УДК 004.7:004.415-025.25 DOI: 10.31673/2412-4338.2019.031723

Рубан І.В., Волк М.О., Рісухін М.В.
Харківський національний університет радіоелектроніки, Харків

МЕТОД САМОВІДНОВЛЕННЯ РОЗПОДІЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ В ГЕТЕРОГЕННИХ КОМП'ЮТЕРНИХ СИСТЕМАХ

Розглянуто сучасні рівні самовідновлення програмного забезпечення. Запропоновано моделі розподілених програмних та гетерогенних комп'ютерних систем, які враховують архітектуру програмних компонентів та гетерогенну природу сучасних обчислювальних ресурсів. Розроблено метод самовідновлення розподілених програмних систем, який дозволяє відновити працездатність програмних компонент в умовах гетерогенних комп'ютерних ресурсів.

Ключові слова: *самовідновлення, розподілене програмне забезпечення, гетерогенні комп'ютерні системи*

Ruban I.V., Volk M.O., Risukhin M.B.
Kharkiv National University of Radio Electronics, Kharkiv

SELF-HEALING METHOD FOR DISTRIBUTED SOFTWARE IN HETEROGENEOUS COMPUTER SYSTEMS

Modern software systems are created and executed in the conditions of continuous evolution of hardware platforms and operating systems. Most of them have a distributed structure, and the software components of the system are hosted on remote heterogeneous computer resources. At some moment one of the programs, computing resource, communications equipment or one of the available services fails. In some cases, there is a likelihood of external influence on the performance of the software components.

In software systems, the term "self-healing" implies the existence of any application, service, or system that may find that it is not working properly, and without any human intervention, makes the necessary changes to restore itself to normal or design state. The system can perform the same actions if its structural elements are likely to fail. The problem is to make a fault-tolerant system that is able to respond to software and hardware changes and self-repair after crashes or take appropriate action in the event of failure.

In in this article levels of software self-healing are considered. Models of distributed software and heterogeneous computer systems are proposed that take into account the architecture of software components and the heterogeneous nature of modern computing resources. A self- healing method for distributed software systems has been developed to restore the functionality of software components in the context of heterogeneous computer resources. The self-healing systems under study can be divided into three levels, depending on the type of resources we track and influence: application software, system software, and hardware level. The developed method allows working on all three levels.

Keywords: *self-healing, distributed software, heterogeneous computer systems*

Рубан И.В., Волк М.А., Рисухин М.В.
Харьковский национальный университет радиоэлектроники, Харьков

МЕТОД САМОВОССТАНОВЛЕНИЯ РАСПРЕДЕЛЕНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ В ГЕТЕРОГЕННЫХ КОМПЬЮТЕРНЫХ СИСТЕМАХ

Рассмотрены современные уровни самовосстановления программного обеспечения. Предложены модели распределенных программных и гетерогенных компьютерных систем,

учитывающие архитектуру программных компонентов и гетерогенную природу современных вычислительных ресурсов. Разработан метод самовосстановления распределенных программных систем, который позволяет восстановить работоспособность программных компонент в условиях гетерогенных компьютерных ресурсов.

Ключевые слова: *самовосстановления, распределенное программное обеспечение, гетерогенные компьютерные системы*

1.Вступ

Сучасні програмні системи створюються та функціонують в умовах постійної еволюції апаратних платформ та операційних систем. Більшість з них мають розподілену структуру, а програмні компоненти системи розміщуються на віддалених різнорідних комп'ютерних ресурсах. Рано чи пізно відбувається збій однієї з програм, обчислювального ресурсу, комунікаційного обладнання або один з наявних сервісів не зможе впоратися з підвищеним навантаженням. В окремих випадках є ймовірність зовнішнього впливу на працездатність програмних компонент. В усіх цих випадках постає проблема відновлення функціонування програмного забезпечення [1].

2. Постановка проблеми

У програмних системах термін «самовідновлення» передбачає наявність будь-якого додатка, служби або системи, яка може виявити, що вона не працює належним чином, і без будь-якого втручання людини внести необхідні зміни, щоб відновити себе в нормальному або проектному стані [1, 2]. Ці ж дії система може виконувати у випадку ймовірності відмови її структурних елементів [3]. Самовідновлення полягає в тому, щоб зробити систему здатною приймати рішення, постійно перевіряючи і оптимізуючи її стан і автоматично адаптуючись до мінливих умов. Проблема полягає в тому, щоб зробити відмовостійку систему, яка здатна реагувати на програмні та апаратні зміни і самовідновлюватися після збоїв або виконувати відповідні дії у випадку передбачення відмов.

3.Аналіз публікацій

Системи самовідновлення можна розділити на три рівні, в залежності від типу ресурсів, які ми відстежуємо і на які впливаємо: прикладного програмного забезпечення, системного програмного забезпечення та рівень обладнання [1-4].

Самовідновлення на рівні програмного додатку - це здатність окремого додатка, програмної системи або служби відновити працездатність зсередини [3,5]. У теорії програмування зазвичай проблеми реєструються за допомогою винятків і, в більшості випадків, реєстр використовується для подальшого аналізу. Коли виникає виключення, можливі різні дії: ігнорувати його і продовжити виконання, зупинити додаток, розробити код, який спробує виявити та виправити помилку та перезавантажити виконання задачі. Згодом з'явилися більш ефективні способи вирішення проблем всередині додатків, наприклад, шаблони проектування, які дозволяють створювати додатки і сервіси з внутрішнім самовідновленням [6].

Самовідновлення на системному рівні застосовується до всіх службових програм рівня операційної системи і додатків незалежно від їх внутрішніх компонентів. Це тип самовідновлення, який ми можемо розробити на рівні всієї комп'ютерної системи. Прикладом рішення системного рівня є перезапуск програмного процесу при його відмові.

Якщо відбулася відмова обладнання, самовідновлення полягає у знаходженні робочих вузлів, на які виконується перерозподіл програмних компонентів з встановленням нових мережних зв'язків. Як і на системному рівні, необхідно періодично перевіряти стан різних компонентів обладнання і діяти відповідно [4,6]. Особлива уваго до цих питань існує у розробників само-адаптивних систем [7].

Найбільш відомі методи самовідновлення програмного забезпечення застосовуються на одному з вказаних рівнях, що накладає обмеження на ефективність організації функціональної стійкості систем.

4. Мета дослідження

Мета дослідження полягає у створенні метода, на основі якого можливо розробляти засоби самовідновлення розподіленого програмного забезпечення з урахуванням особливостей усіх наведених рівнів. Це дозволить проектувати функціонально стійкі програмні системи, значно скоротити час відновлення функціонування таких систем після або в умовах ймовірних відмов.

5. Моделі розподілених програмних та комп'ютерних систем

На рис. 1 представлена узагальнена структура середовища виконання розподіленого програмного забезпечення. Програмний компонент *Pr*, який є елементом множини програмної системи, може бути представлений у вигляді двох структурних складових: коду (реалізує поведінку і є послідовністю виконуваних інструкцій процесора або віртуальної машини) (*Cd*), і даних (*Dt*), що відображають стан процесу в конкретний момент часу. Код програмних компонент, у свою чергу, можна поділити на три основні групи: обчислювальної прикладної задачі (Cd_u), моніторингу та контролю(Cd_m), збереження даних програми (Cd_s). Таким чином, програмна система може бути представлена моделлю:

$$PS = \bigcup_i Pr_i,\ i = \overline{1,N},\ Pr = \{ \bigcup Cd, Dt \} = \{ Cd_u, Cd_m, Cd_s, Dt \} \quad (1)$$

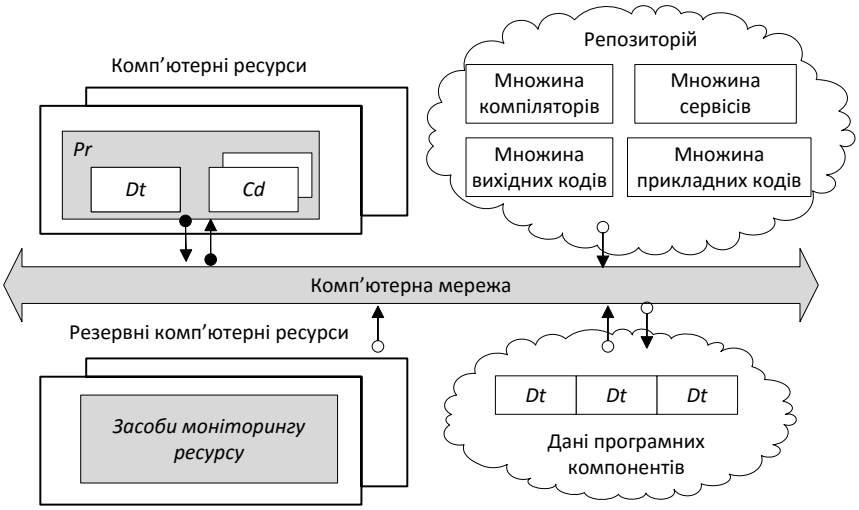

Рис.1. Узагальнена структура середовища виконання розподіленого програмного забезпечення

Відповідно до моделі (1), відновлення функціонування програмної системи містить два етапи: відновлення коду, тобто виконання програми, та відновлення даних, тобто стану програми у момент, який є найближчим до того стану, в якому програма була під час відмови.

Код програмних компонент Cd_u має відображення у множині прикладних кодів та множині вихідних кодів та може відновитися як дублюванням себе на резервному ресурсі, так і завантаженням з репозиторію. Відновлення можливо ініціювати за допомогою коду Cd_m, або, при відмові комп'ютерного ресурсу, засобами системи курування розподіленим обчислювальним процесом.

Особлива ситуація складається, коли немає комп'ютерного ресурсу, який задовольняє умовам виконання програмного компонента. Це можливо, коли резервні комп'ютерні компоненти мають іншу операційну систему, апаратну платформу та інше (згідно умові гетерогенності наявних ресурсів). У цьому випадку, за допомогою хмарних сервісів, знаходяться вихідні коди програмного компонента та належний компілятор для наявної платформи. Після компіляції, прикладний код передається на резервний ресурс на виконання.

Зміна стану програми призводить до зміни даних: $Dt \xrightarrow{Cd_u} Dt'$. Зміна стану моделі відбувається в певні моменти часу T_q, де q є цілим числом і може бути прив'язаний до системного (апаратного або програмного) дискретного таймеру. Таким чином, відновлення програмного забезпечення потребує перезапуск програмного компонента (тобто відновлення коду) та перезавантаження інформації (даних програми).

У роботі [8] введені дві активності, що реалізують запис стану програми одним з двох способів. Перший спосіб – збереження даних програми (дамп пам'яті). Цей спосіб дає надійний механізм повернення програмних компонентів в будь-який момент в минулому, для якого існує дамп пам'яті. Даному способу поставимо у відповідність підпрограму $Pr_d(t_q)$, що дозволяє програмі запам'ятати всі свої дані в момент часу t_q. Другий спосіб - це запам'ятовування історії зміни змінних програми. Якщо обсяг даних програми (сегмент даних) значно перевищує обсяг змінних значень, то можна запам'ятовувати адресу змінюваних даних і їх нове значення. Згодом утворюється ланцюжок таких змін. Підпрограма, що відповідає цьому способу – $Pr_c(Cd^{ad,sz}, t_q)$, де атрибути *ad* і *sz* визначають адресу і розмір змінюваних даних відповідно в момент часу t_q.

Зіставимо кожну з цих підпрограм (1) зворотнім, тобто таким, які повертають програму в один зі станів в минулому: $Pr'_d(t_q)$ і $Pr'_c(Cd^{ad,sz}\ t_q)$. Фактично, зворотна підпрограма Pr' реалізує операцію $Dt' \xrightarrow{Cd_m} Dt$, повертаючи програму в один з попередніх станів. Наявність залежності від дискретного часу дозволяє повертати програму в будь-який момент в минулому.

Будь-який з цих двох способів є достатнім, щоб зберегти інформацію про стани програми під час часу виконання. Однак кожен з них має свої переваги і недоліки. Перший спосіб, зазвичай, веде до значних часових та ресурсних витрат, бо потрібен значний процесорний час на копіювання даних і лінійно зростаючий обсяг пам'яті для зберігання всіх станів програми протягом часу виконання. Другий, хоч і позбавлений значних ресурсних витрат за пам'ятю при збереженні, вимагає підвищених обчислювальних витрат у ході реалізації відкату програми у часі. Для виконання повернення програми у часі, необхідно виконати ланцюжок подій в зворотному порядку. Те, що зміна даних програми відбувається неодноразово, значно збільшує час виконання цього процесу.

Ресурси у комп'ютерній системі також утворюють множину $R = \{ R_j \}, i = \overline{1,M}$, де j – номер обчислювального ресурсу, а N – число ресурсів у комп'ютерній системі. Будь-який обчислювальний ресурс, залучений до обчислювальної системи, так само, як і завдання, описується рядом характеристик, які представлені кортежем (2):

$$R_j = \{ ap_j, os_j, ms_j \},\ j = \overline{1,M}\ , \tag{2}$$

де *ap* – архітектура процесора, *os* – операційна система, ms – об'єм оперативної пам'яті. В залежності від типу завдань, ця множина може бути розширена [9]. Наявність перших двох елементів множини (2) підкреслює гетерогенність комп'ютерних систем.

Множина ресурсів R_j в плині часу динамічно змінюється (видалення, додавання обчислювального ресурсу в систему, зміна характеристик ресурсу). У ролі ресурсів можуть виступати як кластери, так і окремі робочі станції, які надають користувачеві доступ до оперативної і віртуальної пам'яті, дискового простору, а також процесорам обчислювального

ресурсу.

6. Метод самовідновлення розподіленого програмного забезпечення

За основу методологічного підходу до відновлення програмного забезпечення на підставі збереження даних (стану) програми [10]. Нижче наведені етапи розробленого методу самовідновлення розподіленого програмного забезпечення в гетерогенних комп'ютерних системах.

1. Реалізація розподілення програмних компонент за комп'ютерними ресурсами з урахуванням гетерогенних умов комп'ютерних компонентів (2). Завантаження згідно схем призначення програмних елементів на виділені ресурси. Запуск програмних компонент системи, збереження початкового стану розподіленої програмної системи шляхом виконання $Pr(t_0)$ для усіх N програм (q=0).

2. Визначення підмножини вільних ресурсів після реалізації схеми призначення (пункт 1):

$$R^e = \bigcup_{m=1}^{M} R_m \setminus \bigcup_{i=1}^{N} R_r \Big| r = j, \forall\{Pr_i \to R_j\} \quad (3)$$

У разі, коли кількість обчислювальних ресурсів менша або дорівнює кількості програм в завданні, вираз (3) може дати результат $R^e = \varnothing$. В цьому випадку можна прийняти $R^e = \bigcup_{m=1}^{M} R_m$. Можлива також модифікація методу, в якому підмножина упорядковується на основі завантаженості обчислювальних ресурсів.

3. Визначити значення часу t_q, q=q+1, для якого буде виконаний дамп пам'яті програмних компонентів. Перевірити умову закінчення обчислювального процесу. Якщо умова виконана, перехід до пункту 8.

4. При досягненні часу t_q виконати всі підпрограми множин $\bigcup_{i=1}^{N} Cd_u(t_q)$, $\bigcup_{i=1}^{N} Cd_m(t_q)$, $\bigcup_{i=1}^{N} Cd_s(t_q)$. Якщо всі підпрограми виконалися, переходимо до пункту 3. Якщо якась із підпрограм не виконалась (відповідь від неї не одержано), або ініційована процедура самовідновлення (програмним компонентом Cd_m), переходимо до пункту 5.

5. Визначаємо підмножину програмних компонентів завдання, від яких не отримано відповіді (не виконано дамп) $P^{Err} \subset P$.

6. Для підмножини P^{Err} виконати перерозподіл ресурсів з отриманням нової схеми призначення: $Sh^{Err} = \{ P^{Err} \to R^e \}$.

7. Реалізація алгоритму самовідновлення за наступними умовами:

- якщо ресурс ще працює, програмний компонент *Pr* може бути переданий на резервний ресурс без змін;

- якщо ресурс недосяжний та є аналогічний резервний ресурс, програмний компонент *Pr* завантажується з множини прикладних кодів;

- якщо ресурс недосяжний та немає аналогічного резервного ресурсу, але є ресурс з іншою апаратно-програмною платформою, виконується вибірка програмного компонента з множини вихідних кодів, компіляція програми та передача отриманого програмного компонента на резервний ресурс.

Пересилання виконується згідно схеми призначення програмних елементів на виділені ресурси Sh^{Err}. Виконується запуск програмних компонентів, які встановлюють свій статус

розподіленої програмної системи шляхом виконання *Pr'* для всіх програм (q=0) з множини P^{Err}. Перехід до пункту 5 даного методу.

8. Завершення обчислювального процесу, закриття служб підтримки самовідновлення програмного забезпечення.

Цей метод гарантує повне ссамовідновлення функціонування програмного забезпечення, але потребує час для здійснення перезавантаження та пошук і відновлення даних пам'яті програмних компонентів за допомогою дампів пам'яті, а також для компіляції вихідник кодів для нових платформ. У цьому випадку програми системи, або її окремі компоненти, простоюють. Остання властивість може бути подолана за допомогою використання мажоритарного методу забезпечення функціональної стійкості розподіленого обчислювального процесу, але потребує додаткових обчислювальних ресурсів.

7. Висновки

Основною відмінністю розробленого метода самовідновлення розподіленого програмного забезпечення в гетерогенних комп'ютерних системах є можливість відновлення функціонування програмної системи незалежно від наявних апаратних та програмних ресурсів. При цьому наявність цих ресурсів може мати еволюційний характер та враховує появу нових програмних технологій та платформ виконання програм реального часу.

Напрямком подальших досліджень у сфері самовідновлення програмного забезпечення може бути розробка індивідуальних алгоритмів для різних типів апаратних платформ та віртуальних машин. Особливу увагу треба приділити хмарним та туманним обчисленням, які є природним середовищем для рішення завдань даної області.

Список використаної літератури

1. Schneider, C., Barker, A., & Dobson, S. A survey of self- healing systems frameworks. Software: Practice and Experience, 45(10), 2015. pp. 1375-1398.

2. 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.

3. 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.

4. 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.

5. 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.

6. 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.

7. 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.

8. Волк М. А. Журнализация состояний программных распределенных мооделей и ее использование в оптимистических алгоритмах синхронизации. Збірник наукових праць Харківського університету Повітряних Сил. 2010, випуск 1 (23). С.104–107.

9. Filimonchuk T., Volk M., Ruban I., Tkachov V. Development of information technology of tasks distribution for grid-systems using the GRASS simulation environment. Eastern-European

Journal of Enterprise Technologies. Information and controlling system, 2016. Vol. 3/9 (81). pp. 45–53.

10. Ruban I, Filimonchuk T, Ivanisenko I, Risukhin M, Romanenkov Y. The Method for Ensuring the Survivability of Distributed Computing in Heterogeneous Computer Systems. 5th International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T), Kharkiv, Ukraine, October 9-12, 2018, pp. 233-238.

References

1. Schneider, C., Barker, A., and Dobson, S. (2015) "A survey of self- healing systems frameworks." *Software: Practice and Experience*, 45(10): 1375-1398. Print.

2. Manzoor A., Rajput U, Phulpoto N, Abbas F, Rajput M. (2018) "Self-healing in Operating Systems." *IJCSNS International Journal of Computer Science and Network Security,* Vol.18 No.5: 92-98. Print.

3. Hudaib, AA., Fakhouri, HN., Al Adwan, FE., and Fakhouri, SN. (2017) "A Survey about Self-Healing Systems" (*Desktop and Web Application*). Vol.09 No.01: 71-88. Print.

4. Wang, Z., & Wang, J. (2015) "Self-healing resilient distribution systems based on sectionalization into microgrids." *IEEE Transactions on Power Systems*, 30(6): 3139-3149 Print

5 Duarte, DP., Guaraldo, JC., Kagan, H., Nakata, BH., Pranskevicius, PC., Suematsu, AK., and Hoshina, MS. (2016) "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, IEEE*: 301-305. Print.

6. Ansari, B., Simoes, MG., Soroudi, A., and Keane, A. (2016) "Restoration strategy in a self-healing distribution network with DG and flexible loads." *In Environment and Electrical Engineering (EEEIC), 2016 IEEE 16th International Conference*: 1-5. Print.

7. 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. (2013) "Software engineering for self-adaptive systems: a second research roadmap." *In: Software Engineering for Self-Adaptive Systems II*: 1–32 Print

8. Volk M. O. (2010) "The logging of the state of software distributed models and its use in optimistic synchronization algorithms." *Proceedings of Kharkiv University of the Air Force,* Vol. 1 (23): 104–107. Print.

9. Filimonchuk T., Volk M., Ruban I., and Tkachov V. (2016) "Development of information technology of tasks distribution for grid-systems using the GRASS simulation environment." *Eastern-European Journal of Enterprise Technologies. Information and controlling system*, Vol. 3/9 (81): 5–53. Print.

10. Ruban I, Filimonchuk T, Ivanisenko I, Risukhin M, and Romanenkov Y. (2018) "The Method for Ensuring the Survivability of Distributed Computing in Heterogeneous Computer Systems." *5th International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T), Kharkiv, Ukraine*: 233-238. Print.

Автори статті (Authors of the article)

Рубан Ігор Вікторович – д.т.н., професор, перший проректор Харківського національного університету радіоелектроніки (Ihor Ruban – Doctor of Science (Technic), Professor, First Vice-Rector). Phone: +380 (66) 188 18 85. E-mail: ihor.ruban@nure.ua

Волк Максим Олександрович – д.т.н., доцент, професор кафедри Електронних обчислювальних машин (Maksym Volk – Doctor of Science (Technic), Docent, Professor of Computer Engineering Department). Phone: +380(50) 342 42 90, E-mail: maksym.volk@nure.ua

Рісухін Максим Володимирович – аспірант кафедри Електронних обчислювальних машин (Maksym Risukhin – postgraduate student of Computer Engineering Department). Phone: +38 (067) 578 03 31, E-mail: risuhin.max@gmail.com