УДК 004.7:004.415-025.25 **doi: 10.26906/SUNZ.2021.3.074**

М. О. Волк, М. В. Гора, В. Г. Демчук, Т. І. Ольшаньска, О. В. Ткаленко

Харківський національний університет радіоелектроніки, Харків, Україна

МОДИФІКОВАНИЙ МЕТОД САМОВІДНОВЛЕННЯ ПРОГРАМНИХ СИСТЕМ З ВИКОРИСТАННЯМ ДАМПУ ПАМ'ЯТІ

Анотація. Предметом статті є дослідження методів самовідновлення програмних систем. **Мета статті** полягає у підвищенні ефективності засобів самовідновлення програмних систем шляхом розробки модифікованого методу самовідновлення програмних систем з використанням дампу пам'яті. Використаними **методами** є методи декомпозиції складних систем, методи самовідновлення програмних систем, методи теорії множин. Отримані такі **результати**: модифіковано моделі програмних компонент системи та обчислювальних ресурсів, модифіковано метод самовідновлення програмних систем з використанням дампу пам'яті. **Висновки.** Розроблено модифікований метод самовідновлення розподіленого програмного забезпечення програмних систем з використанням дампу пам'яті, який на відміну від існуючого, забезпечує самовідновлення роботи програмної системи на віддалених ресурсах у тому стані, в якому вона перебувала у момент відмови одного з ресурсів, що відбувається в умовах зменшення кількості ресурсів.

Ключові слова: програмна система, самовідновлення, розподілені ресурси, дамп пам'яті.

Вступ

Розподілені програмні системи розробляються в умовах перманентного розвитку апаратних платформ, операційних систем, хмарних сервісів. Часто вони мають складну розподілену структуру, а елементи системи розташовані на віддалених комп'ютерах або віртуальних ресурсах. Іноді відбувається відмова одного з програмних компонентів, гіпервізорів, каналів передачі даних або один з задіяних сервісів не може впоратися з поточним підвищенням навантаження. Також завжди є ймовірність впливу на функціонування програмних компонент зовнішніх факторів. Існує декілька підходів до відновлення функціонування системи у таких випадках, зокрема забезпечення функціональної стійкості та живучості обчислювального процесу [1].

Одним з існуючих механізмів, які забезпечують функціональну стійкість та живучість технічних систем є самовідновлення. У програмних системах термін «самовідновлення» встановлює властивість будь-якої програми, сервісу або операційної системи, яка здатна виявляти ситуації, коли програма не функціонує стандартно, та без будь-якого зовнішнього втручання оператора зробити необхідні дії, які відновлять працездатність системи в звичайному стані [2, 3]. Аналогічні дії така система може виконувати також у випадках ймовірної відмови програм, апаратних компонентів або каналів передачі даних [3]. Процес самовідновлення робить програмну систему здатною періодично перевіряти свій стан, приймати рішення щодо оптимізації, адаптації до різних умов та відновлювати працездатність.

Аналіз публікацій. Засоби самовідновлення можуть бути реалізовані як самостійно, так і з використанням існуючих компонентів комп'ютерних систем. Вони можуть використовувати апаратні засоби (hardware), системне програмне забезпечення (middleware), та стороннє прикладне програмне забезпечення (software або software system) [2-5].

Самовідновлення на рівні прикладного програмного забезпечення - це здатність окремого додатка, програмної системи або процесу відновити працездатність зсередини [3, 5]. Зазвичай, у програмних фрейворках (наприклад, у платформі .NET або віртуальній машині Java), такі випадки обробляються винятками (an exception), які аналізують помилку та можуть здійснювати кроки по її виправленню. У випадку виключення, програміст має можливість різної реакції: ігнорувати збій та виконувати програму далі, перервати виконання (exit) програми, додати функцію (function) обробки, яка виправить ситуацію, або взагалі перезавантажити програму. На сьогодні є ефективні засоби обробки виключень самою програмою, наприклад, існують шаблони проектування (templates), які автоматизовано створюють програмні системи з функціями самовідновлення [6, 7].

Можливе також самовідновлення на системному рівні. Зазвичай виконується операційною системою, фреймворком або гіпервізором. Засоби системного рівня застосовуються до всіх програм незалежно від особливостей їх внутрішньої реалізації та побудовані на стандартних сервісах операційних систем. Ці засоби можна використовувати на будь-якому рівні комп'ютерної системи, що функціонує під керуванням мережної операційної системи. Як приклад відновлення системного рівня можна розглядати повторний запуск процесу, який відмовив або не відповідає.

Якщо відбуваються відмови апаратних засобів, самовідновлення відповідає за пошук нових ресурсів, за якими буде відбуватися перерозподіл програмних компонентів з обов'язковим встановленням нових мережних зв'язків. Ефективність роботи системи у таких випадках залежить від наявності підсистем моніторингу, які накопичують інформацію про стан програм, обладнання, каналів зв'язку, та, у випадках виявлення збоїв, забезпечують перерозподіл комп'ютерних ресурсів [8-12].

Постановка проблеми. Для реалізації механізмів самовідновлення у програмних системах існують ряд методів. Один з них – метод самовідновлення програм з використанням дампу пам'яті. Це один з універсальних методів, який дозволяє зберігати стан програмних компонентів у часі та оновлювати стан

при перезапуску програм. Також цей метод є не складним за реалізацією і багато його етапів можна реалізувати вбудованими засобами операційної системи (наприклад, механізмами баз даних або автозбереження). Але цей метод серед існуючих має найбільшу вартість, виходячи з об'ємів фізичної пам'яті для зберігання даних та часу виконання.

Для зменшення вартості виконання та розширення можливостей засобів самовідновлення в програмних системах, запропоновано модифікацію методу, який дозволяє розміщення декількох програм на одному обчислювальному ресурсі, виконує постійний моніторинг наявних віддалених ресурсів.

Мета статті полягає у підвищенні ефективності засобів самовідновлення програмних систем шляхом розробки модифікованого методу самовідновлення програмних систем з використанням дампу пам'яті.

Моделі програмних компонентів та обчислювальних ресурсів

В роботі було використано моделі та визначення, які були введені в роботі [9]. На рис. 1 наведено структуру обчислювального середовища. Програмні компоненти P_i, які є елементами множини програмної системи, складаються з двох складових: бінарний коду програми, який реалізує поведінку програми (Cd), і локальних даних (Dt), які формують стан процесу в кожен момент часу. Код програм можна розділити на такі групи: прикладний (Cd_u), моніторингу (Cd_m), збереження даних (Cd_s). Так само локальні дані можна поділити на внутрішні дані програми, які фіксують стан (Dt_s) та дані моніторингу або результату (Dt_r), що передаються в систему управління обчислювальним процесом.

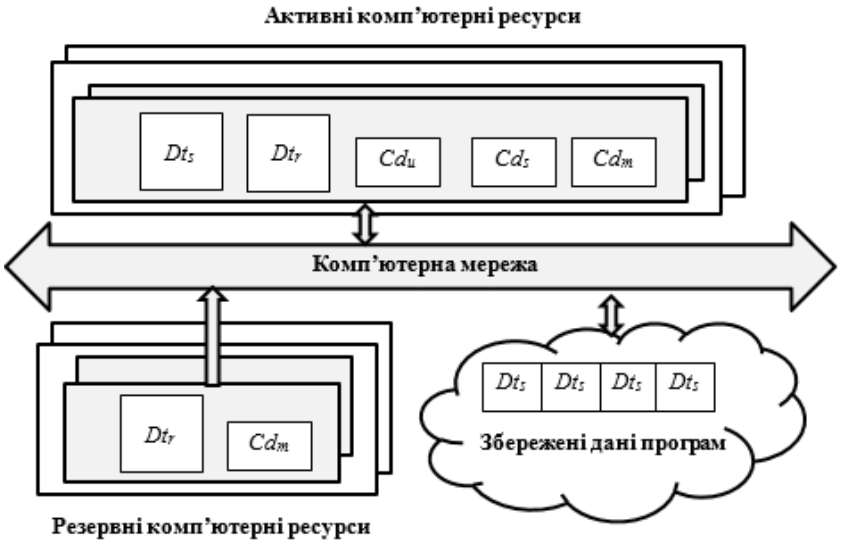

Рис. 1. Структура середовища виконання розподіленої програмної системи

Таким чином, програмна система може бути представлена моделлю:

$$PS = \bigcup_i P_i,\ i = \overline{1, N},$$
$$P = \{\bigcup (Cd,\ Dt)\} = \{Cd_u, Cd_m, Cd_s, Dt_s, Dt_r\} \quad , \quad (1)$$

де N –кількість програм в системі.

Запропонуємо новий компонент системи самовідновлення, який дає можливість при наявності тексту програм та компілятору, який генерує бітовий код програмних компонент під час виконання завдання. Тоді до множини програмних компонент системи додамо компілятор Cd_C, а до даних завдання додаємо файли з програмними компонентами на високорівневих мовах програмування $Dt_{P_i},\ i = \overline{1, N}$. У гетерогенному комп'ютерному середовищі кількість компіляторів $Cd_{Ck},\ k = \overline{1, K}$ залежить від кількості ймовірних операційних систем, на яких можна запустити програмний компонент P_i. Таким чином, множина ресурсів, які використовуються під час процесу обчислень, може набути атрибуту за ознакою можливості отримання компільованої версії програми P_i на обчислювальному ресурсі R_j.

Введемо функцію $\varphi(Dt_{P_i}, R_j)$, що виконує оцінку можливість отримання програми P_i при наявності вхідного коду $Dt_{P_i},\ i = \overline{1, N}$, компілятору для програмно-апаратної платформи $Cd_{Ck},\ k = \overline{1, K}$, для окремого обчислювального ресурсу R_j:

$$\varphi(Dt_{P_i}, R_j) = \begin{cases} P_i^j, \text{ якщо } \exists Cd_{Ck}, k = \overline{1, K}, \\ \varnothing, \text{ в іншому випадку} \end{cases} . \quad (2)$$

Якщо $\varphi(Dt_{P_i}, R_j) \neq \varnothing$, то можлива процедура отримання програмного коду $Dt_{P_i} \xrightarrow{Cd_{Ck}} P_i$ для апаратно-програмної платформи даного комп'ютерного русурсу. Обчислювальні ресурси (зазвичай окремі комп'ютери) становлять множину $R = \{R_j\},\ i = \overline{1, M}$, де j – номер комп'ютера, а N – кількість доступних комп'ютерів у мережі. Серед наявних ресурсів можна виділити активні R_a, які задіяні в обчислювальному процесі та резервні, які з боку системи знаходяться у режимі простою R_r:

$$R = R_a \cup R_r \quad (3)$$

Будь-який комп'ютер, що залучений до процесу розподілених обчислень описується множиною параметрів, наприклад, архітектурою процесора, операційною системою, об'єм оперативної пам'яті тощо. В залежності від типу завдань, ця множина може бути розширена [8]. Множина ресурсів (3) зазвичай змінюється (видалення, додавання комп'ютерів в систему, завантаження сегментів комп'ютерної мережі, еволюція кластерів, операційних систем, драйверів тощо). Обчислювальними ресурсами можуть виступати як окремі комп'ютери, мікроконтролери, так і локальні, глобальні мережі, кластери, віртуальні, хмарні обчислювачі.

Модифікований метод забезпечення самовідновлення з використанням дампу пам'яті програмних компонент

Метод забезпечення самовідновлення з використанням дампу пам'яті програмних компонент серед існуючих методів має найбільшу вартість виходячи з об'ємів фізичної пам'яті та часу збереження і оновлення даних. З іншого боку, він може бути реалізованим вбудованими засобами сучасних операційних систем або віртуальних машин (прикладом можуть бути засоби механізму гібернації), та не є складним з боку програмування [13, 14].

Розглянемо етапи модифікованого методу самовідновлення обчислювального процесу в гетерогенних обчислювальних системах на основі дампу пам'яті програм.

1. На першому етапі виконується розподіл програм за комп'ютерами згідно схеми призначення. Програми завантажуються на віддалені ресурси, проходять ініціалізацію (приводяться до початкового стану), та очікують синхронізації та запуску. Початковий стан усіх програм з системи виконуємо шляхом виклику процедури збереження дампу пам'яті $P_{damp\ i}(t_0)$ для усіх N програм, які знаходяться у програмній системі. Атрибут часу для усіх даних дорівнює індексу початкового часового інтервалу (q=0). Це надає можливість збереження та відновлення початкового стану програмної системи.

2. Знаходження комп'ютерних ресурсів, які є вільними після реалізації схеми призначення (пункт 1) та потенційно можуть бути використані для розміщення програм з завдання:

$$R^e = \bigcup_{m=1}^{M} R_m \setminus \bigcup_{i=1}^{N} R_r \Big| r = j, \forall\{P_i \to R_j\}, \quad (4)$$

якщо $R^e = \varnothing$, $R^e = \bigcup_{m=1}^{M} R_m$.

Можлива ситуація, коли обчислювальних ресурсів недостатньо та множина кількості ресурсів з (3.2) дорівнює

$$R^e = \varnothing..$$

Тоді, можна прийняти

$$R^e = \bigcup_{m=1}^{M} R_m .$$

Можлива модифікація методу, в якому підмножина враховує коефіцієнти використання комп'ютерних ресурсів.

3. Кожному з ресурсів з множини (4) ставиться у відповідність підмножина програмних компонентів, які можливо виконувати на даному обчислювальному ресурсі R^e_{ij} . Для отримання підмножин використовується функція (2). Алгоритм знаходження підмножин полягає у наступному:

3.1. Ініціалізація змінної $j=1$, яка відповідає за перебіг потенційно вільних ресурсів з множини (3).

3.2. Ініціалізація змінної $i=1$, яка відповідає за перебіг усіх програмних компонент.

3.3. Включення в підмножину R^e_{ij} ресурсу, який задовольняє вимогам (2):

$$R^e_{ij} = R^e_{ij} \cup \varphi(Dt_{\mathrm{Pr}_i}, R_j);$$

3.4. Збільшення індексу $i=i+1$. Якщо $i<N$, перехід до п.п. 3.3.

3.5. Збільшення індексу $j=j+1$. Якщо $j<M$, перехід до п.п. 3.2.

4. Визначити наступний момент часу t_q, q=q+1 виконання дампу пам'яті програмних компонентів. Виконати перевірку умов закінчення обчислень. Якщо умова виконана, перехід до пункту 9.

5. При досягненні часу t_q отримати збереження усіх програмних компонент множини

$$\bigcup_{i=1}^{N} P_{dump\,i}(t_q) .$$

У випадку, коли усі програми виконано, переходимо до пункту 3. У випадку, коли залишилися програми, або відповідь від програми одержано, переходимо до пункту 6.

6. Визначаємо підмножину програм з завдання, від яких не отримано даних (дамп пам'яті не виконано) $P^{Err} \subset P$.

7. Для отриманої множини P^{Err} зробити перерозподіл ресурсів за новою схемою призначення:

$$Sh^{Err} = \{P^{Err} \xrightarrow{R^e_{ij}} R^e\}$$

з урахуванням наявності відкомпільованих програмних компонент. Компіляцію можливо виконати заздалегідь або під час перерозподілу.

8. Завантаження програм у відповідності до схеми призначення на обчислювальні ресурси Sh^{Err} . Виконати ініціалізацію програм та відновити стан програми, виконуючи функцію $P'_{damp\ i}(t_q)$ для усіх програм (q=0) з множини P^{Err} . Перехід до пункту 2 даного методу.

9. Завершення виконання програмного завдання, закриття програм управління обчислювальним процесом.

Метод виконує відновлення функціональності програмної системи.

Під час роботи, метод потребує часу для виконання перезавантаження та пошук ресурсу, оновлення даних пам'яті програм зі зроблених дампів пам'яті.

Протягом даного часу система може не відповідати, якщо призупинено критичні програмні компоненти, або продовжувати виконання набору інших функцій.

Висновки

Авторами запропоновано модифікований метод самовідновлення розподіленого програмного забезпечення програмних систем з використанням дампу пам'яті. Серед найбільш важливих можливостей, що надає метод, є самовідновлення роботи програмної системи на віддалених ресурсах у тому стані, в якому вона перебувала у момент відмови одного з ресурсів та зменшення кількості ресурсів в порівнянні з класичним методом. Останнє є важливою особливістю, тому що стандартний метод є дуже ресурсозатратним.

Напрямком подальших досліджень методу самовідновлення може бути розробка модифікацій методу, який зменшує час відновлення програмної системи, що є важливим для програмних систем реального часу.

СПИСОК ЛІТЕРАТУРИ

1. І.П. Саланда, О.В. Барабаш, А.П. Мусієнко. Система показників та критеріїв формалізації процесів забезпечення локальної функціональної стійкості розгалужених інформаційних мереж. Системи управління, навігації та зв'язку. Збірник наукових праць. – Полтава: ПНТУ, 2017. – Т. 1 (41). – С. 122-126.
2. Schneider, C., Barker, A., & Dobson, S. A survey of self- healing systems frameworks. Software: Practice and Experience, 45(10), 2015. pp. 1375-1398.
3. Manzoor A., Rajput U, Phulpoto N, Abbas F, Rajput M. Self-healing in Operating Systems. IJCSNS International Journal of Computer Science and Network Security, Vol.18 No.5, May 2018, pp.92-98.
4. Hudaib, AA., Fakhouri, HN., Al Adwan, FE., & Fakhouri, SN. A Survey about Self-Healing Systems (Desktop and Web Application), Communications and Network, Vol.09 No.01, 2017, pp.71-88.
5. Wang, Z., & Wang, J. Self-healing resilient distribution systems based on sectionalization into microgrids, IEEE Transactions on Power Systems, 30(6), 2015, pp.3139-3149.
6. Duarte, DP., Guaraldo, JC., Kagan, H., Nakata, BH., Pranskevicius, PC., Suematsu, AK., & Hoshina, MS. Substation-based self-healing system with advanced features for control and monitoring of distribution systems. In Harmonics and Quality of Power (ICHQP), 2016 17th International Conference on 2016, October, IEEE, pp. 301-305.
7. Ansari, B., Simoes, MG., Soroudi, A., & Keane, A. Restoration strategy in a self-healing distribution network with DG and flexible loads. In Environment and Electrical Engineering (EEEIC), 2016 IEEE 16th International Conference 2016, June, IEEE, pp. 1-5.
8. De Lemos, R., Giese, H., Muller, H.A., Shaw, M., Andersson, J., Litoiu, M., Schmerl, B., Tamura, G., Villegas, N.M., Vogel, T., et al.: Software engineering for self-adaptive systems: a second research roadmap. In: Software Engineering for Self-Adaptive Systems II, Springer, 2013, pp. 1–32.
9. Рубан І.В., Волк М.О., Рісухін М.В. Метод самовідновлення розподіленого програмного забезпечення в гетерогенних комп'ютерних системах. Телекомунікаційні та інформаційні технології. 2019. № 3 (64), с. 17-23/
10. Коломійцев О. В. Метод розрахунку розміру буферної пам'яті самовідновлювального сегмента телекомунікаційної мережі / Oleksii Kolomiitsev, Alnaeri Frhat Ali, Inna Petrovska // Системи управління, навігації та зв'язку. Збірник наукових праць. – Полтава: ПНТУ, 2021. – Т. 2 (64). – С. 144-147. – doi: https://doi.org/10.26906/SUNZ.2021.2.144.
11. Ткачов, В., Коваленко, А., Кучук, Г., & Ні, Я. (2021). Метод забезпечення живучості високомобільної комп'ютерної мережі. *Сучасні інформаційні системи*, *5*(2), 159–165. https://doi.org/10.20998/2522-9052.2021.2.24
12. Кучук, Н., Шефер, О., Чернева, Г., & Алнаері, Ф. А. (2021). Визначення пропускних здатностей самовідновлювального сегмента мережі. *Сучасні інформаційні системи*, *5*(2), 114–119. https://doi.org/10.20998/2522-9052.2021.2.16
13. Volk M.O., Klenov A.E., Shkruty D.A. Programs survivability method for distributed computing systems. Проблеми інформатизації: Матеріали п'ятої міжнародної науково-технічної конференції. Черкаси, Баку, Бельсько-Бяла, Полтава; 13–15 листопада 2017. С. 27.
14. Волк М. А., Клёнов А. Е. Исследование методов обеспечения живучести системного программного обеспечения. Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління. Матеріали восьмої міжнародної науково-технічної конференції. Полтава – Баку – Харків – Жиліна. 2018. С. 40.

Received (Надійшла) 08.06.2021
Accepted for publication (Прийнята до друку) 25.08.2021

The modified method of self-healing software systems using a memory dump

M. Volk, M. Gora, V. Demchuk, T. Olshanska, O. Tkalenko

Abstract. The article **subject** is the study of methods for self-healing of software systems. The **purpose** of the article is to improve the efficiency of self-healing tools for software systems by developing a modified method for self-healing software systems using a memory dump. The **methods** used are methods of decomposition of complex systems, methods of self-healing of software systems, methods of set theory. The following **results** were obtained: the model of the software components of the system and computing resources was modified, the method of self-recovery of software systems using a memory dump was modified. **Conclusions.** A modified method of self-recovery of distributed software of software systems using a memory dump has been developed, which, unlike the existing one, provides self-healing of the software system on remote resources in the state in which it was at the moment of failure of one of the resources, occurs in conditions of a decrease in the number of resources.

Keywords: software system, self-healing, allocated resources, memory dump.