УДК 621.452.001.57:681.54

Ю. В. Шабатура, к. т. н., доцент; И. Н. Штельмах; М. Ю. Шабатура

НОВАЯ МЕТОДИКА ПОВЫШЕНИЯ ЭФФЕКТИВНОСТИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ ПУТЕМ АВТОМАТИЧЕСКОГО ОПРЕДЕЛЕНИЯ «УЗКИХ МЕСТ»

Рассмотрена актуальная проблема повышения эффективности программного обеспечения вычислительных систем интегрированных в компьютерные сети путем автоматического определения «узких мест». Формализирована задача повышения временной эффективности программного обеспечения в соответствии со стандартом ISO9126, построена модель процесса его работы на основе графов, разработана схема системы автоматизированного повышения эффективности программного обеспечения вычислительных систем на основе уменьшения влияния «узких мест» и раскрыты принципы её работы.

Ключевые слова: *Эффективность, программное обеспечение, узкие места, профайлинг.*

Введение

Вычислительные системы уже давно проникли практически во все сферы человеческой деятельности. Особое распространение получили вычислительные системы, интегрированные в глобальные компьютерные сети, которые характеризуются большим количеством пользователей и значительной нагрузкой. Неотъемлемым условием успешной эксплуатации таких систем является их высокое качество и эффективность.

Вычислительная система (ВС) – система обработки данных, настроенная на решение задач конкретной области применения. ВС включают в себя технические средства и программное обеспечение (ПО). Стандарт ISO 8126 [1] определяет оценочные характеристики качества программного обеспечения ВС. Структура модели качества ПО ВС показана на рис. 1.

Рис. 1. Модель качества ПО соответственно со стандартом ISO9126

Одной из ключевых характеристик ПО является его эффективность. Согласно [1], эффективность определяется как набор атрибутов, которые определяются как отношение

качества функционирования ПО к объему используемых ресурсов.

Показатель временной эффективности демонстрирует способность ПО выполнять указанные действия в интервал времени соответствующий заданным условиям. Ресурсоемкость свидетельствует о минимально необходимых вычислительных ресурсах и количестве обслуживающего персонала, необходимых для эксплуатации ПО.

Постановка задачи

Временную эффективность можно определить как:

$$T_{ef} = \frac{F}{T_{run}}, \tag{1}$$

где F – необходимый объем функциональности, T_{run} – время выполнения программы.

Таким образом, задачу повышения эффективности программного обеспечения можно определить как:

$$t'_{ef} = \max\left(\frac{F}{T_{run}}\right). \tag{2}$$

Следует отметить, что функциональность программного обеспечения, как правило, оговаривается в техническом задании и является четко определённой для конкретной вычислительной системы, поэтому её можно принять постоянной: $F = F_{const}$.

Тогда задача повышения эффективности программного обеспечения сводится к минимизации времени выполнения программы:

$$t'_{ef} = \max_{T_{run}\to\min}\left(\frac{F_{const}}{T_{run}}\right). \tag{3}$$

Модель процесса работы программного обеспечения

Согласно [2], процесс выполнения программы можно представить в виде графа:

$$G = (V, E), \tag{4}$$

где вызовы подпрограмм или выполнение операций создают множество узлов V, а процессы вызова таких операций или подпрограмм являются его ребрами E. Пример такого графа показано на рис. 2.

Введем для каждого ребра вес t_i, который характеризует длительность выполнения операции или подпрограммы. Таким образом, общее время выполнения программы на основе данной модели можно определить как:

$$T_{run} = \sum_{i=0}^{n} t_i, \tag{5}$$

а длительность выполнения всех узлов графа представляет собой множество

$$T = \{t_1, t_2, ..., t_i\}, \tag{6}$$

где n – количество вызовов операций и подпрограмм в процессе выполнения программы в целом. Таким образом, повышение общей эффективности ПО сводится к уменьшению длительности выполнения его отдельных элементов.

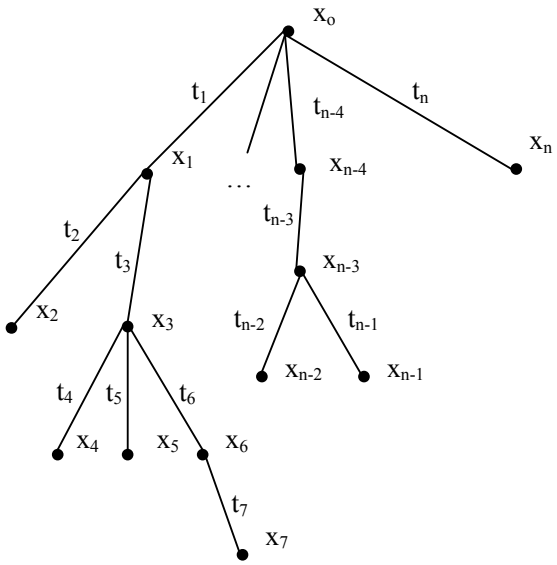

Рис. 2. Представление процесса выполнения программного кода в виде графа вызовов

В инженерии известен феномен под названием «узкое место» (bottleneck), который согласно [3] заключается в ограничении ёмкости или продуктивности системы, вызванной её отдельными элементами. В программировании поиск «узких мест» называется анализом производительности. Процедура поиска таких элементов кода называется профайлингом.

Таким образом, определив множество элементов ПО

$$B = \{x_{k1}, x_{k2}, ..., x_{kb}\}, \tag{7}$$

которые составляют «узкие места», и оптимизировав их, получим уменьшение времени выполнения программы:

$$T_{run} = T_{run} - T'_{run}, \tag{8}$$

где T'_{run} – оптимизированное время выполнения программы за счет оптимизации «узких мест».

На основе (1) можно выразить коэффициент увеличения эффективности ПО в результате оптимизации:

$$k_{ef} = \frac{t'_{ef}}{t_{ef}} = \frac{\overline{\frac{F}{T'_{run}}}}{\overline{\frac{F}{T_{run}}}} = \frac{T_{run}}{T'_{run}}. \tag{9}$$

Таким образом, задача повышения эффективности ПО сводится к двум подзадачам: определению «узких мест» в программном коде и их оптимизации. Схему системы автоматического повышения эффективности ПО вычислительных систем на основе такого подхода показано на рис. 3.

В обычном режиме работы вычислительной системы пользователи осуществляют запросы и получают результат Y. В систему вводится дополнительный блок оптимизации, который позволяет администратору переключать систему в режим оптимизации.

В режиме оптимизации осуществляются тестовые запросы X', на основе которых получается результат Y'. Параллельно, с помощью специальной серверной библиотеки Xdebug [4], поступают данные о последовательности и времени выполнения программы. Такая информация поступает в виде массива данных F, каждый элемент которого содержит информацию о длительности выполнения подпрограммы, её название и место расположения

в файле. Указанный массив легко трансформируется в граф вызовов G, на основе которого осуществляется поиск множества подпрограмм-«узких мест» (7) и их оптимизация.

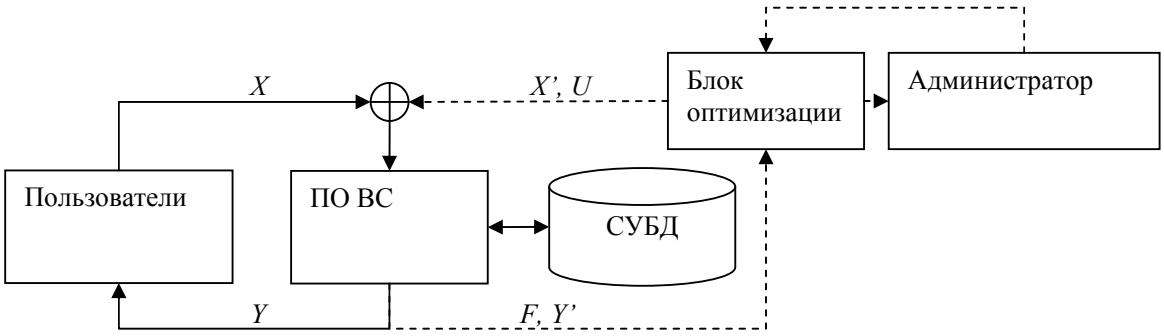

Рис. 3. Схема системы автоматизированного повышения эффективности ПО вычислительных систем на основе уменьшения влияния «узких мест»

Задача автоматического определения «узких мест»

Используя определения «узких мест» в ПО, показанное в [3], термином «узкие места» будем обозначать множество вершин B графа вызовов G для которых длительность выполнения каждой оказывает значительное влияние на суммарную длительность выполнения T_{run}. Степень такого влияния для каждой вершины можно определить как:

$$b_i = \frac{t_i}{T_{run}}. \tag{10}$$

При этом следует учесть, если вершина содержит вызовы других вершин-подпрограмм, то длительность её вызова будет включать длительности всех последующих вызовов. Поэтому если выполняется условие

$$b_i \approx \sum_{j=1}^{d} \frac{c_j}{T_{run}}, \tag{11}$$

где $\{c_1, c_2, ..., c_j\}$ – множество вершин-подпрограмм, вызов которых осуществляется из данной вершины, то такую вершину нельзя считать «узким местом», поскольку значительная продолжительность выполнения данной подпрограммы обусловлена значительным количеством вызовов дочерних подпрограмм. Поэтому такие вершины исключаются из множества «узких мест» B.

Кроме того, в графе G могут существовать вершины, описывающие подпрограммы программных библиотек, которые используются этим ПО. Если данные библиотеки являются стандартизированными и характеризируются высокими качественными показателями, их вызовы нельзя рассматривать как узкие места, скорее они неправильно вызываются разработанным ПО. Именно поэтому вводится множество

$$L = \{l_1, l_2, ..., l_{1b}\}, \tag{12}$$

которое описывает файлы таких библиотек. Подпрограммы, найденные в таких файлах, будут исключаться из процесса поиска.

Скомпоновав алгоритм, описанный в [5], поиска в глубину и ограничения (11) и (12), получим в результате алгоритм поиска множества B всех вершин-подпрограмм системы, для которых возможной является оптимизация. Каждый элемент данного множества характеризируется степенью влияния (10) на общую длительность выполнения программы, а соответственно, и на эффективность (1).

Выбрав подмножество вершин-подпрограмм, имеющих наибольшую степень влияния на

эффективность, получим необходимое множество «узких мест» системы B', над элементами которой выполняется операция оптимизации.

Задача оптимизации «узких мест»

Задача оптимизации «узких мест» в эффективности вычислительной системы заключается в минимизации длительности выполнения подпрограмм-вершин B. Множество «узких мест» B можно разбить на несколько подмножеств, в зависимости от природы возникновения задержек

$$\{B\} = \{\{B_1\},\{B_2\},...,\{B_{bc}\}\}, \quad (13)$$

где bc – количество классов возникновения задержек.

На данном этапе исследований нами выделено 2 основных класса причин возникновения задержек: задержки при выполнении SQL-запросов к БД и задержки в результате использования несовершенных алгоритмов. Для каждого класса задержек используются свои подходы к оптимизации. По первому классу задержек разработана схема генетической оптимизации SQL-запросов, однако необходима последующая работа по расширению количества классов задержек и подходов к их оптимизации.

Практическая реализация и результаты экспериментальных исследований

На основе результатов теоретических исследований, показанных в работе, реализована система поиска временных задержек в вычислительных системах массового пользования с архитектурой PHP/MySQL. Система строит граф вызовов подпрограмм в ПО в виде раскрывающегося иерархического списка (accordion) (рис. 4).

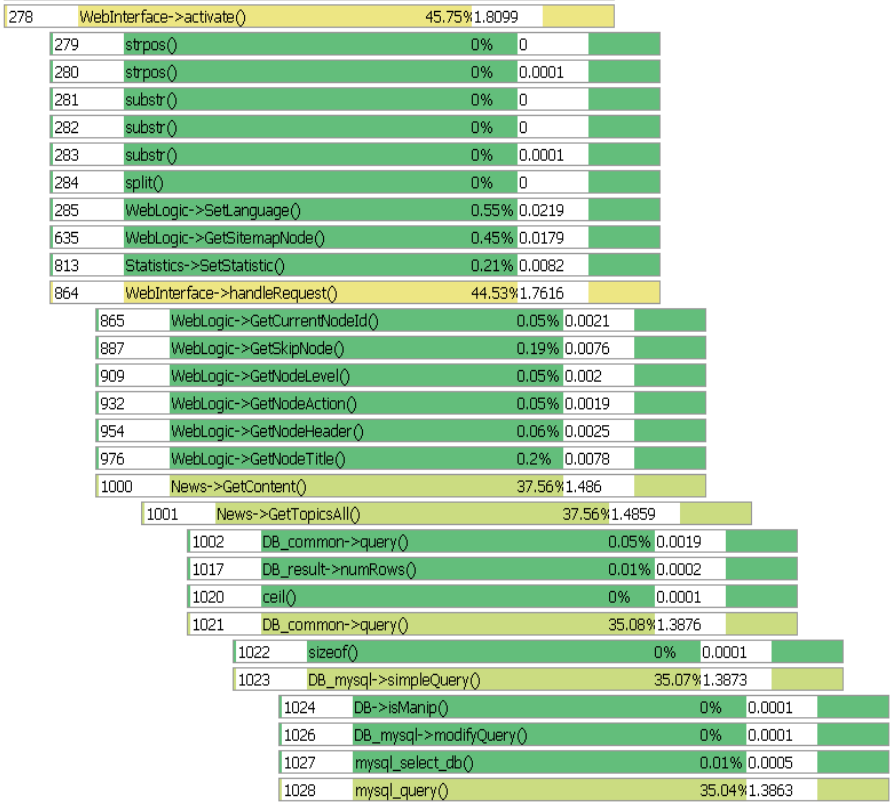

Рис. 4. Визуализация процесса выполнения программного кода

Цвет каждого узла списка устанавливается в зависимости от его влияния (10) на длительность выполнения программы. В результате экспериментальных исследований

данная система показала способность наглядно представлять процесс выполнения ПО вычислительных систем и оптимизировать их работу. В дальнейшем данная система будет расширена функцией автоматического поиска «узких мест» и их оптимизации.

Выводы

1. Разработаны математические модели для реализации автоматического поиска «узких мест» в программном обеспечении вычислительных систем массового использования.
2. Разработана структура, а также алгоритмическое обеспечение для построения системы автоматизированного повышения эффективности ПО вычислительных систем на основе уменьшения влияния «узких мест».
3. Проведены экспериментальные исследования, которые показали перспективность разработанной методики и возможность практического внедрения для уменьшения программных временных задержек различного происхождения.

СПИСОК ЛИТЕРАТУРЫ

1. Scalet R. ISO/IEC 9126 and 14598 integration aspects: A Brazilian viewpoint / Scalet R. // The Second World Congress on Software Quality. – Yokohama: 2000 – 350 p.
2. Новиков Ф.А. Дискретная математика для программистов: Учебник для вузов / Новиков Ф.А. – Спб.: Питер, 2004 – 256 с.
3. Bottleneck (engineering). Wikipedia, the free encyclopedia. Режим доступу: http://en.wikipedia.org/wiki/Bottleneck_(engineering).
4. Derick Rethans. Documentation for: Xdebug 2. Function Traces. Режим доступу: http://www.xdebug.org/docs/execution_trace.
5. Ананий В.О., Левитин Г.В. Алгоритмы: Введение в разрабо тку и анализ / Ананий В.О., Левитин Г.В. – М.: Вильямс, 2006. – с. 212 – 215.

Шабатура Юрий Васильевич – к. т. н., доцент, профессор кафедры МПА. Тел.: (0432) 59-85-71, e-mail: shabatura@vstu.vinnica.ua.

Штельмах Игорь Николаевич – аспирант кафедры МПА. Тел.: (0432) 50-58-55, e-mail: ceo@sbsgroup.com.ua.

Шабатура Максим Юриевич – студент. e-mail: smartmax@i.ua.

Винницкий национальный технический университет.