УДК: 519.852:004.455.2 DOI: 10.31673/2412-4338.2024.022935

Крючкова Лариса Петрівна
Київський столичний університет імені Бориса Грінченка, Київ
ORCID 0000-0002-8509-6659

Яремчук Денис Сергійович
Київський столичний університет імені Бориса Грінченка, Київ
ORCID 0000-0002-8509-6659

Пазинін Андрій Сергойович
Інститут телекомунікацій і глобального інформаційного простору НАН України, Київ
ORCID 0009-0002-9506-9539

ПЛАНУВАННЯ НАВАНТАЖЕННЯ КОНТЕЙНЕРІВ ЗА ДОПОМОГОЮ ЛІНІЙНОГО ПРОГРАМУВАННЯ

Анотація. *Зі зростанням популярності технології розгортання контейнерів Docker, планувальник контейнерів стає ключовим компонентом у процесі розгортання додатків. Для ефективного планування контейнерів необхідно враховувати декілька важливих факторів, таких як споживання електроенергії сервером, час отримання образу віртуальної машини із регістру образів, ціна обміну даними між клієнтом та контейнером. При урахуванні даних факторів, можливо створити систему, що буде ефективно розподіляти задачі між контейнерами, з мінімальним навантаженням на сервер та максимально ефективним використанням ресурсів системи. Проблему планування контейнерів було розглянуто як задачу цілочисельного лінійного програмування. Побудована модель є ефективним та гнучким планувальником, що здатний розподіляти навантаження на контейнери, враховуючи вказані фактори. Для оцінки ефективності нового планувальника, проведено порівняння з планувальником Docker Swarm, що користується Binpack методом планування. Даний алгоритм виконує поставлені на нього задачі, проте його ефективність невисока. Основний принцип алгоритму Binpack полягає у використанні мінімально можливої кількості фізичних вузлів системи для розташування контейнерів. Даний алгоритм не враховує інші фактори, такі як ціну отримання образу системи або ціну обміну даними між клієнтом та сервером. У ході експерименту виявилося, що запропонований метод є більш ефективним. Метод лінійного програмування краще розподіляє контейнери та витрачає на це менше часу та ресурсів. Особливо це помітно зі збільшенням кількості клієнтів та серверів, для яких потрібно вирішити задачу планування. Розроблений метод можливо інтегрувати у фреймворки планування контейнерів.*

Залишок цієї статті організований наступним чином. У «Вступі» описаний принцип роботи контейнерів, визначена проблема використання контейнерів у великій кількості та проаналізовані інші роботи, що спрямовані на вирішення даної проблеми. У «Теоретичних основах дослідження» розглянута технологія Docker та проблема розгортання великої кількості додатків за допомогою даного інструменту. У «Методиці дослідження» описаиа модель, яка вирішує проблему планування контейнерів. У розділі «Результати дослідження» проведений ряд експериментів та порівняння створеного методу із алгоритмом Binpack. У «Висновках» проаналізовані результати експериментів, зроблені у попередньому розділі та запропоновані подальші шляхи розвитку створеної технології планування контейнерів.

Keywords: *linear programming, container, dynamic migration.*

Kriuchkova Larysa
Borys Grinchenko Kyiv Metropolitan University, Kyiv
ORCID 0000-0002-8509-6659

Yaremchuk Denys
Borys Grinchenko Kyiv Metropolitan University, Kyiv
ORCID 0000-0002-8509-6659

Pazynin Andrii

Institute of Telecommunications and Global Information Space of the National Academy of Sciences of Ukraine, Kyiv
ORCID 0009-0002-9506-9539

CONTAINER LOAD PLANNING USING LINEAR PROGRAMMING

Abstract. *With the growing popularity of Docker container deployment technology, the container scheduler is becoming a key component in the application deployment process. For effective container planning, several important factors must be taken into account, such as server power consumption, time to obtain a virtual machine image from the image registry, and the price of data exchange between the client and the container. Taking into account these factors, it is possible to create a system that will effectively distribute tasks between containers, with a minimum load on the server and the most efficient use of system resources. The container planning problem was considered as an integer linear programming problem. The constructed model is an effective and flexible scheduler capable of distributing the load on containers, taking into account the specified factors. To evaluate the effectiveness of the new scheduler, a comparison was made with the Docker Swarm scheduler, which uses the Binpack scheduling method. This algorithm fulfills the tasks assigned to it, but its efficiency is not high. The main principle of the Binpack algorithm is to use the minimum possible number of physical nodes of the system for the location of containers. This algorithm does not take into account other factors, such as the price of obtaining a system image or the price of data exchange between the client and the server. During the experiment, it turned out that the proposed method is more effective. The linear programming method allocates containers better and consumes less time and resources. This is especially noticeable with an increase in the number of clients and servers for which the scheduling problem needs to be solved. The developed method can be integrated into container planning frameworks. The remainder of this article is organized as follows. In the "Introduction" the principle of operation of containers is described, the problem of using containers in large quantities is defined, and other works aimed at solving this problem are analyzed. In "Theoretical Foundations of Research" Docker technology and the problem of deploying a large number of applications using this tool are considered. The "Research Methodology" describes a model that solves the problem of container planning. In the "Research results" section, a number of experiments and a comparison of the created method with the Binpack algorithm are performed. In the "Conclusions" the results of the experiments conducted in the previous section are analyzed and further ways of development of the created container planning technology are proposed.*

Ключові слова: *лінійне програмування, контейнер, динамічна міграція*.

1. Вступ.

Для розгортання додатків у сучасній розробці використовується технологія контейнерів. Дана технологія максимально ефективно ізолює ресурси системи. Контейнери надають можливість створювати середовище для роботи додатку лише з необхідними системними ресурсами. Тобто система, на якій працює декілька контейнерів, для кожного окремого контейнеру виділяє окремий фрагмент пам’яті для роботи та окремі процесорні потужності. Для запуску такого невеликого середовища потрібно небагато часу, ресурси системи використовуються максимально ефективно.

2. Постановка проблеми.

При великому навантаженні на сервери, на яких може бути розгорнуто декілька різних додатків, з’являється необхідність динамічного створення нових контейнерів, кожному з яких необхідно виділити відповідну кількість ресурсів. Даний процес називається плануванням контейнерів, та самим популярним алгоритмом вирішення даної задачі є Binpack. Даний алгоритм може не оптимально розміщувати об’єкти, надаючи пріоритет мінімізації використаних ресурсів. Зі зростанням об’єктів, алгоритм потребує значних обчислювальних ресурсів. Також його ефективність залежить від порядку введення об’єктів, що може збільшити необхідний час для упакування.

3. Аналіз останніх досліджень і публікацій.

Методи планування додатків активно розробляються зі зростанням популярності хмарних обчислень. Серед них можна виділити метод Memory Balancing (MEB), розроблений W.M. Zhao. Суть даного методу полягає у постійному відслідковуванні використання

оперативної пам'яті кожної віртуальної машини та перевизначені об'ємів пам'яті у відповідності до потреб системи [1]. Padala запропонував систему зворотного зв'язку, яка складається з онлайн аналізатора лінійної моделі, що відслідковує залежність ефективності роботи додатку та виділених ресурсів (CPU, memory), і контролера ресурсів, що розподіляє ресурси системи [2]. Than і Thein запропонували два алгоритму розташування ресурсів, що спрямовані на збереження електроенергії. Обидва алгоритми беруть до уваги такі фактори як політика розподілу ресурсів, техніка управління енергією та модель енергоспоживання [3]. Bhardwaj запропонував автономну модель виділення ресурсів, що динамічно виділяє ресурси для ВМ у відповідності до ступеню навантаження віртуальної машини [4]. Khodar запропонував генетичний алгоритм, який створює оптимальний план навантаження контейнерів, що розташовані на віртуальних машинах. На основі історичних даних щодо попередніх навантажень та поточного стану системи, алгоритм робить прогноз та надає рекомендацію щодо розподілення ресурсів ВМ [5]. Rengasamy і Chidambaram запропонували метод розподілення ресурсів серверу для хмарних обчислень, який фокусується на запобіганні майбутніх перенавантажень системи, більше ніж на балансуванні вже створених навантажень [6]. Chen запропонував алгоритм розподілення ресурсів у хмарних сервісах з підтримкою негайних запитів ресурсів. Даний метод перебудовує пріоритети ресурсів для невідкладних викликів віртуальних машин [7]. Yin запропонував LCGA генетичний алгоритм планування, який бере до уваги не тільки фактор навантаження системи, а й час виконання завдань, покладених на контейнер [8].

4. Мета дослідження.

Створення ефективного методу планування додатків, який врахує недоліки алгоритму Binpack та дозволить розгортати додатки з більшою ефективністю.

5. Огляд дослідження.

5.1. Теоретичні основи дослідження.

Контейнери Docker дозволяють упаковувати додаток разом з усіма його залежностями і легко запускати його в будь-якому середовищі. Але при такій гнучкості у створенні контейнерів, виникає проблема виділення відповідної машини, яка відповідала б технічним потребам контейнеру. Дану проблему динамічного виділення віртуальних машин можна вирішити як задачу цілочисельного лінійного програмування, яка буде враховувати різноманітність потреб кожного контейнеру і наявних для цього ресурсів віртуальної машини.

Для виконання контейнеру, його необхідно розгорнути на відповідній машині, яка виділить йому необхідні обчислювальні потужності та пам'ять. Для найбільш ефективного виділення ресурсів, потрібно притримуватися принципів хмарних обчислень, а саме принципу динамічного виділення ресурсів за потребою контейнеру. Дана проблема стає помітною коли необхідно розгорнути велику кількість контейнерів, кожен з яких може належати до різних додатків та мати різні потреби, що призводить до нехватки ресурсів та малоефективного виділення ресурсів [9].

5.2. Методика дослідження.

Для визначення методу управління контейнерами, необхідно дослідити принцип роботи контейнеризованих додатків та визначити основні моменти, які слід врахувати при формулюванні задачі лінійного програмування (ЛП).

Типічні контейнеризовані застосунки реалізовані за таким принципом: клієнт робить запит до серверу через веб сторінку, група серверів має декілька вузлів виконання (контейнерів), кожен з контейнерів має певний ліміт обчислювальних ресурсів. Коли клієнт надає запит на виконання, складність його запиту оцінюється певним числовим значенням. Відповідно до складності задачі, її виконання передається на відповідний контейнер, який найбільше підходить для цього. Також потрібно враховувати обмін даними між клієнтом та сервером, який займає деякий час. Для запуску контейнеру, сервер повинен отримати необхідний образ із репозиторію образів. Сервер може мати деякі необхідні файли, або взагалі не мати ніяких відповідних файлів, тому час отримання може різнитись. На цей процес також витрачаються ресурси.

Враховуючи всі вище згадані процеси, необхідно розробити планувальник контейнерів, який буде відповідати двом критеріям:

1) Всі клієнтські запити контейнеризовані та оброблені сервером.

2) Запити, направлені на відповідні сервери, не повинні перевищувати обчислювальні можливості серверу.

Для виконання першого обмеження, позначимо x(k,t) як навантаження клієнтом t серверу k.

$$\sum_{k=1}^{n} x(k,t) = Workload(t), \tag{1}$$

де n – загальна кількість контейнерів.

Для виконання другого критерію, введемо наступне обмеження:

$$\sum_{t=1}^{m} x(k,t) \le Capacity(k) \tag{2}$$

Це обмеження гарантує те, що всі запущені задачі не перевищують обчислювальних можливостей серверу k.

Ціна обміну даними у мережі визначена як:

$$f_1(x) = \sum_{t=1}^{m}\sum_{k=1}^{n} trans\_\cos t(k,t) * x(k,t) \tag{3}$$

Для розрахунку електроенергії, що використовується сервером, використовується наступна формула [10]:

$$P_k(u) = (P_{k,\max} - P_{k,idle}) * u + P_{k,idle} \tag{4}$$

де $P_k(u)$ і $P_{k,idle}$ є середніми значеннями хосту k, коли сервер перебуває у стані спокою та повного навантаження відповідно. Контейнер у стані спокою не запущений, у той час як повне навантаження означає повне навантаження всіх вузлів серверу відповідними задачами клієнтів.

Символ u у даній функції означає рівень утилізації ресурсів, і розраховується наступним чином:

$$u = \frac{\sum_{t=1}^{m} x(k,t)}{Capacity(k)} \tag{5}$$

Ціна споживання електроенергії визначається як:

$$f_2(x) = \sum_{k=1}^{n}((P_{k,\max} - P_{k,idle}) * \frac{\sum_{t=1}^{m} x(k,t)}{Capacity(k)} + P_{k,idle}) \tag{6}$$

Ціна отримання образу t для запуску контейнера розраховується наступним чином:

$$f_3(y) = \sum_{t=1}^{m}\sum_{k=1}^{n} pull\_\cos t(k,t) * y(k,t) \tag{7}$$

де $x(k,t)$ у тому випадку, якщо вузол k потребує образ з реєстру образів, і $y(k,t)$, якщо образ не потрібен. Якщо навантаження на вузол немає, то і потреби у отриманні образу теж нема. Це можна відобразити обмеженням:

$$x(k,t) \le y(k,t) * Capacity(k) \tag{8}$$

Якщо $y(k,t)$, то і навантаження $y(k,t)$ відповідно.

Враховуючи функції (3), (6), (7), кінцева функція обчислення ціни розгортання контейнеру буде такою:

$$f(x,y) = \alpha f_1(x) + f_2(x) + f_3(y) = \alpha \sum_{t=1}^{m} \sum_{k=1}^{n} trans\_\cos t(k,t) * x(k,t) + \sum_{k=1}^{n}((P_{k,\max} - P_{k,idle}) * \frac{\sum_{t=1}^{m} x(k,t)}{Capacity(k)} + P_{k,idle}) + \sum_{t=1}^{m} \sum_{k=1}^{n} pull\_\cos t(k,t) * y(k,t) \tag{9}$$

де k є змінною для балансування ціни обміну даними.

Так як ми шукаємо мінімально можливі ціни на розташування задач у відповідних контейнерах, цільова функція має наступний вигляд:

$$\min f(x,y) = \alpha f_1(x) + f_2(x) + f_3(y) \quad (10)$$

З відповідними обмеженнями:

$$\sum_{k=1}^{n} x(k,t) = Workload(t), \quad (11)$$

$$\sum_{t=1}^{m} x(k,t) \le Capacity(k), \quad (12)$$

$$x \ge 0 \ and \ x \ is \ \text{integer}, \quad (13)$$

$$y \in (0,1), \quad (14)$$

Звідси маємо задачу цілочисельного лінійного програмування з цільовою функцією (10) та обмеженнями (11-13), де може бути досягнуте оптимальне рішення для $x(k,t)$.

5.3. Результати дослідження.

Для проведення експерименту використовується мова програмування Python, бібліотека PuLP, що дозволяє розв'язувати задачі лінійного програмування. Обидва методи реалізовані за допомогою бібліотеки PuLP, для Binpack змінена поведінка функції ЛП.

Експеримент проводиться у декілька ітерацій, кожного разу кількість серверів та клієнтів буде зростати. Для початку проведено тестування методів на невеликих об'ємах даних, а саме 20 хостів та 30 клієнтів. Далі об'єми збільшуються до 50 серверів та 100 клієнтів, 100 серверів та 200 клієнтів. Навантаження клієнтів, обчислювальні ресурси серверів, ціна обміну даними по мережі і ціна отримання образу для контейнерів генеруються випадково в межах від 10 до 20, 50 до 100, 1 до 4, 0 до 3 відповідно. У ході експериментів визначилось, що випадкова генерація ціни обміну даних впливає на результати, тому = 0.5.

Розроблений алгоритм ЛП порівнюється з алгоритмом Binpack, результати представлені у таблицях 1-3.

Таблиця 1

Загальна ціна розподілу для 20 серверів та 30 клієнтів

№ Ітерації	ЛП	Binpack
1	955	818
2	1168	877
3	899	838
4	743	872
5	923	893

Таблиця 2

Загальна ціна розподілу для 50 серверів та 100 клієнтів

№ Ітерації	ЛП	Binpack
1	3169	6970
2	2377	7225
3	2431	7222
4	2393	6851
5	3123	7475

Таблиця 3

Загальна ціна розподілу для 100 серверів та 200 клієнтів

№ Ітерації	ЛП	Binpack
1	9265	23938
2	9408	22472
3	7997	22900
4	7831	21508
5	9438	22164

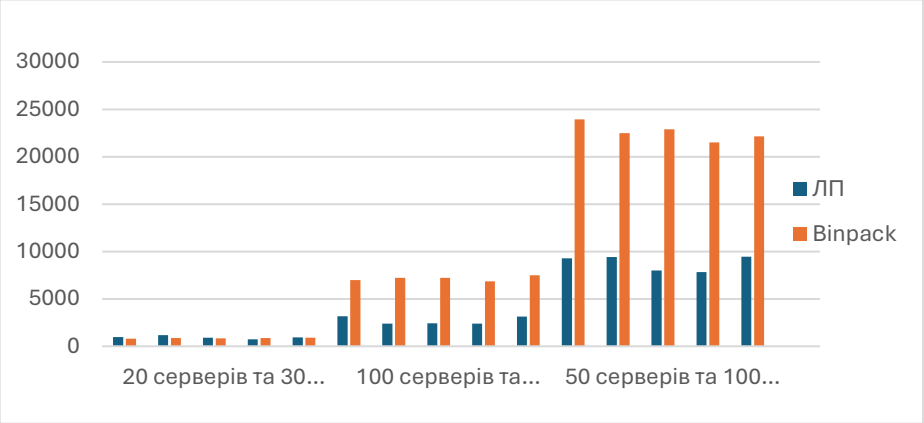

Рис 1. Порівняння загальної ціни двох методів

5. Висновки та перспективи подальших досліджень.

Дана робота запропонувала новий планувальник контейнерів за допомогою представлення проблеми контейнеризації додатків у вигляді задачі цілочисельного лінійного програмування. У результаті можна помітити, що у разі невеликих об'ємів даних (до 20 серверів та 30 клієнтів) метод Binpack є таким же ефективним, або навіть кращим ніж запропонований ЛП метод. Проте зі збільшенням об'ємів даних, ЛП метод має значну перевагу над Binpack. Використання нового методу планування контейнерів підвищить ефективність використання ресурсів серверу, зменшить витрати на обслуговування серверу та підвищить якість сервісу, яким користуються клієнти.

Реалізація динамічної міграції контейнерів, як це вже реалізовано при міграції віртуальних машин до інших фізичних вузлів без переривання роботи (Live Migration), може бути потенціальним шляхом покращення методу планування контейнерів. Також для створення автоматичних систем планування контейнерів можна розробити метод з використанням штучного інтелекту.

Список використаної літератури

1. Zhao W., Wang Z., Luo Y. Dynamic memory balancing for virtual machines. ACM SIGOPS Operating Systems Review. 2009. Vol. 43, no. 3. P. 37–47. (date of access: 07.04.2024).

2. Automated control of multiple virtualized resources / P. Padala et al. the fourth ACM european conference, Nuremberg, Germany, 1–3 April 2009. New York, New York, USA, 2009. (date of access: 07.04.2024).

3. Than M. M., Thein T. Energy-Saving Resource Allocation in Cloud Data Centers. 2020 IEEE Conference on Computer Applications (ICCA), Yangon, Myanmar, 27–28 February 2020. 2020. (date of access: 07.04.2024).

4. Bhardwaj T., Upadhyay H., Sharma S. C. Autonomic Resource Allocation Mechanism for Service-based Cloud Applications. 2019 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS), Greater Noida, India, 18–19 October 2019. 2019. (date of access: 07.04.2024).

5. Khodar A., Al-Afare H. A. F., Alkhayat I. New Scheduling Approach for Virtual Machine Resources in Cloud Computing based on Genetic Algorithm. 2019 International Russian Automation Conference, Sochi, Russia, 8–14 September 2019. 2019. (date of access: 07.04.2024).

6. Rengasamy R., Chidambaram M. A Novel Predictive Resource Allocation Framework for Cloud Computing. 2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS), Coimbatore, India, 15–16 March 2019. 2019. (date of access: 07.04.2024).

7. Chen J. A Cloud Resource Allocation Method Supporting Sudden and Urgent Demands. 2018 Sixth International Conference on Advanced Cloud and Big Data (CBD), Lanzhou, 12–15 August 2018. 2018. (date of access: 07.04.2024).

8. Yin S., Ke P., Tao L. An improved genetic algorithm for task scheduling in cloud computing. 2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA), Wuhan, 31 May 2018 – 02 June 2018. 2018.

9. Peinl R., Holzschuher F., Pfitzer F. Docker Cluster Management for the Cloud - Survey Results and Own Solution. Journal of Grid Computing. 2016. Vol. 14, no. 2. P. 265–282. URL:

10. Feller E., Rilling L., Morin C. Energy-Aware Ant Colony Based Workload Placement in Clouds. 2011 12th IEEE/ACM International Conference on Grid Computing (GRID), Lyon, France, 21–23 September 2011. 2011. (date of access: 07.04.2024).