УДК 004.4 DOI: 10.26102/2310-6018/2020.31.4.014

Метод самоадаптации программных систем на основе технологии трассировки вычислительного процесса

А.М. Бершадский, А.С. Бождай, Ю.И. Евсеева, А.А. Гудков
Пензенский государственный университет,
Пенза, Российская Федерация

Резюме: В статье рассмотрены вопросы разработки метода самоадаптации программного обеспечения на основе технологии трассировки вычислительного процесса. Обосновывается актуальность проблемы создания методов синтеза самоадаптивного программного обеспечения, рассмотрены основные преимущества самоадаптивных программных систем., Приводится описание существующих средств трассировки, обоснован выбор Intel Processor Trace для создания метода самоадаптации программного обеспечения. Рассмотрено определение графа выполнения программы в качестве математического аппарата, лежащего в основе нового метода. Предложена математическая модель поведения самоадаптивной программы, основанная на рассмотренном определении графа вызовов и представляющая собой формализацию полученных с помощью Intel Processor Trace трасс. Рассмотрен алгоритм поиска паттернов в графах выполнения. На основе рассмотренного определения графа выполнения и алгоритма предложен новый метод самоадаптации программной системы, основанный на анализе хода выполнения программы: определяются и в дальнейшем оптимизируются наиболее часто выполняемый участки исходного кода программы (поведенческие паттерны системы). Полученный метод позволит оптимизировать производительность программы, сократив число вычисляемых в процессе выполнения условий.

Ключевые слова: самоадаптивные программные системы, теория графов, трассировка вычислительного процесса, поиск часто встречающихся подграфов, граф выполнения, оптимизация производительности программного обеспечения

Для цитирования: Бершадский А.М., Бождай А.С., Евсеева Ю.И., Гудков А.А. Метод самоадаптации программных систем на основе технологии трассировки вычислительного процесса. *Моделирование, оптимизация и информационные технологии.* 2020;8(4). Доступно по: https://moitvivt.ru/ru/journal/pdf?id=860 DOI: 10.26102/2310-6018/2020.31.4.014

Method of software systems self-adaptation based on the technology of the computing process trace

A.M. Bershadsky, A.S. Bozhday, Y.I. Evseeva, A.A. Gudkov
Penza State University, Penza, Russian Federation

Abstract: The article deals with the development of a software self-adaptation method based on the technology of tracing the computational process. The urgency of the problem of creating methods for the synthesis of self-adaptive software is substantiated, the main advantages of self-adaptive software systems are considered., A description of the existing tracing tools is given, the choice of Intel Processor Trace for creating a method of self-adaptive software is justified. The definition of the program execution graph as the mathematical apparatus underlying the new method is considered. A mathematical model of the behavior of a self-adaptive program is proposed, based on the considered definition of the call graph and representing the formalization of the traces obtained using Intel Processor Trace. An algorithm for searching patterns in execution graphs is considered. On the basis of the considered definition of the execution graph and the algorithm, a new method of self-adaptation of a software system is proposed, based on the analysis of the program execution progress: the most

frequently executed sections of the program source code (behavioral patterns of the system) are determined and further optimized. The resulting method will optimize the performance of the program by reducing the number of conditions calculated during the execution of conditions.

***Keywords**:* self-adaptive software systems, graph theory, computational process tracing, search for common subgraphs, execution graph, software performance optimization

***For citation**:* Bershadskij A.M., Bozhdaj A.S., Evseeva Y.I., Gudkov A.A. Metod samoadaptacii programmnyh sistem na osnove tekhnologii trassirovki vychislitel'nogo processa. *Modeling, optimization and information technology.* 2020;8(4). Available from: https://moitvivt.ru/ru/journal/pdf?id=860 DOI: 10.26102/2310-6018/2020.31.4.014 (In Russ).

Введение

Высокая степень информатизации современного общества предъявляет повышенные требования к производительности и надежности обслуживающих его программных систем и программно-аппаратных комплексов. Для программных приложений, критичных к времени выполнения, крайне полезной может быть способность самостоятельно, без вмешательства разработчиков, оптимизировать собственную производительность. Достичь этого можно путем наделения программного обеспечения (ПО) способностью к самоадаптивному поведению.

Трассировка вычислительного процесса как перспективный способ построения самоадаптивного ПО впервые рассмотрена авторами в работе [1]. Использование трассировки программного кода в методах самоадаптации программных систем осуществляется с целью отслеживания потока управления программы. На основе зафиксированного потока управления можно сформировать паттерн поведения системы. Анализ и последующее преобразование такого паттерна позволит в дальнейшем оптимизировать поведение системы, делая ее работу более эффективной.

Целью данной работы является описание и формализация принципа трассировки вычислительного процесса для создания самоадаптивных программных систем и комплексов широкого прикладного назначения.

Материалы и методы

Трассировка кода используется многими видами программного обеспечения: эмуляторами, динамическими распаковщиками, специализированными средствами для тестирования находящихся в стадии разработки программных продуктов. Традиционные средства трассировки работают по одному из четырех принципов: эмуляция набора инструкций, бинарная трансляция, модификация бинарных файлов для изменения потока управления, работа через отладчик [2].

Эмуляцию набора инструкций используют для отслеживания выполнения программы путем ее запуска вне операционной системы. Подобный подход не изменяет ход выполнения программы, но при этом запуск исследуемой системы на эмуляторе безопаснее обычной отладки. Примером реализации такого подхода является программное решение Bochs, предназначенное для эмуляции процессоров архитектуры x86. Благодаря данному ПО, впервые был осуществлен запуск операционной системы Windows 98 на сотовом смартфоне с операционной системой Android [3].

Под бинарной трансляцией понимается эмуляция одного набора инструкций на другом путем трансляции машинного кода. Примером реализации такого подхода является программное приложение QEMU — средство для эмуляции аппаратного обеспечения различных платформ [4].

Идея модификации бинарных файлов для изменения потока управления реализована в системе Pin — платформе для создания инструментов динамического анализа программ для архитектур x86-64, IA-32, MIC. Pin предоставляет собственный набор функций, предназначенных для фиксирования трасс выполнения программы [5]. Другим примером программно-инструментальной платформы для фиксации трасс на этапе отладки программного обеспечения является Paimai [6].

Большая часть задач, связанных с трассировкой вычислительного процесса, решается с использование перечисленных принципов. Однако компанией Intel на протяжении нескольких лет разрабатывались специализированные, интегрированные в процессор, средства, позволяющие отслеживать поток выполнения запускаемых программ на аппаратном уровне. Предыдущие решения компании были малоэффективны в силу низкой производительности, однако последняя технология — Intel Processor Trace — вышла на качественно новый уровень и позволяет получать трассы выполнения программ с меньшими ресурсозатратами [7].

Основной идеей технологии является протоколирования вызова не каждой программной инструкции, а только условных переходов. Таким образом, объем трассы понижается до минимально необходимого. Для восстановления логики выполнения программы осуществляется объединение записанной трассы с двоичным кодом программы, для чего используются специализированные инструменты Intel PT.

В Таблице 1 приведен пример такой трассировки [7]. Первый столбец содержит исходный ассемблерный код программы (операнды некоторых инструкций опущены), второй — записанную трассу, последний — восстановленный с использованием трассы поток выполнения инструкций. Регистр edx в инструкции «call edx» содержит адрес виртуальной функции; ее фактический адрес можно определить только во время выполнения программы.

Таблица 1 - Трассировка потока выполнения программы с помощью Intel PT
Table 1 - Tracing the program flow using Intel PT

Инструкции	**Трасса Intel PT**	**Поток выполнения**
push		push
mov		mov
add		add
cmp		cmp
je .label	T	je .label
mov		
.label:		.label:
call edx	0x407e1d8	call 0x407e1d8

Intel PT позволяет отслеживать не только поток выполнения команд, но и время их выполнения. В Таблице 2 показан пример такой трассировки [7].

Таблица 2 - Учет времени выполнения инструкций с помощью Intel PT
Table 2 - Tracking the execution time of instructions using Intel PT

Инструкции	**Трасса Intel PT**	**Результат декодирования**	
		Время (нс)	Инструкция
mov		0	mov
jnz	NT	0	jnz
add		2	add
cmp		2	cmp
je .label	TIME 2ns + T	2	je .label
mov			
.label:			.label:
call edx	0x407e1d8	102	call 0x407e1d8
test		102	test
jb	TIME 100ns + NT	102	jb

Преимуществами такого подхода является сравнительно малый объем хранимых трасс и, как следствие, относительно более высокая производительность работы трассировки. Поддержка механизма сбора трасс на аппаратном уровне позволяет не использовать сторонние инструменты трассировки.

Следующим этапом при создании специализированного метода самоадаптации является выбор математического аппарата для анализа записанных трасс. Для выявления устойчивых тенденций в поведении программы необходимо отслеживать, какие последовательности ее инструкций выполняются наиболее часто. В качестве математического аппарата предлагается использовать аппарат теории графов, а задачу поиска наиболее часто встречающихся трасс свести к задаче поиска часто встречающихся подграфов в наборе графов, построенных на базе трасс вычислительного процесса.

Под графом будем понимать пару $G = (V, E)$, где V — множество вершин, $E \subseteq V \times V$ — множество ребер. Графом выполнения будем называть ориентированный граф G с множеством вершин $V = V(G)$, каждый элемент которого абстрагирует линейный блок программных инструкций, и множеством дуг $E(G) = V(G) \times V(G)$, каждый элемент в котором представляет собой переходы между блоками инструкций. Пример графа выполнения показан на Рисунке 1.

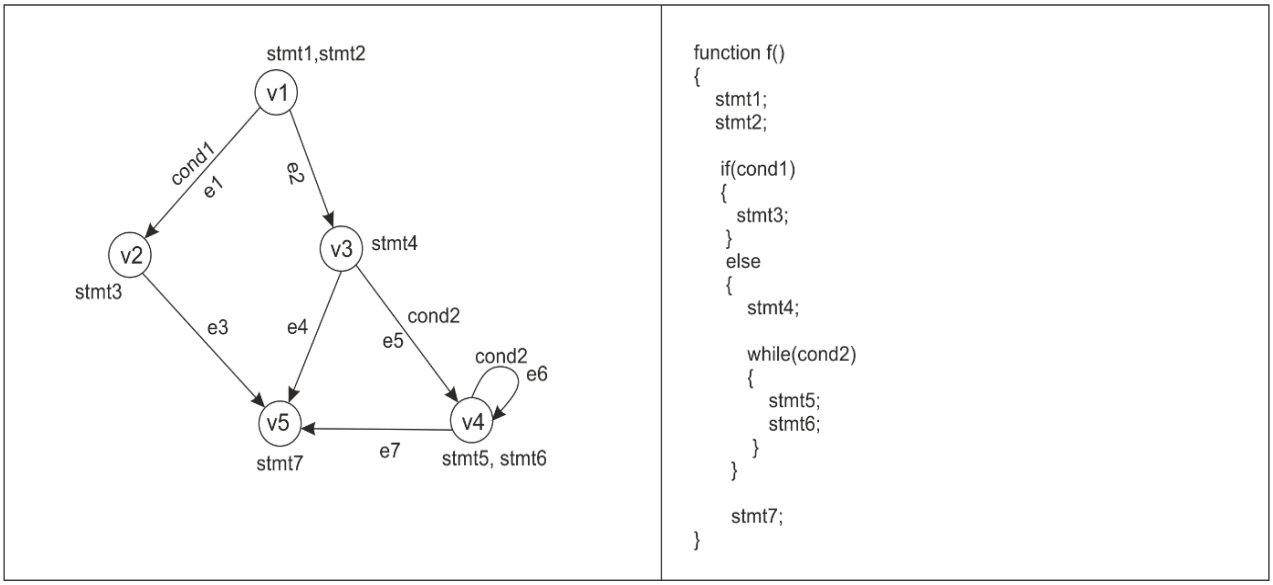

Рисунок 1 — Граф выполнения функции *f*
Figure 1 — Graph of execution of function f

В свою очередь, математическая модель поведения системы будет описываться кортежем

$$M = (F, D), \quad (1)$$

где $F = (X, A)$ — базовый (статический) граф выполнения, абстрагирующий все возможные блоки инструкций программы и все возможные переходы между ними, при этом каждой вершине из множества $X_i = \{x_1, x_2, .., x_m\}$ присвоена метка, определяемая последовательностью инструкций, связанных с данной вершиной: $x_i = (c, b)$, где c — номер вершины, b — связанный с ней блок последовательности линейных инструкций; $D = \{G_1, G_2, ..., G_i, ..., G_n\}$, где $G_i = (V_i, E_i)$ — граф выполнения программы на i-м запуске, при этом $G_i \subseteq F$, $V_i \subseteq X$, $E_i \subseteq A$.

Существующие методы решения задачи о поиске частых подграфов структурно схожи и обычно включают в себя итерации из следующих этапов:

1. Этап формирования кандидатов, на котором осуществляется составление списка подграфов для рассмотрения на следующих этапах.

2. Этап отсечения неподходящих кандидатов.

3. Этап подсчета, который реализует процесс подсчета, сколько раз каждый из конечных кандидатов встречается среди множества графов, поступивших на вход алгоритма.

В качестве примера можно привести алгоритмы на базе Apriori-алгоритма, общая структура которых представлена следующим псевдокодом [8]:

Apriori(D, min_sup, S_{k+1})

На входе: множество графов *D*, пороговое значение поддержки *min_sup*, частые подграфы S_k размером *k*.

На выходе: множество частых подграфов S_{k+1} размерностью *k*+1.

1: $S_{k+1} \leftarrow \emptyset$;

2: для каждого частого подграфа $g_i \in S_k$ выполнять:

3: для каждого частого подграфа $g_j \in S_k$ выполнять:

4: для каждого частого графа *g* размера (*k*+1), сформированного присоединением g_i и g_j выполнять:

5: если *g* является частым в *D* и $g_j \in S_{k+1}$ то

6: добавить *g* к S_{k+1};

7: если $S_{k+1} \neq \emptyset$ то

8: вызвать Apriori(D, min_sup, S_{k+1});

9: выход;

Процесс поиска частых подграфов начинается с подграфов малого размера. На каждой итерации размер найденных частых подграфов увеличивается на единицу. Новые частые подграфы формируются путем объединения схожих найденных ранее подграфов. Далее проверяется частота новых сформированных подграфов.

Помимо алгоритмов, основанных на Apriori-алгоритме, существуют и другие группы алгоритмов поиска частых подграфов. Большинство из них относятся к алгоритмам, основанным на наращивании наборов. Алгоритмы такого типа начинают обход с одной вершины и расширяются за счет добавления на каждой итерации одного ребра. После каждого такого расширения проверяется частота нового графа. Алгоритмы, основанные на наращивании наборов, обладают, по сравнению с Apriori-алгоритмами, более высокой производительностью [8].

Метод самоадаптации программной системы, основанный на технологии трассировки вычислительного процесса, включает в себя следующие этапы:

1. Трассировка выполнения программы на множестве запусков.
2. Сбор полученных трасс и их графовая формализация, построение математической модели поведения системы в соответствии с формулой (1).
3. Обработка множества формализованных трасс *D* таким образом, чтобы номера вершин в них соответствовали номерам вершин из графа *F*.
4. Поиск часто встречающихся подграфов в множестве графов, построенных на основе трасс и являющихся частью математической модели.
5. Нахождение в частых подграфах участков, соответствующих операторам ветвления, которые при всех запусках программы сработали одинаковым образом, то есть всегда были истинны или всегда были ложны.
6. Оптимизация исходного кода программы: удаление инструкций, связанных с вычислением найденных на предыдущем этапе условий, и замена условных переходов на безусловные. Пример оптимизации иллюстрирует псевдокод, представленный в Таблице 3.

Таблица 3 - Оптимизация кода на основе анализа трасс
Table 3 - Code optimization based on trace analysis

Код до модификации	Код после модификации

Вычисление 1
Вычисление условия
if (условие истинно)
{
Вычисление 2
}
else
{
Вычисление 3
}
	Вычисление 1
	if (1*)
	{
	Вычисление 2
	}
	else
	{
	Вычисление 3
	}
	* или 0, в зависимости от того, было условие истинным или ложным при запусках программы

Следует отметить, что программа может перестать корректно работать после внесения в нее изменений по данному методу. Поэтому необходимо также предусмотреть возможность возврата программы к исходному состоянию и возможность предварительного запроса у пользователя необходимости модификации.

Применение на практике указанного метода самоадаптации позволит повысить производительность программной системы за счет сокращения количества выполняемых процессором инструкций, связанных с вычислением условий.

Заключение

В работе предложен метод самоадаптации программной системы, основанный на технологии трассировки вычислительного процесса. С помощью собранных трасс определяются часто исполняемые участки программы, в которых находятся фрагменты кода, расположенные внутри условных операторов. Далее проверяется, исполнялись ли данные фрагменты во всех запротоколированных вызовах и, если исполнялись, то код модифицируется таким образом, что инструкции условного перехода заменяются инструкциями безусловного перехода. Данный метод самоадаптации позволяет оптимизировать производительность программы, сократив количество вычисляемых в процессе исполнения условий.

БЛАГОДАРНОСТИ

Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 18-07-00408.

The study was carried out with the financial support of the Russian Foundation for Basic Research within the framework of scientific project No. 18-07-00408.

ЛИТЕРАТУРА

1. Бершадский А.М., Бождай А.С., Евсеева Ю.И. Гудков А.А. Исследование и разработка методов динамического анализа кода для создания самоадаптивного программного обеспечения . *Моделирование, оптимизация и информационные технологии*. 2018;6(4):108-120.
2. Альтернативные методы трассировки приложений. Безопасность, разработка,

DevOps URL: https://xakep.ru/2014/08/05/app-trace-methods/ (дата обращения: 20.10.20).
3. Bochs User Manual. Bochs: The Open Source IA-32 Emulation Project (Home Page) URL: http://bochs.sourceforge.net/ (дата обращения: 20.10.20).
4. What is QEMU? QEMU URL: https://www.qemu.org/ (дата обращения: 20.10.20).
5. Pin — A Dynamic Binary Instrumentation Tool. Intel URL: https://software.intel.com/content/www/ru/ru/develop/articles/pin-a-dynamic-binary-instrumentation-tool.html (дата обращения: 20.10.20).
6. Assessment war: Windows services. Virus bulletin URL: https://www.virusbulletin.com/virusbulletin/2008/02/assessment-war-windows-services/ (дата обращения: 20.10.20).
7. Enhance performance analysis with Intel Processor Trace . Performance explained easy URL: https://easyperf.net/blog/2019/08/23/Intel-Processor-Trace (дата обращения: 20.10.20).
8. Пыжов В.О., Куликов Г.С., Панов А.В. Задача поиска частых подграфов и алгоритмы ее решения . *Актуальные вопросы современной науки*. 2016;1(48):74-83.

REFERENCES

1. Bershadskij A.M., Bozhdaj A.S., Evseeva Y.I. Gudkov A.A. Issledovanie i razrabotka metodov dinamicheskogo analiza koda dlya sozdaniya samoadaptivnogo programmnogo obespecheniya . *Modelirovanie, optimizaciya i informacionnye tekhnologii*. 2018; 6(4): 108-120.
2. Al'ternativnye metody trassirovki prilozhenij . Bezopasnost', razrabotka, DevOps URL: https://xakep.ru/2014/08/05/app-trace-methods/ (data obrashcheniya: 20.10.20).
3. Bochs User Manual . bochs: The Open Source IA-32 Emulation Project (Home Page) URL: http://bochs.sourceforge.net/ (data obrashcheniya: 20.10.20).
4. What is QEMU? QEMU URL: https://www.qemu.org/ (data obrashcheniya: 20.10.20).
5. Pin — A Dynamic Binary Instrumentation Tool . Intel URL: https://software.intel.com/content/www/ru/ru/develop/articles/pin-a-dynamic-binary-instrumentation-tool.html (data obrashcheniya: 20.10.20).
6. Assessment war: Windows services . Virus bulletin URL: https://www.virusbulletin.com/virusbulletin/2008/02/assessment-war-windows-services/ (data obrashcheniya: 20.10.20).
7. Enhance performance analysis with Intel Processor Trace . Performance explained easy URL: https://easyperf.net/blog/2019/08/23/Intel-Processor-Trace (data obrashcheniya: 20.10.20).
8. Pyzhov V.O., Kulikov G.S., Panov A.V. Zadacha poiska chastyh podgrafov i algoritmy ee resheniya . *Aktual'nye voprosy sovremennoj nauki*. 2016. 1(48): 74-83.

ИНФОРМАЦИЯ ОБ АВТОРАХ / INFORMATION ABOUT THE AUTHORS

Бершадский Александр Моисеевич, заведующий кафедрой САПР ПГУ, доктор технических наук, профессор, Пензенский государственный университет, Пенза, Российская Федерация
e-mail: bam@pnzgu.ru

Bershadskij Aleksandr Moiseevich, Head Of The CAD Department, Dr. Sci. (Tech.), Professor, Penza State University, Penza, Russian Federation

Бождай Александр Сергеевич, профессор кафедры САПР ПГУ, доктор технических наук доцент, Пензенский государственный университет, Пенза, Российская Федерация
e-mail: bozhday@yandex.ru

Bozhdaj Aleksandr Sergeevich, Professor Of The CAD Department, Dr. Sci. (Tech.), Associate Professor, Penza State University, Penza, Russian Federation.

Гудков Алексей Анатольевич, доцент кафедры САПР ПГУ, кандидат технических наук, Пензенский государственный университет, Пенза, Российская Федерация
e-mail: alexei-ag@yandex.ru

Gudkov Aleksej Anatol'evich, Associate Professor Of The CAD Department, Cand. Sci. (Tech.), Penza State University, Penza, Russian Federation.

Евсеева Юлия Игоревна, доцент кафедры САПР ПГУ, кандидат технических наук,
Пензенский государственный университет, Пенза, Российская Федерация
e-mail: shymoda@mail.ru

Evseeva Yuliya Igorevna, Associate Professor Of The CAD Department, Cand. Sci. (Tech.), Penza State University, Penza, Russian Federation.