Сучасний стан наукових досліджень та технологій в промисловості. 2024. №1 (27)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

УДК 004.75:004.453.3
DOI: https://doi.org/10.30837/ITSSI.2024.27.005

М. Волк, М. Гора

МОДИФІКОВАНИЙ МЕТОД САМОВІДНОВЛЕННЯ РОЗПОДІЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ В ГЕТЕРОГЕННИХ КОМП'ЮТЕРНИХ СИСТЕМАХ

Об'єктом дослідження є розподілений обчислювальний процес у гетерогенних комп'ютерних системах. **Предметом** – методи самовідновлення розподіленого програмного забезпечення в гетерогенних комп'ютерних системах. **Мета роботи** – підвищення ефективності систем розподіленого оброблення даних із підтримкою функціональної стійкості обчислювального процесу з допомогою розроблення модифікованого методу самовідновлення розподіленого програмного забезпечення. **Завдання:** дослідити наявні методи відновлення розподіленого обчислювального процесу, зробити висновки про їх переваги та недоліки; на основі математичних моделей завдань, обчислювальних ресурсів і наявних методів розподілу ресурсів розробити модифікацію методу самовідновлення розподіленого програмного забезпечення, беручи до уваги стратегії управління, пошук найкращого рішення для обраних критеріїв, зменшення енергоспоживання під час виконання завдань; провести низку експериментів з порівняння розробленого методу з наявними. **Методи дослідження** ґрунтуються на використанні теорії множин, загальної теорії систем і теорії імітаційного моделювання. **Результати експериментів**, досягнуті в умовах моделювання розподілу програмних завдань на обчислювальні ресурси в імітаційному середовищі моделювання та симуляції обчислювального процесу під час самовідновлення в разі відмов ресурсів, підтверджують ефективність запропонованого методу. Відповідно до результатів дослідження можна зробити **висновки** про те, що застосування методу в системах управління розподіленими обчисленнями не збільшує час, який система витрачає на виконання завдання за відсутності відмов, водночас за наявності відмов дає змогу швидше відновити функціональність програмного завдання та зменшує час виконання на 8–17 %, а енергоспоживання на 7–12 %. Також спостерігається зростання ефективності зі збільшенням розмірності завдань та ймовірності відмов. Напрямами майбутніх досліджень можна визначити розроблення технологій автоматизованого або автоматичного використання методів розподілу ресурсів і самовідновлення.

Ключові слова: методи самовідновлення; програмне забезпечення; розподілені обчислення; комп'ютерні системи; хмарні архітектури; програмні агенти.

Вступ

Сучасні комп'ютерні системи, що підтримують самовідновлення в аварійних ситуаціях, зараз стають усе більш популярними. Це зумовлено тим, що організації та підприємства все частіше впроваджують автоматичні технології та засоби для своєї роботи. У цьому разі бажано мінімізувати час використання та простою техніки та зменшити елементи втручання людини в сам обчислювальний процес. Одним із механізмів, які підтримують функціонування таких систем, є самовідновлення програмного забезпечення, що підвищує надійність і продуктивність обчислень.

Наявні методи самовідновлення основані на резервуванні надлишкових ресурсів, контролі та відтворенні програм і даних. Вони гарантують працездатність системи та повернення її до робочого стану після будь-яких відмов. У разі виходу з ладу програмного компонента засоби самовідновлення можуть автоматично перенаправляти трафік на інші програми в мережі для підтримки продовження обслуговування користувачів. Водночас система сама знаходить новий обчислювальний ресурс і відновлює функціонування програмної системи.

Нині відомо декілька підходів щодо відновлення програмних систем після збоїв. Вони часто використовують такі методи [1–3]:

– перезавантаження, зокрема перезапуск усього програмного забезпечення;

– оновлення програмного забезпечення;

– мікроперезавантаження, що повертають систему до робочого стану до або після виявлення несправності.

Перезапуск програмного забезпечення зазвичай потребує значного часу, що призводить до простою програмної системи, за умови якого система не відповідає на запити та створюється ефект зависання системи загалом.

Оновлення програмного забезпечення використовує динамічне ресурсне середовище та може передбачати

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2024. No. 1 (27)

елементи зміни програми для врахування причин збоїв у нормальній роботі.

Мікроперезавантаження зазвичай працюють швидше та призводять до перезапуску окремих компонентів програмної системи. Такий підхід вимагає аналізу стану програми в момент відмови для розуміння причин відмови.

Основна проблема сучасних методів відновлення функціональної стійкості полягає в значній вартості цього процесу, яка зростає зі збільшенням вимог до часу відновлення. Крім цього, методи, зорієнтовані на зменшення часу відновлення, зазвичай не беруть до уваги обмеження на енергоспоживання, вартість та ефективне використання ресурсів.

Аналіз останніх досліджень і публікацій

Методи відкату до контрольної точки [4] використовуються подібно до перезапуску програми, але водночас забезпечують менший час перезапуску, тому що він виконується з контрольної точки відновлення. У разі такого використання все ще не обробляються окремі помилки, оскільки вони все одно можуть виникати в подальшому.

Існують інші варіанти застосування точки відновлення в поєднанні з виконанням декількох версій програми [5], які враховують детерміновані помилки, якщо відмови відбуваються незалежно одна від одної. Однак вони визначаються значними витратами для програмних систем щодо розроблення, моніторингу, резервування надлишкових ресурсів та виконання кількох версій однієї програми одночасно на різних ресурсах.

Методи автоматичної генерації сигнатур для систем динамічного виявлення та захисту від вразливостей вторгнень [6, 7] фільтрують вхідні дані з метою відсіювання атак. Ключовою проблемою є те, що такі сигнатури можуть спричиняти помилкові спрацьовування, особливо в разі поліморфних атак. Показано, що така атака надто різноманітна, щоб її можна було виявляти за допомогою сигнатур [8].

Наступний проєкт, що застосовує механізм перезапуску з точки відновлення, – це *Rx* [9], який містить механізми зміни середовища виконання завдання заради мети відновлення після помилок. Але в роботі [10] показано, що значна кількість помилок програм не залежить від середовища виконання, а самі помилки є повністю детермінованими й можуть повторюватися. У такому разі відновлення будуть успішними лише за умови використання методів, які закладені в саму програму та реагують на сутність помилки.

Зміна середовища в *Rx* виявилась неефективною через поліморфну поведінку шкідливих запитів [11]. Крім того, *Rx* намагається замаскувати несправності для клієнта, але використовує проксі-сервер застосунку з підтримкою протоколів, що мають бути здатними фільтрувати специфічну інформацію (зокрема часові мітки), яка може заплутати клієнтську програму. Також *Rx* не вирішує проблеми узгодженості за умови встановлення контрольних точок і перезапуску програм, що містять кілька процесів.

Технологія використання "прибиральника" (*Sweeper*) [12] комбінує механізми перезапуску контрольної точки *Rx* і проксі зі специфічними для вразливостей фільтрами виконання (*VSEF*). У разі помилки *Sweeper* аналізує "хибні дані", щоб визначити інформацію на вході, яка призвела до збою, генерує вхідний фільтр для виявлення подібних майбутніх інформаційних запитів, а потім повертає систему до попередньої контрольної точки та поновлює отримання вхідних даних.

Існують підходи до організації обчислень, орієнтовані на "прийнятність" (*acceptability*) [13]. Вони просувають ідею про те, що поточні зусилля з розроблення програм можуть бути неправильно виконаними. Такий підхід ґрунтується на тому факті, що деякі частини програми можуть бути заблокованими без негативного впливу на загальну функціональність системи.

Комп’ютерні обчислення без збоїв [14] є спекулятивною технікою відновлення, що основана на використанні компіляторів реального часу виконання та вимагає ін’єкцію в програмний код для роботи із записами в оперативній пам’яті з допомогою додаткового віртуального буфера. Такий процес забезпечує більш надійну реакцію на помилки, незважаючи на значні накладні витрати на продуктивність, які можуть досягати від 80 % до 500 % для різних програм.

Ще одною спекулятивною технікою відновлення є вибіркова трансакційна емуляція, яка використовується в програмних імунних системах [15]. Вона виявляє функцію, у якій відбулася помилка, а потім частково симулює процедуру реалізації цієї функції, можливо, в більшому діапазоні даних, щоб отримати значення помилок. Підхід застосовує віртуалізацію помилок, щоб визначити евристичне значення помилки в програмній процедурі, у якій вона виникла.

Сучасний стан наукових досліджень та технологій в промисловості. 2024. №1 (27)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

Детальний, формалізований підхід щодо відновлення розподіленого програмного забезпечення наведено в роботах [16, 17], де з використанням моделей програмних компонентів виконується управління станом програмних компонентів у часі за допомогою дампів пам'яті або журналізації зміни даних. Застосування цього підходу до завдань нашого дослідження дасть змогу організувати самовідновлення на низькому програмному рівні (рівні операційної системи, віртуальної машини тощо).

Унаслідок аналізу можна зробити висновок, що жоден із цих методів не є оптимальним. Ці методи зазвичай неефективно долають потік помилок, збоїв та відмов, що призводить до значних витрат на відновлення.

Мета й завдання роботи

Метою роботи є підвищення ефективності підтримки функціональної стійкості гетерогенних комп'ютерних систем завдяки розробленню та впровадженню модифікованого методу самовідновлення програмного забезпечення. Для досягнення мети в роботі виконуються такі завдання: розглядається модель розподіленого обчислювального процесу (РОП) та структура обчислювального середовища виконання; формалізуються стратегії управління РОП; пропонується метод самовідновлення розподіленого програмного забезпечення в гетерогенних комп'ютерних системах та аналізуються результати досліджень розробленого методу.

Модель системи управління розподіленим обчислювальним процесом і стратегії управління

Система управління обчислювальним процесом зазвичай є децентралізованою та має різнорідні компоненти. Пакети завдань та середовище виконання містять безліч програм і комп'ютерних елементів, на яких ці програми виконуються. Методи розподілу, моніторингу, підтримки функціональної стійкості – це окремі програми або програмні системи, якими керує система управління для виконання визначених цілей (рис. 1). Зеленим кольором виділено програмні модулі та методи забезпечення, що впроваджені в процес самовідновлення РОП та потребують зміни для фінальної реалізації системи управління.

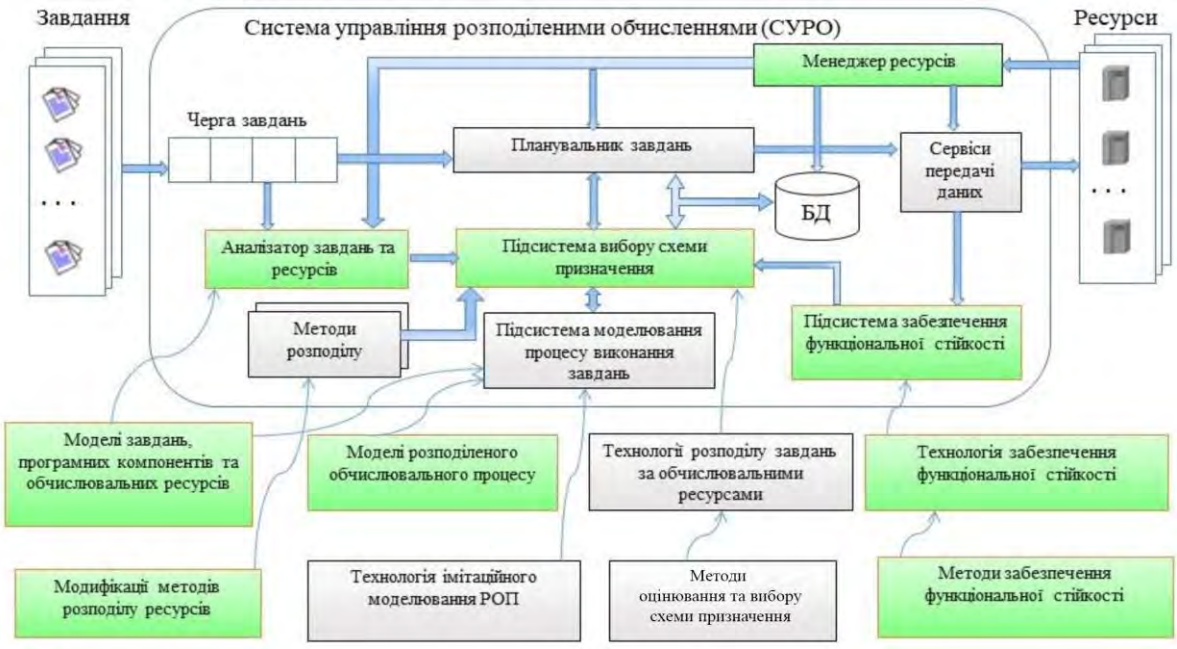


Рис. 1. Структура системи управління розподіленими обчисленнями

За основу подання системи управління розподіленим обчислювальним процесом застосуємо моделі, запропоновані в роботі [18]:

$$\Omega = < Z, R, Mg, Sh, Q, Sy, Ms > , \quad (1)$$

$$O = \left\{ \overline{T_j}, \overline{TR_j}, \overline{TP_j}, \overline{Vz_j}, \overline{Yz_j}, \overline{E_j C_j} \right\}, \ j = \overline{1, N} \ , \quad (2)$$

де Ω – множина елементів системи управління РОП, яка містить такі компоненти: Z – множина

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2024. No. 1 (27)

програмних пакетів завдань, що виконує система; R – множина наявних обчислювальних ресурсів; Mg – сервіси моніторингу системи; Sh – множина схем призначення завдань за ресурсами; Q – множина методів розподілу ресурсів; Sy – сервіси синхронізації; Ms – множина методів самовідновлення; O – множина параметрів, що в кількісному вигляді позначають критерії ефективності РОП: T – час виконання завдання; TR і TP – час роботи та простою ресурсів; Vz – обсяги оперативної пам'яті, яку використовують завдання; Yz – характеристики зв'язності програм у завданні; E – енергоспоживання в процесі виконання завдання; C – відносна вартість виконання завдання; N – загальна кількість завдань.

Вибір сукупності оцінювальних параметрів та елементів системи керування залежить від стратегії управління РОП. Наведемо приклади деяких найбільш поширених стратегій керування та формалізуємо їх.

Стратегія 1. Мінімізувати час виконання завдання за умови обмежень на обсяги пам'яті та час простою ресурсів:

$$St_1 = \begin{cases} TR = f\left(Sh\left(Q,Z,R\right),Sy,Ms\right) \to \min \\ TP \in \left[0, TP_{\max}\right] \\ V \in \left[0, V_{\max}\right] \end{cases} \quad (3)$$

Перша стратегія передбачає використання функції f на просторі об'єктів, до яких належать схеми розподілу Sh, отримані за допомогою методів розподілу ресурсів Q на основі моделей завдання Z та обчислювальних ресурсів R, методів синхронізації програм Sy та методів підтримки функціональної стійкості Ms. Мета функції – задіяти перелічені об'єкти щодо досягнення мінімуму часу виконання завдання. Отримані конфігурації розподілу завдань за ресурсами перевіряються на входження в границі обмежень відповідно до простою та обсягів пам'яті ресурсів, а якщо можна, і мінімізувати їх.

Стратегія 2. Мінімізувати час виконання завдання за умови наявності безлічі методів розподілу та засобів забезпечення функціональної стійкості обчислювального процесу:

$$St_2 = \begin{cases} TR = f\left(Sh\left(Q,Z,R\right),Sy,Ms\right) \to \min \\ Z = const, R = const \\ Q \neq const \end{cases} \quad (4)$$

Стратегія передбачає використання функції f на просторі тих самих об'єктів (1, 2), але за умови незмінності обчислювальних завдань та комп'ютерних ресурсів і наявності множини методів розподілу ресурсів, вибір з яких неоднозначний.

Стратегія 3. Мінімізація вартості виконання завдання на безлічі доступних ресурсів:

$$St_3 = \begin{cases} C = f\left(Sh\left(Q,Z,R\right),Sy,Ms\right) \to \min \\ Q \neq \text{const}, R \neq \text{const} \\ Z = \text{const} \end{cases} \quad (5)$$

Стратегія передбачає використання функції f на просторі тих самих об'єктів (1, 2), але метою функції є мінімізація вартості процесу обчислень.

Стратегія 4. Мінімізація часу відновлення обчислювального процесу:

$$St_4 = \begin{cases} T_{Ms} = f\left(Sh\left(Q,Z,R\right),Sy,Ms\right) \to \min \\ Z = \text{const}; R \neq \text{const} \\ Q \neq \text{const}; Sh \neq \text{const}; Ms \neq \text{const}; Sy \neq \text{const} \end{cases} \quad (4)$$

Ця стратегія важлива для критичних обчислень реального часу, коли необхідно якнайшвидше відновити роботу комп'ютерної системи. Але її реалізація зазвичай призводить до дорогих, затратних рішень.

Стратегія 5. Мінімізувати кількість обчислювальних ресурсів з огляду на обмеження на пам'ять та можливість вибору методу синхронізації:

$$St_5 = \begin{cases} |P| = f\left(Z,R,Sh,Q,V,Sy,Ms\right) \to \min \\ Z = \text{const}; R = \text{const}; TP \to \min; V \to \min \\ Ms \neq \text{const}; Sy \neq \text{const} \end{cases} \quad (5)$$

Зазначена стратегія передбачає використання функції f на просторі об'єктів, до яких належать схеми розподілу Sh, методи розподілу ресурсів Q, моделі завдань Z та обчислювальних ресурсів R, методів підтримки функціональної стійкості Ms. Мета функції – задіяти перелічені об'єкти для досягнення мінімуму потужності множини ресурсів R, які буде використано. Якщо існує декілька рішень, система намагатиметься мінімізувати час простою та обсяги фізичної пам'яті.

Важливо зауважити, що реалізація функції f будується для кожної стратегії окремо, може бути математичною, евристичною або програмною та містити деякі спільні алгоритми обчислень. Імовірно, що стратегій виконання більше, тоді їх реалізація може бути виконана як засобами адміністрування (в автоматичному, автоматизованому або ручному), так і з допомогою розроблення додаткового програмного забезпечення. Одним

Сучасний стан наукових досліджень та технологій в промисловості. 2024. №1 (27)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

із способів реалізації є створення скриптів або використання *XML*-подібної мови розмітки.

Запропонуємо ще одну стратегію, мета якої – мінімізувати енергоспоживання на виконання завдання:

$$St_E = \begin{cases} E = f\left(Z, R, Sh, Q, Ms\right) \to \min; \\ Z = \text{const}; R \neq \text{const}; \\ TP \to \min; Ms \neq \text{const} \end{cases} \quad (6)$$

Стратегія передбачає використання функції f на просторі об'єктів, до яких належать схеми розподілу Sh, методи розподілу ресурсів Q, моделі завдань Z та обчислювальних ресурсів R, методів підтримки функціональної стійкості Ms. Мета функції – задіяти перелічені об'єкти для досягнення мінімуму споживання енергії ресурсами з множини R, які буде застосовано для виконання завдання Z.

У процесі розподілу обчислювальних ресурсів у запропонованому методі використовується ідея методу зграї сальп (англ. *Salp Swarm Algorithm, SSA*), запропонованого в статті [19]. Зазначений метод виконує симуляцію зграйної поведінки сальп під час переміщення в океані та добування їжі. Ключова особливість поведінки сальп, що становить інтерес для цієї роботи, – вибудовування зграї як ланцюжка для досягнення цільової функції (у сальп – це їжа, в роботі – агент). Основні кроки алгоритму, який застосовується, зображено на рис. 2.

Рис. 2. Основні кроки алгоритму розподілу обчислювальних ресурсів

Алгоритм починає процес розподілу з розміщення в просторі агентів випадковим чином. Для кожного агента обчислюється цільова функція, обирається агент із кращим значенням, і положення агента

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2024. No. 1 (27)

позначається як "джерело їжі", за яким полюватиме ланцюжок агентів. За ресурсом, де розташований цей агент, закріплюється основна програма.

Для кожної зграї (завдання) оновлюється позиція основної та підпорядкованих програм. Положення основної програми оновлюється, коли ланцюжок з програм завдання знаходить положення з найкращим значенням цільової функції. Ланцюжок має тенденцію руху в бік глобального оптимуму, що може змінювати своє положення під час оптимізації. На етапі корегування положення програм можлива перевірка встановлених обмежень.

Переваги алгоритму [20]:

– алгоритм зберігає найкраще поточне значення, що не буде втрачено навіть у разі виродження всієї популяції;

– алгоритм оновлює положення провідної програми щодо агента, який відповідає кращому знайденому на поточну мить рішенню; у такий спосіб основна програма здійснює розвідку та пошук;

– алгоритм оновлює положення підпорядкованих програм одна щодо одної, тому вони поступово зміщуються в бік основної програми;

– поступове переміщення підпорядкованих програм запобігає зупиненню алгоритму в точці локального оптимуму;

– алгоритм простий у реалізації та застосуванні.

Переваги алгоритму дають змогу вирішити оптимізаційну задачу як з визначеним, так і невідомим простором пошуку. Алгоритм є адаптивним, що дозволяє уникнути локальні рішення й досягти глобального оптимуму.

У разі отримання декількох рішень система спрямована на мінімізацію часу простою, що зменшує енергоспоживання.

У розробленому методі сальп має бути програмою, що шукає кращий ресурс, але програма не може зробити це самостійно. Тому необхідно розглядати сальп як комбінацію статичного агента, що отримає інформацію про обчислювальний ресурс, і програму, прив'язану до агента. На етапі планування програма може бути прив'язана до агента віртуально, на рівні логічного зіставлення або схеми призначення *Sh*. А після завершення алгоритму планування програма передається агенту на виконання.

Отже, агент виконує чотири функції:

а) збір та передача інформації про обчислювальний ресурс;

б) отримання та розгортання програмного компонента на виконання;

в) підтримка зв'язку між програмними компонентами та системою управління;

г) виявлення відмови програмного компонента та підтримка функціональної стійкості.

Розглянемо етапи метаевристичного методу розподілення завдань за обчислювальними ресурсами в умовах динамічної зміни ресурсів та підтримки функціональної стійкості.

Eman 1. Отримання пулу завдань $Z = \bigcup_{i=1}^{N} P_i$, де N – загальна кількість програм у завданні та нормованих характеристик програм у завданнях $X^P =< E_i^P, T_i^P, V_i^P, Y_i^P, \rho_i^P >$, вибір цільової функції $f\left(x^p\right)$, де $x^p \in X^p$, та одного або декількох обмежувальних параметрів $X_{\lim}^P \subset X^P$.

Eman 2. Отримання доступного пулу ресурсів $R = \left\{R_j\right\}$, $j = \overline{1, M}$, та якісної інформації про ресурси та нормовані характеристики ресурсів X^R. За відсутності можливості отримати характеристики, необхідно доручити це агентам на наступних етапах.

Eman 3. Розсилання агентів $\mathrm{A} = \left\{A_j^{R_j}\right\}$, $j = \overline{1, M}$ на ресурси R. Атрибут R_j агента $A_j^{R_j}$ підкреслює гетерогенність комп'ютерного середовища, коли ресурсу потрібен оригінальний агент. Якщо ресурси однорідні, достатньо одного типу агента – A_j^R або A_j.

Eman 4. Перевірка умови наявності достатньої кількості ресурсів. Умовою може бути кількість агентів, що відповіли, час, що минув з початку розсилання агента. Якщо на етапі 2 було отримано значення характеристик ресурсів, умовою початку алгоритму розподілу може бути $\sum_{i=1}^{N} x_i^P \geq \sum_{j}^{M} x_j^R$, що означає наявність достатньої кількості ресурсів для виконання пулу завдань.

Eman 5. Створення випадкової схеми розподілення ресурсів $Sh = \left\{P \xrightarrow{\theta} R\right\}$. Випадкова функція θ може бути реалізована таким чином. У разі співвідношення потужності множин $|P| \leq |R|$ один програмний компонент з множини P закріплюється за вільним ресурсом з множини P випадковим способом. Найбільш простим та швидким буде виконання алгоритму $P_i \longrightarrow R_i$, $N \leq M$. За умови $|P| > |R|$ можна дочекатися появи

ISSN 2522-9818 (print)
Сучасний стан наукових досліджень та технологій в промисловості. 2024. №1 (27) ISSN 2524-2296 (online)

додаткових ресурсів або розподілити залишкові програмні компоненти рівномірно на наявні ресурси з обмеженнями $X_{\lim}^{P}$). Сортування одночасно векторів P та R за зростанням значень відношень $f\left(x_i^P\right)/f\left(x_i^R\right)$. Унаслідок розподілу можна виділити дві підмножини ресурсів: R' – задіяних у розподілі та R'' – незадіяних у розподілі ресурсів. Тобто більш коректна схема призначення відображає $Sh=\left\{P\xrightarrow{\theta}R'\right\}$. Формування матриць Φ, Φ'. На першій множині P після сортування розташовуватиметься основна програма, яка знайшла найбільш потужний ресурс за критерієм оптимізації.

Необхідно відокремити паралельний процес пошуку нових ресурсів, що паралельно з основним методом буде виконувати дії, аналогічні наведеним в етапах 2 і 3, з розширенням множин R та X^R і матриць Φ, Φ'.

Eman 6. Оновлення підмножини ресурсів $R''\subset R, R''\cap Sh=\varnothing$, які не задіяні в розподілі та для яких визначені параметри цільової функції. Якщо $R''=\varnothing$, то відбувається перехід до етапу 8. Для всіх ресурсів із множини R'' необхідно визначити значення цільової функції $f\left(x_i^{R''}\right)$, $i=\overline{1,|R''|}$. Знайти ресурс, що надає мінімальне значення відношення цільової функції для основної програми до цільової функції ресурсу $\min\left(f\left(x_1^P\right)/f\left(x_i^{R''}\right)\right)$, $i=\overline{1,|R''|}$, запам'ятовування ідентифікатора отриманого ресурсу i'. Якщо отримане мінімальне значення перевищує або дорівнює аналогічному відношенню для ресурсу, на якому зараз розташована основна програма, відбувається перехід до етапу 8.

Eman 7. Перевірка виконання обмежень параметрів множини $X_{i'}^{P_1}$. Якщо обмеження не виконуються, вилучення з множини R'' ресурсу $R''_{i'}$: $R''=R''-R''_{i'}$ без можливості долучення в розподіл знову. Долучення можливе за умови зміни обмежувальних характеристик. Якщо вилучення відбулося, здійснюється перехід до етапу 6. Якщо вичерпано час, відведений на пошук ресурсів, можна переходити до етапу 8.

Eman 8. Зміна прив'язки програм до ресурсів із формуванням нової схеми призначення $Sh=\left\{P_1\longrightarrow R_{i'}\cup P_i\longrightarrow R'_{i-1}\right\}$, $i=\overline{2,N}$. Корегування множини задіяних ресурсів $R'=R'-R'_{|R'|}+R'_{i'}$. Корегування матриць відповідності програм до ресурсів. Перехід до етапу 6.

Eman 9. Ущільнення розподілу ресурсів в умовах розміщення кількох програм на одному ресурсі. Для кожної програми, за винятком основної $\forall P_i$, $i=\overline{2,N}$, перевіряється можливість розміщення на ресурсах, які розташовані до неї в схемі призначення. Переміщення можливе в разі виконання двох умов.

Перша – це сумарне оцінювання відношень цільових функцій програми та ресурсу для одного ресурсу *j*:

$$\sum_{i=0}^{N_{R'_j}}\frac{f\left(x_i^P\right)}{f\left(x_j^{R'}\right)}\le 1, \qquad (7)$$

де $N^{R'_j}$ – кількість програм, розподілених на ресурс *j*.

Друга умова – сумарне виконання обмежувальних функцій $X_j^{P_i}$.

Сортування одночасно векторів P та R відповідно до зростання значень відношень $\dfrac{f\left(x_i^P\right)}{f\left(x_i^R\right)}$.

Eman 10. Передача схеми призначення менеджеру Mg для ініціалізації процесу розподілених обчислень.

Після завершення етапу 10 починається процес виконання завдань. Агенти, продовжуючи сканування ресурсів, починають обслуговувати обмін даними між програмами завдань, підтримуючи функції *RTI*.

Паралельно з цим менеджер починає виконання функцій підтримки функціональної стійкості з реалізацією наступних етапів методу.

Eman 11. Запуск і контроль процесу виконання завдань.

Eman 12. Зберігання даних програми одним методом (або множиною методів). Менеджер періодично отримує стан програмних компонент і передає його модулю відновлення, який зберігає стан програмних компонент у момент часу t_q.

Eman 13. Проведення моніторингу вільних ресурсів R^e, виявлення нових обчислювальних ресурсів, на які можливо перерозподілити програмні компоненти.

Eman 14. Виявлення відмов або збоїв програм чи ресурсів, формування множини P_{Err}.

Eman 15. Якщо відмов не виявлено $\left(P_{Err}=\varnothing\right)$, відбувається перехід до етапу 19.

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2024. No. 1 (27)

Eman 16. Передача множини програм для відновлення P_{Err} та вільних ресурсів R^e від менеджера до модуля відновлення: $Mg\left(R^e, P_{Err}\right) \Rightarrow Ms\left(R^e, P_{Err}\right)$.

Eman 17. Вибір множини нових ресурсів для виконання програм (створення нової схеми призначення): $Sh^{Err} = \left\{P^{Err} \to R^e\right\}$ з виконанням кроків, описаних у етапах 6–9 методу для програм із множини P_{Err}.

Eman 18. Модуль відновлення надсилає відповідний програмний компонент на обраний ресурс, за необхідністю компілює програму для обраної платформи виконання та відновлює стабільну роботу програми, для чого повертає її до стану, найближчому до часу відмови ресурсу.

Eman 19. Перевірка умов завершення процесу виконання завдань. Якщо процес продовжується, відбувається перехід до етапу 12.

Eman 20. Завершення етапу виконання завдань, передача та зберігання даних результатів і статистичної інформації про процес обчислень.

На рис. 3 запропонована діаграма дій, що зображує послідовність виконання етапів методу. Особливістю діаграми, порівняно з граф-схемою алгоритму, є відтворення етапів, що виконуються паралельно. У реалізації методу беруть участь декілька програмних сервісів підтримки розподілених обчислень, які працюють одночасно на різних обчислювальних ресурсах.

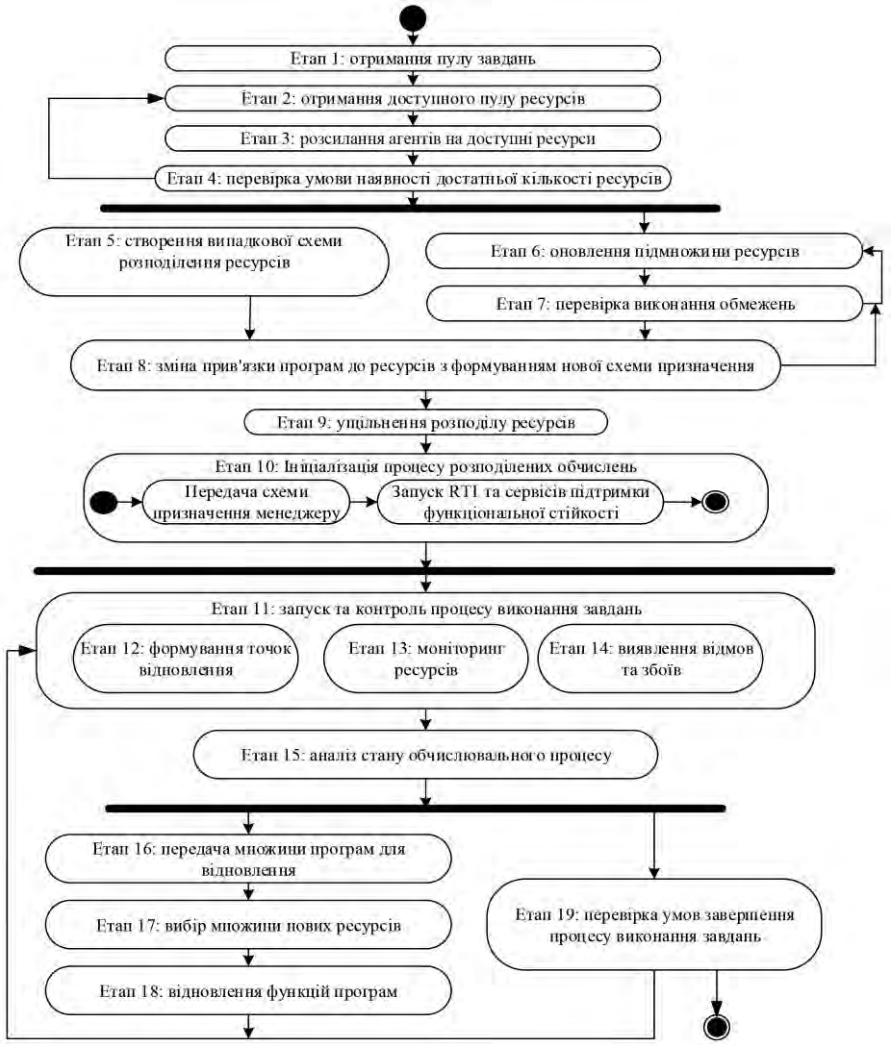


Рис. 3. Діаграма дій метаевристичного методу управління розподіленими обчисленнями

Так, паралельно із системою управління розподіленими обчисленнями виконуються такі сервіси: моніторинг ресурсів, аналізатор ресурсів на відповідність обмеженням, формування точок відновлення (зберігає дані програми у визначеному моменті часу для відновлення), виявлення відмов

Сучасний стан наукових досліджень та технологій в промисловості. 2024. №1 (27)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

та збоїв, аналіз стану обчислювального процесу. Також у паралельному режимі в разі виявлення відмови однієї або декількох програм активується сервіс, що перерозподіляє програмні компоненти на нові ресурси, відновлює їх роботу за допомогою збережених даних у точках відновлення.

Також важливо зауважити, що сервіси, які забезпечують процеси розподілених обчислень, виконуються на віддалених комп'ютерних ресурсах та організовують синхронізацію за допомогою обміну пакетами даних, який організується мережними засобами, засобами *RTI* або спеціально розробленими програмними компонентами.

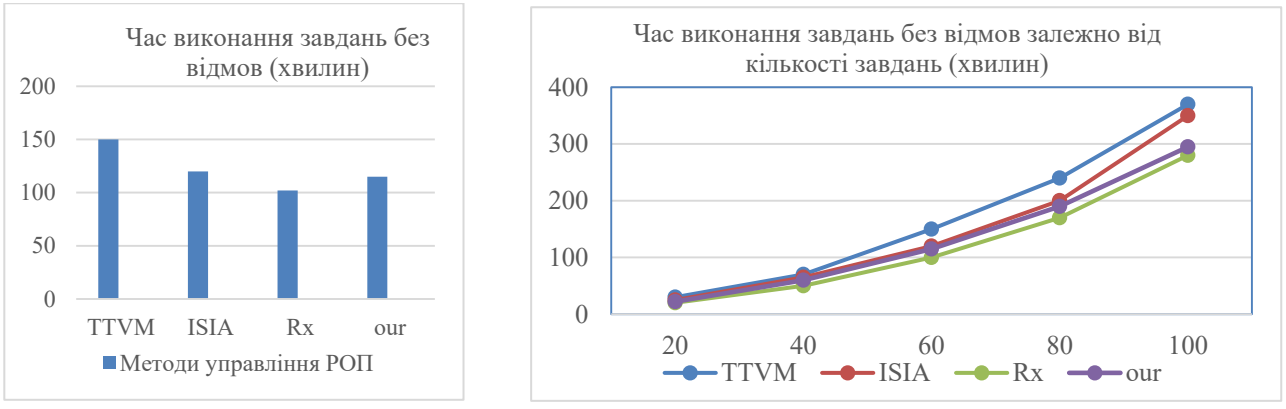
Результати досліджень

Для моделювання розподілених обчислень у хмарних та гетерогенних системах часто використовують *WorkflowSim* [21], який є проєктом із відкритим кодом, що дає змогу виконати порівняння розроблених методів з уже наявними та реалізованими в системі з метою оцінювання їх ефективності. У системі є декілька методів управління ресурсами та забезпечення функціональної стійкості розподілених систем. Для експериментів, що проводилися з метою перевірки корисності розроблених моделей, було обрано такі методи: *time-traveling virtual machine (TTVM)* [5], *instruction stream interrupt assumption (ISIA)* [6] та метод, який використовується в *Rx*-системах [13].

Система *WorkflowSim* дозволяє додати програмні компоненти, які містять нові методи та підсистеми. На рис. 1 зеленим кольором позначені модулі, що зазнали модифікацій у зв'язку з упровадженням розробленого методу. Для імітації відмов ресурсів було розроблено модуль, який із вказаною ймовірністю виводив із діючих ресурсів один або декілька обчислювачів, після чого долучався один із методів самовідновлення.

В експериментах задіяно змінну кількість віртуальних машин і програмні завдання з різними характеристиками. Пакети завдань та ймовірнісні характеристики процесу РОП були ідентичні, щоб виконати порівняльний аналіз розробленого методу з наявними. Кількість завдань визначалася в діапазоні від 10 до 100, кількість програм у завданні – від 5 до 500, кількість задіяних віртуальних машин – 5–40, обсяг фізичної пам'яті кожної віртуальної машини – 8 Gb, обсяг зовнішньої пам'яті – 2Tb, гіпервізор *Xen*. Потоки завдань та відмов генерувалися випадково, застосовувалися стандартні та розроблений методи для відновлення робочого процесу. Результати було усереднено на основі нормування в часі.

На рис. 4 наведено результати моделювання РОП у разі, коли відмови не відбувалися. З рисунка видно, що середній час виконання завдань, отриманий за допомогою запропонованого методу, кращий, ніж у методів *TTVM* та *ISIA,* і не значно перевищує час, отриманий для методу *Rx*. Крім того, із зростанням кількості завдань він наближається до методу *Rx*. Це свідчить про те, що запропонований метод не перевантажує систему управління під час нормальної роботи системи.



а) б)

Рис. 4. Оцінка часу виконання завдань із застосуванням стандартних і запропонованого (*our*) методів самовідновлення: а) середній час виконання, б) час виконання залежно від кількості завдань

На рис. 5 зображено результати експериментів з оцінкою часу виконання завдань, коли з різною ймовірністю η виникають відмови ресурсів, на яких виконуються програми. Дослідження вказує на

ISSN 2522-9818 (print)
ISSN 2524-2296 (online)
Innovative technologies and scientific solutions for industries. 2024. No. 1 (27)

ефективність запропонованого методу за наявності відмов ресурсів, а саме зменшення загального часу обчислювального процесу. Самовідновлення стану виконання завдань виконується швидше, ніж у разі застосування стандартних методів.

Наприклад, для ймовірності відмови $\eta = 0.01$ зменшення часу виконання завдань становило 8 %, для $\eta = 0.02$ – 13 %, $\eta = 0.03$ – 17 %. Зі зростанням імовірності відмов спостерігається збільшення відсотка економії часу проведення обчислень.

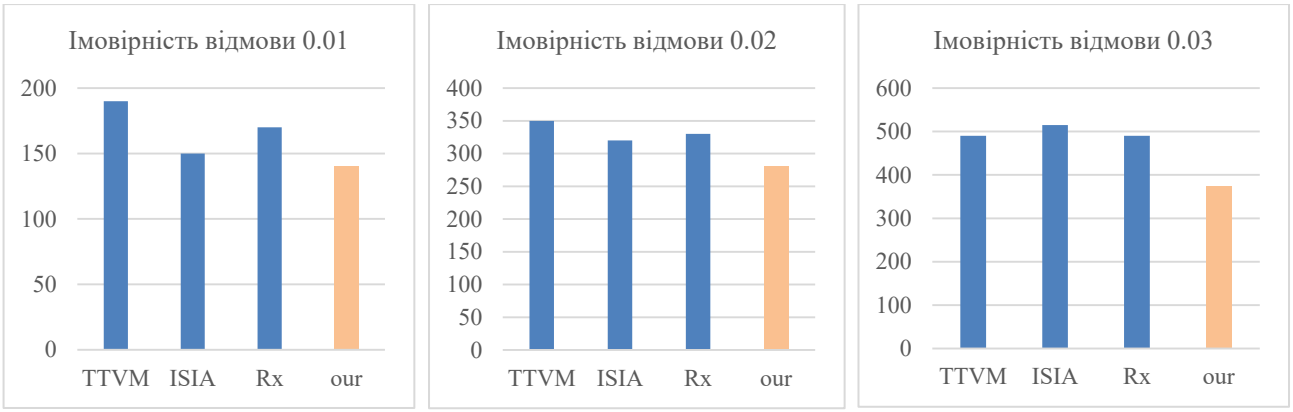


Рис. 5. Оцінка часу виконання завдань із застосуванням стандартних і запропонованого методів самовідновлення за умови різної ймовірності відмов ресурсів

На рис. 6 подані усередненні оцінки для споживання енергії. Для оцінювання енергоспоживання було використано моделі, запропоновані в роботі [18]. Ці моделі беруть до уваги параметри витрати енергії на обчислення, передачу та зберігання даних, синхронізацію та простій ресурсів. Експерименти виконувалися за умов, розглянутих вище. Для отримання середніх значень імовірність відмови визначалася для кожного експерименту випадково за рівномірним законом розподілу в діапазоні $\eta \in [0, 0.03]$.

Рис. 6. Оцінка споживання енергії в процесі виконання завдань із застосуванням стандартних і модифікованих моделей

Результати дослідження показують, що запропонований метод самовідновлення для забезпечення функціональної стійкості обчислювального процесу дає змогу зменшити споживання енергії. На відміну від застосування наявних методів, під час експериментів споживання енергії скорочується на 7–12 %. Є тенденція збільшення відсотка енергоефективності за умови зростання розмірності завдань та ймовірності відмов.

Висновки

Запропонований модифікований метод самовідновлення розподіленого програмного забезпечення

Сучасний стан наукових досліджень та технологій в промисловості. 2024. №1 (27)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

дає змогу підвищити ефективність обчислювального процесу в розподілених комп'ютерних системах. Він оснований на модифікованій теоретико-множинній моделі обчислювального процесу з підтримкою функціональної стійкості, модифікованій моделі оцінювання часу виконання та енергоспоживання.

Під час виконання роботи було змодельовано ситуації щодо розподілу вхідних пулів завдань на обчислювальні ресурси в імітаційному середовищі моделювання, проведено симуляцію обчислювального процесу та самовідновлення в разі відмов ресурсів. Досягнуті результати свідчать про те, що впровадження методу не збільшує час виконання завдання в умовах відсутності відмов. Навпаки, за наявності відмов запропонований метод дає змогу швидше відновити функціональність програмного завдання та зменшує тривалість виконання на 8–17 % з одночасним зменшенням енергоспоживання на 7–12 %. Також спостерігається зростання ефективності із збільшенням розмірності завдань та ймовірністю відмов.

Перспективним напрямом є розроблення технологій автоматизованого або автоматичного використання методів розподілу ресурсів та самовідновлення, особливо для хмарних систем різного призначення.

Список літератури

1. Kumar R., Singla S. A Study of Bug Manifestion Process for Ensuring Software Quality. *2021 10th IEEE International Conference on Communication Systems and Network Technologies (CSNT)*. 8–19 June 2021. P.801–804. DOI: 10.1109/CSNT51715.2021.9509676
2. REPT: Reverse debugging of failures in deployed software / W. Cui et al. *In Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA*, October 2018. P. 17–32. URL: https://www.usenix.org/system/files/osdi18-cui.pdf
3. Hoshino S., Arahori Y., Gondow K. Postmortem accurate IR-level state recovery for deployed concurrent programs. *ACM SIGAPP Applied Computing Review.* Vol. 2021:3. P. 33–48. DOI: https://doi.org/10.1145/3493499.3493502
4. Thakkar A., Lohiya, R. A survey on intrusion detection system: feature selection, model, performance measures, application perspective, challenges, and future research directions. *Artif Intell.* 2022. Rev. 55. P.453–563. DOI: https://doi.org/10.1007/s10462-021-10037-9
5. Yihunie F., Abdelfattah E., Regmi A. Applying machine learning to anomaly-based intrusion detection systems. *In: 2019 IEEE Long Island systems, applications and technology conference (LISAT). IEEE*, 2019. P.1–5. DOI: 10.1109/LISAT.2019.8817340
6. Wressnegger C., Kellner A. and Rieck K. ZOE: Content-Based Anomaly Detection for Industrial Control Systems. *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2018. P. 27–138. DOI: 10.1109/DSN.2018.00025.978-1-5386-5596-2
7. Intrusion Detection and Identification Using Tree-Based Machine Learning Algorithms on DCS Network in the Oil Refinery / K. Ho Kim et al., *IEEE Transactions on Power Systems*. 2022. Vol.37, No.6. P.4673–4682. DOI: 10.1109/TPWRS.2022.3150084
8. Song Y., Locasto M. E., Stavrou A. On the Infeasibility of Modeling Polymorphic Shell-code. *In Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS 2007)*. 2007. P. 541–551. DOI: https://doi.org/10.1145/1315245.1315312
9. Qin F., Tucek J., Sundaresan, J., Zhou. Y. Rx: Treating Bugs as Allergies-A Safe Method to Survive Software Failures. *In Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP 2005)*. 2005. P. 235–248. DOI: 10.1145/1275517.1275519
10. Chen C, Eisenhauer G. and Pande S. Near-Zero Downtime Recovery From Transient-Error-Induced Crashes. *IEEE Transactions on Parallel and Distributed Systems.* 2021. Vol. 33. Issue 5. P. 765–778. DOI: 10.1109/TPDS.2021.3096055
11. Bhat K., Kouwe E., Bos H. and Giuffrida C. FIRestarter: Practical Software Crash Recovery with Targeted Library-level Fault Injection. *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Taipei, Taiwan.* 2021. P. 363–375. DOI: 10.1109/DSN48987.2021.00048.
12. Sweeper: A Lightweight End-To-End System for Defending Against Fast Worms / J. Tucek et al. *In Proceedings of the 2nd European Conference on Computer Systems (Eu-roSys 2007).* 2007. P. 115–128. DOI: 10.1145/1272996.1273010
13. Verma, S., Roy, S. Debug-localize-repair: a symbiotic construction for heap manipulations. *Form Methods Syst Des 58.* 2021. P. 399–439. DOI: https://doi.org/10.1007/s10703-021-00387-z
14. X. Zhao et al. Data backup policies with failure-oblivious computing in reliability theory. *Annals of Operations Research.* 2022. P. 1–12. DOI: 10.1007/s10479-022-04941-8
15. Farzadnia E., Shirazi H, Nowroozi A. A novel sophisticated hybrid method for intrusion detection using the artificial immune system. *Journal of Information Security and Applications,* 2022. Vol. 70. DOI: 10.1016/j.jisa.2020.102721
16. Рубан І., Волк М., Рісухін М. Метод самовідновлення розподіленого програмного забезпечення в гетерогенних комп'ютерних системах. *Телекомунікаційні та інформаційні технології.* 2019. № 3 (64). С. 17–23. DOI: 10.31673/2412-4338.2019.031723

17. М. Волк та ін. Журналізація стану програм для самовідновлення паралельних програмних систем. *Системи управління, навігації та зв'язку*. 2023. Випуск 2(72). С.76-82. DOI: 10.26906/SUNZ.2023.2.080
18. Гора М., Волк М. Моделі управління ресурсами для забезпечення функціональної стійкості процесу розподілених обчислень. *Вісник Херсонського національного технічного університету.* 2023. No 4(87). С. 244–251. DOI: https://doi.org/10.35546/kntu2078-4481.2023.4.28
19. Saremi S., Mirjalili S., and Lewis A. Grasshopper Optimization Algorithm. *Theory and application. Elsevier, Advances in Engineering Software Journal.* 2017. No. 105. P. 30–47. DOI: https://doi.org/10.1016/j.advengsoft.2017.01.004
20. K. Kulkarni et al. An Inertia Weight Concept-Based salp Swarm Optimization Algorithm. *In Proceedings of the 2021 IEEE Madras Section Conference (MASCON), Chennai, India.* 27–28 August 2021. P. 1–6. DOI: 10.1109/MASCON51689.2021.9563412
21. WorkflowSim. URL: https://github.com/WorkflowSim/WorkflowSim-1.0 (дата звернення 06.02.2024)

References

1. Kumar, R., Singla, S. (2021), "A Study of Bug Manifestion Process for Ensuring Software Quality" *2021 10th IEEE International Conference on Communication Systems and Network Technologies (CSNT)*. P. 801–804. DOI: 10.1109/CSNT51715.2021.9509676
2. Cui, W., Ge, X., Kasikci, B., Niu, B., Sharma, U., Wang, R., Yun, I. (2018), "REPT: Reverse debugging of failures in deployed software". *In Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation, OSDI. Carlsbad, CA.* P. 17–32. available at: https://www.usenix.org/system/files/osdi18-cui.pdf
3. Hoshino, S., Arahori, Y., Gondow, K. (2021), "Postmortem accurate IR-level state recovery for deployed concurrent programs". *ACM SIGAPP Applied Computing Review.* Vol. 3. P. 33–48. DOI: https://doi.org/10.1145/3493499.3493502
4. Thakkar, A., Lohiya, R. (2022), "A survey on intrusion detection system: feature selection, model, performance measures, application perspective, challenges, and future research directions". *Artif Intell.* Rev. 55. P. 453–563. DOI: https://doi.org/10.1007/s10462-021-10037-9
5. Yihunie, F., Abdelfattah, E., Regmi, A. (2019), "Applying machine learning to anomaly-based intrusion detection systems". *In: 2019 IEEE Long Island systems, applications and technology conference (LISAT). IEEE.* P. 1–5. DOI: 10.1109/LISAT.2019.8817340
6. Wressnegger, C., Kellner, A. and Rieck, K. (2028), "ZOE: Content-Based Anomaly Detection for Industrial Control Systems." *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. P. 27–138. DOI: 10.1109/DSN.2018.00025.978-1-5386-5596-2
7. Kim, R, Kwak, B., Han, M., Kim, H. (2022), "Intrusion Detection and Identification Using Tree-Based Machine Learning Algorithms on DCS Network in the Oil Refinery". *IEEE Transactions on Power Systems*. Vol.37, No.6. P. 4673–4682. DOI: 10.1109/TPWRS.2022.3150084
8. Song, Y., Locasto, M., Stavrou, A. (2007), "On the Infeasibility of Modeling Polymorphic Shell-code". *In Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS 2007)*. P. 541–551. DOI: https://doi.org/10.1145/1315245.1315312
9. Qin, F., Tucek, J., Sundaresan, J., Zhou, Y. (2005), "Rx: Treating Bugs As Allergies – A Safe Method To Survive Software Failures". *In Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP 2005)*. P. 235–248. DOI: 10.1145/1275517.1275519
10. Chen, C., Eisenhauer, G., Pande, S. (2021), "Near-Zero Downtime Recovery From Transient-Error-Induced Crashes". *IEEE Transactions on Parallel and Distributed Systems.* Vol. 33. Issue 5. P. 765–778. DOI: 10.1109/TPDS.2021.3096055
11. Bhat, K., Kouwe, E., Bos, H., Giuffrida, C. (2021), "FIRestarter: Practical Software Crash Recovery with Targeted Library-level Fault Injection". *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Taipei, Taiwan.* P. 363–375. DOI: 10.1109/DSN48987.2021.00048.
12. Tucek, J., Newsome, J., Lu, S., Huang, C., Xanthos, S., Brumley, D., Zhou, Y., Song., D. (2007), "Sweeper: A Lightweight End-To-End System for Defending Against Fast Worms". *In Proceedings of the 2nd European Conference on Computer Systems (Eu-roSys 2007).* P. 115–128. DOI: 10.1145/1272996.1273010
13. Verma, S., Roy, S. (2021), "Debug-localize-repair: a symbiotic construction for heap manipulations". *Form Methods syst Des 58.* 2021. P. 399–439. DOI: https://doi.org/10.1007/s10703-021-00387-z
14. Zhao, X., Wang, D., Mizutani, S., Nakagawa, T. (2022), "Data backup policies with failure-oblivious computing in reliability theory". *Annals of Operations Research.* P. 1–12. DOI: 10.1007/s10479-022-04941-8
15. Farzadnia, E., Shirazi, H., Nowroozi A. (2019), "A novel sophisticated hybrid method for intrusion detection using the artificial immune system". *Journal of Information Security and Applications,* 2022. Vol. 70. DOI: 10.1016/j.jisa.2020.102721
16. Ruban, I., Volk, M., Risukhin, M. (2019), "A method of self-healing of distributed software in heterogeneous computer systems" ["Metod samovidnovlennya rozpodilenoho prohramnoho zabezpechennya v heterohennykh komp″yuternykh systemakh "]. *Telecommunications and information technologies*. № 3(64). P. 17–23. DOI: 10.31673/2412-4338.2019.031723

Сучасний стан наукових досліджень та технологій в промисловості. 2024. №1 (27)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

17. Volk, M., Hora, M., Labazov, V., Mishenko, A., Barsukiv, A., Goletz, B. (2023), "Journaling of program status for self-healing of parallel software systems" ["Zhurnalizatsiya stanu prohram dlya samovidnovlennya paralel′nykh prohramnykh system"]. *Control, navigation and communication systems.* No 2(72). P. 76–82. DOI: 10.26906/SUNZ.2023.2.080
18. Hora, M., Volk, M. (2023), "Resource management models to ensure the functional stability of the distributed computing process" ["Modeli upravlinnya resursamy dlya zabezpechennya funktsional′noyi stiykosti protsesu rozpodilenykh obchyslen"]. *Bulletin of the Kherson National Technical University.* No 4(87). P. 244- 251. DOI https://doi.org/10.35546/kntu2078-4481.2023.4.28
19. Saremi, S., Mirjalili, S., Lewis, A. (2017), "Grasshopper Optimization Algorithm". *Theory and application. Elsevier, Advances in Engineering Software Journal.* No. 105. P. 30–47. DOI: https://doi.org/10.1016/j.advengsoft.2017.01.004
20. Kulkarni, K. et al. (2021), "An Inertia Weight Concept-Based salp Swarm Optimization Algorithm". *In Proceedings of the 2021 IEEE Madras Section Conference (MASCON), Chennai, India.* 27–28 August 2021. P. 1–6. DOI: 10.1109/MASCON51689.2021.9563412
21. WorkflowSim. available at: https://github.com/WorkflowSim/WorkflowSim-1.0 (last accessed 06.02.2024)

Надійшла 08.02.2024

Відомості про авторів / About the Authors

Волк Максим Олександрович – доктор технічних наук, професор, Харківський національний університет радіоелектроніки, професор кафедри електронних обчислювальних машин, Харків, Україна; e-mail: maksym.volk@nure.ua; ORCID ID: http://orcid.org/0000-0003-4229-9904

Гора Максим Володимирович – Харківський національний університет радіоелектроніки, аспірант кафедри електронних обчислювальних машин, Харків, Україна; e-mail: risuhin.max@gmail.com; ORCID ID: http://orcid.org/0000-0003-0085-3559

Volk Maksym – Doctor of Sciences (Engineering), Professor, Kharkov National University of Radio Electronics, Professor at the Department of Electronic Computer, Kharkiv, Ukraine.

Hora Maksym – Kharkiv National University of Radio Electronics, Postgraduate Student at the Department of Electronic Computers, Kharkiv, Ukraine.

A MODIFIED METHOD OF SELF-RECOVERY OF DISTRIBUTED SOFTWARE IN HETEROGENEOUS COMPUTER SYSTEMS

The **object** of research is the distributed computing process in heterogeneous computer systems. The **subject** of the research is methods of self-healing for distributed software on heterogeneous computer systems. The **goal** is to increase the efficiency of distributed data processing systems with support for the functional stability of the computing process by developing a modified method of self-healing of distributed software. **Tasks**: to investigate the existing methods of restoring the distributed computing process, to draw conclusions about their advantages and disadvantages; on the basis of mathematical models of tasks, computing resources and existing methods of resource allocation, develop a modification of the method of self-recovery of distributed software taking into account management strategies, finding the best solution for the selected criteria, reducing energy consumption during the execution of tasks; conduct a number of experiments comparing the developed method with existing ones. Research **methods** are based on the use of set theory, general systems theory, and simulation modeling theory. The **results** of the experiments obtained during the simulation of the allocation of software tasks to computing resources in a simulated simulation environment and the simulation of the computing process during self-recovery in case of resource failures confirm the effectiveness of the proposed method. **Conclusion**: the application of the method in distributed computing control systems does not increase the time the system spends on performing the task in the absence of failures, at the same time, in the presence of failures, it allows to restore the functionality of the software task faster and reduces the execution time by 8–17%, and energy consumption by 7–12%. There is also an increase in efficiency with an increase in the size of the tasks and the probability of failures. The development of technologies for automated or automatic use of methods of resource allocation and self-recovery can be indicated as areas for future research.

Keywords: self-healing methods; software; distributed computing; computer systems; cloud architectures; software agents.

Бібліографічні описи / Bibliographic descriptions

Волк М. О., Гора М. В. Модифікований метод самовідновлення розподіленого програмного забезпечення в гетерогенних комп'ютерних системах. *Сучасний стан наукових досліджень та технологій в промисловості*. 2024. № 1 (27). С. 5–17. DOI: https://doi.org/10.30837/ITSSI.2024.27.005

Volk, M., Hora, M. (2024), "A modified method of self-recovery of distributed software in heterogeneous computer systems", *Innovative Technologies and Scientific Solutions for Industries*, No. 1 (27), P. 5–17. DOI: https://doi.org/10.30837/ITSSI.2024.27.005