@ispras.ru>*
[1,4] *S.N. Skorik, ORCID: 0000-0002-8316-7302 <skorik@ispras.ru>*
[1,3] *D.Yu. Turdakov, ORCID: 0000-0001-8745-0984 <turdakov@ispras.ru>*
[1] *Ivannikov Institute for System Programming of the RAS,*
25, Alexander Solzhenitsyn Str., Moscow, 109004, Russia
[2] *Yaroslav-the-Wise Novgorod State University,*
173003, Russia, Veliky Novgorod, st. Bolshaya St. Petersburg, 41
[3] *Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russian Federation
[4] *Moscow Institute of Physics and Technology,*
9, Institutskiy per., Dolgoprudniy, 141701, Russia
[5] *Russian-Armenian University,*
123 Hovsep Emin str., Yerevan, 0051 Armenia

Abstract. An electrocardiogram (ECG) is one of the most common medical examinations. High-quality interpretation of a 12-channel electrocardiogram is important for subsequent diagnosis and treatment. One of the important steps in deciphering an ECG is to determine the boundaries of the elements of the PQRST complex. The article discusses mathematical methods for determining the boundaries of the P, T waves and the QRS complex, as well as the R, P and T peaks, presents the shortcomings of mathematical methods for determining the elements of the PQRST complex. And also the values of the metrics obtained as a result of training the neural network segmentation model of the PQRST-complex are given. The experiments performed show the relevance of using neural network and combined approaches to the analysis of the PQRST complex.

Keywords: ECG classification; convolutional neural network; PQRST complex; wavelet transform

For citation: Mashkova O.A., Shaklein V.V., Markin Yu.V., Karpulevich E.A., Ananev V.V., Asatryan A.A., Tigranyan Sh.T., Skorik S.N., Turdakov D.Yu. Methods for determining the elements of the PQRST-complex of the electrocardiogram. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 4, 2022. pp. 229-240 (in Russian). DOI: 10.15514/ISPRAS-2022-34(4)-16

Acknowledgements. The reported study was funded by RFBR and MOST according to the research project № 19-51-06001

1. Введение

Сигнал электрокардиограммы (ЭКГ) является одним из наиболее известных биологических сигналов. Сигнал ЭКГ широко используется для диагностики здоровья людей.

ЭКГ представляет собой результат регистрации электрической активности сердца при помощи электродов на теле человека, фиксирующих разность потенциалов. Данное исследование является одним из важнейших в определении нарушения электролитного баланса, острых патологий сердца, физического состояния сердечной мышцы и т. д. Анализ ЭКГ включает в себя определение положения комплексов QRS и волн P и T.

Обнаружение комплекса QRS является одним из наиболее важных шагов, выполняемых при анализе сигнала ЭКГ. Обнаружение QRS-комплекса, в частности определение зубца R в

сигнале ЭКГ, легче, чем определение других элементов сигнала ЭКГ из-за его структурной формы и большой амплитуды.

Поставленная в исследовании задача – определить начало и конец волн P, T и комплекса QRS, а также пиков R, P и T (см. рис. 1) по заданному ЭКГ-сигналу.

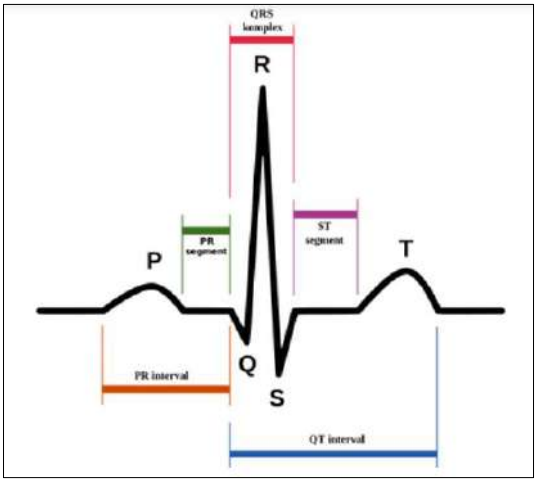


Рис. 1. Зубцы и интервалы ЭКГ
Fig. 1. ECG waves and intervals

В исследованиях применяются различные методы обнаружения комплекса QRS, например, скрытые марковские модели [3], методы дифференцирования [6], а также методы, основанные на нейронных сетях [5].

У большинства представленных методов присутствует фундаментальная проблема, известная как чувствительность к шуму. Для решения этой проблемы могут быть предложены вейвлет-фильтры [1], однако проблема чувствительности к шуму не решается полностью.

В данной статье рассмотрены два подхода к определению элементов PQRST комплекса: на основе математических преобразований и с использованием нейронных сетей.

Работа логически разделена на четыре блока. В первой части статьи описаны математические методы получения начала и конца волн P, T и комплекса QRS, а также пиков R, P и T и приведены результаты их применения к набору данных LUDB [8, 9]. Во второй части публикации приводится анализ недостатков математических методов применимо к анализу сигнала ЭКГ. Далее в статье предлагается использование нейросетевых методов анализа положения пиков и сегментов PQRST комплекса и в заключительной части приводятся результаты применения нейросетевых методов для анализа пиков, положения волн и комплекса QRS.

2. Математические методы определения элементов PQRST-комплекса

2.1 Методология исследования

Теоретической основой исследования явилось каскадное вычисление дискретных свёрток сигнала с вейвлетной функцией методами кратномасштабного анализа [11, 12] для получения коэффициентов детализации на 2-м и 3-м уровнях декомпозиции сигнала (см. рис. 2)

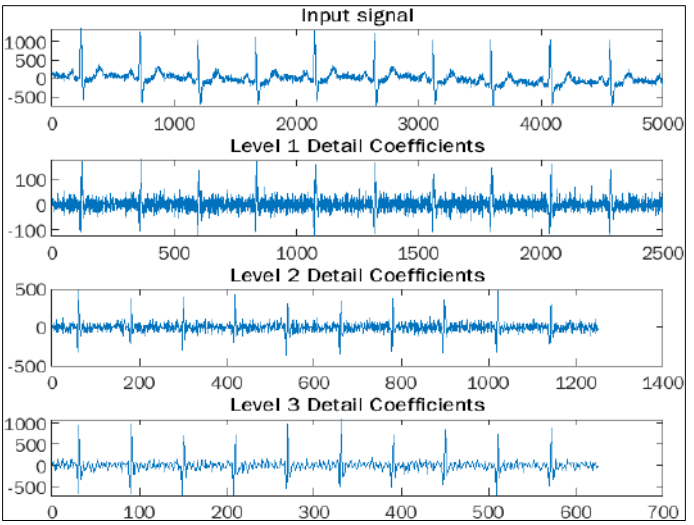


Рис. 2. Входной сигнал и коэффициенты детализации 1-3 уровней
Fig. 2. Input signal and detail factors of 1-3 levels

2.2 Описание алгоритма

В качестве базового алгоритма был выбран алгоритм разметки одного отведения ЭКГ [4]. Выходной сигнал кардиографа передискретизируется к частоте 500 Гц и пропускается через полосовой фильтр Баттерворта 10-го порядка с частотами среза 3 и 30 Гц. Для обработанного сигнала считаются коэффициенты детализации дискретного вейвлет-преобразования для вейвлета Хаара 2-го и 3-го уровней. Нулевое пересечение данных коэффициентов соответствует точкам локального экстремума сглаженного исходного сигнала, а их максимальные абсолютные значения отвечают максимальным амплитудам сигнала [13]. Положение пика R рассчитывается на основе нулевых пересечений коэффициентов 2-го уровня и соответствующих максимальных амплитуд в плавающем окне размера 250 мс. Каждый кандидат проверяется с помощью превышения некоторых эмпирически полученных пороговых значений для коэффициентов 2-го и 3-го уровней в 100 мс-окрестности.

На следующем шаге алгоритма происходит удаление лишних и добавление пропущенных позиций на основе расстояния между соседними пиками, а также сопоставляются результаты по данной позиции по всем отведениям для получения лучшей разметки. Положение начала комплекса QRS определяется левее точки, соответствующей ближайшему слева максимуму модуля положения зубца R на 2-м уровне разложения. Максимумы модуля внутри окна 120 мс также ищутся на основе некоторых пороговых значений коэффициентов. Аналогично находится позиция конца комплекса QRS.

В завершение сегментации комплекса QRS его начало и конец усредняются по всем отведениям. Пик P определяется в окне размера 200 мс левее начала комплекса QRS с помощью нулевых пересечений коэффициентов 3-го уровня с наибольшей амплитудой. Для определения начала волны P используются нулевые пересечения слева от пика P в окне 100 мс, а также предшествующие им максимумы модуля, левее которых ищутся коэффициенты, превышающие некоторый порог. Конец волны P определяется симметрично относительно пика, но не дальше начала комплекса QRS, следующего за данной волной. Полученные координаты корректируются (удаляются лишние точки и добавляются пропущенные) на основе расстояний между соседними величинами, а также пик, начало и конец усредняются по всем отведениям. Блок-схема алгоритма представлена на рис. 3.

Аналогично ищутся параметры волны T. На основе полученных данных считаются продолжительность и амплитуда волн P и T, средняя, минимальная и максимальная ЧСС и R-R расстояние, угол альфа во фронтальной плоскости для комплекса QRS, волн P и T, наклон сегмента ST, корригированные QT интервалы и их дисперсия по формулам Базетта, Фридерика и Саги, а также корнельское произведение.

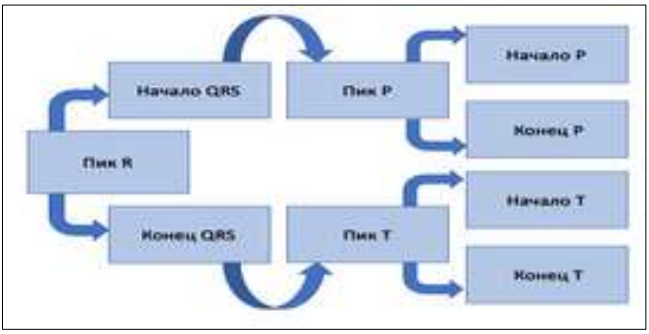


Рис. 3. Блок-схема алгоритма
Fig. 3. Algorithm flowchart

2.3 Оценка работы алгоритма

Алгоритм тестировался на данных LUDB (Lobachevsky University Electrocardiography Database). Обозначим через *TP* (*true positive*) число точек, определенных корректно (т. е. тех, что попадали в симметричную 150 мс-окрестность точки аннотации), через *FP* (*false positive*) – число точек, найденных алгоритмом, но не присутствующих в аннотации, через *FN* (*false negative*) – число ошибочно отсутствующих точек. Для оценки работы алгоритма были взяты метрики из [14]:

$$SE = TP/(TP + FN)$$
$$PPV = TP/(TP + FP)$$
$$F1 = 2TP/(2TP + FP + FN)$$

Для TP-точек также были подсчитаны средняя ошибка и дисперсия ошибок в мс. Для оценки устойчивости работы алгоритма были посчитаны такие же метрики для симметричной 40 мс-окрестности.

40 мс	m ± d	Se	PPV	F1
Начало QRS	-21.5 ± 8.6	99.20 %	99.55 %	99.33 %
Пик QRS	1.8 ± 7.6	99.36 %	99.61 %	99.44 %
Конец QRS	16.5 ± 10.1	99.23 %	99.60 %	99.32 %
Начало P	-21.2 ± 6.8	100.00 %	99.60 %	99.75 %
Пик P	-18.9 ± 5.2	100.00 %	99.74 %	99.85 %
Конец P	0.8 ± 9.5	100.00 %	99.22 %	99.32 %
Начало T	-11.1 ± 10.6	98.15 %	99.58 %	98.33 %
Пик T	-5.2 ± 5.4	99.03 %	99.43 %	99.09 %
Конец T	10.6 ± 10.2	98.06%	98.70 %	98.51 %
150 мс	m ± d	Se	PPV	F1
Начало QRS	-23.5 ± 10	99.25 %	99.55 %	99.36 %
Пик QRS	0.8 ± 9.2	99.37 %	99.62 %	99.46 %
Конец QRS	18.1 ± 13.4	99.36 %	99.59 %	99.44 %
Начало P	-64.1 ± 23	100.00 %	99.82 %	99.90 %
Пик P	-47.4 ± 17.9	100.00 %	99.82 %	99.90 %
Конец P	0.8 ± 14.5	100.00 %	99.74 %	99.86 %
Начало T	-30 ± 18.5	99.11 %	99.78 %	99.34 %
Пик T	-11.2 ± 10.8	99.22 %	99.60 %	99.35 %
Конец T	8.6 ± 17.7	99.16 %	99.07 %	98.99 %

Рис. 4. Полученные результаты
Fig. 4. Results

На рис. 4 представлены результаты работы алгоритма. На рис. 5 представлен пример сегментации ЭКГ-сигнала.

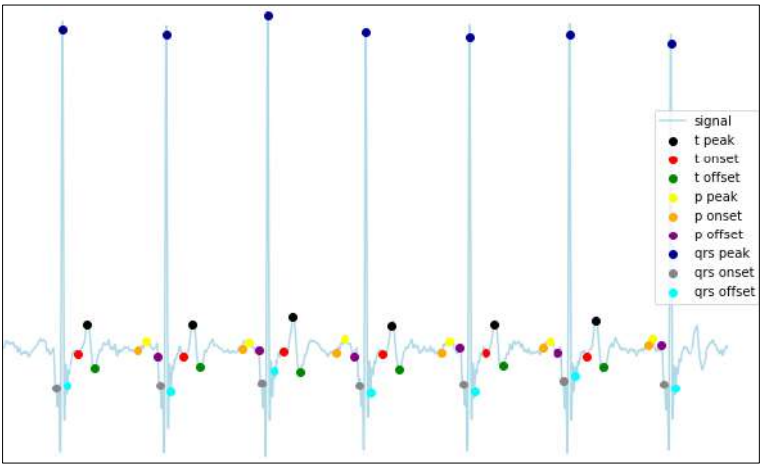


Рис. 5. Пример сегментации ЭКГ-сигнала
Fig. 5. Example of ECG-signal segmentation

Отметим, что аналогичный подход также был применен для оценки качества обученных нейросетей.

3. Определение элементов PQRST-комплекса методами машинного обучения

Математические методы определения элементов PQRST-комплекса не всегда дают положительный результат вследствие зашумленности исходного ЭКГ-сигнала сетевыми помехами, мышечным тремором и т.д., а также искривления изолинии. Особенно трудной задачей является сегментация волны P в силу малости ее амплитуды и, соответственно, совокупной энергии; более того, например, при фибрилляции предсердий, волны P отсутствуют на ЭКГ, заменяясь на специфические волны фибрилляции f, которые не учитываются в стандартных алгоритмах разметки PQRST-комплекса. Сегментация комплекса QRS осложнена большим набором различий его морфологий в зависимости от характера сердечно-сосудистого заболевания (здесь в качестве примера можно привести острый инфаркт миокарда, эффект дигиталиса и блокады ножек пучка Гиса). Таким образом, математические методы сегментации PQRST-комплекса не всегда являются эффективными на реальных данных. Поэтому актуальным направлением для решения поставленной задачи является разработка и реализация алгоритмов глубокого обучения нейронных сетей, поскольку они имеют высокую обобщающую способность.

3.1 Выбор архитектуры нейросети

На текущий момент передовой архитектурой нейронной сети, решающей поставленную задачу, является UNET-подобная архитектура [14]. В то же время, UNET не является единственной архитектурой, решающей подобные задачи, поэтому в рамках эксперимента дополнительно следует адаптировать и протестировать одну из архитектур, решающих задачу сегментации.

В [17] для выделения основных элементов использовалась нейросеть, имеющая архитектуру «автокодировщик» [2] (рис. 6). Как правило, подобные архитектуры используются для

сжатия с минимальными потерями. Также они могут использоваться для сегментации, однако их результаты не превосходят результатов архитектур, заточенных на сегментацию.

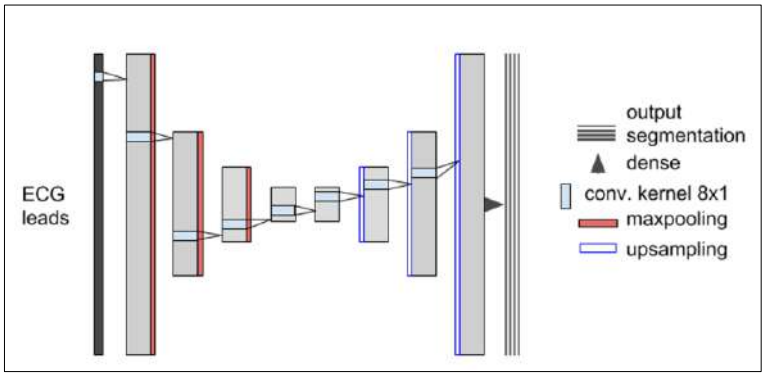


Рис. 6. Архитектура нейросети, использованная в статье [17]
Fig. 6. The neural network architecture used in [17]

Результат, описанный в статье [14], на данный момент, является лучшим из опубликованных. В этой работе используется архитектура UNet1D (рис. 7), а в качестве обучающих данных – закрытый набор LUDB-extended. Воспроизвести результаты статьи [14] невозможно, поскольку веса модели не опубликованы, и получить доступ к использованному набору данных крайне затруднительно.

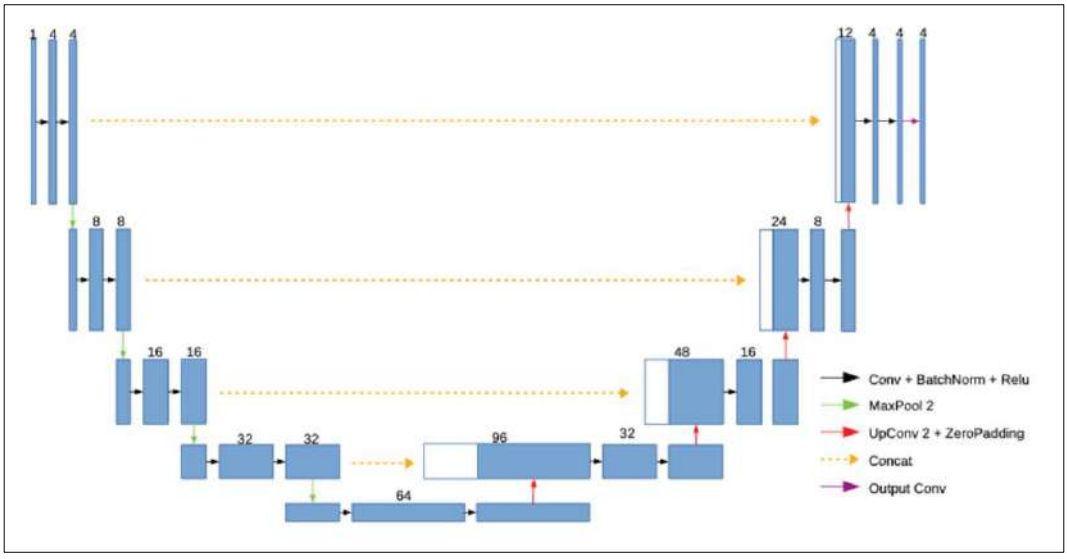


Рис. 7. Архитектура UNet1D
Fig. 7. UNet1D architecture

Авторам рассмотренных исследований удалось добиться высоких результатов. В то же время воспроизвести эти результаты не представляется возможным, поскольку не были опубликованы исходные коды, обученные модели, наборы обучающих данных. Поэтому было решено самостоятельно подготовить обучающий набор данных, а также воспроизвести архитектуру UNet1D и адаптировать архитектуру YOHO, после чего выполнить обучение нейросетей и провести сравнительное тестирование на наборе данных LUDB. В рамках работы рассматриваются две адаптированные архитектуры:

- UNet1D [14] – UNet-подобная архитектура [16];
- YOHO [18] – YOLO-подобная архитектура [15].

3.2 Подготовка обучающего набора данных

Важнейшим фактором при обучении нейронных сетей является наличие большого объема размеченных данных. В рамках решения задачи выделения элементов ЭКГ существует всего несколько открытых наборов данных, причем только один из них содержит записи ЭКГ в 12 отведениях [10]. Для возможности сравнения полученных результатов с ранними работами этот набор будет использоваться в качестве тестовой выборки. В то же время, разделение данного набора на обучающую и тестовую выборки может привести к необъективным данным для сравнения - он содержит малое количество записей с патологиями, влияющими на ритм сердца: всего 4 записи с синусовой тахикардией, 2 записи с нерегулярным синусовым ритмом и 3 записи с типичным трепетанием предсердий [8]. Поэтому нужно было подготовить новый обучающий набор данных. При подготовке обучающего набора данных использовался следующий подход:

- из неразмеченного набора данных были выбраны 10000 записей ЭКГ, на которых присутствуют различные патологии, влияющие на ритм;
- выбранные записи ЭКГ были размечены с использование математического алгоритма, представленного в начале статьи;
- на размеченных данных было проведено обучение нейросети;
- обученная нейросеть была дообучена на наборе данных LUDB;
- из неразмеченного набора данных были выбраны ещё 5000 записей ЭКГ, на которых присутствуют различные патологии, влияющие на ритм;
- дополнительно выбранные записи были размечены с использованием обученной нейросети.

Полученные 5000 размеченных данных использовались для обучения нейросетей в рамках проведенных экспериментов.

3.3 Метрики оценки качества нейросети

Найденные коммерческие решения, реализующие алгоритмы выделения основных элементов ЭКГ, не отображают эти элементы непосредственно, а используют их для подсчёта отображаемых статистик или автоматической постановки диагноза.

Публикация		Начало P	Конец P	Начало QRS	Конец QRS	Начало T	Конец T	Среднее
Finding Morphology Points of Electrocardiographic-Signal Waves Using Wavelet Analysis	SE (%)	98,46	98,46	99,61	99,61	-	98,03	98,83
	PPV (%)	96,41	96,41	99,87	99,87		98,84	98,28
	F1 (%)	97,42	97,42	99,74	99,74		98,43	98,55
ECG Segmentation by Neural Networks: Errors and Correction	SE (%)	95,20	95,39	99,51	99,50	97,95	97,56	97,52
	PPV (%)	82,66	82,59	98,17	97,96	94,81	94,96	91,86
	F1 (%)	88,49	88,53	98,84	98,72	96,35	96,24	94,53
Deep Learning for ECG Segmentation	SE (%)	98,05	98,01	100	100	99,68	99,77	99,25
	PPV (%)	97,73	97,69	99,93	99,93	99,37	99,46	99,02
	F1 (%)	97,89	97,85	99,97	99,97	99,52	99,61	99,14

Рис. 8. Метрики, полученные в других публикациях
Fig. 8. Metrics in other publications

Такой подход является непрозрачным, поскольку врач не может оценить, насколько качественно были обнаружены элементы, на основе которых были рассчитаны статистики.

Поэтому для оценки разработанного решения выполнено сравнение с научными исследованиями, в рамках которых решалась схожая задача.

Работа по сравнению существующих открытых подходов [7, 17, 14] была проделана в статье [14] – результаты представлены на рис. 8. Указанные метрики посчитаны для данных из набора LUDB. На этом же наборе протестировано разработанное решение.

3.4 Оценка качества обученных нейросетей

Обученные нейросети архитектур UNet1D и YOHO были протестированы на наборе данных LUDB – результаты представлены на рисунке 9. При обучении и тестировании выполняется нормализация данных, включающая приведение ЭКГ-записей к одной частоте и заданной продолжительности – обрезка для записей большей продолжительности, дополнение нулевыми значениями для записей меньшей продолжительности.

Нейросеть / публикация		Начало P	Конец P	Начало QRS	Конец QRS	Начало T	Конец T	Среднее
Обученная нейросеть архитектуры UNet1D	SE (%)	96,25	97,70	98,69	98,69	96,84	98,06	97,70
	PPV (%)	97,61	99,17	96,57	96,57	97,51	98,64	97,68
	F1 (%)	96,75	98,25	97,29	97,29	97,57	98,24	97,56
Обученная нейросеть архитектуры YOHO	SE (%)	97,37	97,80	99,72	99,72	97,97	99,11	98,62
	PPV (%)	98,63	99,15	97,98	97,98	97,91	99,04	98,45
	F1 (%)	97,78	98,24	98,73	98,73	97,86	98,99	98,39
Deep Learning for ECG Segmentation	SE (%)	98,05	98,01	100	100	99,68	99,77	99,25
	PPV (%)	97,73	97,69	99,93	99,93	99,37	99,46	99,02
	F1 (%)	97,89	97,85	99,97	99,97	99,52	99,61	99,14

Рис. 9. Метрики, полученные при тестировании обученных нейросетей
Fig. 9. Metrics obtained by testing trained neural networks

4. Заключение

В рамках исследования были разработаны, реализованы и протестированы методы определения элементов PQRST-комплекса электрокардиограммы. Математические методы сегментации PQRST-комплекса не всегда являются эффективными на реальных данных. Алгоритмы глубокого обучения нейронных сетей имеют высокую обобщающую способность, поэтому являются более перспективными.

Список литературы / References

[1] Alexandridi A., Panagopoulos I. et al. R-Peak Detection with Alternative Haar Wavelet Filter. In Proc. of the 3rd IEEE International Symposium on Signal Processing and Information Technology, 2003, pp. 219-222.
[2] Bank D., Koenigstein N., Giryes R. Autoencoders. arXiv:2003.05991, 2020, 22 p.
[3] Coast D.A., Stern R.M. et al. An Approach to Cardiac Arrhythmia Analysis Using Hidden Markov Models. IEEE Transactions on Biomedical Engineering, vol. 37, issue 9, 1990, pp. 826-836.
[4] Di Marco L.Y., and Chiari L. A Wavelet-Based Ecg Delineation Algorithm for 32-Bit Integer Online Processing. Biomedical Engineering Online, vol. 10, issue 1, 2011, pp. 1-19
[5] Fernández-Delgado M., Barro Ameneiro S. MART: A Multichannel Art-Based Neural Network. IEEE Transactions on Neural Networks, vol. 9, issue 1, 1998, pp. 139-150.
[6] Fraden J., Neuman M.R. QRS Wave Detection. Medical and Biological Engineering and Computing, vol. 18, issue 2, 1980. Pp. 125-132.

[7] Kalyakulina A.I., Yusipov I.I. et al. Finding Morphology Points of Electrocardiographic Signal Waves Using Wavelet Analysis. Radiophysics and Quantum Electronics, vol. 61, issue 8-9, 2019, pp. 689-703.
[8] Kalyakulina A.I., Yusipov I.I. et al. Ludb: A New Open-Access Validation Tool for Electrocardiogram Delineation Algorithms. IEEE Access, vol. 8, 2020, pp. 186181-186190.
[9] Kalyakulina A.I., Yusipov I.I. et al. Lobachevsky University Electrocardiography Database (Version 1.0. 0). PhysioNet, 2020, URL: https://physionet.org/content/ludb/1.0.0/.
[10] Kalyakulina A.I., Yusipov I.I. et al. Lobachevsky University Electrocardiography Database (Version 1.0. 1). PhysioNet, 2021, URL: https://physionet.org/content/ludb/1.0.1/.
[11] Mallat S. A Theory for Multiresolution Signal Decomposition: The Wavelet Representation. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 11, issue 7, 1989, pp. 674-693.
[12] Mallat S. Zero-Crossings of a Wavelet Transform. IEEE Transactions on Information Theory, vol. 37 (4): 1991. “1019–33.
[13] Martínez J.P., Almeida R. et al. A Wavelet-Based ECG Delineator: Evaluation on Standard Databases. IEEE Transactions on Biomedical Engineering, vol. 51, issue 4, 2004, pp. 570-581.
[14] Moskalenko V., Zolotykh N., Osipov G. Deep Learning for ECG Segmentation. In International Conference on Neuroinformatics. Studies in Computational Intelligence, vol. 856, 2019, pp. 246-254.
[15] Redmon J., Divvala S. et al. You Only Look Once: Unified, Real-Time Object Detection. In Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779-788.
[16] Ronneberger O., Fischer P., Brox T. U-Net: Convolutional Networks for Biomedical Image Segmentation. Lecture Notes in Computer Science, vol. 9351, 2015, pp. 234-241.
[17] Sereda I., Alekseev S. et al. ECG Segmentation by Neural Networks: Errors and Correction. In Proc. of the International Joint Conference on Neural Networks (IJCNN), 2019, pp. 1-7.
[18] Venkatesh S., Moffat D., and Reck Miranda E. You Only Hear Once: A YOLO-like Algorithm for Audio Segmentation and Sound Event Detection. Applied Sciences, vol. 12, issue 7, 2021, article no. 3293, 16 p.

Информация об авторах / Information about authors

Ольга Анатольевна МАШКОВА, студентка магистратуры механико-математического факультета МГУ. Область научных интересов: анализ и разметка медицинских данных, глубокое обучение, компьютерное зрение и обработка изображений и сигналов.

Olga Anatolyevna MASHKOVA, Master's student of the Mechanics and Mathematics of Moscow State University. Research interests: analysis and labeling of medical data, deep learning, computer vision and image and signal processing.

Всеволод Владиславович ШАКЛЕИН, студент бакалавриата. Научные интересы: машинное и глубокое обучение, анализ данных.

Vsevolod Vladislavovich SHAKLEIN, undergraduate student. Research interests: machine and deep learning, data analysis.

Юрий Витальевич МАРКИН, к.т.н., научный сотрудник ИСП РАН. Научные интересы: машинное и глубокое обучение, анализ биомедицинских данных, информационные системы, обработка изображений и сигналов.

Yury Vitalievich MARKIN, PhD in Technical Sciences, Researcher at ISP RAS. Research interests: machine and deep learning, biomedical data analysis, information systems, image and signal processing.

Евгений Андреевич КАРПУЛЕВИЧ является специалистом отдела «Информационные системы». Сфера научных интересов: применение алгоритмов анализа данных к биомедицинскому домену, разработка систем распределенного хранения и анализа данных.

Evgeny Andreevich KARPULEVICH is a specialist of the Information Systems Department. Research interests: application of data analysis algorithms to the biomedical domain, development of systems for distributed data storage and analysis.

Владислав Валерьевич АНАНЬЕВ – выпускник и ассистент кафедры информационных технологий и систем НовГУ, аспирант ИСП РАН. Сфера научных интересов: анализ и

разметка данных из различных сфер деятельности, глубокое обучение, компьютерное зрение и обработка изображений.

Vladislav Valerievich ANANEV is a graduate and assistant of the Department of Information Technologies and Systems, Novgorod State University, postgraduate student of ISP RAS. Area of research interests: data labeling and analysis for various fields of activity, deep learning, computer vision and image processing.

Ариана Арменовна АСАТРЯН, научный сотрудник. Научные интересы: машинное обучение, анализ данных, обработка изображений и сигналов.

Ariana Armenovna ASATRYAN, researcher. Research interests: machine learning, data analysis, image and signal processing.

Шагане Тиграновна ТИГРАНЯН, магистрант. Научные интересы: машинное обучение, анализ данных, обработка изображений и сигналов.

Shagane Tigranovna TIGRANYAN, graduate student. Research interests: machine learning, data analysis, image and signal processing.

Сергей Николаевич СКОРИК, студент магистратуры МФТИ и сотрудник ИСП РАН. Научные интересы: машинное обучение, методы оптимизации, обработка изображений и сигналов.

Sergej Nikolaevich SKORIK, Master's student at MIPT and an employee of the ISP RAS. Research interests: machine and deep learning, optimization methods, image and signal processing.

Денис Юрьевич ТУРДАКОВ – к.ф.-м.н., заведующий отделом «Информационные системы» ИСП РАН, доцент МГУ. Сфера научных интересов: машинное обучение, интеллектуальный анализ данных, извлечение информации, обработка естественного языка, сложные сети, анализ социальных сетей, большие данные.

Denis Yuryevich TURDAKOV, PhD in Physics and Mathematics, Head of the Information Systems Department at ISP RAS, Associate Professor of the System Programming Department of Moscow State University. Research interests: natural language processing, machine learning, data mining, social network analysis, distributed data processing.

DOI: 10.15514/ISPRAS-2022-34(4)-17

Обзор методов раннего обнаружения меланомы с использованием методов компьютерного зрения

[1]*А.В. Козачок, ORCID: 0000-0002-6501-2008 <a.kozachok@ispras.ru>*
[1]*А.А. Спирин, ORCID: 0000-0002-7231-5728 <a.spirin@ispras.ru>*
[2]*Е.С. Козачок, ORCID: 0000-0003-2912-0754 <muza2804@gmail.com>*

[1] *Институт системного программирования им. В.П. Иванникова РАН,*
109004, Россия, г. Москва, ул. А. Солженицына, д. 25
[2] *Орловская областная клиническая больница*
302028, Россия, г. Орел, ул. Бульвар Победы, д. 1

Аннотация. Меланома является одной из самых агрессивных форм рака, которая поддается лечению только при раннем обнаружении заболевания. В статье рассмотрены существующие алгоритмы и методики визуального диагностирования меланомы. Также рассмотрены системы автоматического диагностирования дерматоскопических изображений и методов, используемых ими. В статье рассмотрены ограничения, препятствующие развитию систем автоматического диагностирования: отсутствие релевантных отечественных наборов данных, позволяющих обучать модели искусственного интеллекта, недостаточный уровень учета метаданных пациентов, низкий охват населения на наличие меланомы при прохождении плановых осмотров. Предложен вариант построения системы поддержки принятия решения врачами общей практики при анализе дерматоскопических изображений кожного покрова.

Ключевые слова: раннее обнаружение меланомы; системы автоматической диагностики; искусственный интеллект в медицине

Для цитирования: Козачок А.В., Спирин А.А., Козачок Е.С. Обзор методов раннего обнаружения меланомы. Труды ИСП РАН, том 34, вып. 4, 2022 г., стр. 241-250. 10.15514/ISPRAS-2022-34(4)-17

Review of methods for early melanoma computer vision detection

[1] *A.V. Kozachok, ORCID: 0000-0002-6501-2008 <ivanov@ispras.ru>*
[1] *A.A. Spirin, ORCID: 0000-0002-7231-5728 <a.spirin@ispras.ru>*
[2]*E.S. Kozachok, ORCID: 0000-0003-2912-0754 <muza2804@gmail.com>*

[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia
[2] *Oryol Regional Clinical Hospital*
10, Pobedy boulevard, Oryol, 302028, Russia

Abstract. Melanoma is one of the most aggressive forms of cancer, which can be treated only with early detection of the disease. The article discusses the existing algorithms and methods of visual diagnosis of melanoma. The systems of automatic diagnosis of dermatoscopic images and the methods used by them are also considered. The article considers the limitations hindering the development of automatic diagnosis systems: the lack of relevant domestic data sets that allow training artificial intelligence models, insufficient level of patient metadata accounting, low coverage of the population for the presence of melanoma during

routine examinations. A variant of building a decision support system by general practitioners in the analysis of dermatoscopic images of the skin is proposed.

Keywords: early detection of melanoma; automatic diagnosis systems; artificial intelligence in medicine

For citation: Kozachok A.V., Spirin A.A., Kozachok E.S. Review of methods for early melanoma detection using computer vision methods. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 4, 2022. pp. 241-250 (in Russian). DOI: 10.15514/ISPRAS-2022-34(4)-17

1. Введение

Меланома – злокачественная опухоль, развивающаяся из меланоцитов – пигментных клеток продуцирующих меланины. Меланома является одной из наиболее агрессивных форм рака кожи, который развивается очень стремительно. Преимущественно локализуется в коже, реже – в сетчатке глаза, слизистых оболочках. Одна из наиболее опасных злокачественных опухолей человека, часто рецидивирующая и метастазирующая лимфогенным и гематогенным путем почти во все органы. Её возникновение и распространение зачастую остается незамеченным, вплоть до формирования неизлечимой стадии болезни. Очень часто меланома похожа на невусы – меланоцитарные образования, не представляющие опасности.

Согласно исследованию [1] международного агентства по изучению рака заболеваемость меланомой кожи к 2040 году возрастет до 510 000 случаев в год, смертность от заболевания составит 96 000 случаев.

Несмотря на развитие методов лечения и развитие фармацевтических препаратов меланома является основной причиной смерти от рака кожи [2].

Как и другие злокачественные новообразования, лечение меланомы дает наилучшие показатели выживаемости при обнаружении на ранней стадии. Когда меланома выявляется до инвазии, лечение имеет высокие показатели излечения только при широком местном иссечении [3, 4, 5], но по мере увеличения глубины инвазии в направлении возможного метастазирования лечение становится все более болезненным, а выживаемость снижается [2].

Вероятность возникновения меланомы зависит от множества признаков: генетических, особенностей окружающей среды, наличия хронических заболеваний, нахождение на солнце, длина теломеров, количество родинок и их размер [6, 7]. Выявление зависимостей возникновения заболевания от различных факторов возможно при использовании искусственного интеллекта и наличия обширных данных по мониторингу состояния здоровья населения [8]. Таким образом дополнительные данные о состоянии здоровья пациентов, их наследственных заболеваниях и ряд других признаков могут повысить точность обнаружения меланомы на ранних стадиях при использовании обученных моделей искусственного интеллекта. Для классификации новообразований по стадиям развития опухоли применяют различные методики, наиболее популярным в настоящее время является стандарт TNM (Tumor – толщина опухоли, Node – количество метастазированных узлов, Metastasis – расположение метастаз) [9, 10].

Для своевременного и безболезненного излечения меланому необходимо обнаруживать на ранних стадиях, поскольку нагрузка на профильных врачей весьма высокая, то можно сделать предположение, что CAD (computer-aided diagnosis – системы автоматической компьютерной диагностики) системы позволят значительно повысить вероятность обнаружения меланомы на ранней стадии.

2. Цель и задачи исследования

В настоящее время существующие подходы к обнаружению меланомы возможно разделить на 2 группы: визуальный осмотр врачом и CAD-системы, представляющие собой аналитические системы поддержки принятия решений, позволяющие врачу экономить время и повышать точность выставления диагноза. Наиболее распространенной системой

поддержки принятия решений врачей-дерматоонкологов является проприетарная система FotoFinder, производства Германии [10]. Ее основным недостатком является высокая стоимость, ограничивающая возможности массового применения для проведения скринингового обследования и раннего выявления меланомы.

В работе [11] авторы делают вывод о недостаточности принимаемых в России мер для ранней диагностики на доинвазивной стадии (pT1) меланомы.

Недостаточная диагностика вызвана, на наш взгляд, несколькими факторами: с одной стороны, недостаточное количество профильных специалистов-онкологов в медицинских учреждениях и как следствие высокая нагрузка и длительные очереди ожидания на прием, что негативно сказывается на вероятности излечения. С другой – отсутствие медицинских информационных систем, способствующих охвату населения при диспансеризации или плановом визите к врачу.

Для снижения времени ожидания очереди на прием к врачу-онкологу CAD-системы могут быть внедрены в непрофильных медицинских учреждениях, что позволит уделять небольшое количество времени врача на осмотр кожных покровов пациента и отправить подозрительные образования для анализа в CAD-системе. Но существует ограничение, в настоящее время подобные системы работают с дерматоскопическими изображениями, следовательно, врач должен обладать дерматоскопом.

Целью исследования является обзор методов диагностирования меланомы на основе применения методов компьютерного зрения и машинного обучения и описание разработанного прототипа системы анализа меланоцитарных изображений. Необходимость его разработки обуславливается отсутствием отечественных аналогов систем и низким процентом обнаружения меланомы на ранней стадии в настоящее время.

3. Методы исследования

При визуальном осмотре пациента применяются различные субъективные методики диагностирования меланомы: правило ABCD(EF) [12], алгоритм CASH [13], 7-точечный список Маки [14], 7-точечное правило Ардженциано [15], метод Мензиса [16], метод TADA [17].

Сведения о чувствительности и специфичность (выражение 1) рассмотренных методов экспертного определения признаков меланомы представлены в табл. 1.

Табл. 1. Чувствительность и специфичность методов экспертной диагностики меланомы
Table 1. Sensitivity and specificity of methods for expert diagnosis of melanoma

Метод	**Чувствительность**	**Специфичность**
Дерматоскопическое правило ABCD	84,1 %	83,5 %
Алгоритм CASH	98 %	68 %
7-точечный список Макки	91,7 %	53,5 %
7-точечное правило Ардженциано	97 %	71 %
Метод Мензиса	92 %	71 %
Метод TADA	94 %	75,5 %

Некоторые из них, несмотря на свою субъективность, могут являться количественными характеристиками для алгоритмов машинного обучения “quantitative imaging”. В исследовании [11] была статистически доказана связь количественных признаков с признаками, применяемыми на качественном уровне правила ABCD: количественной асимметрии образования путем измерения различий в нормализованных распределениях

интенсивности с двух сторон для каждых двух основных осей симметрии на сегментированном поражении в синем диапазоне (430 нм); количественной нерегулярности границ образования, которая является мерой относительной ширины между фактической границей сегментированного поражения и границей эллипса, аппроксимированного так, чтобы соответствовать фактической границе; количественной оценки цвета, которая измеряет пятнистость изображений поражения, полученных в красном диапазоне (700 нм); количественного диаметра поражения – максимального расстояния между самыми удаленными точками новообразования в мм [11].

В исследовании [12] отмечается, что до этапа внедрения механизмов раннего обнаружения заболеваний необходимо решить проблемы получения ложно положительных результатов и как следствие, гипердиагностики и чрезмерного лечения, что влечет дополнительные траты и неоправданно нагружает медицинские учреждения.

HAM 10000 набор дерматоскопических изображений, состоящий из 10015 изображений, разделенных на 7 типов: актинический кератоз, меланома, базальноклеточнаяя карцинома, доброкачественный кератоз, дерматофиброз, меланоцитарный невус, сосудистые поражения кожи.

PH2 набор состоит из 200 изображений, увеличенных в 20 раз размером 768*560. Набор содержит следующие классы: 80 изображений невусов, 80 изображений атипичных невусов и 40 изображений меланомы.

ISIC набор содержит 33126 изображений кожи пациентов, относящихся к 37 классам, в том числе меланома. Преимуществом данного набора является присутствие метаданных: расположение на теле пациента, пол, возраст, анамнез заболеваний родственников и др. Точность обнаружения для описанных наборов и их модификациях представлена в таблице 3.

Глубокие нейронные сети не используют алгоритмы визуального обнаружения признаков меланомы, а определяют информативные признаки в ходе своей работы, основываясь на корреляции признаков и ответов на выходе нейронной сети [18].

В исследовании [19] было доказано, что нейронные сети оказались точнее при выставлении диагноза меланома. Результаты оценки по метрикам чувствительность и специфичность представлены в таблице 2.

Система поддержки принятия решений врачей-дерматоонкологов FotoFinder [19] обладает чувствительностью 86,6 % и специфичностью 89,9 %. Данные показатели являются достаточно высокими и лишь незначительно уступают по чувствительности общей оценке 157 врачей 88,9 % и превосходят врачей по метрике специфичность 75,7 %.

Результаты исследования [20] также подтверждают более высокую точность при классификации меланомы искусственным интеллектом относительно врачей: при одинаковом значении метрики чувствительности в 86,6% специфичность нейросетей составила значение 82,5% против 71,3% у дерматологов. Наличие метаданных позволяет повысить чувствительность при обнаружении меланомы врачами до значения 88,9% согласно кривой ошибок.

Табл. 2. Сравнение результатов оценки новообразований врачами и нейронными сетями
Table 2. Comparison of results of assessment of neoplasms by doctors and neural networks

	Чувствительность (true positive rate)	**Специфичность**
Все участники (n=157)	74,1 %	60,0 %
Университетская больница (n=151)	74,0 %	59,8 %
Врачи-ординаторы (n=6)	76,7 %	65,8 %
Иерархия должностей:		

Начинающие специалисты (n=88)	74,8 %	58,2 %
Врачи (n=15)	72,7 %	60,0 %
Начальники отделений (n=45)	73,0 %	62,3 %
Главные врачи (n=3)	73,3 %	69,2 %
ResNet-50 CNN [3]	87,5 %	60,0 %
Inception v4 CNN (FotoFinder) test-set-300 [7]	95,0 %	80,0%
Врачи [7]	86,6 %	71,3 %

Исследования по применению методов искусственного интеллекта и компьютерного зрения в области классификации изображений новообразований кожи также были отражены в работах [21, 22], результаты представлены в табл. 3.

Табл. 3. Сравнение результатов оценки новообразований нейронными сетями
Table 3. Comparison of the results of assessment of neoplasms by neural networks

Нейронная сеть	Датасет	Точность
VGG-16	HAM 10000	87,42 %
VGG-19		85,02 %
MobileNet		88,22 %
Inception v3		89,81 %
Ансамбль сетей: DenseNet 201, Inception-ResNet v2, Inception v3	PH2	98,8 %
	ISIC-MSK	99,2 %
	ISIC-UDA	97,1 %
	ISBI-2017	95,9 %

Полученные результаты свидетельствуют о высокой точности классификации изображений меланомы и доброкачественных образований нейронными сетями. Однако в ряде исследований отмечается наличие ошибок 1-го и 2-го рода при обнаружении меланомы [19], что оставляет открытым вопрос о совершенствовании архитектур нейронных сетей и формировании более качественных и репрезентативных наборов данных.

В процессе анализа меланоцитарных изображений была выявлена зависимость точности распознавания от типа применяемого оптического дерматоскопа, что обуславливает необходимость дообучения на изображениях от конкретных устройств для повышения точности распознавания.

4. Система раннего диагностирования меланомы

В России существуют системы аналитической поддержки врачей-рентгенологов при скрининнговых маммографических исследованиях, демонстрирующие высокую точность при обнаружении рака груди [23, 24].

Разрабатываемая система относится к системам автоматического диагностирования меланомы на основе анализа меланоцитарных изображений (Computer aided diagnosis) [25-27].

Для повышения вероятности обнаружения меланомы на ранней стадии необходимо использовать 2 существующих подхода совместно. Визуальный осмотр непрофильным специалистом, например, врачом общей практики при диспансеризации (скрининге), поможет выявить подозрительные образования. Фотография подобного образования загружается в систему поддержки принятия решений, которая анализирует изображение и

выдает определенный вероятностный результат, на основании которого врач принимает решение о необходимости дополнительных консультаций у профильного специалиста. Функциональная схема разрабатываемой системы представлена на "рис. 1".

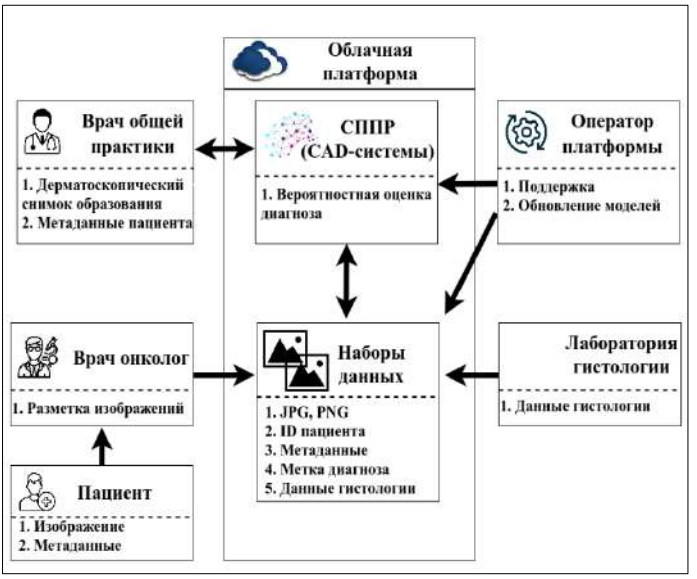


Рис. 1. Функциональная схема разрабатываемой системы
Fig. 1. Functional diagram of the developing system

В целом, задача обнаружения меланомы может включать три подзадачи [5]:

1. Предобработка изображения.

2. Сегментация изображения, выделение участка с новообразованием (определение его размеров в масштабе).

3. Классификация изображений на основе алгоритмов глубокого обучения.

На этапе предобработки выполняется преобразование изображения к стандартным размерам, нормализация по каждому каналу модели RGB.

Сегментация изображения может выполняться на основе цветовой оптимизации каналов изображения [28, 29], методов фиксации уровня [30], методы формирования однородных областей [31], применения глубоких нейронных сетей [32-34].

При обнаружении меланомы используются нейронные сети на основе архитектуры EfficientNet [35].

Разработанный прототип системы включает в себя подсистему разметки изображений врачом-дерматологом для формирования собственного набора данных и подсистему классификации меланоцитарных изображения на предмет наличия меланомы. При оценке точности классификации использовались метрики чувствительность и специфичность:

$$Sensitivity = \frac{TP}{TP + FN}, Specificity = \frac{TN}{TN + FP}, \quad (1)$$

где TP – количество верно распознанных изображений меланомы, TN – количество верно распознанных невусов, FP – количество невусов, распознанных как меланома, FN – количество меланом, распознанных как невусы.

5. Выводы и направление дальнейших исследований

В ходе проведенного анализа была обоснована актуальность разработки системы обнаружения меланомы на неинвазивной начальной стадии, что позволит повысить

вероятность успешного излечения меланомы. Разработанный прототип позволяет анализировать меланоцитарные изображения, полученные от дерматоскопа.

Направление дальнейших исследований связано с формированием собственного набора данных и повышения чувствительности и специфичности при диагностировании меланомы с использованием методов компьютерного зрения.

Список литературы / References

[1] Arnold M., Singh D. et al. Global burden of cutaneous melanoma in 2020 and projections to 2040. JAMA dermatology, vol. 158, issue 5, 2022, pp. 495-503.

[2] Deacon D.C., Smith E.A., Judson-Torres R.L. Molecular Biomarkers for Melanoma Screening, Diagnosis and Prognosis: Current State and Future Prospects. Frontiers in Medicine, vol. 8, 2021, article no. 642380, 26 p.

[3] Gershenwald J.E., Scolyer R.A. et al. Melanoma staging: evidence-based changes in the American Joint Committee on Cancer eighth edition cancer staging manual. CA: A Cancer Journal for Clinicians, vol. 67, issue 6, 2017, pp. 472-492.

[4] Siegel R.L., Miller K.D., Jemal A. Cancer statistics, 2020. CA: A Cancer Journal for Clinicians, vol. 70, issue 1, 2020, pp. 7-30.

[5] Thanh D.N., Prasath V.B. et al. Melanoma skin cancer detection method based on adaptive principal curvatue, colour normalisation and feature extraction with the ABCD rule. Journal of Digital Imaging, vol. 33, issue 3, 2020, pp. 574-585.

[6] Gu F., Chen T.H.et al. Combining common genetic variants and non-genetic risk factors to predict risk of cutaneous melanoma. Human molecular genetics, vol. 27, isuue 23. 2018, pp. 4145-4156.

[7] Landi M.T., Bishop D.T. et al. Genome-wide association meta-analyses combining multiple risk phenotypes provide insights into the genetic architecture of cutaneous melanoma susceptibility. Nature genetics, vol. 52, issue 5, 2020, pp. 494-504.

[8] Siegel R.L., Miller K.D. et al. Cancer statistics, 2022. CA: A Cancer Journal for Clinicians, vol. 72, issue 1, 2022, pp. 7-33.

[9] Teterycz P., Ługowska I. et al. Comparison of seventh and eighth edition of AJCC staging system in melanomas at locoregional stage. World Journal of Surgical Oncology, vol. 17, issue 1, 2019, pp. 1-7.

[10] Gershenwald M.S., Scolyer R.A. Melanoma Staging: American Joint Committee on Cancer (AJCC) 8th Edition and Beyond. Annals of Surgical Oncology, vol. 25, 2018, pp, 2105-2110.

[11] Неретин Е.Ю., Садреева С.Х. «Истинная» заболеваемость меланомой кожи по результатам масштабной трехдневной кампании по ранней диагностике в крупном промышленном центре. Ульяновский медико-биологический журнал, no. 1, 2021 г., стр. 71-83 / Neretin E.Yu., Sadreeva S.Kh. The «true» incidence of melanoma of the skin from the results of a large-scale three-day campaign for early diagnosis in a large industrial center. Ulyanovsk Medical Biological Journal, no. 1, 2021, pp. 71-83 (in Russian).

[12] Srivastava S, Koay E.J. et al. Cancer overdiagnosis: a biological challenge and clinical dilemma. Nature Reviews Cancer, vol. 19, 2019, pp. 349-358.

[13] Glazer A.M., Farberg A.S. et al. Quantitative ABCD parameters measured by a multispectral digital skin lesion analysis device for evaluation of suspicious pigmented skin lesions strongly correlate with clinical ABCD observations. Dermatology Online Journal, vol. 23, no. 8, 2017, 4 p.

[14] Senan E.M., Jadhav M.E. Analysis of dermoscopy images by using ABCD rule for early detection of skin cancer. Global Transitions Proceedings, vol. 2, issue 1, 2021, pp. 1-7.

[15] Saghir U., Devendran V. A Brief Review of Feature Extraction Methods for Melanoma Detection. In Proc. of the 7th International Conference on Advanced Computing and Communication Systems (ICACCS), vol. 1, 2021, pp. 1304-1307.

[16] Kittler H. Evolution of the Clinical, Dermoscopic and Pathologic Diagnosis of Melanoma. Dermatology Practical & Conceptual, vol. 11, no. S1, 2021, pp. 1-10.

[17] Babino G., Lallas A. et al. Melanoma diagnosed on digital dermoscopy monitoring: A side-by-side image comparison is needed to improve early detection. Journal of the American Academy of Dermatology, vol. 85, issue 3, 2021, pp. 619-625.

[18] Saba T. Computer vision for microscopic skin cancer diagnosis using handcrafted and non-handcrafted features. Microscopy Research and Technique, vol. 84, issue 6, 2021, pp. 1272-1283.

[19] Jaimes N., Marghoob A.A. Triage amalgamated dermoscopic algorithm. Journal of the American Academy of Dermatology, vol. 82, issue 6, 2020, pp. 1551-1552.
[20] Gillies R.J., Schabath M.B. Radiomics improves cancer screening and early detection. Cancer Epidemiology and Prevention Biomarkers, vol. 29, issue 12, 2020, pp. 2556-2567.
[21] Brinker T. J. et al. Deep learning outperformed 136 of 157 dermatologists in a head-to-head dermoscopic melanoma image classification task //European Journal of Cancer. – 2019. – Т. 113. – С. 47-54.
[22] Haenssle H. A., Fink C. et al. Man against machine: diagnostic performance of a deep learning convolutional neural network for dermoscopic melanoma recognition in comparison to 58 dermatologists. Annals of oncology, vol. 29, issue 8, 2018, pp. 1836-1842.
[23] Ratul M.A.R., Mozaffari M.H. et al. (2020). Skin lesions classification using deep learning based on dilated convolution. BioRxiv, 2019, 860700.
[24] Akram T., Lodhi H.M.J. et al. A multilevel features selection framework for skin lesion classification. Human-centric Computing and Information Sciences, vol. 10, issue 1, 2020, pp, 1-26.
[25] Павлович П.И., Бронов О.Ю. и др. Сравнительное исследование результатов анализа данных цифровой маммографии системы на основе искусственного интеллекта «Цельс» и врачей-рентгенологов. Digital Diagnostics, том 2, no 2S, 2021 г., стр. 22-23 / Pavlovich P.I., Bronov O.Yu. et al. Comparative study of the digital mammography data analysis system based on artificial intelligence “Celsus” and radiologists. Digital Diagnostics, vol. 2, no. 2S, 2021, pp. 22–23 (in Russian).
[26] Карпов О.Э., Бронов О.Ю. и др. Компаративное исследование результатов анализа данных цифровой маммографии системы на основе искусственного интеллекта «цельс» и врачей-рентгенологов. Вестник Национального медико-хирургического Центра им. НИ Пирогова, том 16, вып. 2, 2021 г., стр. 86-92 / Karpov O.E., Bronov O.Yu. et al. Comparative study of data analysis results of digital mammography ai-based system «Celsus» and radiologists. analysis of clinical cases. Bulletin of Pirogov National Medical & Surgical Center, vol. 16, issue 2, 2021, pp. 86-92 (in Russian).
[27] Foahom Gouabou A.C., Damoiseaux J.L. et al. Ensemble Method of Convolutional Neural Networks with Directed Acyclic Graph Using Dermoscopic Images: Melanoma Detection Application. Sensors, vol. 21, issue 12, 2021, article no. 3999, 19 p.
[28] Luna-Benoso B., Martínez-Perales J.C. et al. Melanoma detection in dermoscopic images using a cellular automata classifier. Computers, vol. 11, issue 1, 2022, 13 p.
[29] Rizzi M., & Guaragnella C. A Decision Support System for Melanoma Diagnosis from Dermoscopic Images. Applied Sciences, vol. 12, issue 14, 2022, article no. 7007, 15 p.
[30] Thanh D.N.H., Hien N.N. et al. Adaptive Thresholding Skin Lesion Segmentation with Gabor Filters and Principal Component Analysis. Advances in Intelligent Systems and Computing, vol. 1125, 2019, pp. 811-820.
[31] Thanh D.N.H., Erkan U. et al. A Skin Lesion Segmentation Method for Dermoscopic Images Based on Adaptive Thresholding with Normalization of Color Models. In Proc. of the IEEE 2019 6th International Conference on Electrical and Electronics Engineering, 2019, pp. 116-120.
[32] Thanh D.N.H., Hien N.N. et al. Automatic Initial Boundary Generation Methods Based on Edge Detectors for the Level Set Function of the Chan-Vese Segmentation Model and Applications in Biomedical Image Processing. Advances in Intelligent Systems and Computing, vol. 1125, 2019, 2018, pp. 171-181.
[33] Wong A., Scharcanski J., Fieguth P. Automatic Skin Lesion Segmentation via Iterative Stochastic Region Merging. IEEE Transactions on Information Technology in Biomedicine, vol. 15, issue 6, 2011, pp. 929-936.
[34] Jaisakthi S.M., Mirunalini P., Aravindan C. Automated skin lesion segmentation of dermoscopic images using GrabCut and k-means algorithms. IET Computer Vision, vol. 12, issue 8, 2018, pp. 1088-1095. doi: 10.1049/iet-cvi.2018.5289.
[35] Burdick J, Marques O et al. Rethinking Skin Lesion Segmentation in a Convolutional Classifier. Journal of Digital Imaging, vol. 31, issue 4, 2018, pp. 435-440.
[36] Al-Masni M.A., Al-Antari M.A. et al. Skin lesion segmentation in dermoscopy images via deep full resolution convolutional networks. Computer Methods and Programs in Biomedicine, vol. 162, 2018, pp. 221-231.
[37] Ha Q., Liu B., Liu F. Identifying melanoma images using efficientnet ensemble: Winning solution to the siim-isic melanoma classification challenge. arXiv preprint arXiv:2010.05351, 2020, 6 p.

Информация об авторах / Information about authors

Александр Васильевич КОЗАЧОК – доктор технических наук, доцент, заведующий лабораторией безопасного программного обеспечения и анализа данных. Сфера научных интересов: методы и системы защиты информации, кибербезопасность, машинное обучение, анализ данных.

Alexander Vasilievich KOZACHOK – Doctor of Technical Sciences, associate professor, Head of the Laboratory of Secure Software and Data Analysis. Research interests: algebraic structures in the information security methods and systems, cybersecurity, machine learning, data analysis.

Андрей Андреевич СПИРИН – кандидат технических наук, научный сотрудник. Его научные интересы включают распознавание образов, системы искусственного интеллекта.

Andrei Andreevich SPIRIN is a Candidate of Technical Sciences, Research Associate. His research interests include pattern recognition, artificial intelligence systems.

Елена Сергеевна КОЗАЧОК – врач-дерматолог. Сфера научных интересов: машинное обучение, анализ меланоцитарных изображений.

Elena Sergeevna KOZACHOK – Dermatologist. Her research interests include artificial intelligence systems, melanocytic image analysis.