Avalanche: применение параллельного и распределенного динамического анализа программ для ускорения поиска дефектов и уязвимостей

М. К. Ермаков, А. Ю. Герасимов[1]
mermakov@ispras.ru, agerasimov@ispras.ru

Аннотация. В статье рассматривается подход к уменьшению времени динамического анализа программ при помощи применения параллельных вычислений при проверке выполнимости ограничений, а также при применении распределенных вычислений в процессе динамического анализа программ. Приводятся результаты практического применения данного подхода, реализованного в рамках инструмента динамического анализа Avalanche. На основе полученных результатов даётся оценка увеличения эффективности динамического анализа и рассматриваются возможности дальнейшего развития разработанных методов.

Ключевые слова: динамический анализ, обнаружение дефектов, параллельные вычисления, распределенные вычисления.

1. Введение

Решение задачи поиска дефектов и уязвимостей в программах при помощи динамического анализа программ инструментом Avalanche [1] сводится к решению следующих подзадач:

- Отслеживание потока помеченных данных в анализированной программе с параллельным сбором информации об условных переходах на трассе анализа программы при помощи дополнительного модуля Tracegrind для среды динамической инструментации программ Valgrind [2].
- Вычисление новых входных данных для программы с целью воспроизведения дефекта в потенциально опасной операции и обхода ещё не пройденных ветвей в программе при помощи решателя STP [3].

1 Работа проводится в рамках научно исследовательских работ Института системного программирования РАН в 2012-2013 годах

- Выбор наиболее предпочтительного набора входных данных для следующей итерации анализа приложения

Изначально, в инструменте Avalanche применялся последовательный подход при анализе программы (рис. 1).

1. Запускался анализ на некотором начальном наборе входных данных, собиралась трасса выполнения программы, на которой запоминались данные о потенциально опасных операциях и условных переходах, зависящих от входных данных.
2. При помощи решателя STP последовательно вычислялись новые наборы входных данных для
    - каждой потенциально опасной операции с целью генерации входных данных, подтверждающих наличие дефекта в программе
    - каждого условного перехода, зависящего от входных данных, с целью обхода не пройденных частей программы
3. Производился выбор наиболее перспективного набора входных данных для следующей итерации анализа на основе метрики наибольшего прироста не проанализированных базовых блоков программы.

Рис. 1: Общая схема работы инструмента Avalanche

В связи с тем, что инструментация и выполнение инструментированной программы средой Valgrind является достаточно тяжеловесной операцией, а также вычисление выполнимости формулы решателем STP в общем случае является NP-полной задачей, оптимизация вычисления которой в случае применения вычислителя при проведении динамического анализа программ описана в работе Вартанова и

Сидорова [4], то проведение динамического анализа программы занимает существенное время. В связи с этим встает задача оптимизации работы инструмента Avalanche для ускорения процесса анализа программы. В данной статье будут рассмотрены подходы к распараллеливанию вычислений и реализация распределенных вычислений в рамках инструмента Avalanche.

2. Параллельный динамический анализ

При анализе возможных вариантов решения задачи распараллеливания работы инструмента Avalanche рассматривались варианты распараллеливания работы дополнительного модуля Tracegrind для среды Valgrind и распараллеливания вычисления новых входных данных для следующих запусков анализа программы.

Реализация распараллеливания работы дополнительного модуля Tracegrind оказалась невозможна в связи с ограничениями инструментирующей среды Valgrind. В связи с этим предложен подход к распараллеливанию вычисления новых входных данных для следующей итерации динамического анализа программы. На рис. 2 показана принципиальная схема распараллеливания вычислений.

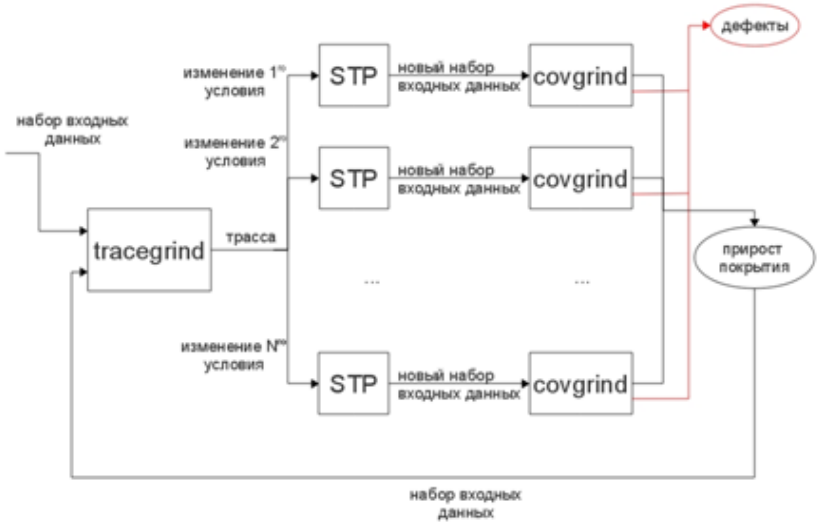

Рис. 2: Схема проведения параллельного анализа

В процессе работы дополнительного модуля Tracegrind для среды Valgrind собирается информация о трассе выполнения программы, а именно о потенциально опасных операциях и условных переходах, зависящих от входных данных. Количество условных переходов и

потенциально опасных операций на пути выполнения программы может достигать достаточно большого количества.

Для каждой потенциально опасной операции, найденной на трассе выполнения программы, необходимо произвести расчет булевской формулы при помощи STP, которая может подтвердить или не подтвердить возможность воспроизведения ошибки при выполнении потенциально опасной операции. В то же время для каждого условного перехода, зависящего от входных данных программы, необходимо вычислить булевскую формулу при помощи SAT, которая позволит определить входные данные для обхода альтернативной ветки условного перехода.

Исходя из данных, приведенных в работе [1], время работы вычислителя может занимать более 99% общего времени работы инструмента для некоторых проектов, а вычисление булевской формулы может производиться в от дельном вычислительном потоке, то, выполняя вычисление булевских формул параллельно, можно значительно сократить общее время анализа программы.

В табл. 1 приведены данные экспериментальных запусков инструмента Avalanche на наборе тестовых приложений с включенным и выключенным режимом распараллеливания работы STP. Для всех указанных проектов были использованы одинаковые начальные параметры, время работы было ограничено 2 часами (7200 секундами).

Табл. 1. Сравнительные результаты работы инструмента Avalanche в однопоточном и параллельном режимах

	Однопоточный режим анализа		Параллельный режим анализа		
Программа	Количество итераций	Количество дефектов	Количество итераций	Количество дефектов	Ускорение
qtdump	142	2	68	2	-115%
cjpeg	62	1	109	1	+76%
sndfile-mix-to-mono	48	0	313	1	+552%
avibench	64	2	171	1	+167%
mpeg2dec	200	0	661	1	+230%
mpeg3dump	151	2	283	2	+87%

Применение алгоритма определения лучшего прироста покрытия анализируемой программы, работающего при параллельном режиме в отдельном потоке на отдельной булевой формуле, вводит элемент неопределенности при выборе следующего наилучшего набора входных данных программы. Если вычисление каждой из нескольких формул занимает примерно одинаковое время, то существует вероятность получения лучшего набора данных для увеличения покрытия анализируемой программы для разных булевых формул в зависимости от работы планировщика потоков вычислительной среды, что вводит некоторую неопределенность при оценке времени нахождения определенного дефекта в анализируемой программе. Решить эту проблему можно введением некоторого арбитра, который будет накапливать набор данных для дальнейшего анализа и выбирать лучший из нескольких полученных.

На практике данная особенность может приводить к различного рода последствиям, как положительным с точки зрения качества анализа, так и отрицательным. В частности, выбор наилучшей ветки для проведения следующей итерации, отличающейся от ветки, выбранной при использовании последовательной схемы анализа, может как позволить определить ранее не обнаруженный дефект, так и не провести проверку определённой ветки выполнения с потерей ошибки, воспроизводимой на данной ветке.

Так, например, на проекте avibench, несмотря на увеличения количества итераций анализа, был не обнаружен дефект, связанный с разыменованием нулевого указателя, воспроизводящимся на единственной ветке выполнения. В другом случае, на проекте qtdump детерминизм, вносимый параллельными вычислениями, вызвал более быстрое увеличение “глубины” просмотра точек ветвления программы. Подобное “углубление” привело к росту сложности трасс и, соответственно, запросов для решателя STP. В конечном итоге большая часть времени анализа была затрачена на проверку выполнимости полученных трасс, что снизило количество итераций анализа. Падение производительности было зафиксировано только на проекте qtdump и наиболее вероятно связано с особенностями трасс, получаемых при выполнении программы.

3. Распределенный динамический анализ

В работе Исаева и Сидорова [1] выполнение каждой итерации анализа моделируется как вычисление на одной из веток дерева достижимых путей в программе. И делается предположение, что используя данную модель можно добавить возможность запуска инструмента Avalanche на некотором поддереве достижимых путей в программе, используя одно из ветвлений на пути выполнения программы как корень поддерева.

Применение подобной возможности позволит выполнять запуск инструмента Avalanche на нескольких компьютерах или в распределенной вычислительной среде, используя её вычислительные возможности.

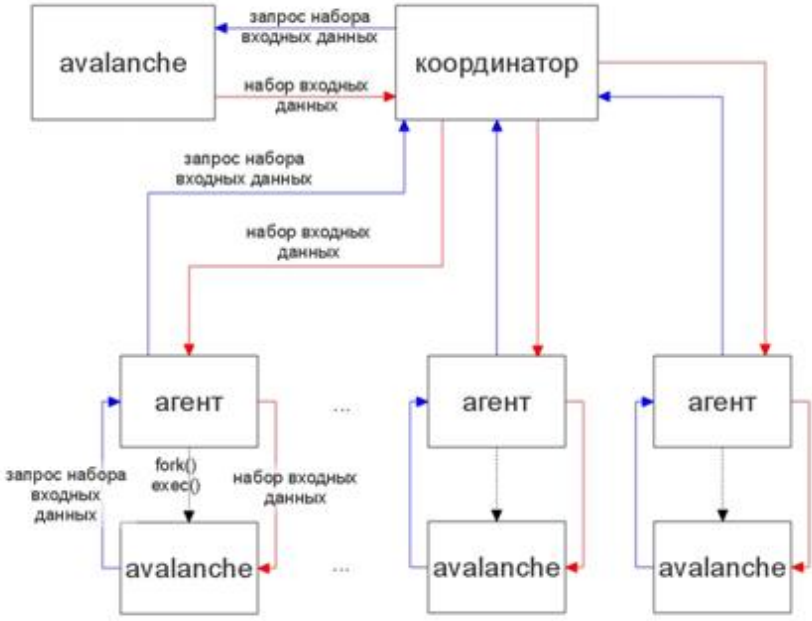

Рис. 3: Схема работы инструмента Avalanche при использовании распределённых вычислительных ресурсов

Для реализации этой возможности модифицирован алгоритм работы инструмента Avalanche:

1. В качестве входного параметра работы инструмента на ряду с набором входных данных анализируемой программы передается информация о глубине условного перехода, начиная с которой должно производиться инвертирование условия условного перехода для вычисления входных данных с целью обхода альтернативного пути выполнения программы.
2. Реализован координатор, который накапливает вычисленные наборы входных данных анализируемой программы и имеет возможность передачи набора входных данных и глубины условий для инвертирования на свободный узел вычислительной среды.

3. Реализован промежуточный агент, которая запускается на узле вычислительной среды и:
    - ожидает получения данных для анализа целевой программы от координатора
    - запускает инструмент Avalanche для анализа поддерева возможных путей программы.

Каждый агент по сути превращает узел вычислительной среды в независимый анализатор программы, который позволяет производить полноценный анализ подмножества возможных путей выполнения программы, вычисляя входные данные для обхода новых путей выполнения в программе и подтверждения дефекта для потенциально опасной операции. При этом подмножества путей, анализируемые на различных вычислительных узлах, не пересекаются между собой, за исключением редких случаев недетерминизма внутри самой исследуемой программы.

Особенность текущей реализации распределенного режима инструмента Avalanche позволяет агентам получать данные для анализа только от координатора. Координатор работает на одном из узлов распределенной вычислительной среды и накапливает наборы входных данных для анализа альтернативных путей в программе. Особенность текущей реализации координатора заключается в том, что когда на узле, где работает анализатор, проанализированы все достижимые пути на тех поддеревьях, которые не переданы для анализа на агенты, то анализ на узле координатора заканчивается. Также, данные о дефектах, найденных на агенте накапливаются на узле, где был запущен агент, что при использований общего сетевого диска не является серьезным ограничением.

Результаты работы инструмента Avalanche в распределенном режиме представлены в табл. 2. Данные результаты были получены при использовании сети из 4 вычислительных узлов со сходными характеристиками, время работы анализа было ограничено 2 часами (7200 секундами). Для сравнения эффективности анализа используются два критерия: количество обнаруженных дефектов и время обнаружения данных дефектов, рассчитываемое с момента старта анализа (для распределённого режима приводится наименьшее время обнаружения соответствующего дефекта среди всех вычислительных узлов).

Табл. 2. Сравнительные результаты работы инструмента Avalanche в обычном и распределённом режиме

	Стандартный режим анализа		Распределённый режим анализа	
	Время обнаружения дефектов, с	Количество дефектов	Время обнаружения дефектов, с	Количество дефектов
qtdump	2872 3110	2	137 6810 7048	3
cjpeg	1114	1	330 2160	2
mpeg2dec	-	0	410 5128	2
mpeg3dump	1 706	2	1 103 280	3

Результаты применения инструмента в распределённом режиме свидетельствуют о достаточной эффективности использования данного режима при наличии соответствующей вычислительной базы. Разделение проверки поддеревьев общего дерева возможных путей выполнения приводит к тому, что за ограниченное время происходит обнаружение большего числа дефектов. Также происходит уменьшение времени, необходимого для обнаружения дефектов, по сравнению с использованием только одного вычислительного узла.

Распределённый анализ, подобно паралелльному анализу, может вносить достаточную степень недетерминизма в процесс выбора наборов входных данных для последующих итераций. При использовании только одного вычислительного узла информация о приросте покрытия ещё не рассмотренных путей выполнения программы применяется на всех итерациях анализа. При использовании нескольких вычислительных узлов информация о приросте покрытия является локальной для каждого вычислительного узла, что может изменить непосредственные значения прироста для различных путей выполнения (по сравнению со значениями, которые бы наблюдались при стандартном режиме анализа). Так, как и при использовании параллельного анализа, недетерминизм привёл к увеличении времени обнаружения дефектов в проекте qtdump, однако позволил обнаружить дополнительную уязвимость.

Стоит отметить, что работа в распределенном режиме может быть улучшена путем применения распределенного алгоритма работы

координатора, который бы позволил эффективно обрабатывать ситуацию, когда на вычислительном узле координатора у анализатора Avalanche заканчиваются данные о достижимых, но не проанализированных путях в анализируемой программе. Это позволит осуществлять более полный анализ целевой программы за ограниченное время. В этом случае одним из вариантов продолжения анализа программы может стать получение достижимых, но ещё не проанализированных путей от агентов и передача входных данных на другого агента, который закончил анализ переданного ему поддерева. Снижение детерминизма, вносимого применением распределённых вычислений, также является одним из перспективных направлений дальнейших исследований.

4. Заключение

В статье рассмотрены способы улучшения производительности динамического анализа программ при помощи параллельного и распределенного анализа на примере решений реализованных в инструменте динамического анализа программ Avalanche. Описаны схемы реализации параллельного и распределенного режимов работы анализатора, ограничения инструмента и технологий, на которых построен инструмент, при применении рассмотренных методов и приведены результаты подтверждающие обоснованность предложенных решений, а также указаны направления дальнейших исследований.

Список литературы.

[1]. И. К. Исаев, Д. В. Сидоров. «Применение динамического анализа при генерации входных данных, демонстрирующих критические ошибки и уязвимости в программах». Программирование, №4, 2010, с. 1-16 .

[2]. Nethercote N., Seward J. “Valgrind: A framework for heavyweight dynamic binary instrumentation”. PLDI, 2007.

[3]. V. Ganesh and D. Dill. “A decision procedure for bit-vectors and arrays”. In CAV 2007, LNCS 4590, pages 519-531

[4]. С.П. Вартанов, Д. В. Сидоров «Оптимизация задачи проверки выполнимости булевских ограничений при помощи кэширования промежуточных результатов». Труды института системного программирования РАН, том 22, 2012, с. 281-292.

Avalanche: adaptation of parallel and distributed computing for dynamic analysis to improve performance of defect detection

M. K. Ermakov, A. Y. Gerasimov
mermakov@ispras.ru, agerasimov@ispras.ru
ISP RAS, Moscow, Russia

Abstract. This paper focuses on dynamic program analysis optimization through the use of distributed computing scheme and parallel computing for checking satisfiability of Boolean constraint sets. The paper is organised as follows: section 1 contains an overview of the Avalanche tool and identifies the key points of its work flow and module structure applicable to distributed and parallel optimizations. Sections 2 and 3 provide general schemes of parallel and distributed program analysis respectively: parallel approach is based on the relative independence of several execution constraint sets; distributed approach is based on a protocol for communication between a number of computing units targeted at sharing sub-trees of execution paths in the analysed program. An overview of results obtained from applying the practical implementation of parallel and distributed schemes of dynamic analysis to a number of open-source applications is given. The paper presents a detailed evaluation of the increased efficiency (in terms of the number of defects detected, as well as relative detection time) of dynamic analysis achieved while applying developed techniques. Section 4 concludes the paper with an overview of possible issues related to non-deterministic nature of parallel and distributed approaches and discusses future directions for research on the Avalanche tool.

Keywords: dynamic analysis, defect detection, parallel computing, distributed computing.

References

[1]. I. K. Isaev, D. V. Sidorov. The use of dynamic analysis for generation of input data that demonstrates critical bugs and vulnerabilities in programs. Programming and Computing Software. Volume 36 Issue 4, July 2010. pp. 225-236. doi:10.1134/S0361768810040055

[2]. Nethercote N., Seward J. Valgrind: A framework for heavyweight dynamic binary instrumentation. Proceedings of ACM SIGPLAN conference on Programming language design and implementation, 2007. pp. 89-100. doi: 10.1145/1250734.1250746

[3]. V. Ganesh and D. Dill. A decision procedure for bit-vectors and arrays. Proceedings of the 19th international conference on Computer aided verification, 2007. pp. 519-531

[4]. S. P. Vartanov, D. V. Sidorov. Optimizatsiya zadachi proverki vypolnimosti bulevskikh ogranichenij pri pomoshhi kehshirovaniya promezhutochnykh rezul'tatov. [Optimization of Boolean satisfiability solver by caching intermediate results]. Trudy ISP RAN [The Proceedings of ISP RAS], 2012, vol. 22, pp. 281-292 (in Russian).