DOI: 10.15514/ISPRAS-2021-33(3)-7

High Performance Distributed Web-Scraper

D.S. Eyzenakh, ORCID: 0000-0003-1584-1745 <eisenachdenis@gmail.com>
A.S. Rameykov, ORCID: 0000-0001-7989-6732 <arcane561@gmail.com>
I.V. Nikiforov, ORCID: 0000-0003-0198-1886 <igor.nikiforovv@gmail.com>
Peter the Great St.Petersburg Polytechnic University
29, Polytechnicheskaya, St.Petersburg, 195251, Russia

Abstract. Over the past decade, the Internet has become the gigantic and richest source of data. The data is used for the extraction of knowledge by performing machine leaning analysis. In order to perform data mining of the web-information, the data should be extracted from the source and placed on analytical storage. This is the ETL-process. Different web-sources have different ways to access their data: either API over HTTP protocol or HTML source code parsing. The article is devoted to the approach of high-performance data extraction from sources that do not provide an API to access the data. Distinctive features of the proposed approach are: load balancing, two levels of data storage, and separating the process of downloading files from the process of scraping. The approach is implemented in the solution with the following technologies: Docker, Kubernetes, Scrapy, Python, MongoDB, Redis Cluster, and CephFS. The results of solution testing are described in this article as well.

Keywords: web-scraping; web-crawling; distributed data collection; distributed data analysis

For citation: Eyzenakh D.S., Rameykov A.S., Nikiforov I.V. High Performance Distributed Web-Scraper. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 3, 2021, pp. 87-100. DOI: 10.15514/ISPRAS-2021-33(3)-7

Высокопроизводительный распределенный веб-скрапер

Д.С. Эйзенах, ORCID: 0000-0003-1584-1745 <eisenachdenis@gmail.com>
А.С. Рамейков, ORCID: 0000-0001-7989-6732 <arcane561@gmail.com>
И.В. Никифоров, ORCID: 0000-0003-0198-1886 <igor.nikiforovv@gmail.com>
Санкт-Петербургский Политехнический университет Петра Великого,
Россия, Санкт-Петербург, 195251, ул. Политехническая, д. 29

Аннотация. За последнее десятилетие Интернет стал гигантским и богатейшим источником данных. Данные используются для извлечения знаний путем выполнения машинного анализа. Чтобы выполнить интеллектуальный анализ данных веб-информации, данные должны быть извлечены из источника и помещены в аналитическое хранилище. Это ETL-процесс. Разные веб-источники имеют разные способы доступа к своим данным: либо API по протоколу HTTP, либо парсинг исходного кода HTML. Статья посвящена подходу к высокопроизводительному извлечению данных из источников, не имеющих API для доступа к данным. Отличительными особенностями предлагаемого подхода являются: балансировка нагрузки, двухуровневая подсистема данных и отделение процесса загрузки файлов от процесса парсинга. Подход реализован в решении со следующими технологиями: Docker, Kubernetes, Scrapy, Python, MongoDB, Redis Cluster и CephFS. Результаты тестирования решения также описаны в этой статье.

Ключевые слова: веб-скрапинг; веб-краулинг; распределенный сбор данных; распределенный анализ данных

Для цитирования: Эйзенах Д.С., Рамейков А.С., Никифоров И.В. Высокопроизводительный распределенный веб-скрапер. Труды ИСП РАН, том 33, вып. 3, 2021 г., стр. 87-100 (на английском языке). DOI: 10.15514/ISPRAS–2021–33(3)–7.

1. Introduction

Due to the rapid development of the network, the World Wide Web has become a carrier of a large amount of information. The data extraction and use of information has become a huge challenge nowadays. Traditional access to the information through browsers like Chrome, Firefox, etc. can provide a comfortable user experience with web pages. Web sites have a lot of information and sometimes haven't got any instruments to access over the API and preserve it in analytical storage. The manual collection of data for further analysis can take a lot of time and in the case of semi-structured or unstructured data types the collection and analyzing of data can become even more difficult and time-consuming. The person who manually collects data can make mistakes (duplication, typos in the text, etc.) as far as the process is error-prone.

Web-scraping is the technique which is focused on solving the issue of the manual data processing approach [1]. Web scraping is the part of ETL-process and is broadly used in web-indexing, web-mining, web data integration and data mining. However, many existing solutions do not support parallel computing on multiple machines. This significantly reduces performance, limiting the system's ability to collect large amounts of data. A distributed approach allows you to create a horizontally scalable system performance of which can be increased depending on the user's needs.

The article proposes an approach to organize distributed, horizontally scalable scraping and distributed data storage. Using an orchestration system greatly simplifies the interaction with the system, and the support of automatic load balancing avoids overloading individual nodes.

2. Existing Web Scraping Techniques

Typically, web scraping applications imitate a regular web user. They follow the links and search for the information they need. The classic web scraper can be classified into two types: web-crawlers and data extractors (fig. 1).

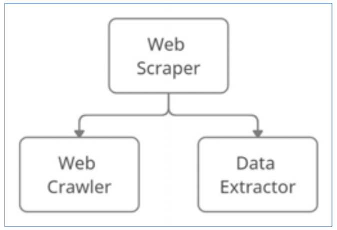

Fig. 1 Web-scraper structure

A web-crawler (or called a spider, spiderbot) is the first type of data web-scraping. The crawler is a web robot also known as an Internet bot that scans the World Wide Web typically operated by search engines for the purpose of Web indexing [2]. The crawling procedure starts with the list of seed URLs. The program identifies all the links that exist on seed pages and stores them. After that, the list of all links is recursively visited. This process continues until all URLs will be visited. There are several types of web-crawlers, but all of them can be divided into a common crawler and focused crawler.

Focused crawler searches for the most suitable pages according to the topic that is defined by the user. This goal is achieved by using algorithms of intelligent text analysis. It ensures that web pages can only be crawled for information related to the specific topic. In the server's perspective, there are single machine crawlers or distribution crawlers. The information crawling can be achieved by dividing into several nodes and their cooperation, which improves the efficiency and performance of the crawler.

The second type of web scraper is a data extractor [3]. The website contains a large amount of information and the analyst cannot spend a lot of time manually collecting and converting this data into the desired format. Besides that, a web page can contain a lot of unstructured data that means it can contain noise or redundant data. Data extractors can easily extract large and unstructured data and convert them into a comprehensive and structured format. The extraction process starts with indexing or crawling. In the crawling process, the crawler finds a list of the relevant URLs that the data extractor will process. In these web pages a lot of junk and useful data is mixed. The data extractor extracts the needed information from the web-pages. Data extractor contains a lot of techniques [4] for extraction data from HTML pages.

3. Comparison Analysis of Systems

Here is an overview and comparison of web scraping frameworks for fast scanning any kind of data, distributed scraping systems for increasing the performance, and orchestration systems.

3.1 Scraping tools

There are various tools for working with web scrapers. They can be divided into three categories: libraries, frameworks, and desktop or web-based environments.

3.1.1 Libraries

Modern web resources may contain various information. Due to this circumstance, certain flexibility is required for configuring and implementing web scraping tools. The libraries guarantee access to the web resource. Most of the library implement the client side of the http protocol, then the resulting web page is parsed and the data is retrieved using string functions such as regular expressions, splitting and trimming, etc. [5]. Also, third-party libraries can help with implementing more complex analysis, for example, building an html-tree and XPATH mappings.

One of the most popular site access libraries is *libcurl*. It supports the major features of the HTTP protocol, including SSL certificates, HTTP POST, HTTP PUT, FTP uploading, HTTP form-based upload, proxies, cookies and HTTP authentication. Moreover, it can work with many programming languages. In Java, the Apache HttpClient package emulates HTTP main features, i.e., all request methods, cookies, SSL and HTTP authentication, and can be combined with HTML parsing libraries. Java also supports XPath and provides several HTML cleaning libraries, such as *jsoup*. Programs like *curl* (libcurl) and *wget* implement the HTTP client layer, while utilities such as *grep*, *awk*, *sed*, *cut* and *paste* can be used to parse and transform contents conveniently.

3.1.2 Desktop or web application

Desktop applications are implementations of web scrapers that are designed for noncoding professionals. This kind of web scraper contains a graphical shell that makes it easier to create and support web robots. Typically, these applications include an embedded web browser, where the user can navigate to a target web resource and interactively select page elements to extract them, avoiding any kind of *regex*, *XPath* queries, or other technical details. In addition, modules are capable of generating several kinds of outputs, such as CSV, Excel and XML files, and queries that are inserted into databases. The main disadvantages of desktop solutions are commercial distribution and limited API access, which make it difficult to embed these web scrapers into other programs.

Table 1. Comparison of scraping frameworks

Feature/ Framework	Scrapy	PySpider	NodeCralwer	Apify SDK	Selenium
Builtin Data Storage Supports	Customizable	CSV, JSON	CSV, JSON, XML	JSON, CSV, XML, HTML	Customizable
Suitable for Broad Crawling	Yes	No	Yes	Yes	No
Builtin Scaling	Yes	Yes	No	Yes	No
Support AJAX	No	Yes	No	Yes	Yes
Available Selectors	CSS, Xpath	CSS, Xpath	CSS, Xpath	CSS	CSS, Xpath
Builtin Interface for Periodic Jobs	No	Yes	No	Yes	No
Speed (Fast, Medium, Slow)	Fast	Medium	Medium	Medium	Very slow
CPU Usage (Fast, Medium, Slow)	Medium	Medium	Medium	Medium	High
Memory Usage (High, Medium, Low)	Medium	Medium	Medium	High	Medium
Github Forks	9000	3600	852	182	6200
Github Stars	39600	14800	5800	2700	19800
License	BSD License	Apache License 2.0	MIT	Apache License 2.0	Apache License 2.0

3.1.3 Frameworks

Programming libraries have their limitations. For example, you need to use one library for accessing a web page, another for analyzing and extracting data from HTML pages. The architecture designing and the compatibility of the library's checking process can take a significant amount of time. Frameworks are a complete solution for developing web scrapers. Comparison results of popular frameworks for implementing web scrapers are presented in the article as well (Table 1).

Comparison is made according to the following criteria. Built-in Data Storage Supports - supporting types of files or other storage.

Suitable for Broad Crawling – this type of crawler covers a large (potentially unlimited) number of domains, and is only limited by time or another arbitrary constraint, rather than stopping when the domain has already been crawled to completion or when there are no more requests to perform. These are called *broad crawls*, which are the typical crawlers used by search engines. Speed, CPU usage, and memory usage can represent system performance. GitHub Forks, GitHub Starts, Last Update can inform the state of the framework, support and community activity.

3.2 Orchestration and containerization systems

Since our system should support horizontal and vertical scaling, it must support the orchestration system. The orchestration system will monitor the status of services, distribute the load among the nodes in the cluster, taking into account the resources of each of these nodes. An orchestration system is a support for the compatibility of software products that communicate with each other through remote procedure calls (RPC). There are many solutions on the market today, but some of them are bound to specific companies. Such systems can impose many different restrictions such as: territorial limitations, bounded choice of cloud computing service, the chance to be left without data due to any external factors. And so, at the moment, there are the following tools: Kubernetes, Docker Swarm, Apache Mesos.

Based on the paper [6], we can conclude that the Kubernetes orchestration system has a large coverage of the required technologies. It is also the de facto standard today, as evidenced by the fact that it is the only orchestration system that has been accepted into the Cloud Native Computing Foundation [7]. It is also used by such large companies as Amazon, Google, Intel, etc.

We were also faced with the choice of virtualization or containerization system. Referring to the fact that Kubernetes can work with virtual machines, we chose containerization, due to it consuming less resources, which was confirmed by research [8]. Also, Kubernetes in its delivery recommends to work with containers. Thus, our web scraper will be delivered as a container running Docker.

3.3 Distributed scraping system review

3.3.1 Research

Scrapy does not provide any built-in facility for running spiders in a distributed (multi-server) manner. However, there are several ways to organize work. Some of the popular solutions are Frontera, Scrapy Redis, Scrapy Cluster, and Scrapyd.

Frontera is a distributed crawler [9] [10] system. Based on the description of the project, we can say that the system is a separately distributed web crawler. It is designed to collect a large number of URLs as data sources. The system does not have a built-in data extractor and it is not known whether it is possible to add one. Hence, we can say that the system is intended for other tasks.

Scrapy Redis [11] [12] is a system architecture that supports distributed web-scraping. In the process of crawling, the Redis database can mark the links that have been already crawled and add to the queue links that haven't crawled yet, avoiding the repeated crawling problem in the distributed process. From the Scrapy Redis description follows that Redis database is used as main storage. Redis, as the main storage, does not satisfy the ACID theorem, namely CD, which carries the consequences of losing part of the tasks, if the cluster is in an emergency state, the consequences of the loss of tasks can carry different types of damage, from re-scanning the page, which entails a decrease in production.

Scrapy Cluster [13] was taken after Scrapy Redis, which offloads Requests to a Redis instance. It has the same problem with the Redis database. Based on the project description we can find out that the system does not imply the usage of an orchestration system, this opportunity is provided to the user. It is a big disadvantage due to the reason that it is not clear how the system will behave when the basic functions of maintaining the cluster will be launched by the orchestrator. For instance, operation “liveness check” could not behave correctly and conflict with internal monitoring of the system or doesn’t work at all. Scrapy Cluster also uses the Apache Kafka message broker as a connection between the system components. The parallelism of Kafka lies in the number of sections in the topic [14]. All data that falls into the topic is balanced between sections. From one section, following the documentation, only one instance of the application can read data. Several disadvantages can be distinguished from this:

- Adding new topics will semantically separate the data, which means that the data that has been already stored in Kafka is not rebalanced internally [15]. This means that if you try to add a new instance, the spider will slow down the system until all new jobs will be balanced against the new partitions.
- In the case of Scrapy Cluster, the spiders are bound to the partition, and according to the scrapy cluster documentation, control signals for the spider can be sent to the partition (stop the scraping job) [16]. This makes it impossible to add new partitions until the end of the complete scraping session. Since concerning to the balancing formula in Kafka:
    - $hash(key)\ \%\ number\_of\_partition$

  the control signal can be sent to the wrong spider.
- Removing topics from the Kafka is impossible, which can lead to a situation when several dozen spiders stop in the cluster, the producer (sending the task) cannot rebalance the queue between the remaining spiders. This can lead to imbalance – overloading one spider and no load on others.

One of the most popular distributed scraping solutions using the Scrapy framework is Scrapyd [17]. Scrapyd allows you to deploy and manage multiple Scrapy projects using the json API. The wrapper provides the ability to store a queue in a database, and also allows you to manage several machines at once. With all the visible advantages, the system has not been without drawbacks. The lack of a balancing system does not allow the use of a larger number of nodes. When sending a request for scraping, you must specify the ip address of a specific node and before that make sure that it is not 100% loaded.

3.3.1 Conclusion

After conducting research and analysis of various web scraping tools, it was decided to use the Scrapy framework. It does not restrict the developer by its license or capabilities and is also used in many companies [18]. At the same time, it has a convenient architecture for building any web scraper. A convenient mechanism for adding additional software easily compensates drawbacks of the framework. Such as support of JavaScript, etc

Considering all of the above, we decided to develop a system that would solve the problems of existing solutions, allowing programs written in the Scrapy framework to work, and also use the Kubernetes orchestration system.

4. Methodology

4.1 Overall architecture of our distributed scraping system

Distributed scraper architecture presents 3 functional layers (fig. 2):

User interface layer;

Web scraping layer;

Data storage layer.

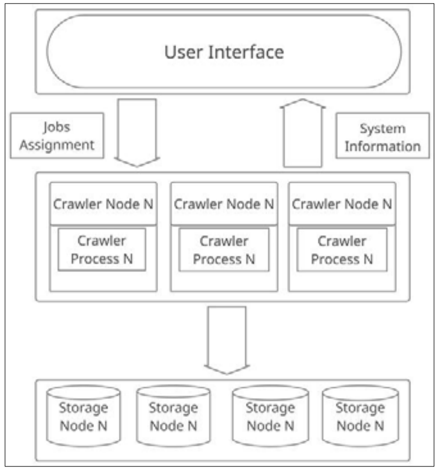

Fig. 2 Distributed approach for data extraction

The user interface layer is responsible for interacting with the end-user, the user can send control commands to the cluster, receive a response and see the scraping statistics at a given point of time. The web scraping layer is a layer of distributed web spiders that do not store state by themselves, that is to say, they receive it from the user interface layer, so they can be multiplied, so this layer is responsible for all the scraping logic of sites. The data storage layer is responsible for storing all collected information, which includes texts and media content.

4.2 System design

Fig. 3 shows a detailed diagram of the system operation.

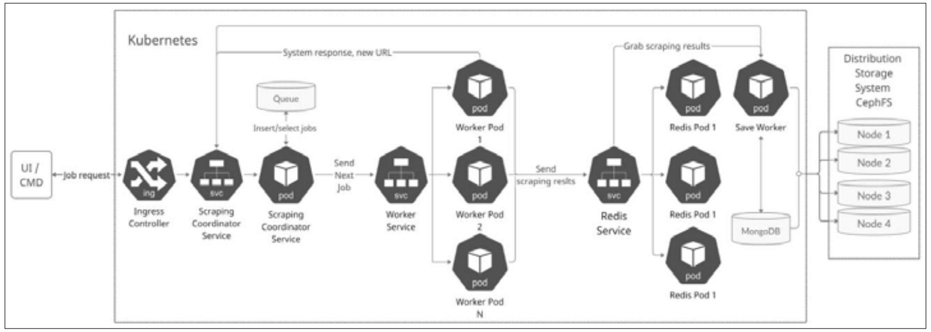

Fig. 3. System design

4.2.1 Ingress Controller

An ingress controller is an element of the orchestrator infrastructure, the main task of which is to proxy external traffic to services within the cluster. It also performs other tasks such as SSL termination and balancing and routing of traffic based on names and URLs.

Insert reference to the figure, the ingress controller is the entry point to the cluster for the end-user and, based on the hostnames, redirects requests to a particular environment.

4.2.2 Services

There are three services in our system: Scrapy Coordinator Service, Spider Service, Redis Service. Pods in the cluster are not permanent, they can be stopped, they can change the IP address, they can be moved to another element of the cluster (node). Because of this, we faced the problem of controlled access to pods. Service solves this problem by acting as an abstraction that provides access to pods, has access sharing mechanisms, and also has mechanisms for discovering all services that match the conditions. The end-user or program does not need to know the IP address of a particular pod, he can refer to the domain name (for example crawler.company 1, where crawler is the generic pod name, and company 1 is the namespace name) and gain access to one of the pods.

Service also balances traffic using the Round-robin method. This algorithm is already built into the standard delivery of Service Kubernetes and allows you to evenly distribute tasks across all working nodes.

4.2.3 Scraping Coordinator and Workers

As we saw in fig. 3, the scraping coordinator and workers run in separate containers. Scraping coordinator working with PostgreSQL [19] database. PostgreSQL works in Master Slave mode, namely with asynchronous replication, asynchronous replication allows you not to wait for a response from the master, thereby not slowing down its work, and replication will not stop if the slave is disabled, this solution is quite simple and reliable in the sense that this replication mode is shipped with the database distribution and solves most fault tolerance problems. It works as a storage of the queue of requests and processed links.

Worker consists of scrapy spiders written in scrapy and HTTP-wrapper. The wrapper is a web server written in Python with micro-framework Klein. Klein framework based on Twisted – the most powerful Python asynchronous framework, provides easy integration with Scrapy that uses Twisted for all HTTP traffic.

The following is the sequence followed by the system when initiating a scraping request from the perspective of the scraping coordinators and workers.

1) Each scraping coordinator receives a list of URLs to crawl and the number of workers instances a user would like to use. The scraping coordinator checks in the PostgreSQL database if the seed URL has already been processed. If not, the coordinator creates a job that includes the seed URL and custom settings for the spider. After that, it adds those jobs to a queue along with a user scrape endpoint of the worker based on the number of received from the user.
2) The scraping coordinator pops jobs from the queue and passes them to the scraper's URLs.
3) HTTP-wrapper takes a job, takes settings, and starts Scrapy spider. If a worker is free and accepts the URL, it sends back an acceptance message. If it is busy and has no free threads to handle the request, it replies with a rejected message. The scraping coordinator adds those URLs for which it received a rejection message, back into the queue.
4) Spider scrape data and send it to the Redis cluster. If the user has enabled the deep scraping function, extracts all the child URLs (HREF elements in the web page) and passes them to the scraping coordinator.

4.2.4 Scrapy Spider

It is a standalone spider program using the Scrapy framework. Has ample opportunities, such as: JavaScript processing. Since most pages have dynamically generated content these days, it is no longer enough to just browse static pages. If this factor is not taken into account, a large amount of data can be lost during screening. Scrapy in the standard delivery cannot work with dynamic content. But there is a possibility of connecting additional modules – headless web browsers

Supports processing web pages with pagination. Unlike traditional search engines, which write every next page to the seed URL queue. This can lead to high code interfacing, poor readability, and spider startup costs. Using Scrapy's system call-back mechanism as a bridge, URL queue creation and content crawling operation are performed separately, which solves the shortcomings of traditional crawlers [20].

The system also can use middleware to dynamically change IP proxy, as well as the User-Agent value. All this significantly reduces the chance of blocking by a web resource.

4.2.5 Redis-Cluster

A memory database is needed to quickly save results, thereby blocking save operation minimizes waiting on the part of the scraper, increasing its performance. Redis Cluster acts as an intermediate caching layer, it provides fast storage of information, since all data is stored in RAM. Spiders do not stand idle waiting for information to be saved, this is important because the write speed in distributed file systems is rather low [21]. Redis stores information primarily as a dictionary, that is, on a key-value basis. The key is the site URL, and the value is the result of scraping in the form of a JSON file.

4.2.6 Save Worker

Save Worker performs the task of post-processing information. It scans keys in Redis at a certain frequency. After the information reaches the desired size (recommended 4+ Gb [21]) or is not updated, it starts downloading information to itself and simultaneously deleting data from Redis. After that, it starts scanning the scraping result, looking for certain marks in it in order to load the missing files into a distributed file storage. Thus, the spiders unload the waiting time for downloading large files with an indefinite download time. After that SaveWorker saves all results to the database with one request.

4.2.7 MongoDB

MongoDB [22] is well adapted to our problem. It is a document-oriented database management system and does not require a description of the schema and tables.

MongoDB has the ability to scale horizontally.

4.2.8 CephFS

It is a software-defined distributed file system that can scale flexibly to store petabytes of data. Ceph is able to replicate data between nodes, as well as balance the load between them. When a node fails, Ceph can self-heal without downtime, thereby improving system availability. Ceph offers 3 types of interface for interacting with storage, a POSIX compatible [23] file system, an S3 storage and a block device, thus providing higher compatibility with already written software.

4.2.9 Sharing resources between users

To restrict manual cluster management, unique user environment settings are specified. After that, they are automatically added to the declarative description of the cluster configuration file. This file is stored alongside the project output in YML format in the Git [24] repository.

The system supports resource sharing using the tools provided by the orchestration system, namely namespace and ingress controller. Ingress controller redirects the user to a particular namespace, according to the URL. The software located in one namespace does not have direct access to the resources of another namespace, just as each namespace can be allocated quotas for processor time and the amount of RAM.

5. Approbation and Testing

In order to verify the effect of distributed crawlers builds an experimental stand. The three servers use the 64-bit Debian 9 operating system.

5.1 Scalability

Scrapers do not have an internal state, this is confirmed by the fact that the coordinator transfers the state to each individual scraper and does not store the subsequent state, but writes it to the in-memory database and to the coordinator, therefore such a system scale well horizontally.

5.2 Chaos monkey testing

Chaos monkey is a set of software tools that allows you to simulate crashes on a live system. It analyzes the system for all its critical components and disables them in different sequences. This allows you to observe the actions of the system during emergency operation, as well as identify critical points of the system. During testing, a situation occurred when, in aggressive mode, the testing software disabled both the Master and Slave PostgreSQL servers.

5.3 System speed test

The first experiment examines the overall speed of the system. For this, the Scrapy project was developed and deployed in the system, data collection takes place within a single website. The main database for storing data is the MongoDB database management system based on CephFS distributed file storage. Text data is stored as JSON files, images and other data are downloaded directly to file storage.

Testing was carried out for 5 hours, the same number of seed-URLs were chosen as input parameters, which makes it possible to ensure that the web crawler will follow the same route from the links. The performance test results are presented in the table. Each node ran 5 instances of Scrapy. This value was chosen empirically based on the load on the systems, and also on the network.

Table 2. Result of performance test

Time/h	One	Three	Five
(Single node) Pages	6650	19267	31 920
Elements	77200	223 880	370 560
(Two-node) Pages	11970	34 114	58 653
Elements	131500	368 200	631 200
(Tree-node) Pages	15960	46 922	76 927
Elements	257000	724 740	1 259 300

It can be seen by reproducing the above experiments (table 2) that the data acquisition experiment results of the distributed scraping shows that the efficiency of a node crawling is lower than that of two nodes crawling at the same time. Compared with the stand-alone spider, it can get more pages and run more efficiently.

5.4 Balancing test

Since the system contains several fail-safe elements and has the ability to distribute the load, its stability should be checked.

At first, the work of the system for distributing the load on the nodes was checked. The launch was carried out on 100 seed URLs, which contain tabular data, text, as well as data stored in various types of files. During the scanning process, the crawler worked in a limited wide scan mode, that is, it could click additional links within the same domain.

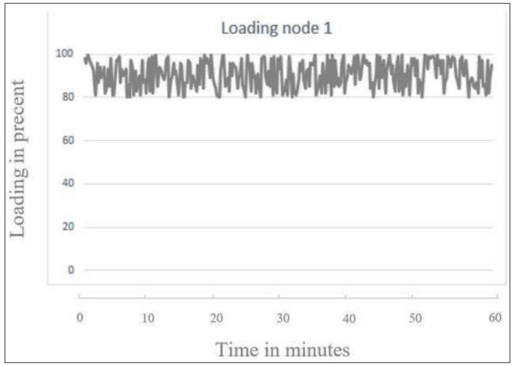

Fig. 4. Loading node №1

Testing was carried out within 5 hours. During testing, there were no emergencies.

On the graphs of the load of nodes (fig. 4-6), we can see that the balancer copes well with its task; the system is working quite stably, there are no strong drawdowns and spikes in performance.

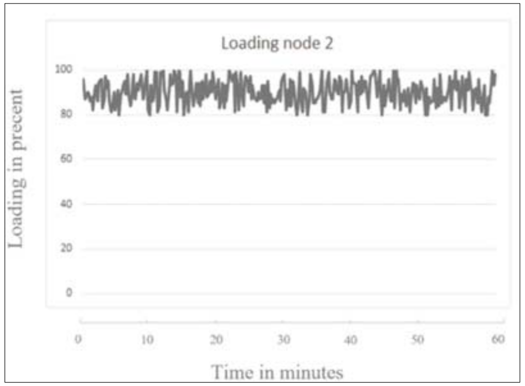

Fig. 5 Loading node №2

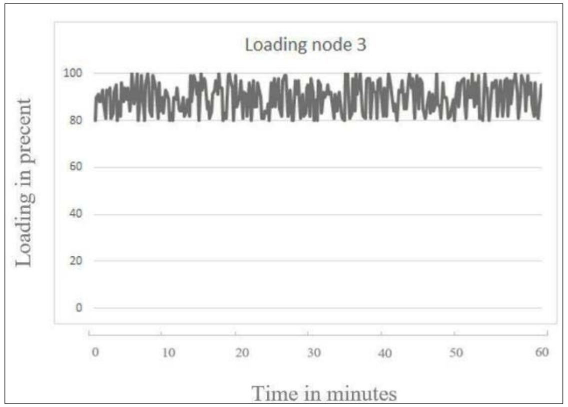

Fig. 6 Loading node №3

Table 3. Sability test

Time/h	One	Two	Three	Four	Five
Pages	15343	14903	16013	14850	15025

On the table 3, we can see that the number of collected items for each hour is almost the same. This fact shows the stability of the system. This is facilitated by the work of the balancing algorithm, which allows not to overload the system nodes and the optimal load.

6. Conclusion

As part of the work, the following work has been done:

- Classification of information extraction solutions has been introduced;
- A review of existing distributed web scrapers implementations has been conducted;
- A comparative analysis of the considered solutions is carried out.

In the course of the comparative analysis, deficiencies were found in existing solutions, namely, the lack of an orchestration system, problems with horizontal scalability implementations, and deployment of applications.

An architecture has been proposed, which is headed by the Kubernetes orchestration system, which monitors the health of each element of the cluster and shares access to resources.

During the analysis of the work of the resulting system, it showed its viability. Reducing the time required to retrieve data from web resources, connecting additional work nodes to work. And so, the increased stability of the system due to replication of the data storage, the use of a load balancer, and an intermediate storage layer in the Redis Cluster.

References

[1] Deepak Kumar Mahto, Lisha Singh. A dive into Web Scraper world. In Proc. of the 3rd International Conference on Computing for Sustainable Global Development (INDIACom), 2016, pp. 689-693.

[2] Web Cralwer. Available at: https://webbrowsersintrodu ction.com/.

[3] Momin Saniya Parvez, Khan Shaista Agah Tasneem, Shivankar Sneha Rajendra, Kalpana R. Bodke. Analysis of Different Web Data Extraction Techniques. In Proc. of the International Conference on Smart City and Emerging Technology (ICSCET), 2018, pp. 1-7.

[4] Anand V. Saurkar, Kedar G. Pathare, Shweta A. Gode. An Overview On Web Scraping Techniques and Tools. International Journal on Future Revolution in Computer Science & Communication Engineering, vol. 4, no. 4, 2018, pp. 363-367.

[5] Rohmat Gunawan, Alam Rahmatulloh, Irfan Darmawan, Firman Firdaus. Comparison of Web Scraping Techniques: Regular Expression, HTML DOM and Xpath. In Proc. of the International Conference on Industrial Enterprise and System Engineering, 2018, pp. 283-287.

[6] Isam Mashhour Al Jawarneh, Paolo Bellavista et al. Container Orchestration Engines: A Thorough Functional and Performance Comparison. In Proc. of the 2019 IEEE International Conference on Communications (ICC), 2019, pp. 1-6.

[7] CNCF certificate. Available at: https://www.cncf.io/certification/software-conformance/.

[8] S. Vestman. Cloud application platform - Virtualization vs Containerization. Student Thesis. Blekinge Institute of Technology, Sweden, 2017, 45 p.

[9] Distributed Frontera: Web crawling at scale. Available at: https://www.zyte.com/blog/distributed-frontera-web-crawling-at-large-scale/.

[10] Frontera documentation. Available: at https://frontera.r eadthedocs.io/en/latest/.

[11] Scrapy-Redis documentation. Available at: https://scrapy-redis.readthedocs.io/en/v0.6.x/readme.html#.

[12] Fulian Yin, Xiating He, Zhixin Liu. Research on Scrapy-Based Distributed Crawler System for Crawling Semi-structure Information at High Speed. In Proc. of the 2018 IEEE 4th International Conference on Computer and Communications (ICCC), 2018, pp. 1356-1359.

[13] Scrapy-Cluster documentation. Available at: https://scrapy-cluster.readthedocs.io/en/latest/.

[14] Kafka documentation. Intro. Available at: https://kafka.apache.org/documentation/#introduction.

[15] Kafka official documentation. Basic_ops_modify_topic. Available: at https://kafka.apache.org/documentation.html #basic_ops_modify_topic.

[16] Scrapy-Cluster documentation. Core Concepts. Available at: https://scrapy-cluster.readthedocs.io/en/latest/topics/introduction/overview.html.

[17] Scrapyd documentation. Available at: https://scrapyd.re adthedocs.io/en/stable/.

[18] Official Scrapy framework web-site. List of companies using Scrapy. Available at: https://scrapy.org/companies/.

[19] Regina O. Obe, Leo S. Hsu. PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database. 3rd Edition, O'Reilly Media, Inc., 2017, 314 p.

[20] Deng Kaiying, Chen Senpeng, Deng Jingwei. On optimisation of web crawler system on Scrapy framework. International Journal of Wireless and Mobile Computing, vol. 18, no. 4, 2020, pp. 332-338.

[21] Jia-Yow Weng, Chao-Tung Yang. Chih-Hung Chang. The Integration of Shared Storages with the CephFS. In Proc. of the 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), 2019. pp. 93-98

[22] Shannon Bradshaw, Eoin Brazil, Kristina Chodorow. Customers who viewed MongoDB: The Definitive Guide: Powerful and Scalable Data Storage. 3rd edition, O'Reilly Media, Inc., 2019, 514 p.

[23] Abutalib Aghayev, Sage Weil et al. File systems unfit as distributed storage backends: lessons from 10 years of Ceph evolution. In Proc. of the 27th ACM Symposium on Operating Systems, 2019. pp. 353–369

[24] N. Voinov, K. Rodriguez Garzon et al. Big data processing system for analysis of GitHub events. In Proc. of the 22nd International Conference on Soft Computing and Measurements, 2019, pp. 187-190.

Информация об авторах / Information about authors

Денис Сергеевич ЭЙЗЕНАХ, студент. Область интересов: технологии контейнеризации, распределенные системы, параллельная обработка данных.

Denis EYZENAKH, student. Research interests: containerization technologies, distributed systems, parallel data processing.

Антон Сергеевич РАМЕЙКОВ, студент. Область интересов: технологии сбора данных, параллельная обработка данных, системы распределенных вычислений.

Anton RAMEYKOV, student. Research interests: data collection technologies, parallel data processing, distributed computing systems.

Игорь Валерьевич НИКИФОРОВ, к.т.н., доцент. Область интересов: параллельная обработка данных, системы распределенного хранения данных, большие данные, верификация программного обеспечения, автоматизация тестирования.

Igor NIKIFOROV, PhD (Computer Science), Assistant Professor. Research interests: parallel data processing, distributed storage systems, big data, software verification, test automation.