УДК 621.452.001.57:681.54

Ю. В. Шабатура, к. т. н., доц.; І. М. Штельмах; М. Ю. Шабатура

НОВА МЕТОДИКА ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ОБЧИСЛЮВАЛЬНИХ СИСТЕМ ШЛЯХОМ АВТОМАТИЧНОГО ВИЗНАЧЕННЯ «ВУЗЬКИХ МІСЦЬ»

Розглянуто актуальну проблему підвищення ефективності програмного забезпечення обчислювальних систем інтегрованих у комп'ютерні мережі шляхом автоматичного визначення «вузьких місць». Формалізовано задачу підвищення часової ефективності програмного забезпечення відповідно до стандарту ISO9126, побудовано модель процесу його роботи на основі графів, розроблено схему системи автоматизованого підвищення ефективності програмного забезпечення обчислювальних систем на основі зменшення впливу «вузьких місць» і розкрито принципи її роботи.

Ключові слова: *Ефективність, програмне забезпечення, вузькі місця, профайлинг.*

Вступ

Обчислювальні системи вже давно заполонили практично всі сфери людської діяльності. Особливого поширення набули обчислювальні системи, інтегровані в глобальні комп'ютерні мережі, які характеризуються великою кількістю користувачів і значним навантаженням. Неодмінною умовою успішної експлуатації таких систем є їхня висока якість та ефективність.

Обчислювальна система (ОС) – це система обробки даних, налаштована на вирішення задач конкретної галузі застосування. ОС складаються з технічних засобів і програмного забезпечення (ПЗ). Стандарт ISO 8126 [1] визначає оціночні характеристики якості програмного забезпечення ОС. Структуру моделі якості ПЗ обчислювальних систем наведено на рис. 1

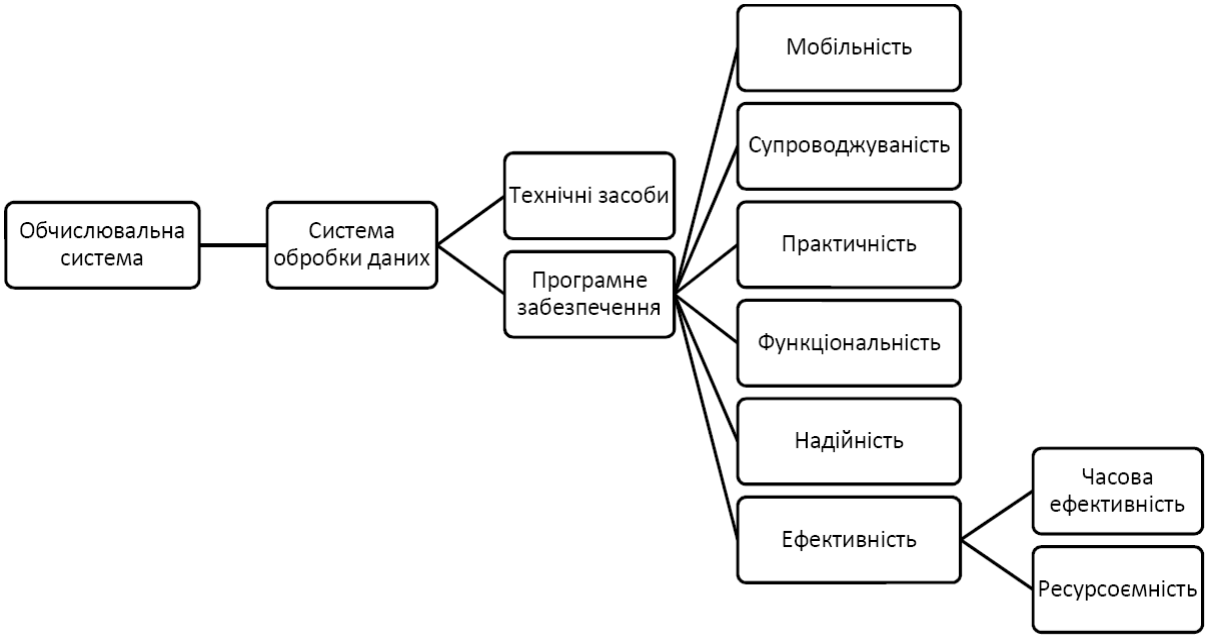

Рис. 1. Модель якості програмного забезпечення згідно зі стандартом ISO9126

Однією з ключових якісних характеристик ПЗ є його ефективність. Згідно з [1]

ефективність визначається як набір атрибутів, які відносяться до співвідношення якості функціонування ПЗ і об'єму використовуваних ресурсів.

Показник часової ефективності демонструє здатність ПЗ виконувати задані дії в інтервал часу, який відповідає заданим вимогам. Ресурсоємність показує мінімально необхідні обчислювальні ресурси та число обслуговуючого персоналу для експлуатації ПЗ.

Постановка завдання

Часову ефективність можна визначити як:

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

де F – необхідний об'єм функціональності, T_{run} – час виконання програми.

Отже, задачу підвищення ефективності програмного забезпечення можна виразити як:

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

Слід зазначити, що функціональність програмного забезпечення, зазвичай, обговорюється в технічному завданні і є чітко визначеною для конкретної обчислювальної системи, тому її можна прийняти сталою: $F = F_{const}$.

Тоді задача підвищення ефективності програмного забезпечення зводиться до мінімізації часу виконання програми:

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

Модель процесу роботи програмного забезпечення

Згідно з [2], процес виконання програми можна представити у вигляді графа:

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

де виклики підпрограм або виконання операцій утворюють множину вузлів V, а процеси виклику даних операцій або підпрограм є його ребрами E. Приклад такого графа показано на рис. 2.

Введемо для кожного ребра вагу t_i, яка характеризує тривалість виконання операції або підпрограми. Отже, загальний час виконання програми на основі описаної моделі можна виразити, як:

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

а тривалість виконання всіх вузлів графу викликів представляє собою множину

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

де n – кількість викликів операцій та підпрограм у процесі виконання програми в цілому. Отже, підвищення загальної ефективності ПЗ зводиться до зменшення тривалості виконання його окремих елементів.

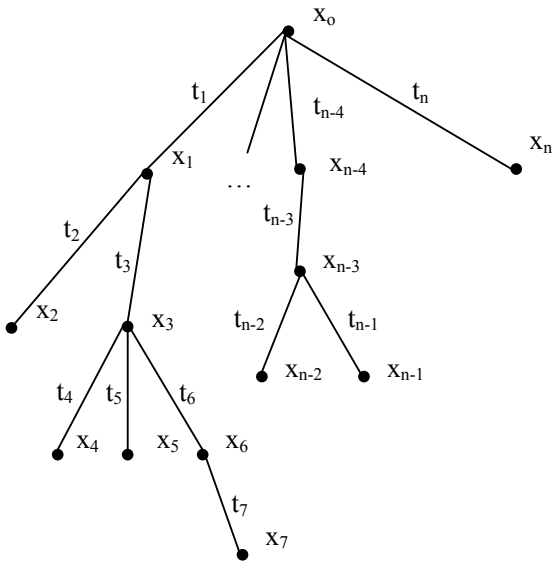

Рис. 2. Представлення процесу виконання програмного коду у вигляді графа викликів

В інженерії відомий феномен під назвою «вузьке місце» (bottleneck), який згідно з [3] полягає в обмеженні ємності або продуктивності системи, викликаної її окремими елементами. У програмуванні пошук «вузьких місць» називається аналізом продуктивності. Процедура пошуку таких елементів коду називається профайлингом.

Отже, виявивши множину елементів ПЗ

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

які утворюють «вузькі місця», та оптимізувавши їх, одержимо зменшення часу виконання програми:

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

де T'_{run} – оптимізований час виконання програми за рахунок оптимізації «вузьких місць».

На основі (1) можна виразити коефіцієнт збільшення ефективності ПЗ:

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

Отже, завдання підвищення ефективності ПЗ зводиться до двох підзадач: виявлення «вузьких місць» у програмному коді та їхньої оптимізації. Схему системи автоматичного підвищення ефективності ПЗ обчислювальних систем на основі такого підходу показано на рис. 3.

У звичайному режимі роботи обчислювальної системи користувачі здійснюють запити й одержують результат Y. У систему вводиться додатковий блок оптимізації, який дозволяє адміністратору переводити систему в режим оптимізації.

У режимі оптимізації здійснюються тестові запити X', на основі яких одержується результат Y'. Паралельно, з допомогою спеціальної серверної бібліотеки Xdebug [4], отримуються дані про послідовність і час виконання програми. Така інформація накопичується у вигляді масиву даних F, кожен елемент якого містить інформацію про тривалість виконання підпрограми, її назву та місце розташування у файлі. Зазначений масив легко трансформується у граф викликів G, на основі якого здійснюється пошук множини підпрограм-«вузьких місць» (7) та їх подальша оптимізація.

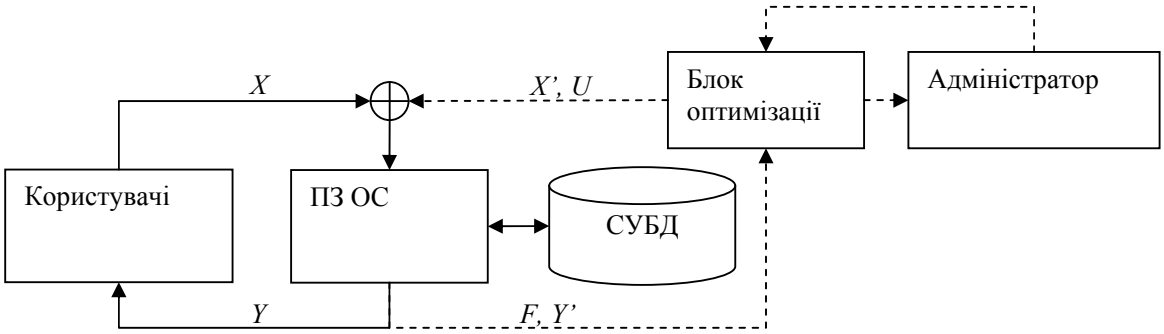

Рис. 3. Схема системи автоматизованого підвищення ефективності ПЗ обчислювальних систем на основі зменшення впливу «вузьких місць»

Завдання автоматичного виявлення «вузьких місць»

Використовуючи визначення «вузьких місць» у ПЗ, наведене в [3], терміном «вузькі місця» позначатимемо множину вершин B графа викликів G, тривалість виконання кожної з яких здійснює значний вплив на сумарну тривалість виконання програми T_{run}. Ступінь такого впливу для кожної вершини можна визначити як:

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

При цьому слід врахувати, якщо вершина містить виклики інших вершин-підпрограм, то тривалість її виклику буде складатися з тривалості всіх наступних викликів. Тому, якщо стверджується умова

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

де $\{c_1, c_2, ..., c_j\}$ – множина вершин-підпрограм, виклик яких здійснюються з даної вершини, то таку вершину не можна вважати «вузьким місцем», оскільки значна тривалість виконання цієї підпрограми зумовлена значною тривалістю викликів дочірніх підпрограм. Тому такі вершини виключаються з множини «вузьких місць» B.

Окрім того, в графі G можуть існувати вершини, які описують підпрограми програмних бібліотек, що використовуються цим ПЗ. Якщо дані бібліотеки є стандартизованими й характеризуються високими якісними показниками, їх виклики не можна розглядати як вузькі місця, імовірніше вони неправильно викликаються розробленим ПЗ. Саме тому доцільно ввести множину

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

яка описує файли таких бібліотек. Підпрограми, знайдені в таких файлах, будуть виключатися з процесу пошуку.

Скомпонувавши алгоритм, наведений в [5], пошуку в глибину, та обмеження (11) і (12), одержимо в результаті алгоритм пошуку множини B всіх вершин-підпрограм системи, для яких можлива оптимізація. Кожен елемент цієї множини характеризується ступенем впливу (10) на загальну тривалість виконання програми, а відповідно, і на ефективність (1).

Вибравши підмножину вершин-підпрограм, які мають найбільший ступінь впливу на ефективність, одержимо необхідну множину «вузьких місць» системи B', над елементами якої здійснимо оптимізацію.

Задача оптимізації «вузьких місць»

Задача оптимізації «вузьких місць» в ефективності обчислювальної системи полягає в мінімізації тривалості виконання підпрограм-вершин B. Множину «вузьких місць» B можна розбити на декілька підмножин, залежно від природи виникнення часових затримок

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

де bc – кількість класів виникнення затримок.

На цьому етапі досліджень нами виділено 2 основних класи виникнення затримок: затримки при виконанні SQL-запитів до БД та затримки в результаті використання недосконалих алгоритмів. Для кожного класу затримок використовуються свої підходи до оптимізації. По першому класу затримок розроблено систему генетичної оптимізації SQL-запитів, однак необхідна подальша робота з розширення кількості класів затримок і підходів щодо їх оптимізації.

Практична реалізація та результати експериментальних досліджень

На основі результатів теоретичних досліджень, наведених у цій роботі, реалізовано систему пошуку часових затримок в ОСІКМ з архітектурою PHP/MySQL. Система будує граф викликів ПЗ у вигляді ієрархічного списку (accordion) (рис. 4).

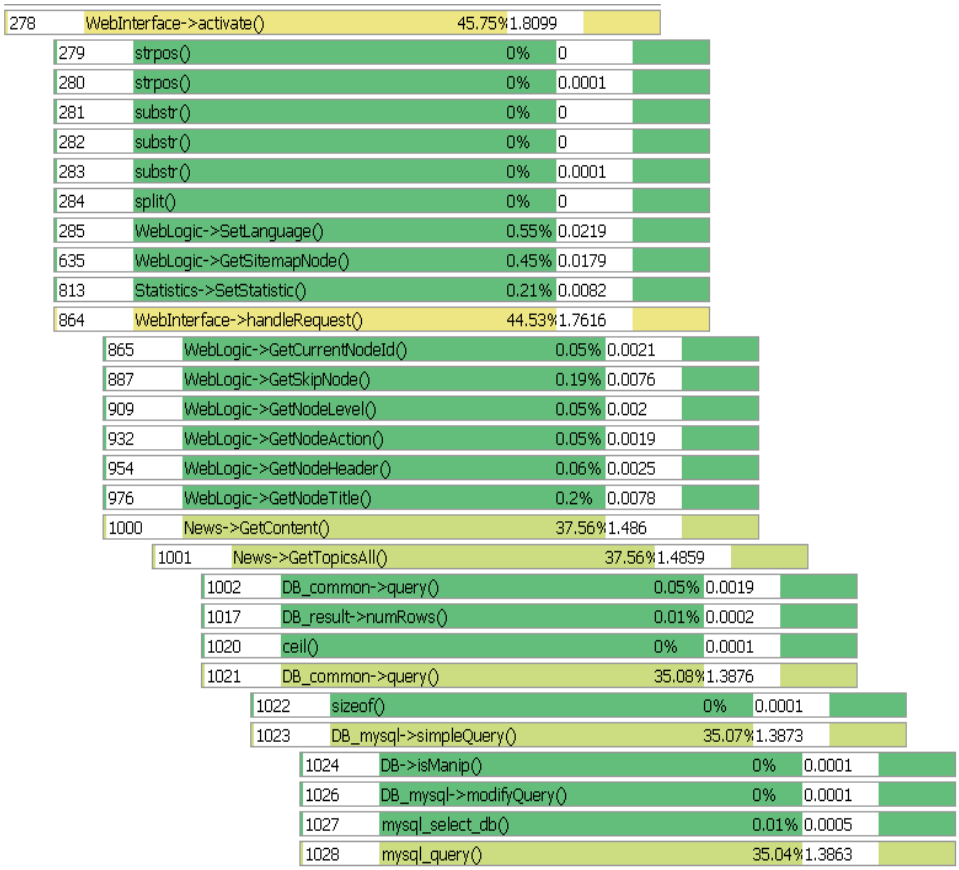

Рис. 4. Візуалізація процесу виконання програмного коду

Колір кожного вузла цього списку встановлюється залежно від його впливу (10) на тривалість виконання програми. У результаті експериментальних досліджень зазначена система засвідчила здатність наочно представляти процес виконання ПЗ обчислювальних систем та оптимізувати їх роботу. У подальшому цю систему буде розширено функцією

автоматичного пошуку «вузьких місць» та їх оптимізації.

Висновки

1. Розроблено математичні моделі для реалізації автоматичного пошуку «вузьких місць» у програмному забезпеченні обчислювальних систем масового користування.

2. Розроблено структуру, а також алгоритмічне забезпечення для побудови системи автоматизованого підвищення ефективності ПЗ обчислювальних систем на основі зменшення впливу «вузьких місць».

3. Проведено експериментальні дослідження, які засвідчили перспективність розробленої методики та можливість її практичного впровадження для зменшення програмних часових затримок різного походження.

СПИСОК ЛІТЕРАТУРИ

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 разработку и анализ / Ананий В.О., Левитин Г.В. – М.: Вильямс, 2006. – с. 212 – 215.

Шабатура Юрій Васильович – к. т. н., доцент, професор кафедри МПА. Тел.: (0432) 59-85-71, e-mail: shabatura@vstu.vinnica.ua.

Штельмах Ігор Миколайович – аспірант кафедри МПА. Тел.: (0432) 50-58-55, e-mail: ceo@sbsgroup.com.ua.

Шабатура Максим Юрійович – студент. E-mail: smartmax@i.ua.

Вінницький національний технічний університет.