УДК 519.6:004.4:004.942

МАТЕМАТИЧНА МОДЕЛЬ ПРЕДСТАВЛЕННЯ ІНФОРМАЦІЇ В ПАМ'ЯТІ ДЛЯ АНАЛІЗУ ПРОДУКТИВНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Мітіков М.Ю., аспірант[*],
mitikov.m22@fpm.dnu.edu.ua, ORCID ID: 0009-0002-1297-5676
Дніпровський національний університет імені Олеся Гончара
(м. Дніпро, Україна)

У сучасному цифровому ландшафті ефективність програмного забезпечення має вирішальне значення для успіху ключових секторів економіки, включаючи фінанси, торгівлю, охорону здоров'я, транспортні перевезення та енергетику. Неоптимальна продуктивність програмного забезпечення може мати серйозні наслідки, такі як зниження продуктивності обчислювальної системи, репутаційні та потенційні втрати.

Виявлення проблем з продуктивністю програмного забезпечення є багатогранним і складним завданням. Проблеми можуть виникати з різних причин: низька ефективність алгоритмів, застарілі технології з низькою продуктивністю, мови програмування, які не використовують повною мірою обчислювальні можливості, а також неправильне управління ресурсами або архітектурою програмного забезпечення. Ці фактори можуть взаємодіяти у складний, непередбачуваний спосіб, ускладнюючи ефективну діагностику та вирішення проблем, що призводить до непередбачуваних збоїв, втрати даних та помилкової роботи системи.

Важливість вирішення цих проблем посилюється поширеністю використання розподілених систем, де незначні дефекти можуть суттєво погіршити продуктивність. Таким чином, здатність виявляти, аналізувати та вирішувати проблеми з програмним забезпеченням має вирішальне значення при розробці та супроводі програмних систем. Отже, завдання розробки нових математичних моделей для виявлення та аналізу проблем продуктивності програмного забезпечення є актуальним. В IT-секторі аналіз програмного забезпечення з метою його оптимізації є необхідним для розробки конкурентоспроможних програмних рішень. Надійність програмного забезпечення стає вирішальним фактором при виборі розробника і корелює зі стійкістю фірми на ринку. Якісне програмне забезпечення, що використовує менше обчислювальних ресурсів, зменшує

[*] Науковий керівник – д-р фіз.-мат. наук, професор Гук Н.А.

витрати, що є життєво важливим для конкурентоспроможності. Надійність, безпека та стійкість до вразливостей є ключовими факторами для запобігання витоку персональних даних або іншої конфіденційної інформації.

Враховуючи ці фактори, вивчення та вирішення проблем продуктивності програмного забезпечення є не лише академічно важливим, але й стратегічним пріоритетом глобальної IT-індустрії. У цьому дослідженні розглядаються проблеми продуктивності програмного забезпечення, представлені підходи до виявлення неефективності та оптимізації програмного забезпечення. Проведено аналіз методів виявлення проблем продуктивності програмного забезпечення, запропоновано рекомендації та обмеження щодо застосування цих методологій.

Ключові слова: програмне забезпечення, продуктивність, знімок пам'яті, дублювання, гранулярність.

Постановка проблеми. В сучасному цифровому світі ефективність роботи програмного забезпечення (ПЗ) є критичним фактором, що забезпечує успіх для цілих секторів економіки, зокрема, фінансового, торгівельного, охорони здоров'я, вантажоперевезень, енергетичного, особливо в період військового стану в Україні. Неоптимальна робота ПЗ може призвести до серйозних збитків, зокрема зниження продуктивності обчислювальних систем [1], репутаційних та потенційних втрат [2].

Виявлення проблем у роботі ПЗ є багатогранною та складною задачею. Проблеми можуть виникати з низки причин, що пов'язані з низькою продуктивністю алгоритмів [3], використанням застарілих технологій з низькою швидкодією, застосуванням мов програмування, що не можуть повністю використати можливості обчислювальної техніки, а також неправильним керування ресурсами та проблемами з архітектурою ПЗ [4]. Ці фактори можуть взаємодіяти між собою у складний та непередбачуваний спосіб, що унеможливлює ефективну діагностику та не дозволяє вирішити проблеми з непередбачуваними поломками, втратою даних, невірним функціонуванням системи.

Важливість вирішення цієї проблеми підсилюється, оскільки зараз для обчислень найчастіше використовуються великі та розподілені системи [5], де незначний відсоток недоліків може призвести до значних втрат в продуктивності. В цьому контексті здатність виявляти, аналізувати та вирішувати проблеми в роботі ПЗ стає життєво важливим аспектом розробки та обслуговування програмних систем. Саме тому задача розробки нових математичних моделей для виявлення та аналізу проблем продуктивності роботи програмного забезпечення є актуальною.

В сучасному IT-секторі аналіз ПЗ з метою його оптимізації дозволяє

створювати конкурентоспроможні програмні рішення, оскільки ненадійність в роботі програм стає каталізатором під час вибору розробника і корелює з перспективами існування підприємства в ринкових умовах [6]. Якісне розроблене програмне забезпечення використовує меншу кількість обчислювальних ресурсів, тож має нижчу собівартість, що є критичним для підтримання конкурентоспроможності, а надійність та безпека, стійкість до уразливості дозволяє запобігти витоку персональних даних або іншої чутливої інформації [2].

З урахуванням цих факторів, вивчення та вирішення проблем в роботі програмного забезпечення стає не тільки академічно важливим питанням, але й стратегічним пріоритетом для глобальної ІТ-індустрії. В роботі розглядаються проблеми продуктивності програмного забезпечення, наводяться підходи до виявлення недоліків у роботі ПЗ та оптимізації. Проведено аналіз методів виявлення проблем у функціонуванні ПЗ, сформульовано рекомендації та обмеження по використанню наведених підходів.

Метою роботи є розробка методики виявлення проблем у роботі ПЗ на основі аналізу знімків пам’яті.

Аналіз останніх досліджень і публікацій. На сьогодні для аналізу програмного забезпечення в залежності від типу проблеми та контексту застосування можуть використовуються різні підходи, серед яких модульне, інтеграційне та системне тестування, аналіз подій та стану програми на основі логування, трасування з метою виявлення послідовності подій та можливих проблеми з логікою програми, аналіз вхідного коду вручну або із застосуванням автоматизованих інструментів, моніторинг системи у режимі реального часу. Однак, зазначені методи аналізу потребують значних зусиль та часових витрат, не дозволяють виявляти проблеми, що можуть виникнути в процесі експлуатації ПЗ. Більш ефективним інструментом для виявлення помилок, проблем з продуктивністю та надмірною витратою ресурсів під час роботи ПЗ є аналіз знімків пам'яті. Знімок пам'яті (memory dump) – це запис вмісту оперативної пам'яті програми у певний момент часу.

Традиційні методи аналізу знімків пам'яті базуються на ручному аналізі та використовують примітивні інструменти для ідентифікації проблем в роботі ПЗ [7]. Ці методи можуть включати в себе збір показників пам'яті, таких як зайнятий та вільний розмір [8], певні шаблони та вільні блоки. Це може бути ефективним для виявлення проблем у роботі простих програм, але є малоефективним і трудомістким для складних систем.

Сучасні методи аналізу знімків пам'яті набули розвитку завдяки впровадженню автоматизованих систем та спроб залучити методи машинного навчання. Ці системи можуть автоматично виявляти аномалії [4, 9], виконувати аналіз структур даних та оперативної пам'яті і корелювати

знайдені проблеми з програмним кодом. Новітні методи використовують візуалізацію знімків пам'яті для спрощення аналізу структури та динаміки. Це не тільки полегшує процес виявлення проблем, але і дозволяє пришвидшити аналіз.

Автоматичні методи не потребують високого рівня технічної експертизи, але є обмеженими за сценаріями і даними, на яких відбувалось тренування в разі застосування систем штучного інтелекту.

Найбільш ефективним підходом до аналізу знімків пам'яті може бути комбінування традиційних та сучасних методів. Із застосуванням гібридного підходу автоматизовані системи слугують "першим фільтром", що ефективно сканує великі об'єми даних та виявляє потенційно проблемні аспекти, а подальший аналіз інженером дозволяє виявити першоджерела проблеми, враховувати природу походження запитів, виявити недостатнє виділення пам’яті для перевикористання попередніх результатів обчислень.

Під час виконання аналізу знімків пам'яті необхідно враховувати технологію програмування, що застосується. У контексті .NET-технологій існує ряд вбудованих діагностичних інструментів, що дозволяють провадити автоматизований аналіз пам’яті та потоків, наприклад, Event Tracing Providers. Ці інструменти спрощують ідентифікацію проблемних аспектів як-то витоки пам'яті. Оскільки .NET оперує на високому рівні абстракції, він дозволяє інтеграцію з розподіленими системами моніторингу, тим самим забезпечуючи можливість аналізу комплексних систем без значущих технічних обмежень.

Використання мови C++, що являє собою низькорівневу мову, вимагає від інженерів глибокого розуміння механізмів управління пам'яттю. Інструменти для аналізу пам'яті в C++ фокусуються, переважно, на ручному виявленні витоків пам'яті та оптимізації алгоритмів управління ресурсами.

Найбільш поширеними проблемами у роботі ПЗ є надмірне використання пам'яті та її виток. Аналіз попередніх досліджень показав, що більшість робіт спрямована саме на вивчення проблеми витоку пам’яті, тому в роботі розглядаються інші аспекти, що призводять до надмірного використання пам’яті, а саме дублюванням незмінної інформації та виділення завеликих обсягів оперативної пам’яті для зберігання даних.

Формулювання цілей статті. З використанням знімків пам’яті необхідно виявити дублювання незмінної інформації та надмірне виділення обсягів оперативної пам’яті для зберігання даних. Оцінити відсоток надмірного використання пам’яті на реальний промисловій системі.

Основна частина.

Матеріали та методи. Знімок пам'яті програми може бути зображеним у вигляді структурованого масиву байтів в такий спосіб:

$$M^{Time} = \{b_1, b_2, ..., b_n\}, \quad (1)$$

де $Time$ – момент часу, у який відбувається зняття знімку пам’яті, b_i – байти з яких складається знімок, $i = 1,...,n$, n – кількість байтів в знімку пам’яті.

O^{Time} – множина об’єктів знімку пам’яті .

$o_{j,k}$ – об’єкт множини O^{Time}, $j = 1,...,m$, m – кількість об’єктів в знімку пам’яті.

T – множина типів об’єктів, категорія даних, що визначає розмір, діапазон та операції над значеннями.

t_k – елемент множини T, $k = 1,...,p$, p – кількість типів об’єктів в знімку пам’яті.

Об’єктом $o_{j,k}$ є екземпляр типу t_k, під який виділяється пам'ять в діапазоні пам’яті, що ідентифікується посиланням на його адресу $\&o_{j,k}$.

На рис.1 зображений приклад об’єкту $o_{j,k}$ адреса якого $\&o_{j,k} = 25daffe2c0$ і тип $t_k = "System.String"$.

Позначимо $s(o_{j,k})$ розмір об’єкту в байтах. В даному випадку $s(o_{j,k}) = 34$.

Поля об’єкту FLD о визначаються типом t_k. Для типу даних $t_k = "System.String"$ $FLD_k = (m\_stringLength, m\_firstChar)$, де $m\_stringLength$ – довжина рядка, $m\_firstChar$ – перший символ рядка.

Таким чином, екземпляр об’єкту в пам’яті задається в вигляді сукупності

$$o_{j,k} = (\&, t_k, s, FLD). \quad (2)$$

```
0:000> !mdt 225daffe2c0
00000225daffe2c0 (System.String) Length=4, String="true"
0:000> !do 225daffe2c0 1
Name:        System.String
MethodTable: 00007ff8d05a59c0 2
EEClass:     00007ff8d0582ec0
Size:    3   34(0x22) bytes
File:        C:\Windows\Microsoft.Net\assembly\GAC_64\mscorlib\v4.0_4.0.0.0__b77a5c561934e089\mscorlib.dll
String:      true
Fields:
              MT    Field   Offset                 Type VT     Attr            Value Name       4
00007ff8d05a85a0  4000283        8        System.Int32  1 instance                4 m_stringLength
00007ff8d05a6838  4000284        c         System.Char  1 instance               74 m_firstChar
00007ff8d05a59c0  4000288       e0       System.String  0   shared           static Empty
                                    >> Domain:Value  00000221f9b0b680:NotInit  00000226fd0157a0:NotInit  <<
0:000> du 225daffe2cc
00000225`daffe2cc  "true"
0:000> db 225daffe2cc
00000225`daffe2cc  74 00 72 00 75 00 65 00-00 00 00 00 00 00 00 00  t.r.u.e.........
00000225`daffe2dc  00 00 00 00 00 00 00 00-00 00 00 00 c0 59 5a d0  .............YZ.
```

Рис. 1. Приклад об'єкту у знімку пам'яті

Інтерпретація знімку пам'яті $P\left(M^{Time}\right)$ в момент часу $Time$ складається з аналізу стеку викликів та реєстру процесора, сканування пам'яті, визначення типів об'єктів, та може бути записна у вигляді:

$$P\left(M^{Time}\right)=\left(O^{Time},THR^{Time}\right), \tag{3}$$

де $THR^{Time}=\left\{thr_1,thr_2,\ldots,thr_d\right\}$ – множина потоків виконання, d – кількість потоків.

Об’єм пам’яті, що керується CLR, дорівнює сумі розмірів усіх об’єктів у знімку:

$$V_{Original}^{Time}=s\left(O^{Time}\right). \tag{4}$$

Позначимо O_{Opt}^{Time} множину об’єктів O^{Time}, що не містять дублювання, тоді об’єм пам’яті, що вони займають:

$$V_{Opt}^{Time}=s\left(O_{Opt}^{Time}\right). \tag{5}$$

Для оцінки надмірного використання пам’яті будемо застосовувати відношення:

$$\frac{V_{Original}^{Time}}{V_{Opt}^{Time}}. \tag{6}$$

Математичний апарат аналізу знімків пам’яті.

Маємо множину об'єктів O^{Time} та множину типів T, які відображені у знімку пам’яті знятому у момент часу $Time$.

Враховуючи, що кожен об’єкт $o_{j,k}$ множини O^{Time} є екземпляром типу t_k множини T, $j=1,\ldots,m$, m – кількість об’єктів в знімку пам’яті, $k=1,\ldots,p$, p – кількість типів об’єктів в знімку пам’яті.

Введемо функцію групування G_1, яка групує елементи вхідної множини об'єктів O^{Time} за типами t_k та визначає кількість об'єктів кожного типу c_k^{Time}:

$$G_1: O^{Time}\rightarrow\left(O_k^{Time},c_k^{Time}\right),\ k=1,\ldots,p \tag{7}$$

тобто результатом групування G_1 до O^{Time} буде пара O_k^{Time} – множина унікальних об’єктів та їхня кількість c_k^{Time} відповідно.

В результаті операції групування отримано підмножини $O_k^{Time}\subseteq O^{Time}$, які містять усі об’єкти типу T.

Математична модель детектування витоку пам’яті.

Витік пам'яті (англ. memory leak) – процес неконтрольованого зменшення обсягу вільної оперативної пам'яті або віртуальної пам'яті комп'ютера, пов'язаний з помилками в програмах, що вчасно не звільняють пам'ять від непотрібних даних, або з помилками системних служб контролю

пам'яті.

Для послідовності знімків пам’яті отриманих з певним інтервалом $\left\{P\left(M^{Time_1}\right),P\left(M^{Time_2}\right),...,P\left(M^{Time_r}\right)\right\}$, де r – кількість знімків пам’яті, застосуємо функцію G_1 для групування об’єктів, що отримані з послідовно знятих знімків пам’яті з певним інтервалом. Це дозволить виявляти типи об’єктів, кількість екземплярів яких монотонно збільшується у часі:

$$\left\{\left(O_k^{Time_1},c_k^{Time_1}\right),\left(O_k^{Time_2},c_k^{Time_2}\right),...,\left(O_k^{Time_r},c_k^{Time_r}\right)\right\}, \quad (8)$$

де $k=1,...,p$, p – кількість типів об’єктів в знімку пам’яті, r – кількість знімків пам’яті.

Враховуючи, що типи в об’єктно-орієнтовному програмуванні декларуються у коді програми і змінюються виключно при внесенні змін до нього, в послідовних знімках пам’яті одного процесу кількість типів не змінюється, тобто $T^{Time_1}=T^{Time_2}=...=T^{Time_r}$.

Критерієм витоку пам’яті будемо вважати монотонне зростання кількості об’єктів певного типу у послідовно знятих знімках пам’яті:

$$c_k^{Time_1}\le c_k^{Time_2}\le ...\le c_k^{Time_r}, \quad (9)$$

де $k=1,...,p$, p – кількість типів об’єктів в знімку пам’яті, r – кількість знімків пам’яті.

Множина L_f містить об’єкти які спричиняють виток пам’яті:

$$L_f=\left\{\left(O_k^{Time_1},c_k^{Time_1}\right),\left(O_k^{Time_2},c_k^{Time_2}\right),...,\left(O_k^{Time_r},c_k^{Time_r}\right):c_k^{Time_1}\le c_k^{Time_2}\le ...\le c_k^{Time_r}\right\}, \quad (10)$$

де f – кількість типів об’єктів з витоком пам’яті.

Математична модель представлення незмінюваної інформації.

Розглянемо складніший для діагностування сценарій дублювання незмінюваної інформації. Виділимо з множини типів T підмножину типів $R\subseteq T$, які мають властивість незмінності. Властивість незмінності полягає у тому, що створений екземпляр об’єкту не має змінних полів всередині і не може бути модифікованим.

Враховуючи, що кожен об’єкт $o_{j,k}$ є екземпляром типу t_k, з вхідної множини об’єктів виділимо множину об’єктів, що мають тип з властивістю незмінності, отримаємо множину IM, що складається виключно з об’єктів, які є екземплярами незмінних типів:

$$IM=\left\{IM_y\right\}=\left\{O^{Time_r}:s\left(t_k^{Time_1}\right)=s\left(t_k^{Time_2}\right)=...=s\left(t_k^{Time_r}\right)=const\right\}, \quad (11)$$

де $y=1,...,z$, z – кількість об’єктів незмінного типу, $k=1,...,p$, p – кількість типів об’єктів в знімку пам’яті.

Згідно з визначенням, об’єкт має фіксовані адресу у пам’яті та розмір, тож фізично об’єкт складається з набору послідовних байтів. Враховуючи

властивість незмінності, два незмінних об’єкти $IM_{y1} = IM_{y2}$ будуть рівними, якщо байти, з яких вони складаються, будуть дорівнюватимуть один одному.

Наступним кроком є отримання множини об’єктів унікальних типів серії знімків пам’яті.

Введемо функцію групування G_2, яка групує елементи вхідної множини об'єктів IM, якщо байти з яких складаються екземпляри $\{IM_y\}$ однакові:

$$G_2 : IM \to (U_v, c_v), \tag{12}$$

де v – кількість об’єктів унікальних типів.

Результатом групування G_2 множними об’єктів IM буде пара U_v – множина об’єктів унікальних типів та їхня кількість c_v відповідно.

Наведемо приклад зі знімку пам’яті.

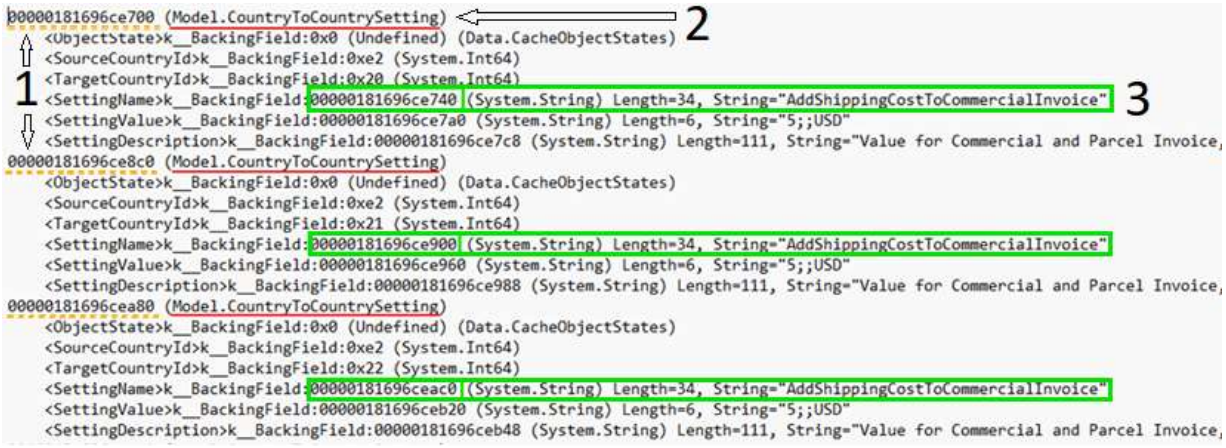

Рис. 2. Приклад дублювання інформації

На рис. 2 відображено результат групування об’єктів O^{Time} за типом $t_k = "Model.CountryToCountrySetting"$ (підкреслено, зона 2) з адресами а (пунктир, зона 1) та значенням полів. У рамках (зона 3) позначені фізично різні рядки з однаковим вмістом. У CLR тип «String» має властивість незмінності, тож наявність різних об’єктів з однаковим змістом викликає дублювання інформації.

Для зменшення обсягів пам’яті, що використовується, незмінні об’єкти не мають дублюватись, тож замість повторюваних об’єктів достатньо використовувати єдиний екземпляр. Цей підхід з перевикористанням єдиної копії незмінного об’єкту має назву пул об’єктів («object pool»).

Розроблена модель виявлення проблем у роботі ПЗ була реалізована у вигляді програмного додатку за допомогою .NET та бібліотеки ClrMD.

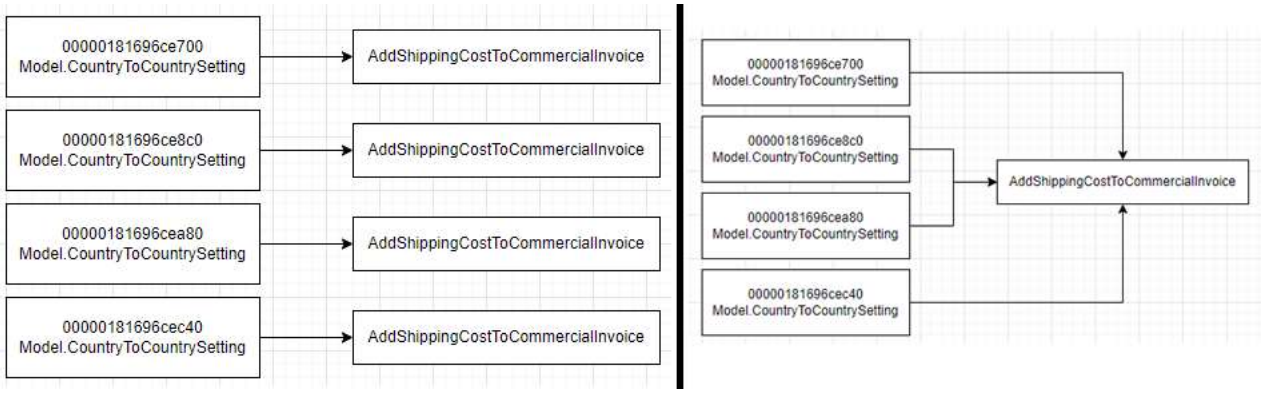

Рис. 3. Різниця між дублюванням та перевикористанням об’єктів

Результати та обговорення.

Для застосування та аналізу ефективності запропонованого підходу розглядався знімок пам’яті системи промислового масштабу розміром більше 20ГБ ($M^{Time} = \{b_1, b_2, ..., b_n\}$, $n > 2 \cdot 10^{10}$). Під час роботи система вимагала фізичного збільшення кількості операційної пам’яті. При застосуванні запропонованого підходу було виявлено дублювання рядків.

Таблиця 1

Топ 5 дублікатів рядків у знімку, що аналізується

Значення рядку	Дублікати	Адреса	Сумарний об'єм
true	635181	225daffe2c0	2.4 MB
HideStandardShippingMethod	390073	225ce525758	9.7 MB
false	318273	225daffe228	1.5 MB
Restricted due to ticket	257600	2258d5f9e80	8.4 MB
visa	241913	225ca1a7d70	945 KB

Було знайдено 220 тисяч унікальних рядків, які дублюються 14.5 мільйонів разів. Загалом, знімок містив 33 мільйона рядків, тож 43 % з них виявилися дублікатами. Загальний обсяг пам’яті становив 2164 МБ, з яких дублікати займають 930 МБ. Усунення дублікатів суттєво зменшило обсяг пам’яті для виконання обчислювального процесу.

Висновки. В роботі здійснено аналіз проблем в роботі ПЗ, пов'язаних з неефективним використанням пам'яті. Модель пам'яті представлена у вигляді знімків в різні моменти часу. Запропоновано математичну модель представлення інформації в пам’яті. Наведено приклад виявлення та усунення проблеми надмірного використання пам’яті в системі промислового масштабу.

Література

1. Labuschagne, Adriaan, Laura Inozemtseva, and Reid Holmes. Measuring the Cost of Regression Testing in Practice. 2017 р. Режим доступу:

https://www.cs.ubc.ca/~rtholmes/papers/fse_2017_labuschange.pdf .
2. Rahul Telang, S. Wattal. An Empirical Analysis of the Impact of Software Vulnerability Announcements on Firm Stock Price. IEEE Transactions on Software Engineering, 2007. Volume 33, Issue 8. Pages 544 – 557. DOI: 10.1109/TSE.2007.70712.
3. Bentley, Jon Louis. Writing efficient programs. Pittsburgh, Pennsylvania: Prentince-hall, 1982. 192p. ISBN:978-0-13-970251-8.
4. Gregg, Brendan. Systems Performance, Enterprise and the Cloud, 2nd ed. Boston: Addison–Wesley Professional, 2020. 928p ISBN-13: 978-0-13-682015-4.
5. Salah T., Zemerly M.J., Yeun C.Y., Al‑Qutayri M., Al‑Hammadi Y. The evolution of distributed systems towards microservices architecture // 2016 11th International Conference for Internet Technology and Secured Transactions (ICITST 2016). – Barcelona, Spain. – 5–7 Dec 2016. – P. 318–325. – IEEE. DOI: 10.1109/ICITST.2016.7856721.
6. J. Christopher Westland. The cost of errors in software development: evidence from industry. *Journal of Systems and Software*, 2002. Vol. 62. Pp.1-9. https://doi.org/10.1016/S0164-1212(01)00130-3.
7. Hewardt M. Advanced .NET Debugging. Boston: Pearson Education, 2010. ISBN 978-0-321-57889-1.
8. Ferrandez T. Debugging .NET Core memory issues (on Linux) with dotnet dump. Microsoft, 2021. Режим доступу: https://www.tessferrandez.com/blog/2021/03/18/debugging-a-netcore-memory-issue-with-dotnet-dump.html.
9. Microsoft. Smart detection in Application Insights. Режим доступу: https://learn.microsoft.com/en-us/azure/azure-monitor/alerts/proactive-diagnostics. (дата звернення: 25.04.2023р).
10. Osipov A. Selecting C# Profiling Tools. Режим доступу: https://urn.fi/URN:NBN:fi:amk-2019061016294.
11. Assiri B., Busch C. Transactional memory scheduling using Machine Learning Techniques [Preprint]. *Proceedings of the 2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, 2016. DOI:10.1109/pdp.2016.21.
12. Noël C. Extensible Software Transactional Memory [Preprint]. *Proceedings of the Third* C *Conference on Computer Science and Software Engineering*, 2010. 12 pp. DOI:10.1145/1822327.1822331.
13. Baghsorkhi S.S., Gelado I., Delahaye M., Hwu W.-M.W. Efficient performance evaluation of memory hierarchy for highly multithreaded graphics processors [Preprint]. *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '12).* New Orleans, LA, USA. Feb 25–29, 2012. pp. 23–34. DOI:10.1145/2145816.2145820.
14. Ustiugov D., Petrov P., Kogias M., Bugnion E., Grot B. Benchmarking,

Analysis, and Optimization of Serverless Function Snapshots. Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS ’21). Virtual, USA, 19–23 Apr 2021. pp. 559–572. DOI:10.1145/3445814.3446714.

MATHEMATICAL MODEL OF INFORMATION REPRESENTATION IN MEMORY FOR SOFTWARE PERFORMANCE ANALYSIS

Mykola Mitikov

In today's digital landscape, the efficacy of software is pivotal for the success of key economic sectors, including finance, commerce, healthcare, freight, and energy. Suboptimal software performance can incur serious repercussions, such as diminished computing system productivity and reputational and potential losses.

Software performance issue detection is multifaceted and challenging. Problems can stem from various causes: low algorithm efficiency, outdated technologies with poor performance, programming languages failing to fully leverage computing capabilities, and mismanagement of resources or software architecture. These factors may interact in complex, unpredictable ways, complicating effective diagnosis and resolution, leading to unforeseeable breakdowns, data loss, and erroneous system operations.

The gravity of solving these issues is accentuated by the prevalent use distributed systems, where minor defects can significantly impair productivity. Thus, the capability to identify, analyze, and resolve software issues is crucial in software system development and maintenance. Hence, the task of developing new mathematical models for detecting and analyzing software performance problems is of current interest. In the IT sector, software analysis aimed at optimization is essential for devising competitive software solutions. Software reliability becomes a decisive factor in developer selection and correlates with a firm's market sustainability. Quality software utilizing fewer computing resources reduces costs, vital for competitive viability. Reliability, security, and vulnerability resilience are key in preventing personal data leaks or other sensitive information exposure.

Considering these factors, examining and addressing software performance issues is not only academically significant but a strategic global IT industry priority. This study examines software performance problems, presenting approaches for identifying software inefficiencies and optimization. It conducts an analysis of methods for detecting software performance issues, offering recommendations and constraints on the application of these methodologies.

Key words: software, performance, memory snapshot, duplication, granularity.

References

1. Labuschagne, Adriaan, Laura Inozemtseva, and Reid Holmes. (2017). *Measuring the Cost of Regression Testing in Practice*.
2. Rahul Telang, S. Wattal. (2007) An Empirical Analysis of the Impact of Software Vulnerability Announcements on Firm Stock Price. *IEEE Transactions on Software Engineering*, 33(8). DOI: 10.1109/TSE.2007.70712.
3. Bentley, Jon Louis. (1982). Writing efficient programs. Pittsburgh, Pennsylvania : Prentince-hall. ISBN:978-0-13-970251-8.
4. Gregg, Brendan. (2021). *Systems Performance, Enterprise and the Cloud, second edition*. ISBN-13: 978-0-13-682015-4.
5. Salah T., Zemerly M.J., Yeun C.Y., Al‐Qutayri M., Al‐Hammadi Y. (2016). The evolution of distributed systems towards microservices architecture. *ICITST 2016*. Barcelona, Spain. Pp. 318–325. DOI: 10.1109/ICITST.2016.7856721.
6. J. Christopher Westland. (2002). The cost of errors in software development: evidence from industry. *Journal of Systems and Software*, 62. 1-9. DOI: 10.1016/S0164-1212(01)00130-3.
7. Hewardt M. Advanced .NET. (2010). Debugging. Boston: Pearson Education. ISBN 978-0-321-57889-1.
8. Ferrandez T. (2021). Debugging .NET Core memory issues (on Linux) with dotnet dump. Microsoft. Retrieved from: https://www.tessferrandez.com/blog/2021/03/18/debugging-a-netcore-memory-issue-with-dotnet-dump.html .
9. Microsoft. Smart detection in Application Insights. (2023). Retrieved from: https://learn.microsoft.com/en-us/azure/azure-monitor/alerts/proactive-diagnostics .
10. Osipov A. Selecting C# Profiling Tools. (2019). Retrieved from: https://urn.fi/URN:NBN:fi:amk-2019061016294.
11. Assiri. B., Busch. C. (2016) Transactional memory scheduling using Machine Learning Techniques. *24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)* [Preprint]. DOI: 10.1109/pdp.2016.21.
12. Noël, C. (2010). Extensible software transactional memory. *Proceedings of the Third C* Conference on Computer Science and Software Engineering*. DOI:10.1145/1822327.1822331.
13. Baghsorkhi, S.S. et al. (2012). Efficient performance evaluation of memory hierarchy for highly multithreaded graphics processors. P*roceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming. 23-34*. DOI:10.1145/2145816.2145820.
14. Ustiugov, D. et al. (2021). Benchmarking, analysis, and optimization of serverless function snapshots. *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 559-572. DOI:10.1145/3445814.3446714.