DOI 10.34185/1562-9945-5-161-2025-19
УДК 519.6:004.4:004.942

М.Ю. Мітіков, Н.А. Гук

АРХІТЕКТУРА ЕКСПЕРТНОЇ СИСТЕМИ ДЛЯ АНАЛІЗУ ЗНІМКІВ ПАМ'ЯТІ

Анотація. Запропоновано архітектуру експертної системи на основі правил, що призначена для аналізу знімків пам'яті програмних додатків з метою виявлення і усунення неефективного використання пам'яті. Система використовує продукційні правила, що ґрунтуються на знаннях про типи об'єктів та їхні властивості (зокрема, незмінність), статистику алокацій та розподіл об'єктів у пам'яті. Розроблено формальну модель представлення знімку пам'яті як сукупності об'єктів з певними атрибутами, введено критерії виявлення дублікатів незмінної інформації та надлишкового резервування пам'яті. Описано механізми групування об'єктів за типами і значеннями для діагностики дублювання даних, а також методи визначення неефективних алокацій, зокрема, надмірного розміру структур даних відносно їх корисного вмісту). Система генерує рекомендації у вигляді інструкцій щодо оптимізації коду, а саме впровадження кешування або пулів об'єктів, перерозподіл структури даних тощо. В системі здійснюється оцінювання очікуваного виграшу в пам'яті. Наведено математичні формалізації для опису вхідних даних, операцій групування, фільтрації та порівняння об'єктів.
Ключові слова: експертна система, знімок пам'яті, дублювання даних, неефективні алокації, продукційні правила, пул об'єктів, оптимізація пам'яті.

Вступ. Ефективне використання пам'яті є важливою складовою продуктивності програмного забезпечення. Неналежне управління пам'яттю, зокрема дублювання даних або використання неефективних структур даних, призводить до надмірного використання оперативної пам'яті і погіршення швидкодії програмних додатків [1]. Більшість існуючих досліджень зосереджені на проблемі *витоків пам'яті*, тоді як інші аспекти, такі як дублювання незмінної інформації, виділення надмірних обсягів пам'яті під дані, вивчені менш ретельно [2]. Метою роботи є побудова *експертної системи* для автоматизованого аналізу знімків пам'яті та виявлення надмірного використання пам'яті з подальшою генерацією рекомендацій щодо їх усунення. Експертні системи широко використовуються у сфері кібербезпеки для кластеризації ознак аномалій і кібератак для їх виявлення [3]. Експертна система продемонструвала точність розпізнавання атак у межах 76–99%, що є співставним із результатами провідних нейромережевих методів. Схоже застосування запропоновано у моделі адаптивного виявлення аномалій на основі логічних процедур [4]. Автори розробили систему "*Threat Analyzer*",

яка автоматично формує матриці ознак для різних класів аномалій та атак, що забезпечує ефективну ідентифікацію відхилень у критично важливих комп'ютерних системах.

У роботі [5] розглядається проблема виявлення аномалій у сенсорних даних процесу буріння нафтових свердловин за умов невизначеності та неповноти вимірювань. Запропоновано алгоритм детектування відхилень, що базується на правилах із коефіцієнтами впевненості, який дозволяє враховувати нечіткість, шум і похибки сенсорних показників. Експертні знання про нормальну динаміку параметрів (тиску, температури, швидкості обертання тощо) формалізуються у вигляді правил, що сигналізують про відхилення від нормативних значень.

У роботі [6] продемонстровано підхід до діагностики аномалій у складних програмних системах на основі строгих математичних моделей. Використання проєкційних перетворень і сингулярного розкладу дозволило досягти високої точності виявлення аномальних станів при помірних затратах обчислювальних ресурсів. Це підтверджує ефективність поєднання математичного моделювання та аналізу знімків стану системи для ідентифікації проблем у її роботі.

У роботі [7] зроблено акцент на формальному критерії виявлення аномалій. Введений авторами інтегральний показник базується на перевірці однорідності вибірок вхідних параметрів, що дозволяє своєчасно відокремити аномальні режими роботи системи. Експериментально підтверджено високий рівень достовірності методу, що свідчить про перспективність такого підходу для підвищення надійності моніторингу станів програмно-апаратних комплексів.

Для вирішення поставленої задачі по виявленню збільшеного використання пам'яті реалізовано рішення у вигляді *рекомендаційної системи*, яка має допомогти інженерам визначати джерела неефективного використання пам'яті та запропонувати ефективні рішення на основі кількісної оцінки.

Для досягнення поставленої мети сформовано базу знань, що включає відомості про інтерпретацію знімків пам'яті, типи та властивості об'єктів (зокрема *незмінність*), статистичні характеристики алокації (розміри масивів, коефіцієнти заповнення структур даних тощо), а також правила (евристики) типу *умова → дія*. Умови правил описують характерні *сценарії неефективного використання пам'яті* (наприклад, наявність багатьох однакових рядкових констант або структур даних із великим невикористаним резервом), а дії містять рекомендації щодо оптимізації – такі, як застосування пулу об'єктів, інтернування рядків чи зміна структури даних.

Постановка задачі. Розглядається задача побудови експертної системи, яка здатна на основі інформації зі знімків пам'яті автоматично виявити дублювання незмінної інформації, надмірне резервування пам'яті, надмірний розмір колекцій, та інші сценарії неефективного використання пам'яті під час роботи програмного додатку. Експертна система повинна надати рекомендації щодо усунення проблем надмірного використання пам'яті, а саме зменшення дублювання, перерозподіл пам'яті, зміну типу зберігання даних, та оцінити ефект оптимізації у відсотках економії пам'яті відносно загального обсягу використання. Необхідно: розробити формальні моделі, що описують знання

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

про предметну область; виявити типові ситуації надмірного використання пам'яті з подальшою генерацією рекомендацій щодо їх усунення; розробити архітектуру експертної системи, механізм виведення; провести тестування розробленої експертної системи на даних знімків пам'яті, отриманих з промислових систем.

Модель знань. Об'єктом дослідження в експертній системі є *знімок пам'яті* програми (memory dump), який зафіксовано станом керованої пам'яті у визначений момент часу. Знімок пам'яті можна подати у вигляді структурованого масиву байтів в такий спосіб: $M_{time} = [b_1, b_2, ..., b_N]$, де *time* – момент часу, у який відбувається зняття знімку пам'яті; $b_1, b_2, ..., b_n$ – байти, з яких складається знімок; N – кількість байтів [8].

Введемо поняття середовища виконання Common Language Runtime (CLR), об'єкту *o* та типу *t*. CLR керує виконанням програм та забезпечує управління пам'яттю, безпеку типів, обробку виняткових ситуацій (помилок). Тип (*t*) – це категорія даних, що визначає розмір, діапазон та операції над значеннями. У C# кожна змінна, константа, вираз, параметр та значення, що повертається, має тип. Типи можуть бути попередньо визначеними (наприклад, String, Integer тощо), або визначеними користувачем. Об'єктом *o* є екземпляр типу *t*, для зберігання якого виділяється пам'ять в діапазоні пам'яті. Об'єкт *o* можна ідентифікувати та маніпулювати ним за допомогою посилання на його адресу (*a*). На рис.1 зображено приклад об'єкту o_x в знімку пам'яті, в рамці 1 позначено адресу $a \in [b_1, b_2, ..., b_n]$, за якою об'єкт o_x знаходиться у пам'яті, $a(o_x) = 25daffe2c0$; в рамці 2 позначено тип $t(o_x)$ = "*System.String*" об'єкту *o*, що характеризується адресою, за якою визначений тип; пунктирною лінією (3) позначено розмір (*s)* об'єкту *o в* байтах, *s(o)* = 34 байта; поля *fl* об'єкту *o* визначаються типом *t* і містять дані, що позначені у рамці 4, $fl(o) = \{ fl_{m\_stringLength}, fl_{m\_firstChar} \}$. Таким чином, екземпляр об'єкту задається в вигляді $o_m = (a, t, s, fl)$.

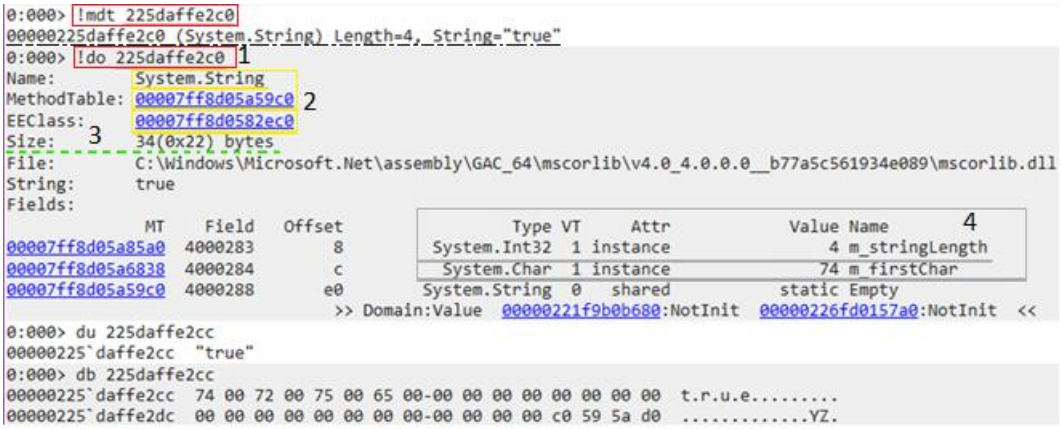

Рисунок 1 - Приклад об'єкту у знімку пам'яті

Обсяг пам'яті, що керується CLR у знімку, дорівнює сумі розмірів пам'яті, що виділяється для зберігання кожного з об'єктів:

$$OriginalHeapSize_{time} = \sum_1^m (S(o_m)).$$

Для подальшого аналізу визначимо *Opt* підмножину об'єктів *opt*, що не містять дублювання, тоді:

$$OptHeapSize_{time} = \sum_1^k (S(opt_k)),$$

де k – загальна кількість унікальних об’єктів у знімку пам’яті, що знятий в момент часу time.

Для оцінки надмірного використання пам’яті будемо застосовувати відношення:

$$\frac{OptHeapSize_{time}}{OriginalHeapSize_{time}}$$

Іншими словами, якщо у знімку присутні кілька різних об’єктів з однаковим внутрішнім станом (значеннями полів), до *Opt* належить лише один представник кожної унікальної комбінації значень. На основі цього вводиться показник *надмірного використання пам’яті* як відношення:

$$E = \frac{S_{\text{total}} - S_{Opt}}{S_{\text{total}}} \times 100\%,$$

де S_{opt} – сумарний розмір об’єктів без дублювання. Показник E (виражений у відсотках) відображає, яка частка пам’яті витрачається надміру через дублювання даних або неефективні алокації. Наприклад, якщо $E=43\%$, це означає, що майже половина пам’яті зайнята зайвими копіями даних, які потенційно можуть бути усунені або через шаблон пул об’єктів, або через шаблон енкодер.

Групування та фільтрація об’єктів. Для аналізу пам’яті у знімку система застосовує *операції групування* об’єктів за певними критеріями, що відповідають гіпотезам про неефективність. Маємо множину об'єктів O_{time} (містить усі об’єкти *o)* та множину типів T_{time} (усі типи *t),* які відображені у знімку пам’яті знятому у момент часу *time*. Враховуючи, що кожен об’єкт *o* є екземпляром типу *t*, введемо функцію групування *G*, яка групує елементи вхідної множини об'єктів *O* за типами:

$$G(O): O \rightarrow T.$$

В результаті операції групування отримано підмножини O_t, які містять усі об’єкти типу *t:*

$$G(O_{time}) = \left\{\{O_{time,t_1}, t_{time,t_1}\} \dots \{O_{time,t_i}, t_{time,t_i}\}\right\}, \text{де } \{t_1 \dots . t_i\} \subseteq T$$

Ця операція дозволяє отримати статистику кількості об’єктів кожного типу, аналізувати розподіли об’єктів за значеннями і структурними характеристиками.

Пошук дублікатів незмінних об’єктів. Виділимо з множини типів *T* підмножину типів *R*, які мають властивість незмінності [9]. 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 є екземпляром типу, з вхідної множини об’єктів виділимо об’єкти, які мають тип з властивістю незмінності:

$$\begin{cases} Immutable: T \rightarrow \{true, false\} \\ R = \{t \in T \mid Immutable(t)\} \\ O_R = \{o_{t_{im}} \in O \mid t_{im} \in R\} \end{cases}.$$

В результаті отримаємо множину O_R, що складається виключно з об’єктів, які є екземплярами незмінних типів.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

Для кожного типу $r \in R$ виконується групування об'єктів цього типу за їхнім внутрішнім станом (набором значень полів). Оскільки об'єкти незмінні, можна ототожнити їх стан із сукупністю байтів, що описують об'єкт у пам'яті. Нехай $Val(o_r)$ позначає відображення об'єкта o_r на його значення (послідовність байтів).

$$\begin{cases} O_r \subseteq O \\ Val : O_r \to V_r \\ Val(o) = v \\ o_1 \sim o_2 \Leftrightarrow Val(o_1) = Val(o_2)\,. \\ V_r = \{\, Val(o) \mid o \in O_r \} \\ G_{val}(O_r) = \frac{O_r}{\sim} = \{\, O_{r,v} \mid v \in V_r \} \\ \mu_r = \frac{|O_r|}{|G_{val}(O_r)|} \end{cases}$$

Тоді групування за значенням розбиває множину об'єктів O типу r на підмножини $O_{r,v}$ з однаковим значенням v. Кожна така підмножина $O_{r,v}$ містить об'єкти типу r з ідентичним вмістом.

Правила експертної системи сформулюємо у вигляді продукцій «ЯКЩО ситуація, ТО дія». Тоді, в разі виявлення середньої кратності повторення μ_r більшої за задану N, можна сформулювати експертне правило, яке надає рекомендацію використання інтернування, в такий спосіб:

$$\textbf{\textit{IF}}\ r \in R \wedge \mu_r > N\ \textbf{\textit{THEN}}\ Recommend(Intern(r)).$$

Виявлення надмірного діапазону значень. Ще однією типовою ситуацією надмірного використання обсягу пам'яті є застосування діапазону значень більшого, ніж потрібно для виконання певної операції [10]. Позначимо $T_{inv} \subseteq T$ – множину обраних типів. Для кожного типу $t \in T_{inv}$ введено множину обраних незмінних полів цього типу $FL_{inv}(t)$. Для кожного поля $fl_t \in FL_{inv}(t)$ діапазон значень визначається:

$$\begin{cases} T_{inv} \subseteq T\,, O_t \subseteq O\,, t \in T_{inv} \\ Val_{fl} : O_t \to Dom(fl) \\ ActualRange_{t,fl} = \left[\min_{o \in O_t} Val_{fl}(o), \max_{o \in O_t} Val_{fl}(o)\right]. \\ TheoRange(fl) = [minDom(fl), maxDom(fl)] \\ \eta(t,fl) = \frac{\max ActualRange_{t,fl} - \min ActualRange_{t,fl} + 1}{\max TheoRange(fl) - \min TheoRange(fl) + 1} \\ 0 < \eta \leq 1 \end{cases}$$

Для кожного типу t з множини T_{inv} виконується пошук діапазону фактичних значень кожного з обраних полів fl_{inv}. З відношення фактичного діапазону даних до теоретично виділеного можна обчислити коефіцієнт використання діапазону значень. Отриманий коефіцієнт застосовано для формулювання правила про обрання типу даних, який потребує менше пам'яті, та з надлишком покриває знайдений діапазон значень:

$$\textbf{IF } t \in T_{inv} \wedge fl \in FL_{inv}(t) \wedge \eta(t, fl) \leq \tau \textbf{ THEN } Recommend(fl \rightarrow ShrinkType(ActualRange_{t,fl}))$$

Виявлення втрат пам'яті через вирівнювання (padding) об'єктів CLR. У середовищі виконання CLR об'єкти мають мінімальний розмір S_{min}, який складається з заголовку об'єкту *Header* та зарезервованого місця під дані *Reserved*. Об'єкти підлягають вирівнюванню *Align(x)* за межами, кратними розміру машинного слова (64 біт, або 8 байт). Це призводить до збільшення фактичного обсягу пам'яті, зайнятого об'єктом, порівняно з обсягом корисних даних. Для кожного типу визначається фактичний розмір об'єкта в пам'яті $S_{actual}(t)$ та теоретичний розмір $S_{raw}(t)$, обчислений як сума розмірів його полів без урахування вирівнювання. На основі співвідношення між цими величинами оцінюється частка пам'яті $Loss_{padding}(t)$, втраченої внаслідок зсуву:

$$\begin{cases} S_{min} = Header + Reserved = 16 + 8 = 24 \\ Align(x) = 8 * \left\lceil \frac{x}{8} \right\rceil \\ S_{fields}(t) = \sum_{i=1}^{m} s(fl_i) \\ S_{raw}(t) = Header + \max\left(S_{fields}(t),\ Reserved\right) \\ S_{actual}(t) = Align\left(S_{raw}(t)\right) \\ Loss_{padding}(t) = \frac{S_{actual}(t) - S_{raw}(t)}{S_{raw}(t)} \end{cases}.$$

Отримане значення дозволяє сформулювати правило для зменшення розміру полів за умови досягнення граничного значення τ_{pad}:

$$\textbf{IF } Loss_{padding}(t) \geq \tau_{pad} \textbf{ THEN } Recommend(RepactFields(t)).$$

Це правило посилює ефект рекомендації наданої правилом щодо «*Виявлення надмірного діапазону значень*».

Виявлення надмірної ємності колекцій. Виявлення надмірної ємності колекцій [11] є узагальненням підходу щодо групування значень полів, оскільки застосовується до типів-колекцій T_{col} та здійснює групування за структурними характеристиками – фактичною кількістю елементів та ємністю внутрішніх буферів. Введемо функцію $FieldType(fl)$, яка повертає тип значення поля та за типом виявляє тільки колекції:

$$\begin{cases} FieldType: FL \rightarrow T \\ Collection: T \rightarrow \{true, false\} \\ FL_{col}(t) = \{\ fl \in FL_{inv}(t) \mid Collection(FieldType(fl))\} \\ Collection(List < T >) = true, Collection(Dictionary < K, V >) = true \end{cases}$$

До типів колекцій відносяться, зокрема, *List<T> (список)* та *Dictionary<K,V> (словник)*, які є найбільш поширеними структурами у .NET-системах. Список (*List<T>*) – це впорядкована змінна структура, що реалізується на основі масиву з автоматичним збільшенням ємності. При досягненні межі поточної місткості відбувається виділення

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

нового масиву подвоєного розміру, копіювання всіх наявних елементів та заміна посилання. Словник (*Dictionary<K,V>)* – це хеш-таблиця, яка асоціює ключі з відповідними значеннями. Зі збільшенням кількості елементів до критичного порогу виконується виділення нового масиву bucket-осередків та оновлення структури зберігання. Обидві структури дозволяють динамічне збільшення кількості елементів за рахунок виділення внутрішніх буферів більших за фактичний об'єм елементів. Для контейнерів групування може проводитись за *ємністю* (розміром виділеного під внутрішні буфери масиву або хеш-таблиці) $CollectionCapacity$ та за *фактичною кількістю* збережених елементів $CollectionElementsCount$:

$$\begin{cases} CollectionElementsCount: O \to N \\ CollectionCapacity: O \to N \\ Val_{fl}: O_t \to O \\ \phi(t,fl) = \frac{1}{|O_t|}\sum_{o \in O_t} \frac{CollectionElementsCount(Val_{fl}(o))}{CollectionCapacity(Val_{fl}(o))} \\ 0 < \phi(t,fl) \leq 1 \end{cases}.$$

Отримане відношення дозволяє сформулювати правило для зменшення розміру колекції:

$$\boldsymbol{IF}\ t\ \in T_{inv}\ \wedge fl\ \in FL_{col}(t)\ \wedge\ \phi(t,fl)\ \leq \tau_{cap}\ \boldsymbol{THEN}\ Recommend(ReduceCapacity(t,fl)).$$

Оцінка вартості зберігання інформації у хеш-колекції типу словник (Dictionary<K,V>). Хеш-колекції типу словник використовують службові структури *W* (масиви *buckets*, *entries*, допоміжні поля *views* та об'єкти представлень ключів/значень) для зберігання інформації *U*. Цей підхід потребує створення додаткових об'єктів, які можуть займати більший обсяг пам'яті ніж збережені значення для невеликої кількості елементів. На основі співвідношення між обсягом корисної інформації *U*, та загальним обсягом *M* оцінюється вартість збереження корисної інформації η:

$$\begin{cases} Collection_{dict}: T \to \{true, false\} \\ T_{dict} = \{t \in T \mid Collection_{dict}(t)\}, \\ O_t = \{o \in O | G(o) = t\}, t\ \in T_{dict} \\ U(o) = \sum_{i=1}^{CollectionElementsCount(o)} (S(key_i) + S(value_i)) \\ W(o) = S(o) + S\big(buckets(o)\big) + S\big(entries(o)\big) + S(views(o)) \\ M(o) = W(o) + U(o) \\ \eta(o) = \frac{U(o)}{M(o)} \\ \eta_t = \frac{\sum_{o \in O_t} U(o)}{\sum_{o \in O_t} M(o)}, 0 < \eta(o) \leq 1 \end{cases}.$$

При зростанні кількості елементів в колекції, додатковий об'єм пам'яті *W* займає меншу частку. Правило експертної системи спрацьовує для колекцій з невеликою кількістю елементів і рекомендує замінити хеш колекцію на колекцію з фіксованою кількістю елементів:

$$\boldsymbol{IF}\ \eta_t\ \leq\ \eta_{min}\ \boldsymbol{THEN}\ Recommend(ReplaceWithNonHashCollection(t))$$

Оцінка потенціалу зменшення використання пам'яті при стисканні масивів байтів. Стискання масивів байтів у пам'яті дозволяє усунути надлишковість використання пам'яті без втрати даних, що дозволяє скоротити споживання пам'яті, підвищити щільність зберігання, тому доцільною є оцінка застосування алгоритмів стискання $Zip(o)$ для O_{t_b} об'єктів типу *System.Byte[]*. Введемо параметр L_{min} , який відповідає мінімальній довжині масиву *C(o)* для ефективної роботи алгоритму стискання:

$$\begin{cases} C: O_{t_b} \to N \\ L_{min} \in N \\ O^*_{t_b} = \{o \in O_{t_b}; C(o) \geq L_{min}\} \\ Zip: O^*_{t_b} \to O^{zip}_{t_b} \\ INFO(o) = INFO(Zip(o)) \\ C(o) \geq C(Zip(o)) \end{cases}.$$

Для оцінювання операції застосуємо метрику ефективності стискання, яка обчислюється як відношення суми довжин всіх наявних масивів з довжиною більшою, або рівною, заданому пороговому значенню з використанням компресії, до суми оригінальних довжин, що використовувалися до застосування алгоритмів компресії:

$$\begin{cases} Ratio_{compress} = \dfrac{\sum_{o \in O^*_{t_b}} C(Zip(o))}{\sum_{o \in O^*_{t_b}} C(o)} \\ 0 < Ratio_{compress} \leq 1 \end{cases}.$$

Отримана метрика дозволяє сформулювати експертне правило по застосуванню алгоритмів компресії:

$$\boldsymbol{IF} \mid O^*_{t_b} \mid > 0 \ \wedge Ratio_{compress} \leq \tau_{zip} \ \boldsymbol{THEN} \ Recommend(ApplyCompression(t_b))$$

Виявлення надмірного споживання пам'яті через надлишкову кількість блоків синхронізації. Для забезпечення потокобезпечного доступу до даних використовується сегментована синхронізація у колекціях, які підтримують одночасний доступ з декількох потоків виконання. Кількість блоків синхронізації $Locks_{cd}$ визначає гранулярність блокувань і впливає на накладні витрати пам'яті. У середовищі .NET тип *ConcurrentDictionary<K,V>* реалізовує потокобезпечний одночасний доступ до елементів колекції словника. За замовченням, кількість блоків синхронізації дорівнює кількості обчислювальних ядер системи, і зростає з кількістю елементів всередині колекції:

$$\begin{cases} G(O): O \to T, \ O_t = o \in O \mid G(o) = t \\ IsConcurrentDict: T \to \{true, false\} \\ T_{cd} = \{t \in T \mid IsConcurrentDict(t)\} \\ Locks_{cd}: O_{cd} \to N \\ KV: T_{cd} \to K \times V, \ KV(t_{cd}) = (K,V), \ t_{cd} \subseteq T_{cd} \end{cases}.$$

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

В разі, коли кількість екземплярів словників $O_{t_{K,V}}$ однакового типу (*K x V)* перевищує порогове значення N_{cd}, ймовірність інтенсивного одночасного доступу з боку великої кількості потоків до кожного окремого словника зменшується, тому рівень гранулярності більший за L_{max} не є виправданим:

$$\begin{cases} N_{cd} \in N, L_{max} \in N \\ O_{K,V} = O_{t_{K,V}},\ t_{K,V} \in T_{cd} \wedge KV(t_{K,V}) = (K,V) \\ O_{K,V}^{locks} = \{o \in O_{K,V} \mid Locks_{cd}(o) > L_{max}\} \end{cases}.$$

Отримані підмножини для визначених типів (*K x V*) містять лише екземпляри об'єктів словників з більшою гранулярністю блокувань і дозволяють сформулювати правило для зменшення гранулярності:

$$\boldsymbol{IF}\ t_{K,V} \in T_{cd} \wedge \left|O_{K,V}^{locks}\right| > N_{cd}$$

$$\boldsymbol{THEN}\ Recommend(ReduceConcurrency(K,V)).$$

Архітектура rule-based експертної системи

Загальна структура. Розроблена експертна система побудована за класичною архітектурою продукційної системи, що включає три основні компоненти: база фактів, база знань та машина виведення. База фактів містить дані, отримані зі знімка пам'яті, що представлені у формалізованому вигляді: множини об'єктів *O*, типів *T*, групувань O_t, $O_{r,v}$ тощо. База знань складається з набору правил виду «ЯКЩО ситуація, ТО дія», де умови сформульовано на основі введених понять про об'єкти і їх групи, а дії відповідають рекомендаціям, що призводять до оптимізації використання пам'яті. Машина виведення реалізує алгоритм прямого виведення для аналізу поточної ситуації, до наявних фактів застосовуються релевантні правила та здійснюється генерація висновків.

Правила в системі можна розподілити на дві групи: правила виявлення проблеми та правила рекомендацій. Перші визначають, чи присутній у знімку пам'яті певний сценарій неефективності, другі – яку саме пораду надати розробнику. Наприклад, правило виявлення може бути сформульоване так: *«Якщо знайдено більш ніж 1000 рядків типу System.String з однаковим значенням, яке повторюється понад 100 разів кожне, то зафіксувати випадок дублювання рядкових констант»* Це правило спрацьовує на основі аналізу груп $O_{string,v}$, отриманих при групуванні рядків за значенням *v*. Правило рекомендації, пов'язане з ним, можна записати в такий спосіб: *«Якщо в системі виявлено значне дублювання незмінних рядків, рекомендувати застосувати інтернування рядків або кешування їхніх значень в єдиному екземплярі (пул об'єктів).»* Іншим прикладом є формулювання правила виявлення для неефективної колекції, яке можна сформулювати так: *«Якщо об'єкт типу Dictionary має фактичне заповнення менше 20% від розміру його внутрішнього буфера, позначити цей словник як кандидата для здійснення оптимізації».* Тоді відповідна рекомендація матиме таке формулювання: *«Розглянути можливість зменшення початкової ємності словника або запропонувати використан-*

ня альтернативної структури (наприклад, списку або масиву), якщо обсяг даних невеликий».

Таким чином, продукційна система послідовно аналізує факти, отримані зі знімка пам'яті, співставляючи їх із умовами правил. При виконанні умови спрацьовує права частина правила – формування нового факту (виявленої проблеми) або безпосередньо рекомендації. Важливо, що правила можуть мати пріоритети або пороги спрацювання, щоб знизити кількість хибно-спрацьовуючих порад. Знання для правил були отримані на основі аналізу характеристик промислових .NET систем з використанням бібліотеки ClrMD для доступу до внутрішньої структури пам'яті.

Джерела знань та інженерія бази правил. Правила створено на основі декількох джерел знань: (а) аналіз коду та статистики утиліти для дослідження знімків пам'яті, (б) публікації авторів з результатами експериментів, та (в) загальні принципи оптимізації з області розробки ПЗ. Інструментальним базисом для (а) став програмний додаток, що виконує автоматизований збір статистики знімка пам'яті (використовує *Microsoft.Diagnostics.Runtime*, відому як ClrMD). Цей додаток генерує звіти на які і спираються умови правил.

Аналіз результатів. Запропонована архітектура експертної системи на основі правил була апробована на реальних знімках пам'яті великого промислового веб-додатку (20–120 ГБ кожен). Система успішно виявила типові проблеми, підтверджуючи дані попередніх досліджень. Зокрема, було знайдено і рекомендовано до усунення значне дублювання рядкових даних (до 43% рядків – дублікати) [22], що вказує на потребу впровадження кешування константних строкових значень. Також ідентифіковано численні випадки неефективного використання колекцій: словники із заповненням <20%, масиви з великою кількістю невикористаних елементів, тисячі порожніх масивів та списків. Кожна така знахідка супроводжувалась конкретною порадою, як покращити код або конфігурацію. Наприклад, для великого словника налаштувань було запропоновано зменшити його розмір або розбити на декілька менших словників за категоріями, оскільки 80% його bucket-осередків лишалися порожніми. В іншому випадку масив об'єктів був заповнений менш ніж на 10%, і система порадила перейти до використання структури даних список, який збільшується в разі додавання елементів, замість використання масиву фіксованого обсягу.

Отримані рекомендації були надані розробникам додатку, що підлягав тестуванню. Експериментальні оцінки показали, що реалізація цих рекомендацій дозволила знизити пікове використання пам'яті на 15–30% в залежності від сценарію використання, що є суттєвим покращенням продуктивності без додаткових апаратних ресурсів. Таким чином, правило-орієнтована експертна система довела свою ефективність як інструмент аналізу і оптимізації пам'яті програмних додатків.

Висновки. У роботі розглянуто підхід до аналізу ефективності використання пам'яті комп'ютеру під час роботи програмного додатку. Розроблено модель представлення знімку пам'яті комп'ютера як множини об'єктів, надано опис операції групування об'єктів за типами і значеннями, визначено показники надмірного використання

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

пам’яті та ефективності збереження інформації. Виявлено та надано формалізований опис типових ситуацій надмірного використання пам’яті, зокрема ситуацій, пов’язаних з дублюванням даних та надлишковим резервуванням пам’яті для зберігання інформації.

Запропоновано архітектуру експертної системи на основі продукційних правил, яка здійснює аналіз знімків пам’яті для виявлення ситуацій надмірного використання. Надано формулювання продукційних правил та побудовано базу знань, що охоплює інтерпретацію знімків пам’яті, властивості типів (незмінність), статистичні показники алокації. Експертна система на основі правил була апробована на реальних знімках пам’яті великого промислового веб-додатку, реалізація рекомендацій експертної системи дозволила знизити пікове використання пам’яті на 15–30% в залежності від сценарію використання.

Запропонований підхід є незалежним від конкретної реалізації програмного коду та базується виключно на фактичному стані пам’яті, зафіксованому у знімку.

Перспективи подальших досліджень можуть бути спрямовані на розширення бази правил для врахування інших типів неефективності (наприклад, надлишкове дублювання коду або обчислень), інтеграцію статистичних методів і машинного навчання для автоматичного налаштування порогів спрацювання правил, а також проведення масштабних експериментів на різних платформах (Java, C++) для перевірки універсальності підходу. Розроблена рекомендаційна система є основою інструменту, що підвищує ефективність роботи програмних додатків шляхом глибокого аналізу використання пам’яті під час їх роботи та експертної оцінки знайдених аномалій.

ЛІТЕРАТУРА / REFERENCES

1. Mitikov, N. and Guk, N. (2024) ‘Investigation of software application performance issues’, *Mathematical and computer modelling. Series: Technical sciences*, 25, pp. 22–36. doi:10.32626/2308-5916.2024-25.22-36.

2. Nataliia, H. and Nikolay, M. (2024) ‘Modern problems of Anomaly Identification in enterprise systems’, *System technologies*, 5(154), pp. 146–153. doi:10.34185/1562-9945-5-154-2024-15.

3. Lakhno, V., Tkach, Y., *et al.* (2016) ‘Development of adaptive expert system of information security using a procedure of clustering the attributes of anomalies and cyber attacks’, *Eastern-European Journal of Enterprise Technologies*, 6(9 (84)), pp. 32–44. doi:10.15587/1729-4061.2016.85600.

4. Lakhno, V. *et al.* (2016) ‘Design of adaptive system of detection of cyber-attacks, based on the model of logical procedures and the coverage matrices of features’, *Eastern-European Journal of Enterprise Technologies*, 3(9(81)), p. 30. doi:10.15587/1729-4061.2016.71769.

5. Protsiuk, V. (2024) ‘Anomaly detection models of for sensor data of oil and gas well drilling process under uncertainty’, *Herald of Khmelnytskyi National University. Technical sciences*, 333(2), pp. 177–188. doi:10.31891/2307-5732-2024-333-2-29.

6. Meleshko, Y. *et al.* (2024) ‘Development a set of mathematical models for anomaly detection in high-load complex computer systems’, *Eastern-European Journal of Enterprise Technologies*, 6(4 (132)), pp. 14–25. doi:10.15587/1729-4061.2024.316779.

7. Semenov, S. *et al.* (2022) ‘Devising a procedure for defining the general criteria of abnormal behavior of a computer system based on the improved criterion of uniformity of input data samples’, *Eastern-European Journal of Enterprise Technologies,* 6(4 (120)), pp. 40–49. doi:10.15587/1729-4061.2022.269128.
8. Mitikov, N.Y. and Guk, N.A. (2023) ‘Detection of software problems based on memory dump analysis’, *Problems of applied mathematics and mathematic modeling,* pp. 171–178. doi:10.15421/322318.
9. Mitikov, N.Y. and Guk, N.A. (2024) ‘Modeling and automation of the process for detecting duplicate objects in memory snapshots’, *Herald of Advanced Information Technology,* 7(2), pp. 147–157. doi:10.15276/hait.07.2024.10.
10. Guk, N., Mitikov, N. and Selivyorstova, T. (2024) ‘Detecting extraordinary application memory use by analyzing memory screenshots’, *Science and Technology Today* [Preprint], (10(38)). doi:10.52058/2786-6025-2024-10(38)-39-49.
11. Mitikov, N., & Guk, N. (2025). Enhancing collection performance in software through Memory Snapshot Analysis and mathematical modeling. *Modern Problems of Modeling*, (27), 109–122. https://doi.org/10.33842/2313-125x-2025-19-109-122
12. Mitikov, M. (2025). Mathematical model of information representation in memory for software performance analysis. Modern Problems of Modeling, (28), 96–107. https://doi.org/10.33842/2313-125x-2025-30-96-107

Received 03.11.2025.
Accepted 17.11.2025.

Architecture of an expert system for analysing memory snapshoots

This paper proposes a rule-based expert system architecture designed to analyse software application memory snapshots in order to identify and eliminate inefficient memory usage. The system uses production rules based on knowledge about object types and their properties (in particular, immutability), allocation statistics, and object distribution in memory. A formal model for representing a memory snapshot as a set of objects with certain attributes has been developed, and criteria for identifying duplicates of immutable information and excessive memory reservation have been introduced. Mechanisms for grouping objects by type and value to diagnose data duplication are described, as well as methods for identifying inefficient allocations, in particular, excessive data structure size relative to their useful content. The system generates recommendations in the form of instructions for code optimisation, namely the implementation of caching or object pools, redistribution of data structures, etc. The system evaluates the expected memory savings. Mathematical formalizations are provided to describe the input data, grouping operations, filtering, and object comparison.

Keywords: expert system, memory snapshot, data duplication, inefficient allocations, production rules, object pool, memory optimisation.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

Гук Наталія Анатоліївна – завідувачка кафедри комп'ютерних технологій, доктор фізико-математичних наук, професорка, Дніпровський національний університет імені Олеся Гончара, huk_n@fpm.dnu.edu.ua, ORCID ID: 0000-0001-7937-1039

Мітіков Микола Юрійович – аспірант кафедри прикладної математики, Дніпровський національний університет імені Олеся Гончара, mitikov.m22@fpm.dnu.edu.ua, ORCID ID: 0009-0002-1297-5676

Huk Nataliia - Head of the Department of Computer Technologies, Doctor of Physical and Mathematical Sciences, Professor, Oles Honchar Dnipro National University, huk_n@fpm.dnu.edu.ua, ORCID ID: 0000-0001-7937-1039

Mitikov Nikolay - PhD student, Department of Applied Mathematics, Oles Honchar Dnipro National University, mitikov.m22@fpm.dnu.edu.ua, ORCID ID: 0009-0002-1297-5676