НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА СИСТЕМНОГО ПРОГРАМУВАННЯ І
СПЕЦІАЛІЗОВАНИХ КОМП'ЮТЕРНИХ СИСТЕМ

«На правах рукопису»
УДК 004.7

«До захисту допущено»

Завідувач кафедри СПСКС

_____ В.П.Тарасенко_
(підпис) (ініціали, прізвище)

"___"_____2018р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 123 «Комп'ютерна інженерія»
«Комп'ютерні системи та компоненти»

на тему: Спосіб планування та динамічного балансування навантаження на модулі хмарного середовища

Виконала: студентка ІІ курсу, групи КВ-61м

Телелейко Інна Сергіївна_____ _____
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник: доцент кафедри СПіСКС, к.т.н.,

доцент Орлова Марія Миколаївна _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент_____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – другий (магістерський)
Спеціальність 123 «Комп'ютерна інженерія»
«Комп'ютерні системи та компоненти»

ЗАТВЕРДЖУЮ
Завідувач кафедри СПСКС
_____ _В.П.Тарасенко__
(підпис) (ініціали, прізвище)
«___»_____2018р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Телелейко Інні Сергіївні

1. Тема дисертації: спосіб планування та динамічного балансування навантаження на модулі хмарного середовища,
науковий керівник дисертації: доцент кафедри СПіСКС, к.т.н., доцент Орлова М.М.,
затверджені наказом по університету від «22» березня 2018 р. №986-с

2. Термін подання студентом дисертації 11 травня 2018 р.

3. Об'єкт дослідження: системи балансування робочого навантаження та планування ресурсів в хмарних технологіях.

4. Предмет дослідження: методи і алгоритми балансування навантаження на модулі у хмарному середовищі, критерії їх порівняння, узагальнена модель хмарного додатку, обґрунтування вибору правил балансування навантаження.

5. Перелік завдань, які потрібно розробити:
- проведення систематизації методів і алгоритмів балансування навантаження в хмарних системах;
- розробка порівняльної характеристики розповсюджених систем балансування навантаження;
- розробка модифікованого способу динамічного балансування навантаження в хмарному середовищі на основі методу міграції задач та алгоритму Weighted Least Connections;
- експериментальне дослідження ефективності створених правил балансування навантаження.

6. Перелік ілюстративного матеріалу:
   - узагальнена характеристика Cloud Computing;
   - схема компонентів планування завдань у Cloud Computing;
   - схема планування завдання;
   - загальна схема запропонованого способу динамічного балансування навантаження
   - результати проведених досліджень;
   - діаграма отриманих результатів порівняння запропонованого способу.

7. Перелік публікацій:
   - Телелейко І. С., Орлова М.М. Спосіб динамічного балансування навантаження в хмарному середовищі // Науковий журнал «Інтернаука. – 2018. – №7. – Електронні дані. – Режим доступу: https://www.inter-nauka.com/issues/2018/7/3687/ .
   - Телелейко І.С., Клятченко Я.М. Інтеграція сервіс-орієнтованої архітектури з Grid-системами // ПМК-2016, Київ, 20-22 квітня 2016 р.: Збірник тез доповідей. – К.: Просвіта, 2016. – С.60-64.
   - Телелейко І.С., Орлова М.М. Сумісність підходів хмарних обчислень та грід-технологій // SAIT 2017, Київ, 22-25 травня 2017 р. / ННК «ІПСА» НТУУ «КПІ ім. Сікорського». – К.: ННК «ІПСА» НТУУ «КПІ ім. Ігоря Сікорського», 2017. – 340 с.
   - Телелейко І. С., Орлова М.М. Аналіз методів динамічного балансування навантаження в хмарному середовищі // ПМК-2018, Київ, 21-23 березня 2018р.: Збірник тез доповідей. – К.: Просвіта, 2018.

8. Дата видачі завдання 5 вересня 2016 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Ґрунтовне ознайомлення з предметною галуззю	16.12.2016	
2.	Визначення структури магістерської дисертації; вивчення літератури, пошук додаткової літератури, патентний пошук	05.03.2017	
3.	Робота над першим розділом магістерської дисертації; проведення наукового дослідження	15.05.2017	
4.	Проведення наукового дослідження; робота над другим розділом магістерської дисертації; розроблення програмного забезпечення	10.10.2017	
5.	Проведення наукового дослідження; робота над статтею за результатами наукового дослідження	17.12.2017	
6.	Проведення наукового дослідження; робота над третім розділом магістерської дисертації;	21.02.2018	
7.	Завершення роботи над основною частиною магістерської дисертації; підготовка ілюстративного матеріалу;	16.04.2018	
8.	Оформлення текстової і графічної частини магістерської дисертації	20.04.2018	
	Попередній розгляд магістерської дисертації на кафедрі	26.04.2018	

Студент _____ Телелейко І. С.
(підпис)

Науковий керівник дисертації _____ Орлова М. М.
(підпис)

РЕФЕРАТ

Актуальність теми. Потребу в плануванні навантаження необхідно вирішувати на початковій стадії розвитку будь-якого проекту. Проблеми недостатньої продуктивності сервера в зв'язку зі зростанням навантаження можна вирішувати шляхом нарощування потужності сервера або ж за рахунок оптимізації використовуваних алгоритмів, програмних кодів тощо. Але рано чи пізно настає момент, коли і ці заходи виявляються недостатніми та неефективними. Основними характеристиками хмарних обчислень є масштабованість, еластичність, мобільність, необмежений обсяг даних, що оброблюються, та можливість нарощувати ресурси. Актуальним є аналіз існуючих підходів балансування навантаження в хмарному середовищі та експериментальне дослідження ефективності створених правил балансування навантаження .

Об'єкт дослідження – системи балансування робочого навантаження та планування ресурсів в хмарних технологіях.

Предмет дослідження – методи і алгоритми балансування навантаження на модулі у хмарному середовищі, критерії їх порівняння, узагальнена модель хмарного додатку, обґрунтування вибору правил балансування навантаження.

Методи досліджень – проведення класифікації методів і алгоритмів балансування навантаження в хмарних системах, розробка порівняльної характеристики розповсюджених систем балансування навантаження, надання практичних рекомендацій по налаштуванню системи балансування навантаження, експериментальне дослідження ефективності створених правил балансування навантаження.

Мета роботи: підвищення функціональної ефективності та ступеня рівномірного завантаження модулів хмарної інфраструктури. Для цього визначено завдання, які вирішуються в роботі:

1. Проведення систематизації методів і алгоритмів балансування навантаження в хмарних системах.

2. Розробка порівняльної характеристики розповсюджених систем балансування навантаження.

3. Розробка модифікованого способу динамічного балансування навантаження в хмарному середовищі на основі методу міграції задач та алгоритму Weighted Least Connections.

4. Експериментальне дослідження ефективності створених правил балансування навантаження.

Наукова новизна одержаних результатів полягає в наступному:

1. Проведено аналітичний огляд літературних джерел технології хмарних обчислень, їх загальні характеристики, особливості, структуру та архітектуру та показано, що на сьогоднішній день залишається невирішеним питання балансування навантаження у хмарному середовищі.

2. Запропоновано модифікований спосіб динамічного балансування навантаження в хмарному середовищі, який відрізняється від відомих способів балансування одночасним застосуванням методу міграції задач та алгоритму Weighted Least Connections, що дозволяє зменшити час обробки завдань.

3. Проведено експериментальні дослідження, які показали, що час обробки завдань при використанні методу міграції задач та алгоритму Weighted Least Connections зменшується від 30% до 10% (у порівнянні з алгоритмом FCFS та з алгоритмом Max-Min відповідно).

Практична цінність одержаних в роботі результатів дозволяють зменшити час обробки завдань при навантаженні на модуль вище максимально допустимої границі завданнями та роботи до 10 активних модулів, застосувавши метод міграції задач та алгоритму Weighted Least Connections, а саме алгоритм динамічного навантаження з урахуванням часу проходження та середнього коефіцієнта використання ресурсів як параметра.

Апробація роботи. Основні положення і результати роботи представлені та обговорені на:

- VIII науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг – ПМК'2016» (Київ, 20 – 22 квітня 2016 року);
- 19-ій Міжнародній науково-технічній конференції «Системний аналіз та інформаційні технології» SAIT 2017 (Київ, 22-25 травня 2017 року);
- X конференції молодих вчених «Прикладна математика та комп'ютинг – ПМК'2018» (Київ, 21 – 23 березня 2018 року).

Публікації. За темою досліджень опубліковані 4 наукові праці, з яких 1 стаття в науковому фаховому виданні, що реферується наукометричними базами та 3 тези доповідей на конференціях.

Структура та обсяг роботи. Магістерська дисертація складається з вступу, чотирьох розділів та висновків.

У *вступі* подано загальну характеристику роботи, зроблено оцінку сучасного стану проблеми, обґрунтовано актуальність напрямку досліджень, сформульовано мету і задачі досліджень, показано наукову новизну отриманих результатів і практичну цінність роботи.

У *першому розділі* наведено аналітичний огляд літературних джерел технології хмарних обчислень, їх загальні характеристики, особливості, структуру та архітектуру.

У *другому розділі* наведено та проаналізовано розповсюджені алгоритми та методи динамічного балансування навантаження. Визначено ті алгоритми, з якими будемо проводити порівняльний аналіз на основі експериментальних досліджень.

У *третьому розділі* розроблено правил балансування та проведено навантажувальне тестування. Для оцінки запропонованого методу була розроблена модель динамічного балансування навантаження в хмарному середовищі на основі методу міграції задач та модернізованого алгоритму Weighted Least Connections.

У *четвертому розділі* наведені та проаналізовані результати експериментальних дослідження ефективності створеного способу динамічного балансування навантаження.

У *висновках* представлені результати проведеної роботи.

Робота представлена на 70 аркушах, містить 10 рисунків, 5 таблиць та посилання на список використаних літературних джерел з 21 найменувань.

Ключові слова: хмарне середовище, еластичність, ресурси робочого навантаження, ефективність планування, алгоритми балансування навантаження, масштабування.

РЕФЕРАТ

Актуальность темы. Потребность в планировании нагрузки необходимо решать на начальной стадии развития любого проекта. Проблемы недостаточной производительности сервера в связи с ростом нагрузки можно решать путем наращивания мощности сервера или же за счет оптимизации используемых алгоритмов, программных кодов и тому подобное. Но рано или поздно наступает момент, когда и эти меры оказываются недостаточными и неэффективными. Основными характеристиками облачных вычислений является масштабируемость, эластичность, мобильность, неограниченный объем данных, обрабатываемых и возможность наращивать ресурсы. Актуальным является анализ существующих подходов балансировки нагрузки в облачной среде и экспериментальное исследование эффективности созданных правил балансировки нагрузки.

Объект исследования - системы балансировки рабочей нагрузки и планирования ресурсов в облачных технологиях.

Предмет исследования - методы и алгоритмы балансировки нагрузки на модули в облачной среде, критерии их сравнения, обобщенная модель облачного приложения, обоснование выбора правил балансировки нагрузки.

Методы исследований - проведение классификации методов и алгоритмов балансировки нагрузки в облачных системах, разработка сравнительной характеристики распространенных систем балансировки нагрузки, предоставление практических рекомендаций по настройке системы балансировки нагрузки, экспериментальное исследование эффективности созданных правил балансировки нагрузки.

Цель работы: повышение функциональной эффективности и степени равномерной загрузки модулей облачной инфраструктуры. Для этого определены задачи, которые решаются в работе:

1. Проведение систематизации методов и алгоритмов балансировки нагрузки в облачных системах.

2. Разработка сравнительной характеристики распространенных систем балансировки нагрузки.

3. Разработка модифицированного способа динамической балансировки нагрузки в облачной среде на основе метода миграции задач и алгоритма Weighted Least Connections.

4. Экспериментальное исследование эффективности созданных правил балансировки нагрузки.

Научная новизна исследования заключается в следующем:

1. Проведен аналитический обзор литературных источников технологии облачных вычислений, их общие характеристики, особенности, структуру и архитектуру и показано, что на сегодняшний день остается нерешенным вопрос балансировки нагрузки в облачной среде.

2. Предложен модифицированный способ динамической балансировки нагрузки в облачной среде, который отличается от известных способов балансировки одновременным применением метода миграции задач и алгоритма Weighted Least Connections, что позволяет уменьшить время обработки заданий.

3. Проведены экспериментальные исследования, которые показали, что время обработки заданий при использовании метода миграции задач и алгоритма Weighted Least Connections уменьшается от 30% до 10% (по сравнению с алгоритмом FCFS и с алгоритмом Max-Min соответственно).

Практическая ценность полученных в работе результатов позволяют уменьшить время обработки задач при нагрузке на модуль выше максимально допустимого предела задачами и работы до 10 активных модулей, применив метод миграции задач и алгоритма Weighted Least Connections, а именно алгоритм динамической нагрузки с учетом времени прохождения и среднего коэффициента использования ресурсов в качестве параметра.

Апробация работы. Основные положения и результаты работы представлены и обсуждены на:

• VIII научной конференции магистрантов и аспирантов «Прикладная математика и компьютинг – ПМК-2016» (Киев, 20 - 22 апреля 2016)

• девятнадцатой Международной научно-технической конференции «Системный анализ и информационные технологии» SAIT 2017 (Киев, 22-25 мая 2017 года);

• X конференции молодых ученых «Прикладная математика и компьютинг – ПМК-2018» (Киев, 21 - 23 марта 2018).

Публикации. По теме исследований опубликованы 4 научные работы, из которых 1 статья в научном профессиональном издании, реферируется наукометрическими базами и 3 тезисы докладов на конференциях.

Структура и объем работы. Магистерская диссертация состоит из введения, четырех глав и выводов.

Во *введении* представлена общая характеристика работы, произведена оценка современного состояния проблемы, обоснована актуальность направления исследований, сформулированы цели и задачи исследований, показано научную новизну полученных результатов и практическую ценность работы.

В *первом разделе* приведены аналитический обзор литературных источников технологии облачных вычислений, их общие характеристики, особенности, структуру и архитектуру.

Во *втором разделе* приведены и проанализированы распространенные алгоритмы и методы динамической балансировки нагрузки. Определены те алгоритмы, с которыми будем проводить сравнительный анализ на основе экспериментальных исследований.

В *третьем разделе* разработаны правила балансировки и проведения нагрузочное тестирование. Для оценки предложенного метода была разработана модель динамической балансировки нагрузки в облачной среде на основе метода миграции задач и модернизированного алгоритма Weighted Least Connections.

В *четвертом разделе* приведены и проанализированы результаты экспериментальных исследования эффективности созданного образа динамической балансировки нагрузки.

В *выводах* представлены результаты проведенной работы.

Работа представлена на 70 листах, содержит 10 рисунков, 5 таблиц и ссылки на список использованных литературных источников из 21 наименований.

Ключевые слова: облачная среда, эластичность, ресурсы рабочей нагрузки, эффективность планирования, алгоритмы балансировки нагрузки, масштабирование.

ABSTRACT

Actuality of theme. The need for load planning needs to be addressed at the initial stage of development of any project. Problems of inadequate server performance due to increased load can be solved by increasing the server capacity or by optimizing the algorithms, program codes used, etc. However, eventually the moment comes when these measures are inadequate and ineffective. The main characteristics of cloud computing are scalability, elasticity, mobility, unlimited amount of data being processed, and the ability to build resources. An actual analysis of existing load balancing approaches in cloud environments and an experimental study of the effectiveness of load balancing rules created are relevant.

The object of the study - systems for balancing workload and resource planning in cloud-based technologies.

Subject of research - methods and algorithms for balancing load on modules in cloud environments, criteria for their comparison, generalized model of cloud application, justification for choosing load balancing rules.

Research methods - the classification of methods and algorithms of load balancing in cloud systems, the development of comparative characteristics of distributed load balancing systems, the provision of practical recommendations for adjusting the load balancing system, and the experimental study of the effectiveness of the created load balancing rules.

The purpose of the work: increase the functional efficiency and the degree of uniform loading of cloud infrastructure modules. To do this, the tasks that are solved in the work are determined:

1. Conducting systematization of methods and algorithms of load balancing in cloud systems.

2. Development of comparative characteristics of distributed load balancing systems.

3. Development of the modified method of dynamic balancing of load in a cloud environment based on the method of task migration and the Weighted Least Connections algorithm.

4. Experimental study of the efficiency of the created load balancing rules.

The scientific novelty of the results obtained is as follows:

1. An analytical review of the literary sources of cloud computing technology, their general characteristics, features, structure and architecture has been carried out and it has been shown that the problem of load balancing in the cloud environment remains unresolved for today.

2. A modified method for dynamically balancing load in a cloud environment is proposed, which is different from the known methods of balancing the simultaneous application of the task migration method and the Weighted Least Connections algorithm, which reduces the time spent processing tasks.

3. Experimental studies have been carried out which showed that the time of task processing using the method of task migration and the Weighted Least Connections algorithm is reduced from 30% to 10% (compared with the FCFS algorithm and the Max-Min algorithm, respectively).

The practical value of the results obtained in the work can reduce the time of task handling with the load on the module above the maximum permissible boundary of the task and work up to 10 active modules, using the method of problem migration and the Weighted Least Connections algorithm, namely the dynamic load algorithm, taking into account the passage time and the average usage factor resources as a parameter.

Test work. The main provisions and results of work are presented and discussed at:

• VIII scientific conference of masters and postgraduate students "Applied Mathematics and Computer - PMK'2016" (Kyiv, April 20 - 22, 2016);

• 19th International Scientific and Technical Conference "System Analysis and Information Technologies" SAIT 2017 (Kyiv, May 22-25, 2017);

• X Conference of Young Scientists "Applied Mathematics and Computing - PMK'2018" (Kyiv, March 21 - 23, 2018).

Publications. On the subject of research published 4 scientific works, of which 1 article in the scientific professional publication, which is referenced by science-based bases and 3 theses of reports at conferences.

Structure and scope of work. The master's thesis consists of an introduction, four chapters and conclusions.

The *introduction* gives a general description of the work, assesses the current state of the problem, substantiates the relevance of the research direction, formulates the purpose and objectives of the research, shows the scientific novelty of the results obtained and the practical value of the work.

The first section provides an analytical review of literary sources of cloud computing technology, their general characteristics, features, structure and architecture.

In *the second section*, distributed algorithms and methods of dynamic load balancing are presented and analyzed. The algorithms with which we will conduct a comparative analysis on the basis of experimental research are determined.

In *the third section*, the rules of balancing and load testing are developed. To evaluate the proposed method, a dynamic load balancing model in a cloud environment was developed based on the problem migration method and the modernized Weighted Least Connections algorithm.

The fourth section presents and analyzes the results of experimental studies of the effectiveness of the created method of dynamic load balancing.

The conclusions are the results of the work.

The work is presented on 70 sheets, contains 10 figures, 5 tables and a link to the list of used literary sources of 21 titles.

Keywords: cloud environment, elasticity, workload resources, planning efficiency, load balancing algorithms, scaling.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface, набір визначень взаємодії різнотипного програмного забезпечення.

ARUR – Average Resource Utilization Ratio.

CORBA – Common Object Request Broker Architecture, це запропонований консорціумом OMG технологічний стандарт розробки розподілених застосунків.

DDOS-атаки – Distributed Denial of Service, атака на комп'ютерну систему з наміром зробити комп'ютерні ресурси недоступними користувачам, для яких комп'ютерна система була призначена.

DNS – Domain Name System, програма, призначена для відповідей на DNS-запити за відповідним протоколом.

GUI – Graphical user interface, ип інтерфейсу, який дозволяє користувачам взаємодіяти з електронними пристроями через графічні зображення та візуальні вказівки, на відміну від текстових інтерфейсів, заснованих на використанні тексту, текстовому наборі команд та текстовій навігації.

HAProxy – Hidh Availability Proxy, вільне програмне забезпечення, проксі-сервер і балансувальник наватаження в системах з високою доступністю.

HTTP – HyperText Transfer Protocol, протокол прикладного рівня, де обмін повідомленнями йде за звичайною схемою «запит-відповідь».

HTTPS – HTTP Secure, хема URI, що синтаксично ідентична http: схемі, яка зазвичай використовується для доступу до ресурсів Інтернет.

IaaS – Infrastructure-as-a-service, е модель обслуговування, в межах якої споживачу надається можливість керувати засобами обробки та збереження, комунікаційними мережами, та іншими фундаментальними обчислювальними ресурсами.

IPv6 – нова (шоста) версія протоколу IP (Internet Protocol), яка прийшла на зміну четвертій версії IPv4.

IT – Information Technologies, сукупність методів, виробничих процесів і програмно-технічних засобів.

PaaS – Platform as a Service, модель надання хмарних обчислень, при якій споживач отримує доступ до використання інформаційно-технологічних платформ.

QoS – Quality of service, якість послуг, які надає комунікаційна мережа, набір методів для управління ресурсами пакетних мереж.

RMI – Remote Method Invoking, технологія звернення до віддалених методів, що дозволяє істотно спростити розробку розподілених систем.

RPC – Remote Procedure Calling, протокол, що дозволяє програмі, запущеній на одному комп'ютері, звертатись до функцій (процедур) програми, що виконується на іншому комп'ютері, подібно до того, як програма звертається до власних локальних функцій.

RTSP – Real Time Streaming Protocol, Потоковий протокол реального часу.

SaaS – Software as a Service, модель поширення через хмарне середовище програм споживачам, при якій постачальник розробляє веб-програму, розміщує її й управляє нею

SOAP – Simple Object Access Protocol, протокол обміну структурованими повідомленнями в розподілених обчислювальних системах, базується на форматі XML.

SSL – Secure Sockets Layer, протокол, що забезпечує конфіденційність та цілісність даних, котрі пересилаються між веб-серверами і веб-браузерами.

TCP/IP – (Transmission Control Protocol — «протокол керування передаванням») набір протоколів мережі Інтернет.

UDP – це один з найпростіших протоколів транспортного рівня моделі OSI, котрий виконує обмін повідомленнями (датаграмами) без підтвердження та гарантії доставки.

URL-адреса – Uniform Resource Locator, єдиний вказівник на ресурс, стандартизована адреса певного ресурсу.

VDI– Virtual Desktop Infrastructure.

VMM – Virtual Machine Manager.

W3C – World Wide Web Consortium, головна міжнародна організація, що розробляє й впроваджує технологічні стандарти для всесвітньої павутини.

WLC – Weighted Least Connections

WSDL – Web Services Description Language, мова визначення інтерфейсу веб-сервісу заснована на XML, що описує функціональність веб-сервісу і спосіб доступу до нього.

XML – Extensible Markup Language, стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками.

ПЗ – програмне забезпечення.

ВСТУП

Сьогодні все більше установ і організацій прагнуть створити власні веб-сервіси як у мережі Інтернет так для корпоративних потреб на базі приватної або публічноїхмарної інфраструктури. Оскільки такі системи належать до високонавантажених, то можливості масштабування і балансування навантаження при їх побудові мають особливе значення. Від вибору і налаштування системи балансування навантаження, які б найкращим чином відповідали задачам створення конкретного сервісу в конкретних умовах, напряму залежить ефективність роботи сервісу і безперебійність надання послуг.

При побудові хмарної інфраструктури важливо вміти розподіляти навантаження між компонентами, проте існуючі рішення з балансування мають ряд обмежень. Потребу в плануванні навантаження необхідно вирішувати на початковій стадії розвитку будь-якого проекту. Проблеми недостатньої продуктивності сервера в зв'язку зі зростанням навантаження можна вирішувати шляхом нарощування потужності сервера або ж за рахунок оптимізації використовуваних алгоритмів, програмних кодів тощо. Але рано чи пізно настає момент, коли і ці заходи виявляються недостатніми та неефективними. Актуальним є аналіз існуючих підходів динамічного балансування навантаження в хмарному середовищі.

Основними характеристиками хмарних обчислень є масштабованість, еластичність, мобільність, необмежений обсяг даних, що оброблюються, та можливість нарощувати ресурси [1]. У даній роботі проводиться аналіз та досліджується балансування навантаження саме динамічного виду, методи та алгоритми.

Проблема балансування обчислювального навантаження розподіленого додатка виникає з тих причин, що:

- неоднорідна структура розподіленого додатка: різні логічні процеси вимагають різні обчислювальні потужності;

- неоднорідна структура обчислювального комплексу: різні обчислювальні вузли характеризуються різною продуктивністю;
- неоднорідна структура міжвузлової взаємодії: лінії зв'язку, що з'єднують вузли, можуть мати різні характеристики пропускної спроможності.

Метою даної роботи є оцінка можливостей використання алгоритмів балансування навантаження на ресурси хмарної інфраструктури та методів масштабування в хмарному середовищі, проведення систематизації, порівняння основних методів і алгоритмів балансування навантаження та методів масштабування додатків в хмарному середовищі, проведення порівняння розповсюджених систем балансування навантаження додатків, дослідження налаштування системи балансування навантаження, підвищення функціональної ефективності та надання відповідних практичних рекомендацій.

РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД ТЕХНОЛОГІЇ ХМАРНИХ ОБЧИСЛЕНЬ

1.1. Технології хмарних обчислень

Важливим аспектом стало застосування механізмів управління ресурсами в локально та глобально розподілених середовищах. Під «ресурсами» маємо на увазі все, що так чи інакше бере участь в обробці даних: обчислювальні кластери, сховища даних, файлові системи, програмне забезпечення, мережеве устаткування, яке забезпечує з’єднання ресурсів в єдину систему.

При виконанні розподіленої програми, комп’ютери, які приймають участь в обробці даних, з’єднуються з мережею Інтернет. Обмін повідомленнями відбувається за протоколом HTTP, оскільки обмін даними мережею за іншими портами та в іншому форматі без додаткових налаштувань фільтрується більшістю брандмауерами та проксі.

До появи веб-сервісів у світі вже існували технології, що дозволяли додаткам взаємодіяти на відстані, де одна програма могла викликати будь-який інший ресурс, який при цьому міг бути запущений на комп'ютері, розташованому в іншому місті або навіть країні. Такий віддалений виклик процедур називається RPC (Remote Procedure Calling). Прикладом таких технологій є CORBA (Common Object Request Broker Architecture), RMI (Remote Method Invoking). Ідея веб-сервісу полягала в створенні такого RPC, який би взаємодіяв з HTTP пакетом.

Перш ніж викликати віддалену процедуру, необхідно описати виклик в XML файлі формату SOAP (Simple Object Access Protocol, це XML розмітка, яка використовується в веб-сервісах). Все, відправляється через HTTP, спочатку перетворюється в XML опис SOAP, потім застосовується в HTTP пакеті та надсилається на інший комп'ютер в мережі по TCP/IP. WSDL (Web Services Description Language) представляє собою формат XML файлу для опису сервісів мережі як набору кінцевих операцій, які працюють за допомогою повідомлень, які містять документо-орієнтовану інформацію. Документ WSDL повністю описує зовнішній інтерфейс веб-сервісу. Він

надає інформацію про послуги, які можна отримати, скориставшись методами сервісу, і способи звернення до цих методів [7].

Далі розглянемо загальний підхід використання веб-сервісу. У веб-сервісах завжди є клієнт і сервер. Сервер – це веб-сервіс (endpoint: кінцева точка, куди доходять SOAP повідомлення від клієнта).

Основні кроки реалізації:

- опис інтерфейсу веб-сервісу;
- реалізація інтерфейс;
- запуск веб-сервісу;
- створення клієнта і віддалений виклик потрібного методу веб-сервісу.

Веб-сервіси складають єдину концепцію створення таких додатків, функції яких використовують за допомогою стандартних протоколів Інтернет. Реалізація виконується за допомогою технологій, які стандартизовані World Wide Web Consortium (W3C) [7]:

Але все частіше з’являється потреба у користувача в будь-який момент часу мати можливість в нехтуванні ресурсами, яка робоча станція не в змозі забезпечити. Підвищення продуктивності обчислень до прийнятного рівня часто можна досягнути шляхом перерозподілу обчислювальних ресурсів між задачами.

1.2. Основні характеристики хмарного середовища

Швидке зростання хмарних обчислень надає величезний потенціал ефективності, економії коштів та інновацій уряду, бізнесу та іншим. Ключовими властивостями хмари є можливість масштабувати та забезпечувати зберігання даних та обчислювальну потужність в динамічному, економічному вигідному стані, без потреби користувача керувати основною частиною технології.

Досягнення повного потенціалу хмарних обчислень вимагає співпраці між урядом, промисловістю та окремими користувачами. Щоб реалізувати цей трансформаційний потенціал, необхідно:

- зміцнити впевненість у хмарі у користувача;
- розробляти стандарти та необхідну інфраструктуру;
- розтлумачити закони та політику, які спрямовані на збільшення інвестицій у хмарні обчислення.

Користувачі хмар потребують впевненості в тому, що ризики у безпеці, пов'язані зі збереженням своїх даних та запуском програм у хмарних системах, розуміються та належним чином підпорядковуються. Це так само справедливо для урядових користувачів, оскільки вони є приватними користувачами.

Для досягнення необхідного рівня безпеки постачальники хмарних сервісів повинні прийняти в комплексі практичне використання та відомі процедури безпеки, включаючи:

- визнані, прозорі та перевірені критерії безпеки.
- ідентифікація, автентифікація та механізми контролю доступу, що відповідають ступеню чутливості даних.
- всебічне та постійне тестування безпеки до та після розгортання.

1.3. Принципи роботи

Гіпервізор - це монітор віртуальної машини (VMM), який дозволяє одночасно запускати численні віртуальні операційні системи в комп'ютерній системі. Ці віртуальні машини також називаються гостьовими машинами, і всі вони використовують обладнання фізичної машини, таке як пам'ять, процесор, сховище і інші пов'язані ресурси. Це покращує і покращує використання основних ресурсів.

Гіпервізор ізолює операційні системи від основної машини. Завдання гіпервізора - задовольнити потреби гостьовий операційної системи і ефективно управляти нею. Кожна віртуальна машина незалежна і не заважає один одному, хоча вони працюють на одній і тій же машині. Вони ніяк не пов'язані один з одним. Навіть часом одна з віртуальних машин виходить з

ладу або стикається з будь-якими проблемами, інші машини продовжують нормально працювати.

Гіпервізори поділяються на два типи. Перший тип - це гіпервізор з білим металом, який розгортається безпосередньо над системним обладнанням хоста без будь-яких базових операційних систем або програмного забезпечення. Деякі приклади гіпервізора типу 1 - гіпервізор Microsoft Hyper-V, VMware ESXi, Citrix XenServer. Другий тип - це розміщений гіпервізор, який працює як програмний рівень в одному фізичному операційному середовищі. Гіпервізор працює як окремий другий рівень поверх апаратного забезпечення, а операційна система працює як третій рівень. У число розміщених гіпервізора входять Parallels Desktop і VMware Player.

1.4. Моделі хмарного розміщення

В механізмі «4-3-2» середня цифра означає три основні моделі хмарних послуг. Всі вони можуть працювати як окремо, так і в комбінації один з одним.

Хмарні сервіси мають властивості, в деяких випадках автоматично, швидкого розгортання та швидкого розповсюдження, що забезпечує швидке масштабування. Для споживача такі властивості є доступними як послуга, причому мають необмежений характер і можуть бути доступними у будь-якій кількості в будь-який час.

Існують три основні моделі хмарних послуг:

- Software as a Service (програмне забезпечення як послуга) – постачальник послуг забезпечує програмним забезпеченням та програмами в мережі Інтернет. Користувачі підписуються на ПО і отримують доступ до нього через веб-інтерфейс або API вендора.
- Platform as a Service (Платформа як послуга) – постачальник послуг пропонує доступ до хмарного середовища, в якому користувач може створювати додатки. Підтримка базової інфраструктури здійснюється постачальником.

- Infrastructure asaservice (Інфраструктура як послуга) – постачальник надає клієнтам доступ до сховища, мережі, серверу та іншим обчислювальним ресурсам у хмарному середовищі з оплатою по факту використання.

Далі наведено головні особливості кожної з моделі. IaaS – це хмарне рішення, де користувачі використовують свої власні платформи та додатки в інфраструктурі постачальника послуг. Головними перевагами є:

- користувачі оплачують IaaS за запитом;
- інфраструктура масштабується залежно від потреб у комп'ютерній потужності та пам'яті;
- можна заощадити витрати на покупку та обслуговування свого власного програмного забезпечення;
- дані знаходяться в хмарі, тому немає єдиної точки відмови;
- підтримується віртуалізація задач адміністрування, що дозволяє виділити час для виконання іншої роботи.

PaaS – це хмарне рішення, в якому користувач може в межах хмарного середовища створювати розробку та керування додатками. Головними перевагами є:

- PaaS надає платформу з інструментами для тестування, розробки та розміщення додатків у тому ж середовищі;
- постачальник керує захистом, операційними системами, серверним програмним забезпеченням та резервним копіюванням;
- значно полегшується спільна робота, навіть якщо співробітники працюють віддалено.

SaaS – це хмарне рішення, де користувачі не встановлюють додаток на свій власний локальний пристрій, додатки знаходяться у віддаленій хмарній мережі, доступ до якої здійснюється через веб-інтерфейс або API. Такий підхід дозволяє користувачам зберігати і аналізувати дані та спільно

працювати над проектами незалежно від локального місця знаходження. Головними перевагами є:

- постачальник надає користувачу програмне забезпечення та програми на основі підписки;
- користувачам не потрібно встановлювати або оновлювати ПО, а також керувати ним;
- захист даних: будь-який збій обладнання не призводить до втрати даних.
- використання ресурсів масштабується в залежності від потреб в услугах.
- додатки доступні майже з будь-якого пристрою, підключеного до мережі Інтернет.

Для користувача всі три рівні методу можуть слугувати корисним інструментом як для розробки, так і для бізнесу. Але під даною концепцією лежить існування двох сторін одночасно. Всі три моделі мають свої переваги та недоліки, та розділяють зону відповідальності на зону користувача послуги та провайдера послуги. В таблиці 1 показано зона відповідальності, яку надає провайдер послуги.

Застосування хмарного сервісу не має супроводжуватися зниженням рівня захищеності. Передаючи на обробку провайдера свої дані, користувач повинен бути впевнений в тому, що дані обробляються строго відповідно до встановленого технологічного процесу, що враховує як організаційні, так і технічні вимоги щодо забезпечення безпеки. Порівняльна таблиця всіх трьох методів представлено в таблиці 1.4.1.

Тобто захист даних в хмарі починається на стороні клієнта і закінчується на стороні провайдера. Вибір методів захисту, так само як і принцип вибору бізнес-процесів для перенесення в хмару, повинен ґрунтуватися на аналізі ризиків.

Таблиця 1.4.1 Модель представлення хмарних послуг, розробка автора

	IaaS	**PaaS**	**SaaS**
Мережа	так	так	так
Програми	ні	ні	так
Сховище	так	так	так
Дані	ні	ні	так
Сервер	так	так	так
Віртуалізація	так	так	так
Операційна система	ні	так	так
Підпрограмне забезпечення	ні	так	так
Середовище виконання	ні	так	так

1.5. Переваги та недоліки

Однією з основних причин, по яким багато компаній все ще не наважуються перенести свої IT-технології в хмарне середовище, - це існування ризиків у безпеці, яку гарантують користувачам. Але існує безліч переваг, які все більше переконують у потребі використанні хмарних обчислень. Деякі з ризиків, пов'язані з хмарними обчисленнями, включають в себе:

(a)Віддаленість даних від споживача: хмарні сервери розташовані в віддалених місцях від фізичного місцезнаходження користувача. Це дає їм відчуття невпевненості в тому, що вони не контролюють свої дані, на відміну від традиційних форм, де сервери знаходяться в центрах обробки даних.

(b) Безпека: загроза безпеці в хмарному середовищі сильно відрізняється від різних типів атак, таких як шкідливе програмне забезпечення, атаки з використанням вірусів, розподілена відмова від обслуговування (DDOS-атаки,

Distributed Denial of Service) тощо. Кібер-кримінальна діяльність теж розвивається за допомогою цієї технології, і рано чи пізно вонивинайдуть нові форми атак, які можуть створити серйозні проблеми для бізнесу.

(c)Підтримка: можлива крадіжка даних і витік даних до конкурента незадоволеними співробітниками. Не всі постачальники хмарного хостингу пропонують аналогічні послуги. Рівень підтримки, який вони пропонують, буде відомий тільки тоді, коли фактично почнуть використовувати сервіс.

(d) Можливості підключення: хмарні обчислення вимагає стабільного і надійного підключення до Інтернету. Небезпека втрати підключення до Інтернету може заподіяти серйозної шкоди для бізнесу. З'єднання має бути швидким.

(e)Оприлюднення даних: у світлі подій останніх років існує ймовірність вторгнення правоохоронних органів до даних, які передаються або зберігаються в хмарному середовищі.

До головних переваг хмарних обчислень можна віднести:

(a) Економічність: дозволяє користувачам легко отримувати доступ до своїх даних з будь-якого куточка планети та економити час і ресурси. Користувачам надається рахунок за використання певної моделі хмарного середовища, яка дає їм можливість купувати ресурси, які їм потрібні.

(b) Безпека: провайдери хмарного середовища реплікують дані та зберігають їх на певних серверах. Дані шифруються при передачі, для впевненості у тому, що вони мають несанкціоноване використання, навіть якщо вони перехоплені.

(c) Оптимізована робота в команді: співробітники можуть підтримувати зв'язок один з одним та оновлювати інформацію, навіть якщо вони не знаходяться у службових приміщеннях і продовжують працювати без фізичної звітності в своєму офісі.

(d) Надійність: хмарне середовище забезпечує безперервність бізнесу. Навіть якщо один з серверів стикається з проблемою, служби доставляються на веб-сайти з інших серверів в кластері.

(e) Автоматизоване оновлення хмарного середовища: всі оновлення програмного забезпечення виконуються автоматично без будь-якого людського втручання.

(f) Аварійне відновлення: у концепції хмарного середовища сервери можуть розташовуватися де завгодно. Провайдери оновлюють системи автоматично, включаючи оновлення безпеки. Це заощаджує час і гроші, немає необхідності це робити самостійно.

1.6. Масштабування в хмарному середовищі

Балансування навантаження в хмарному середовищі відрізняється від класичної моделі архітектури балансування навантаження та її впровадження за допомогою серверів для виконання балансування навантаження, оскільки важко передбачити кількість запитів, які будуть передаватися на сервер. Це забезпечує нові можливості, а також представляє свій унікальний комплекс завдань. Балансування навантаження є однією з центральних проблем в області хмарних обчислень [9]. Це механізм, який рівномірно розподіляє динамічне локальне навантаження на всіх вузлах по всій хмарі, щоб уникнути ситуації, коли деякі вузли сильно завантажуються, тоді як інші не працюють. Це допомагає досягти підвищення кількості користувачів та використання ресурсів, а отже, покращення загальної ефективності та

економічної вигоди ресурсів системи. Він також гарантує, що кожен обчислювальний ресурс розподіляється ефективно та справедливо [10].

Масштабування серверів стає надзвичайно важливим, коли є потреба у збільшенні ресурсів, але ресурси сервера не можуть забезпечити таку технологічну підтримку, яку потрібно плавно розвивати. Якщо вам потрібно на постійній основі більше ресурсів, масштабування серверів є найкращим способом для досягнення цього.

Автоматичне масштабування - це один з найпопулярніших способів застосування нових серверів. Система може вгадати потрібні вам ресурси та автоматично масштабувати сервери для зміни перенавантаження. Необхідну кількість серверів можна як підключити для обробки запитів, але й відключити якщо навантаження на систему не перерозподіляє на них навантаження. Це може заощадити фінансові ресурси.

Споживачі хмарних обчислень можуть значно зменшити витрати на інфраструктуру інформаційних технологій в короткостроковому та/або середньостроковому планах і швидко реагувати на зміни обчислювальних потреб, використовуючи властивості обчислювальної еластичності хмарних послуг.

Для визначення ефективності алгоритму балансування навантаження, виділяємо наступні властивості [5]:

- рівномірне завантаження ресурсів системи;
- масштабування: алгоритм повинен зберігати працездатність при збільшенні навантаження;
- передбачуваність: потрібно чітко розуміти, в яких ситуаціях і при яких навантаженнях алгоритм буде ефективним для вирішення поставлених завдань.

Балансування навантаження додатку має прямий вплив на швидкість, яку потрібно досягти, а також на продуктивність паралельної системи [3, 4].

Проблема балансування обчислювального навантаження розподіленого додатка виникає з тих причин, що [5]:

- неоднорідна структура розподіленого додатка: різні логічні процеси вимагають різні обчислювальні потужності;
- неоднорідна структура обчислювального комплексу: різні обчислювальні вузли характеризуються різною продуктивністю;
- неоднорідна структура міжвузлової взаємодії: лінії зв'язку, що з'єднують вузли, можуть мати різні характеристики пропускної спроможності.

Первинною метою оптимізації балансування навантаження є перерозподіл збалансованого навантаження за допомогою завдань та мінімізація потреб між процесами зв'язку з оптимальним використанням ресурсів та часом роботи. Отже, покращення продуктивності обчислювальних вузлів шляхом вирівнювання робочих навантажень елементів обробки є метою балансування навантаження.

Алгоритм балансування навантаження має такі головні властивості [6]:

- продуктивність: збільшення ефективності системи оптимальним чином, наприклад, скоротити час відповіді завдання, зберігаючи при цьому затримку відповіді;
- одноманітність роботи: обробляти всі робочі місця в системі однаково незалежно від їхнього походження або положення;
- відмовостійкість: зберігати витривалість роботи при частковій відмові в системі;
- модифікованість: має можливість модифікувати себе відповідно до будь-яких змін або розширити в конфігурації розподіленої системи;
- системна стабільність: можливість обліку в надзвичайних ситуаціях, таких як раптове збільшення навантаження, так що продуктивність системи не погіршується за межі певного порогового значення.

Балансування навантаження може здійснюватися за допомогою як апаратних, так і програмних інструментів. Застосування динамічного

балансування з урахуванням поточного завантаження серверів дозволяє побудувати хмарне середовище, яке оптимально використовує всі наявні ресурси.

Балансування навантаження досягається в середовищі хмари у два етапи: по-перше, це розподіл завдання серед вузлів, другий полягає в тому, щоб відстежувати віртуальну машину та виконувати операцію балансування навантаження за допомогою міграції завдань або підходу міграції віртуальної машини. Метою планування завдань є створення графіка і присвоєння кожного завдання вузлу (віртуальній машині) за певний період часу, так що всі завдання виконуються за мінімальний проміжок часу. Планування завдань – кількість та розмір завдань дуже швидко змінюється в середовищі хмари. Важко підрахувати всі можливі картографічні завдання-ресурси в хмарному середовищі. Тому необхідний оптимальний алгоритм планування виконання завдань, який може ефективно поширювати завдання, щоб менша кількість віртуальної машини була перевантажена або завантажена. Розподіляючи завдання на віртуальну машину, планувальник задач розподілу навантаження починає виконувати операцію балансування навантаження, щоб завдання перерозподілялися від перевантаженої до завантаженої віртуальної машини, і таким чином зберігається рівновага системи. Три основні етапи потрібні для планування завдання в хмарному середовищі (рисунок 1.6.1).

На етапі користувача подаються робочі місця через графічний інтерфейс користувача або веб-інтерфейс з вимогою на обслуговування в частині якості обслуговування (QoS, Quality of service), апаратного забезпечення, програмного забезпечення тощо.

На етапі планувальника фазових завдань виконуються всі завдання планування завдання та операції балансування навантаження. Обробник запиту на роботу переадресовує автентичний запит до планувальника завдань для подальшої обробки, де формується відповідь всім завданням відповідної

віртуальної машини та призначеного завдання. Планувальник завдань містить інформацію про стан всіх вузлів (незайняті або зайняті).

На останньому етапі в хмарному середовищі формується базова архітектура планування виконання завдань. Такий етап називається фазою на рівні хмари. Центр обробки даних містить хости, і кожен хост містить віртуальну машину. Кількість віртуальних машин може збільшуватися і зменшуватися під час виконання та залежить від потужності хосту і кількості майбутніх запитів користувача.

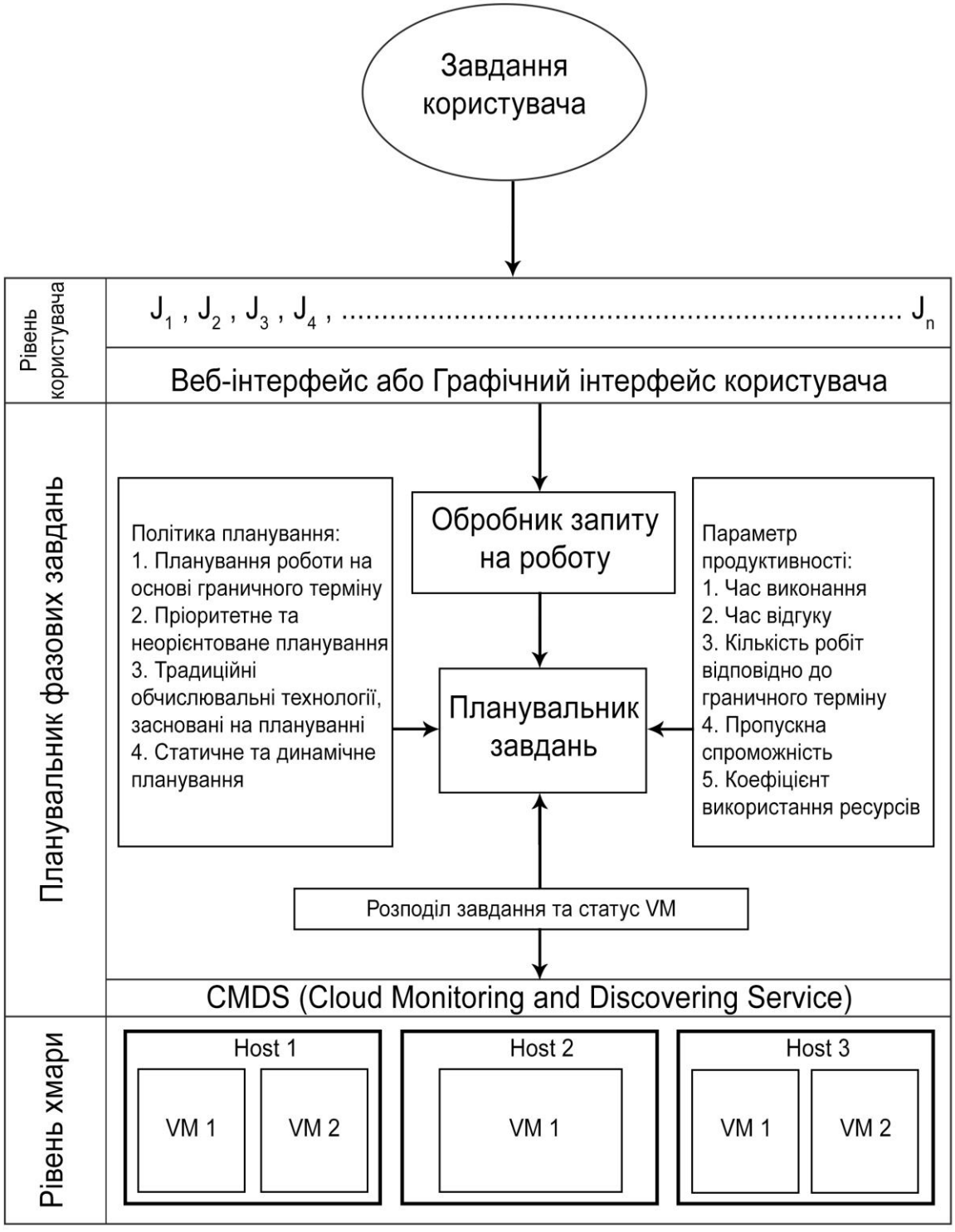


Рисунок 1.6.1 – Компоненти, необхідні для планування завдань у Cloud computing

Висновки до розділу 1

На основі проведеного деталізованого аналітичного огляду технології хмарних обчислень показано, що споживачі хмарних обчислень можуть значно зменшити витрати на інфраструктуру інформаційних технологій (в

короткостроковому і середньостроковому планах) і гнучко реагувати на зміни обчислювальних потреб, використовуючи властивості обчислювальної еластичності хмарних послуг.

Наведено та проаналізовано принципи роботи хмарного середовища, проведено аналіз особливостей моделей хмарного розміщення, а саме: програмне забезпечення як послуга (SaaS), платформа як послуга (PaaS) та інфраструктура як послуга (IaaS). Виділено переваги та недоліки хмарних технологій, які показують, що завдяки об’єднанню ресурсів та непостійному характеру споживання з боку користувачів, можна забезпечити економію в масштабі системи, використовуючи менші апаратні ресурси, ніж при виділенні апаратних потужностей для кожного споживача, а завдяки автоматизації модифікації виділення ресурсів значно знижуються витрати на обслуговування.

Але при цьому використання хмарного середовища має ряд невирішених питань. Гострим на сьогодні залишається питання безпеки, конфіденційності та балансування навантаження у хмарному середовищі.

РОЗДІЛ 2. ПОРІВНЯЛЬНИЙ АНАЛІЗ РОЗПОВСЮДЖЕНИХ АЛГОРИТМІВ ТА МЕТОДІВ БАЛАНСУВАННЯ НАВАНТАЖЕННЯ

2.1. Головні поняття про балансування навантаження

При проектуванні будь-яких інформаційних систем однієї з істотних завдань є планування її продуктивності. Адже основним критерієм якості є отримання відповіді на надіслані запити та швидка реакція на дії. Значно легше досягнути це застосувавши більш потужне обчислювальне обладнання, але як будь-який ресурс воно має свою вартість. Тому в реальних умовах є необхідність у знаходженні розумного співвідношення між очікуваним навантаженням на проектовану інформаційну систему та характеристиками її апаратної частини. Пошук правильного номінального рівня продуктивності представляє особливий потенціал у сфері публічних інформаційних систем типу соціальних мереж, інтернет-магазинів, дошок оголошень, відео хостингу тощо. Наприклад, після якої-небудь реклами навантаження на даний вид послуги може за лічені години змінитися навіть не в рази, а на порядок. У комерційному плані, це - дуже добре, але якщо навантаження на сайт перевищити на певний критичний поріг, він просто перестане працювати, і гроші, витрачені на рекламу, не принесуть користі. Навпаки, непрацюючий сайт може викликати зворотний, негативний, ефект.

Потреба змінення продуктивності інформаційної системи може виникнути і в менш гострих ситуаціях. Наприклад, при поступовому збільшенні навантаження або при очевидній надмірності спочатку запланованої потужності.

Розуміючи можливість виникнення таких ситуацій, існує необхідність у передбаченні масштабування, тобто можливість зміни її продуктивності за допомогою невеликих змін, що не порушують фундаментальну архітектуру системи.

2.2. Основні цілі та вимоги до системи балансування

В першу чергу, балансування застосовується для розподілу навантаження між нашими серверами. По-друге, за рахунок балансування ми можемо підвищити відмовостійкість нашої системи, тобто, наприклад, якщо один з наших серверів в кластері виходить з ладу, то навантаження на себе бере другий, якщо зможе її потягнути. Завдяки балансуванню також досягається якась захист від деяких видів атак, наприклад, атаки на всі з'єднання.

До балансуванню, як і до будь-якій системі, пред'являються якісь вимоги. Основні цілі:

- розподілення навантаження між серверами;
- збільшення відмовостійкості;
- захист від деяких видів атак.

Необхідні вимоги до системи балансування:

- справедливість;
- ефективність;
- зменшення часу виконання запиту;
- зменшення часу відгуку;
- передбачуваність;
- рівномірне завантаження ресурсів системи;
- масштабування.

Метою даного дослідження є оцінка можливостей запропонованого методу динамічного балансування навантаження в хмарному середовищі та визначення такого розподілу завдань, який забезпечує приблизно однакове обчислювальне навантаження та мінімальні витрати на передачу даних між ними. Динамічне балансування передбачає перерозподіл обчислювального навантаження на вузли під час виконання програми.

Головні вимоги до програмного забезпечення для реалізації динамічного балансування:

- завантаження вузлів;
- пропускна спроможність ліній зв'язку;
- частота обмінів повідомленнями між логічними процесами розподіленого додатка.

Технологія управління розподіленими ресурсами є одною з найважливіших задач, яка направлена на забезпечення керування інформаційної інфраструктури в умовах значного зростання навантаження та збільшення кількості компонентів мережі.

2.3. Методи балансування навантаження в хмарних системах

При збільшенні навантаження можна вирішити дане питання у два способи: збільшити продуктивність сервера або додати нові сервера і об'єднати їх у кластер. Обидва способи допоможуть впоратися з навантаженням, але у першого методу не передбачається забезпечення відмовостійкості, так як мало користі від потужного сервера, якщо він не працює. У кластері забезпечується цей аспект: «впав» один сервер - підхопили інші. Межі для масштабування у другому варіанті немає.

Для об'єднання декількох серверів в кластер постає актуальність та необхідність розподіляти навантаження між ними. Це завдання вирішує балансувальник.

Концептуально схеми балансування навантаження можна розділити на два типи: статичні та динамічні. Статичне балансування виконується на етапі проектування розподіленого додатка. Дуже часто при розподілі логічних процесів на процесори використовується досвід попередніх запусків, застосовуються генетичні алгоритми. Однак попереднє розміщення логічних процесів між процесорами неефективне.

Статичне балансування навантаження

Балансування навантаження - це один з типів планування, який є або статичним, або динамічним середовищем щодо конфігурації хмари. Статичний алгоритм послідовно розподіляє трафік на сервери. Цей алгоритм

вимагає попереднього знання системних ресурсів. Отже, рішення не залежить від поточного стану системи. Динамічні алгоритми, які приймають рішення, базуються на фактичному стані системи, і дозволяють переміщатися з машини з перевантаженням на невикористану машину в режимі реального часу. Алгоритм змішаного завантаження балансу фокусується на симетричному розподілі

заданого обчислювального завдання та зниження вартості зв'язку розподілених обчислювальних вузлів.

Необхідно чітко розуміти, що хмарні обчислення потрапляють в динамічне середовище, щоб ми хотіли б зосередити алгоритми динамічного навантаження на навантаження. Цей алгоритм можна класифікувати на два аромати. Один з них називається періодичним плануванням пакетного режиму, тобто після збору прибуття завдання призначаються відповідним ресурсам, а інший називається плануванню негайного режиму, який передбачає, що завдання призначаються ресурсам відразу на основі мінімального часу завершення та мінімального часу виконання.

Алгоритм планування повинен впливати на такі характеристики:

- мінімальний час очікування;
- мінімальний час відгуку;
- максимальна пропускна спроможність;
- максимальна завантаженість процесора.

Динамічне балансування навантаження

Динамічне балансування передбачає перерозподіл обчислювального навантаження на вузли під час виконання програми. Програмне забезпечення, що реалізує динамічне балансування, враховує:

- завантаження обчислювальних вузлів;
- пропускну спроможність ліній зв'язку;
- частоту обмінів повідомленнями між логічними процесами розподіленого додатка тощо.

Балансування навантаження може здійснюватися за допомогою як апаратних, так і програмних інструментів. Застосування динамічного балансування з урахуванням поточного завантаження серверів дозволяє побудувати хмару хмарне середовище, яке оптимально використовує всі наявні ресурси.

Алгоритм динамічного навантаження не передбачає жодних попередніх знань про дії в роботі або глобальному стані системи, тобто рішення щодо балансування навантаження. Вони базуються виключно на існуючому або поточному стані системи. У розподіленій системи алгоритм динамічного навантаження виконується всіма вузлами, які присутні в системі, і завдання балансування навантаження розподіляється між ними. Взаємодія між вузлами для здійснення балансування навантаження може мати дві форми: кооперативну та некооперативну.

У кооперативній формі вузли працюють поруч один за одним, щоб досягти спільної мети, наприклад, для просування загального часу відгуку. У некооперативному варіанті кожен вузол працює незалежно в напрямку локальної для нього мети, наприклад, для просування часу відповіді локального завдання. Алгоритми динамічного розподілу навантаження, що мають розподілений характер, часто виробляють більше повідомлень, ніж нерозподілені, оскільки кожен з вузлів у системі повинен взаємодіяти з кожним іншим вузлом. Перевага такого підходу полягає в тому, що навіть якщо один або декілька вузлів не активуються, це не призведе до зупинки всього процесу балансування навантаження. Така система впливає на продуктивність системи.

У нерозподіленому типі або один вузол, або група вузлів виконують задачу балансування навантаження. Алгоритми динамічного розподілу навантаження нерозподіленого характеру можуть мати дві форми: централізовану та напіврозподілену. У централізованому алгоритмі балансування навантаження розподіляється лише через один вузол у загальній системі: центральний вузол. Він відповідає за балансування

навантаження всієї системи. Інші вузли взаємодіють лише з центральним вузлом. Однак у напіврозподіленій формі вузли поділяються на кластери, де балансування навантаження в кожному кластері має централізовану форму. У кожному кластері вибирається центральний вузол за допомогою відповідної методики вибору, яка забезпечує балансування навантаження всередині цього кластера, тобто балансування навантаження повної системи здійснюється через центральні вузли кожного кластера.

Централізоване динамічне навантаження вимагає менше повідомлень для прийняття рішення, оскільки кількість загальних взаємодій у системі різко зменшується в порівнянні з напіврозподіленою формою. Проте централізовані алгоритми можуть створювати вузьке місце в системі на центральному вузлі, а процес балансування навантаження виявляється безнадійним, коли центральний вузол виходить з ладу. Тому цей алгоритм, в основному, підходить для мереж невеликого розміру.

2.4. Алгоритми розподілення навантаження

Алгоритм динамічного навантаження не передбачає жодних попередніх знань про дії в роботі або глобальному стані системи, тобто рішення щодо балансування навантаження. Вони базуються виключно на існуючому або поточному стані системи. У розподіленій системи алгоритм динамічного навантаження виконується всіма вузлами, які присутні в системі, і завдання балансування навантаження розподіляється між ними. Взаємодія між вузлами для здійснення балансування навантаження може мати дві форми: кооперативну та некооперативну.

Round Robin або алгоритм кругового обслуговування, являє собою перебір по круговому циклу: перший запит передається одному серверу, потім наступний запит передається іншому і так до досягнення останнього сервера, а потім все починається спочатку, процеси розподілені між усіма процесорами [2]. Найпоширенішою імплементацією цього алгоритму є, звичайно ж, метод балансування Round Robin DNS. Однак розподіл робочого

навантаження між процесорами однаковий, а час обробки завдання для різних процесів різний. Тому в будь-який момент часу лише деякі вузли можуть бути дуже завантажені. Round Robin DNS часто використовується для завантаження запитів балансу між кількома веб-серверами. Головні показники ефективності: пропускна спроможність, час відгуку, витрати, використання ресурсів, продуктивність.

Max-Min алгоритм балансування навантаження виконує ідентичну процедуру алгоритму Min-Min. Цей алгоритм обчислює час завершення виконання всіх завдань. Максимальний час завершення приймається і призначається відповідним ресурсам. Цей алгоритм найкраще в ситуаціях, коли кількість завдань з максимальним часом завершення і забирає голодування. Мінімальний час завершення завдання чекав у замовленій черзі, доки не буде завершено інше завдання максимального часу завершення. Тут ми можемо зрозуміти, що цей алгоритм добре працює в статичному середовищі, і обидва алгоритми мають свої переваги та недоліки на основі середовища. Продуктивність не залежить від обраного алгоритму. Алгоритми Min-Min і Max-Min однаково виконуються у статичному хмарному середовищі.

Connection Mechanism або алгоритм балансування навантаження також може базуватися на найменшому механізмі зв'язку, який є компонентом алгоритму динамічного планування. Це вимагає підрахунку кількості підключень для кожного сервера динамічно для приблизного навантаження. Цей алгоритм балансування навантаження відстежує номер підключення кожного сервера. Кількість посилань додається, коли до неї надсилається нове з'єднання, і зменшується, коли закінчується з'єднання або відбувається тайм-аут [3]. Головні показники ефективності: пропускна спроможність, відмовостійкість, використання ресурсів, масштабування.

A Task Scheduling Algorithm Based on Load Balancing або алгоритм планування задач, що базується на балансуванні навантаження: дворівневий метод планування виконання завдань, що базується на балансуванні

навантаження для сприйняття динамічних вимог користувачів і отримання високого використання ресурсів. Це забезпечує балансування навантаження за допомогою першого планування завдань для віртуальних машин, зберігає розміщення ресурсів, збільшує час відповіді ресурсу, споживання ресурсів та загальної продуктивності середовища обчислень у хмарі.

Алгоритм балансування мінімальної місткості (Min-Min) – простий статичний алгоритм і пропонує відмінну продуктивність у плануванні завдань. Хмарний менеджер служби знаходить час завершення кожного завдання. Нове завдання чекало черги для виконання. Цей алгоритм присвоює завдання ресурсу на основі того, яке завдання має мінімальний час виконання для завершення.

Алгоритм Randomized має тип статичного характеру. У цьому алгоритмі процес можна обробляти певним вузлом *n* з ймовірністю *p*. Порядок розподілу процесу зберігається для кожного процесора незалежно від розподілу з віддаленого процесора. Цей алгоритм добре забезпечує процес рівномірного завантаження. З іншого боку, проблема виникає, коли навантаження мають різні обчислювальні складності. Рандомізований алгоритм не підтримує детерміністичний підхід.

Active Clustering – це кластерний алгоритм, який вводить поняття кластеризації в хмарні обчислення. У хмарних обчисленнях існує багато алгоритмів балансування навантаження. Кожен алгоритм має свої переваги та недоліки. Залежно від вимоги, використовується один з алгоритмів. Продуктивність алгоритму може бути покращена шляхом створення кластера вузлів. Кожен кластер можна вважати групою. Принцип активної кластеризації – об'єднати подібні вузли, а потім працювати над цими групами. Процес створення кластера обертається навколо концепції вузла збігання. Продуктивність системи підвищується завдяки високій доступності ресурсів, тим самим збільшуючи пропускну спроможність. Це збільшення пропускної спроможності пояснюється ефективним використанням ресурсів.

Головні показники ефективності: час міграції, витрати, використання ресурсів.

2.5. Основні системи балансування навантаження

Hidh Availability Proxy

Дана система балансування навантаження для сервісів TCP, який підходить для вирівнювання навантаження HTTP, оскільки підтримує постійне виконання сесії. HAProxy – це безкоштовне, уже швидке та надійне рішення , що пропонує високу доступність, балансування навантаження та проксі-сервер для програм. Така система призначена для веб-сайтів з дуже високим рівнем трафіку. На сьогоднішній день він став стандартним балансувальником відкритого ресурсу. HAProxy дуже часто використовують (розгортають) за замовчуванням у хмарних платформах.

Режим роботи такої системи робіть її інтеграцію в існуючу архітектуру дуже легкою та без ризиків. Але все ж пропонують не використовувати нестійкі веб-сервери в мережі.

HAProxy має декілька версій, які відрізняються одна від одної набором функцій. Наведемо основні з них для кожної з останніх версій:

- версія 1.8 – багатопотокова, HTTP / 2, кеш-пам'ять, додавання / видалення сервера, безперервне перезавантаження, DNS SRV, апаратні SSL-двигуни тощо;
- версія 1.7 – додана серверна швидка реконфігурація, містить агенти обробки вмісту, багатоструктурні сертифікати тощо;
- версія 1.6 – підтримка роздільної спроможності DNS, мультиплексування HTTP-з’єднання, повна реплікація стільникового з’єднання, стиснення безготівкових даних тощо.
- версія 1.6 – додані SSL, IPv6, захист DDoS тощо.

Кожна версія визначає свій набір функцій, які накладаються на попередню.

Підвищення сумісності є дуже важливим аспектом HAProxy, і навіть версія 1.5 може працювати з конфігураціями, зробленими для версії 1.0 13 років тому.

HAProxy передбачає декілька методів, що зазвичай зустрічаються в архітектурі операційної системи для досягнення абсолютної максимальної продуктивності [15]:

- модель, яка керується подіями, значно зменшує вартість контекстного перемикача та використання пам'яті. Можлива обробка декількох сотень завдань за мілісекунду, а використання пам'яті становить близько декілька кілобайт на сеанс, тоді як споживана пам'ять на попередньо встановлених або жорстких серверах більше декількох мегабайтів на процес;
- перевірка подій на системах, які цепідтримують (Linux і FreeBSD), що дозволяє миттєво виявляти будь-яку подію в будь-якому з'єднанні серед десятків тисяч;
- затримка оновлень перевірки подій за допомогою «ледачого» кеша подій; це гарантує, що ми ніколи не оновимо подію, якщо це не є обов'язковим. Це заощаджує багато системних викликів;
- буферизація без будь-якої копії даних між читанням та записом, коли це можливо. Це заощаджує багато циклів процесора та корисної пропускної спроможності пам'яті. Найчастіше вузьким місцем стає шина вводу-виводу між процесором і мережевими інтерфейсами. При 10-100 Гбіт/с пропускна спроможність пам'яті також може стати вузьким місцем;
- перенаправлення нульової копії, яка можлива за допомогою системного виклику splice() під Linux, в результаті якого здійснюється реальна нульова копія, починаючи з Linux

3.5. Це дозволяє невеликому пристрою під 3 Вт, пересилання HTTP-трафіку на один Гбіт/с;

- розподілювач пам'яті MRU з використанням пулів пам'яті з фіксованим розміром, що віддає перевагу ділянкам«гарячої» кеш-пам'яті над «холодними» кешами. Це значно скорочує час, необхідний для створення нового сеансу;
- можливість обмежувати кількість accept() для ітерації при роботі в багатопроцесному режимі, так що навантаження рівномірно розподіляється між процесами;
- адаптація до апаратного забезпечення та максимально можливого ядра процесора, який керує мережевими адаптерами, але не суперечить йому;
- оптимізована черга таймера;
- оптимізований аналіз заголовків HTTP: заголовки аналізуються інтерпретованими на льоту, а синтаксичний аналіз оптимізований, щоб уникнути повторного читання будь-якої раніше прочитаної області пам'яті. Контрольна точка використовується, коли кінцевий буфер досягається з неповним заголовком, так що синтаксичний аналіз не починається з самого початку, коли більше даних зчитаються. Розбір середнього HTTP-запиту зазвичай займає півмісяця на швидкому Xeon E5;
- ретельне зменшення кількості дорогих системних дзвінків . Більша частина роботи виконується в користувальному просторі за замовчуванням, наприклад, читання часу, агрегування буферу, включення та вимкнення файло-дескриптора;
- контент-аналіз оптимізовано для того, щоб носити лише покажчики на оригінальні дані та ніколи не копіювати, якщо дані не потрібно трансформувати. Це гарантує, що дуже малі структури переносяться і що вміст ніколи не реплікується, коли це не є абсолютно необхідним.

Пропозиція Cloud Computing варіює від пропозиції конкретної IT-інфраструктури до розгортання складних програм та програмних рішень. Вивчаючи модель доставки послуг Cloud Computing, виникає завдання управляти сотнями тисяч запитів користувачів і програм. Тому постачальник Cloud Computing повинен розглянути розумну інтеграцію інфраструктури, щоб створити пропозицію Cloud Computing, яка забезпечує прозорість, масштабованість, безпеку та найвищу якість (QoS).

CloudSim

Оцінка та оцінка хмарних обчислень є обов'язковою для постачальників Cloud, які планують надавати певні послуги та користувачі Cloud, які мають намір перенести свою IT-інфраструктуру, платформу або програмне забезпечення в Cloud (Інтернет у випадку публічної хмари або Intranet у випадку приватного хмари) Незважаючи на те, що використання реальної інфраструктури для тестування та оцінки розгортання хмар може дати дослідникам реальний підхід до прийняття критичного рішення про рух вперед з цією моделлю обчислень, в більшості випадків це може бути дуже дорогим:

- інфраструктури, платформи та програмне забезпечення великі витрати;
- необхідність тестування на масштабованих середовищах (більше інфраструктури);
- витрати на управління та технічне обслуговування.

Окрім цього критичного чинника, ми можемо додати споживання часу, щоб протестувати конкретний сценарій:

- інфраструктура установки та конфігурації;
- повторювані та змінні тести;
- налагодження та усунення несправностей.

Більш життєздатною альтернативою є використання інструментів моделювання. Ці інструменти дозволяють оцінити гіпотезу (дослідження

порівняльного аналізу застосування) в контрольованому середовищі, де можна легко відтворити результати.

Існує безліч інструментів симулятора для моделювання та моделювання великомасштабних середовищ хмарного обчислення.

Можна визначити два типи симуляторів: графічний інтерфейс користувача (GUI) або симулятори на основі мови програмування (наприклад, Java).

Як описано в попередньому розділі, існує різноманітна кількість симуляторів Cloud Computing, кожна з яких має специфічні характеристики та орієнтована на конкретну ціль. Вибір найкращого симулятора Cloud Computing - складна місія. Наскільки нам відомо, було важко визначити CloudSim як основну платформу для найбільш часто використовуваних симуляторів Cloud до цього моменту. CloudSim був створений як розширення симулятора GridSim, щоб представити шар віртуалізації Cloud Computing, який не був присутній на оригінальному симуляторі.

Підходящою альтернативою є використання інструментів моделювання, що дає можливість оцінити гіпотезу перед розробкою програмного забезпечення в середовищі, де можна відтворити тести. Зокрема, у випадку обласного обчислення, коли доступ до інфраструктури здійснює платежі в реальній валюті, підходи на основі моделювання дають значні переваги, оскільки це дозволяє Cloud клієнтам протестувати свої послуги в повторюваному та контрольованому середовищі безоплатно, а також настроювати продуктивність вузькі місця до розгортання на реальних хмарах. На стороні постачальника, середовища симуляції дозволяють оцінити різні види сценаріїв лізингу ресурсів при різному розподілі навантаження та ціноутворення. Такі дослідження могли б допомогти постачальникам оптимізувати вартість доступу до ресурсів з упором на підвищення прибутку.

CloudSim - це симулятор, заснований на мові програмування, і навіть якщо він не підтримує графічний інтерфейс користувача для моделювання,

він пропонує CloudAnalyst (який є розширенням CloudSim) для дослідників, які вважають за краще використовувати зручний інтерфейс для проведення своїх досліджень. CloudSim представляє себе дослідникам хмарних обчислень як Java-платформу, яка підтримує основні характеристики Cloud Computing (IaaS) з підтримкою віртуалізації та плануванням завдань (PaaS і SaaS) і відкриває двері для виникнення, інтеграції та тестування нових алгоритмів для планування завдань або розробка нових характеристик, що допомогло в доставці нових тренажерів.

Основні функції CloudSim:

- підтримка моделювання та моделювання великих центрів обробки даних хмарних обчислень;
- підтримка моделювання та моделювання віртуалізованих серверних хостів з налаштовуваною політикою для забезпечення ресурсів хоста віртуальних машин;
- підтримка моделювання та моделювання контейнерів для додатків;
- підтримка моделювання та моделювання енергозберігаючих обчислювальних ресурсів;
- підтримка моделювання та моделювання мережевих топологій центрів обробки даних та програм передачі повідомлень;
- підтримка моделювання та моделювання федеративних хмар;
- підтримка динамічної вставки елементів симуляції, зупинка та відновлення симуляції;
- підтримка визначеної користувачем політики для виділення хостів на віртуальні машини та політики для розподілу ресурсів хоста на віртуальні машини.

Висновки до розділу 2

На основі проведеного порівняльного аналізу існуючих алгоритмів та методів балансування навантаження у технології хмарних обчислень показано, що хмарні обчислення пропонують сервіс по мережі. Важливою проблемою хмарних обчислень є балансування навантаження. Перевантажена система забезпечує дуже низьку продуктивність і не забезпечує сервіс ефективно. Тому для сценарію потрібен економічний алгоритм балансування навантаження для створення сервісу, не розбиваючи його. У цьому розділі розглянуто методологічний аналіз різних алгоритмів балансування навантаження для послуг у мережі хмарності, концентруючись на балансуванні навантажень.

Також проаналізовані основні алгоритми балансування навантаження за різними показниками продуктивності, тобто для кожного алгоритму вказані показники ефективності, які в ньому використовуються. Існуючі способи балансування навантаження та алгоритми зосереджені на скороченні накладних витрат, зменшенні часу міграції та підвищення продуктивності. Це важливі завдання при розробці хмарної платформи, які дозволяють підвищити пропускну спроможність. Для подальших досліджень важливе розуміння всіх розглянутих методів балансування навантаження та підходів. Питання балансування навантаження хмарних сервісів потребує вдосконалення, а в багатьох аспектах – першочергових розробок та напрацювань.

Визначено алгоритм, який зосереджується на збільшенні навантаження, щоб задовольнити вимоги користувача для продовження сервісу шляхом розподілу робочого навантаження в балансовому порядку, щоб отримати максимальне використання ресурсів та зменшити час простою системи.

РОЗДІЛ 3. МОДИФІКОВАНИЙ СПОСІБ ДИНАМІЧНОГО БАЛАНСУВАННЯ НАВАНТАЖЕННЯ НА МОДУЛІ ТА ЙОГО АНАЛІЗ

3.1. Основні кроки балансування навантаження

Віртуальний сервер, налаштований на використання алгоритму Least Connection балансування навантаження, для передачі ресурсів вибирає службу з найменшою кількістю активних зв’язків. Це типовий метод, оскільки в більшості випадків він забезпечує найкращу продуктивність.

NetScaler Unified Gateway консолідує інфраструктуру віддаленого доступу для надання технології єдиного входу в усіх додатках, розташованих в центрі обробки даних або в хмарі, що використовуються у вигляді SaaS. Він дозволяє співробітникам отримувати доступ до будь-якої програми з будь-якого пристрою за єдиною URL-адресою. NetScaler Unified Gateway забезпечує кращу в своєму класі безпеку і бездоганну комфортність роботи користувача завдяки наданню безпечного доступу і технології єдиного входу за єдиною URL-адресою для корпоративних, VDI-, веб- і SaaS-додатків, розміщених в будь-якому центрі обробки даних або хмарі.

Для служб TCP, HTTP, HTTPS та SSL_TCP пристрій NetScaler включає в список існуючих з'єднань наступні типи з'єднання:

- активні підключення до сервісу;
- очікування зв'язків у наростаючій черзі.

У першому випадку, це з'єднання, що представляють запити, які клієнт відправив на віртуальний сервер і які віртуальний сервер переслав у службу. Для служб HTTP та HTTPS активні з'єднання представляють лише ті запити, на які ще не отримали відповіді. У другому випадку, будь-які з'єднання з віртуальним сервером, які чекають у черзі нагору і ще не пересилаються до послуги. Підключення можуть накопичуватися в черзі зависання в будь-який час з наступних причин:

- служби мають ліміт зв'язку, і всі послуги у конфігурації балансування навантаження мають таку межу;

- функція захисту від перенапруги налаштовується та активується за рахунок надходження запитів на віртуальний сервер;
- сервер із збалансованим навантаженням досяг внутрішньої межі, тому не відкриває жодних нових з'єднань.

Коли віртуальний сервер використовує Least Connection, він вважає, що очікувані з'єднання належать до певної служби. Тому він не відкриває нові зв'язки з цими службами.

Для служб UDP, з'єднання, що розглядає найменший алгоритм з'єднання, включають всі сеанси між клієнтом і службою. Ці сесії є логічними, тимчасовими об'єктами. Коли з'являється перший пакет UDP у сеансі, пристрій NetScaler створює сеанс між вихідною IP-адресою та портом та IP-адресою та портом призначення.

Для з'єднань потокового потоку (RTSP) пристрій NetScaler використовує кількість активних керуючих з'єднань для визначення найменшої кількості підключень до служби RTSP.

Наступний приклад показує, як віртуальний сервер вибирає службу для балансування навантаження за допомогою найменшого методу з'єднання. Розглянемо наступні три служби:

- сервіс-HTTP-1 - обробка 3 активних транзакцій;
- сервіс-HTTP-2 обробляє 15 активних транзакцій;
- сервіс-HTTP-3 не обробляє жодних активних транзакцій.

Наступна діаграма показує, як пристрій NetScaler пересилає вхідні запити при використанні найменшого методу з'єднання.

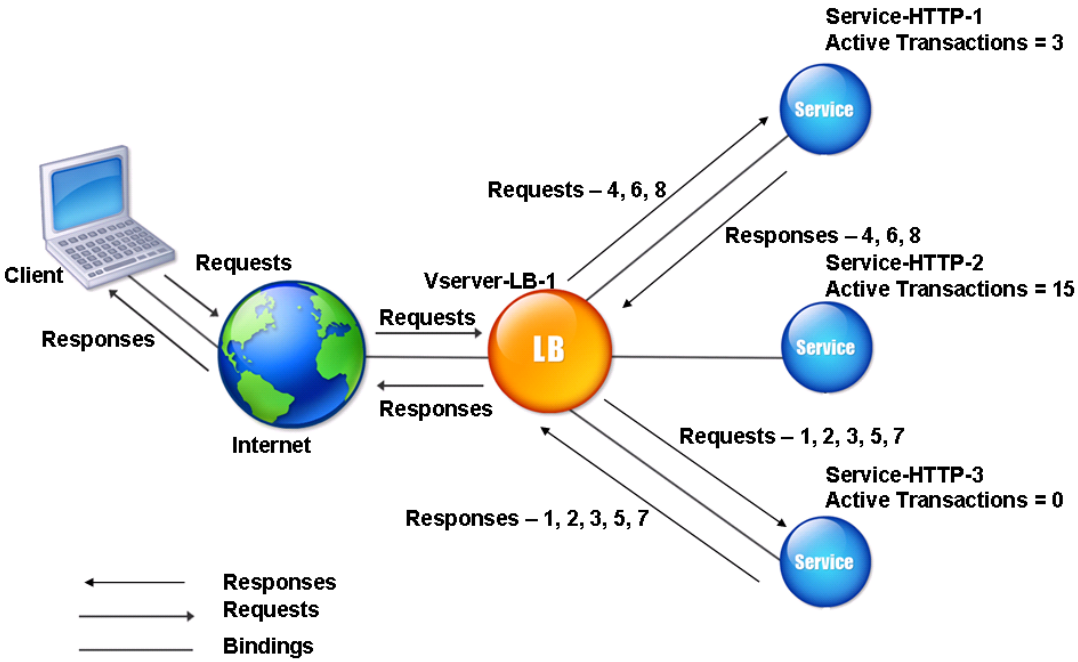


Рисунок 3.1.1 – Зображення дії алгоритму Least Connection балансування навантаження

На рисунку 3.1.1 віртуальний сервер вибирає послугу для кожного вхідного з'єднання, вибираючи сервер з найменшими активними транзакціями.

Підключення пересилаються наступним чином:

- сервіс HTTP-3 отримує перший запит, оскільки він не обробляє жодних активних транзакцій. Примітка. Сервіс без активної транзакції вибирається спочатку;
- сервіс HTTP-3 отримує другий і третій запити, оскільки служба має наступну найменшу кількість активних транзакцій;
- сервіс-HTTP-1 отримує четвертий запит. Оскільки Service-HTTP-1 та Service-HTTP-3 мають однакову кількість активних транзакцій, віртуальний сервер використовує круглий метод, щоб вибирати між ними;

- сервіс-HTTP-3 отримує п'ятий запит;
- сервіс-HTTP-1 отримує шостий запит тощо, доки служба-HTTP-1 та Service-HTTP-3 не оброблятимуть таку ж кількість запитів, що й Service-HTTP-2. У той час пристрій NetScaler починає перенаправляти запити до Service-HTTP-2, коли це найменш завантажений сервіс, або його черга з'являється в черговій черзі. Примітка: якщо з'єднання з Service-HTTP-2 закриті, він може отримати нові з'єднання, перш ніж кожна з двох інших служб матиме 15 активних транзакцій.

Прилад NetScaler також може використовувати найменший спосіб підключення, коли вага присвоюється службам. Він вибирає службу за допомогою значення (Nw) такого виразу:

Nw = (кількість активних транзакцій) * (10000 / вага)

Наступний приклад показує, як пристрій NetScaler вибирає службу для балансування навантаження, використовуючи найменший спосіб підключення, коли ваги присвоюються службам. У попередньому прикладі припустимо, що Service-HTTP-1 присвоюється вага 2, Service-HTTP-2 призначається вага 3, а Service-HTTP-3 присвоюється вага 4.

Підключення пересилаються наступним чином:

- сервіс-HTTP-3 отримує перше, оскільки служба не обробляє будь-які активні транзакції. Примітка. Якщо служби не обробляють жодних активних транзакцій, пристрій NetScaler використовує круглий метод, незалежно від ваг, призначених для кожного з цих служб;
- сервіс-HTTP-3 отримує друге, третє, четверте, п'яте, шосте та сьоме запити, оскільки служба має найменше значення Nw;
- сервіс-HTTP-1 отримує восьмий запит. Оскільки Service-HTTP-1 та Service-HTTP-3 тепер мають таке ж значення

Nw, NetScaler виконує балансування навантаження круглим способом. Тому Service-HTTP-3 отримує дев'ятий запит.

Наступний рисунок показує, як пристрій NetScaler використовує найменший спосіб підключення, коли ваги присвоюються службам на рисунку 3.1.2.

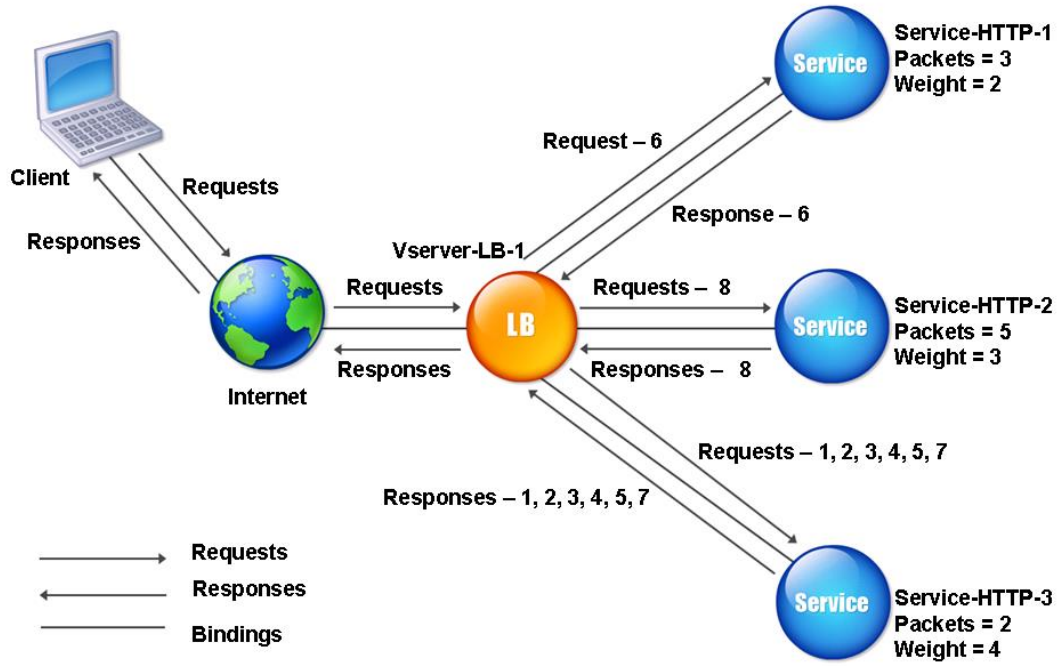


Рисунок 3.1.2 – Механізм Метод балансування навантаження з найменшими зв'язками при присвоєнні маси

3.2. Спосіб розподілення навантаження

Планування виконано *n* завдань в *m* вузлах (віртуальних машинах) має бути виконано таким чином, щоб користувач хмарного середовища міг виконувати своє завдання за мінімальний час роботи з максимальним використанням ресурсів. Планувальник задач отримує *N* запитів на завдання (задачі) $T_1, T_2 T_3 T_4 \dots \dots \dots \dots T_N$. У даній роботі не розглядається якість параметрів сервісу: пріоритет, вартість тощо. Всі завдання не є пріоритетними та незалежними, кожне завдання має довжину $T L_1$, p

швидкість обробки, кількість процесорів q, обсяг основної пам'яті r та обов'язково смугу пропускання B. Планувальник задач (рис. 3.2.1) в хмарному середовищі містить інформацію про віртуальну машину M, а саме швидкість обробки процесора, кількість процесорів, пам'ять, пропускну спроможність $VM_1, VM_2 VM_3 VM_4 \dots \dots \dots \dots VM_N$.

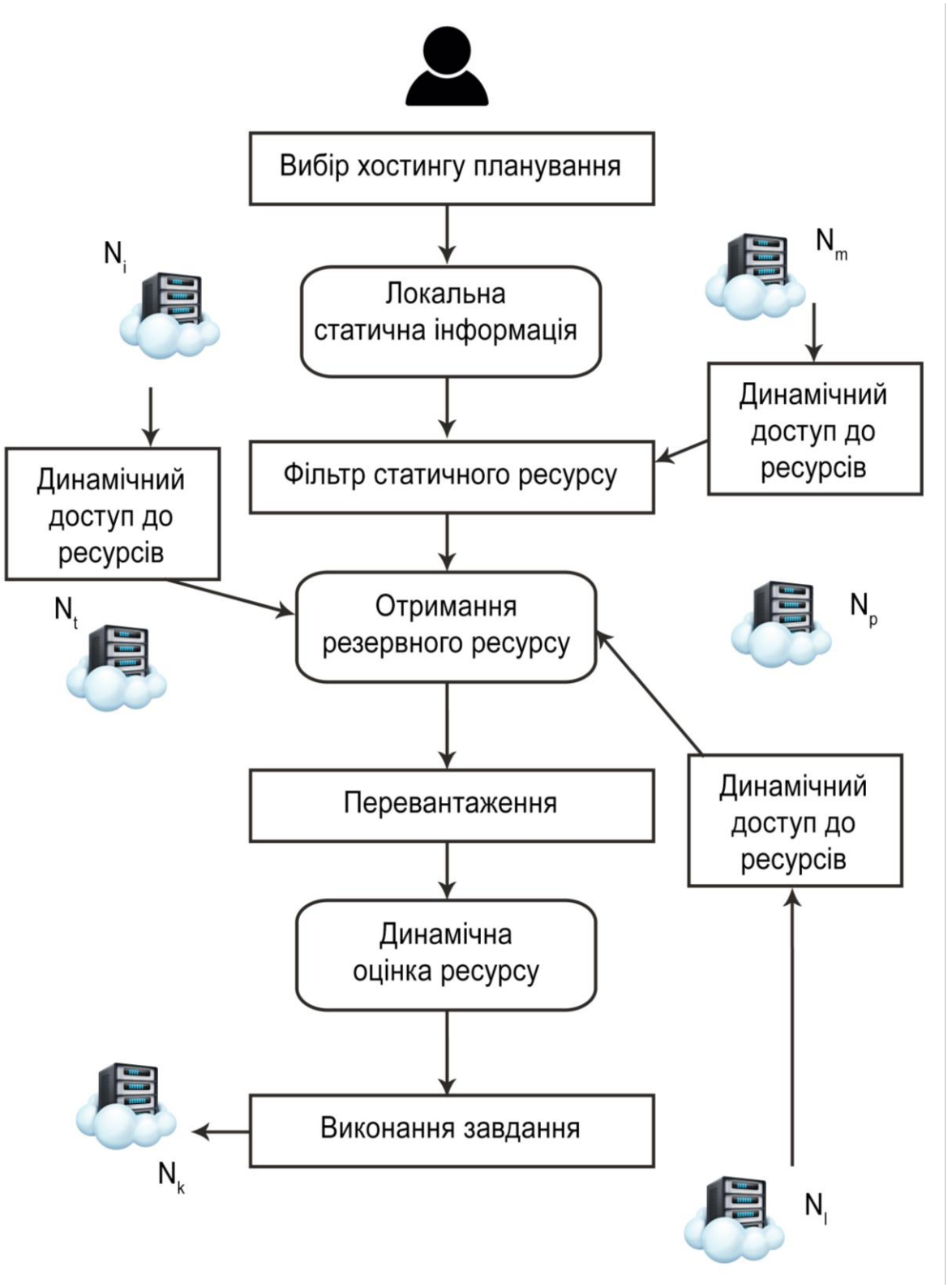


Рисунок 3.2.1 – Опис процесу планування завдань

Для обчислення потенціал окремої віртуальної машини та ємність використовуємо формулу:

$$C_{VM} = p * q \quad (1)$$

де p - швидкість обробки процесора в мільйонах інструкцій за секунду; q - кількість процесорів зайнятих для виконання завдання.

Завантаження інформації на віртуальну машину: планувальник завдань із хмарного середовища розподіляє завдання на віртуальну машину, кожна віртуальна машина має чергу для зберігання навантаження. Загальна довжина черги у віртуальній машині визначається як навантаження на цю віртуальну машину. Завантаження віртуальної машини можна розрахувати як

$$LVM_{i,t} = \frac{K * TL_{i(t)}}{S(VM_{i,t})} \quad (2)$$

де $K = 1,2,3 \ldots . N$ завдання. $S(VM_{i,t})$ визначається як швидкість обслуговування віртуальної машини в момент часу t, який можна виразити у формі потужності p та кількості процесорів q як $p * x(t)$, де $x = 1,2,3 \ldots . q$.

Навантаження віртуальної машини за часом t обчислюємо як кількість задач на конкретній віртуальній машині, поділену на швидкість обслуговування віртуальної машини. Отже, загальне навантаження на всю віртуальну машину дорівнює

$$L = \sum_{j=1}^{M} LVM_j \quad (3)$$

Якщо заплановане робоче навантаження всієї системи є більшим, ніж її потенціал, то центр обробки даних не зможе впоратися з усім майбутнім запитом, тому або відмовиться від майбутнього запиту на завдання, або збільшить віртуальну машину за допомогою концепції еластичності. Якщо майбутнє робоче навантаження менше, ніж потенціал всієї віртуальної системи, то знаходиться навантаження на кожну окрему віртуальну машину та визначається, чи є віртуальні машини, які перевантажені або завантажені. Якщо всі віртуальні машини перевантажені, то завдання переноситься на

завантажену віртуальну машину так, щоб всі завдання могли бути виконані за мінімальний час.

Час передачі завдання може бути розрахований *TT* = довжина завдання (*TL*) / пропускна спроможність (*B*).

Знаходимо час виконання завдання T_i на віртуальній машині VM_j

$$T_{exej} = \frac{\sum_i^N E_{ij} * TL_i}{p * q} \quad (4)$$

де $E_{ij} = 1$, якщо завдання T_i призначене віртуальній машині VM_j, інакше значення = 0.

Необхідно визначити час, який складається з часу виконання та передачі завдання (за умови, що будь-яке завдання може переміщуватися з перевантаженої віртуальної машини до завантаженої).

$$MST = \max\left\{\sum_{j=1}^{M} T_{exej}\right\} \quad (5)$$

3.3. Метод міграції задач

Міграції для вузла, який визначає, чи є вигідною міграція одного процесу або суперпроцесу, вона потребує лише знання про:

(а) обсязі даних, обмінюваних для зовнішнього зв'язку процесів, на яких він розміщений,

(б) обсяг даних, що обмінюються для внутрішньої (локальної) комунікації процесів, на яких він розміщується;

(в) найближчих сусідів, залучених до вищезгаданого зовнішнього зв'язку.

Оскільки вся ця інформація вже відома вузлам всередині системи, виконання Weighted Least Connections (WLC) базується лише на локальній інформації.

Алгоритм WLC не вимагає ніякої локальної інформації, яка ще не доступна або яка не може бути зібрана та мати незначну вартість. Прикладом такої інформації є характеристики навантаження вузлів / процесів, які

алгоритм не залежить. Це правда, що для правильного функціонування алгоритму кожен процес повинен знати розташування суміжних процесів. Розумно припустити, що або ця інформація є легкодоступною для того, щоб відбуватися процес комунікації, або це може бути поширюватися ліниво, коли процеси спілкуються один з одним.

Щоб відобразити динамічні зміни в обсязі трафіку між двома процесами, приймаємо метод усереднення.

Стандартизація атрибута ресурсу

Серед рішень проблем багатоцільового атрибута спосіб квантованості атрибутів безпосередньо впливає на об'єктивну оцінку. У процесі оцінки ресурсів використовується кілька показників; кожен індикатор має іншу одиницю і розмірність. Щоб краще відображати фактичний стан ресурсів, кожен індикатор потрібно порівнювати за уніфікованими показниками; тому, перш ніж оцінювати кожен ресурс обчислювальної вузла всебічно, нам потрібні вимірювання та стандартизація відповідних показників. У роботі використовується метод середнього геометричного методу в методі стандартизації векторів [5] для стандартизації атрибутів ресурсу, метод простий для розрахунку та впливу екстремального значення менше. Спосіб полягає в наступному:

$$Z_{ij} = \frac{x_{ij}}{\sqrt{\Sigma(x_{ij})^2}} \tag{6}$$

У визначенні p означає частоту процесора, *k* означає число ядра, *m* - об'єм пам'яті, *d* - пропускна спроможність периферійного зберігання, а *c* - пропускна спроможність.

Оскільки різний атрибут ресурсу має інший вимір, використовуємо формулу (6) для розміру та стандартизуємо атрибути статичного ресурсу, як показано нижче:

$$p_i = \frac{p_i * k_i}{\sqrt{\sum_{j=1}^{u} (p_j * k_j)^2}} \qquad m_i = \frac{m_i}{\sqrt{\sum_{j=1}^{u} m_j{}^2}}$$

$$d_i = \frac{d_i}{\sqrt{\sum_{j=1}^{u} d_j{}^2}} \qquad \widetilde{c_i} = \frac{c_i}{\sqrt{\sum_{j=1}^{u} c_j{}^2}} \tag{7}$$

У формулі $i = \{1,2,\dots,u\}$ означає всі доступні обчислені ресурси. У формулі (8) все відповідно представляє стандартизований результат процесора, пам'яті, жорсткого диска, мережі, стандартизації єдиного обчислювального вузла.

Кількісне розміщення динамічних атрибутів

Оцінка перевантаження усуває обчислювальні ресурси з великим завантаженням і визначає число хостів, які мають кращу продуктивність в системі та є обчислювальними вузлами з відносно легким навантаженням. Алгоритм обирає оптимальні обчислювальні вузли для подання завдань з них за допомогою динамічної оцінки. У даній роботі було використано алгоритм WLC. Завдяки наявності зваженого коефіцієнта застосування модифікованого алгоритму Least Connections визначає ключову роль у представленому способі.

Перед динамічною оцінкою необхідно стандартизувати динамічну інформацію обчислювальних вузлів. Оскільки кожен обчислювальний процесор може мати кілька ядер, процес індикаторів виглядає наступним чином:

$$p_i'(t) = \sum_{i=1}^{k_i} \left[\left(1 - p_i(t)\right) * p_i \right] \tag{8}$$

У формулі k_i означає основний номер хоста N_i.

Ми можемо знайти стандартизоване значення індикатора динамічного атрибута одиночного обчислюваного ресурсу, використовуючи формули(7), (7) та (8), виглядає наступним чином:

$$p_i(t) = \frac{p_i * k_i}{\sqrt{\sum_{j=1}^{g}(p_i'(t))^2}} \qquad m_i(t) = \frac{(1-m_i(t)) * m_i}{\sqrt{\sum_{j=1}^{g}[(1-m_j(t)) * m_j]^2}}$$

$$d_i(t) = \frac{(1-d_i(t)) * d_i}{\sqrt{\sum_{j=1}^{g}[(1-d_j(t)) * d_j]^2}} \qquad \widetilde{c_i}(t) = \frac{(1-c_i(t)) * c_i}{\sqrt{\sum_{j=1}^{g}[(1-c_j(t)) * c_j]^2}} \qquad (9)$$

Завантаження інформаційної системи зворотного зв'язку

Після фільтрування ресурсів за вимогою та квантованості статичних атрибутів використовуємо вікно перехоплення продукту v, щоб виключити обчислювальні вузли з низькою продуктивністю та виділити l резервні обчислювальні ресурси з v ресурсів з кращою продуктивністю, використовуючи стохастичну вагу. У процесі вибору резервних ресурсів коефіцієнт пропорції γ безпосередньо впливає на розподіл ймовірності видобутку ресурсів. Коли γ дорівнює 1, то ймовірність вибору сортованих обчислювальних ресурсів для резервування близька; коли λ відстає від 1, то ймовірність вибору відмінного обчислювального вузла у фронті буде збільшуватися. Наприклад, коли v беруться 4, λ прийнято 0,5 , можна отримати вірогідність вибору чотирьох сортованих обчислювальних ресурсів відповідно:

$$a_1 = \frac{8}{15} a_2 = \frac{4}{15} a_3 = \frac{2}{15} a_4 = \frac{1}{15} \qquad (10)$$

У наведеному вище прикладі ймовірність ресурсу з найбільш відмінними статичними атрибутами становить більше 0,5.

Опис алгоритму Weighted Least Connections. Запропонований алгоритм є динамічним, адже використовується у кожній перевантаженій станції та контролюється з головної станції. Алгоритм зважених найменших зв’язків використовує «ваговий» компонент на основі відповідних можливостей кожного сервера. Але необхідно заздалегідь вказати або визначити «вагу» кожного сервера.

Балансувальник навантаження, який реалізує алгоритм зважених найменших зв’язків враховує два параметри: вагу (ємність) кожного сервера та поточну кількість клієнтів, які в даний час підключені до кожного сервера. Вибір вузла системи даним способом балансування навантаження виконується на основі кількості активних з’єднань, тобто потужності сервера. Алгоритм WLC динамічний та здатний назначати «вагу» серверам «на льоту».

Аналіз алгоритму WLC. Недоліки алгоритму: немає ніякого сенсу застосовувати цей алгоритм для задач з короткими сесіями, характерними, наприклад, для HTTP протоколу. Для такого виду задач кращим застосуванням буде Round Robin.

Переваги алгоритму: даний алгоритм застосовується для задач, пов'язаних з тривалими з'єднаннями. Наприклад, для розподілу навантаження між серверами бази даних. Якщо на деяких вузлах буде занадто багато активних підключень, нових запитів вони вже не отримають, і навпаки.

3.4. Навантажувальне тестування

Розрахунок часу обробки: виділяється завдання всій віртуальній машині за алгоритмом планування, після чого розпочинається моніторинг системи віртуальних машин. Завдяки цьому визначаємо стан кожної віртуальної машини, де з’ясовуємо необхідність балансування навантаження – перенесення завдань з перевантаженої віртуальної машини до менш завантаженої. Для досягнення максимальної оптимізації та виявлення достовірних результатів збільшуємо кількість повідомлень та їх довжину. Граничним значенням стає значення перевантаження всієї системи.

Середній коефіцієнт використання ресурсів (Average Resource Utilization Ratio, ARUR): основна мета розподілу навантаження полягає в тому, щоб максимально використовувати ресурси. Середній коефіцієнт використання ресурсів розраховуємо за формулою:

$$ARUR = \left(\frac{\text{середній час}}{\text{час обробки}}\right) * 100 \quad (11)$$

де середній час = Σ часу, витраченого ресурсом (VM_j), щоб закінчити всю роботу ресурсу, де $j = \{1, 2, 3 \ldots\ldots\ldots M\}$. Діапазон середнього коефіцієнта використання ресурсів становить від 0 до 1, максимальне значення для ARUR становить 1 (використання ресурсу становить 100%), найгірше - 0 (ресурс знаходиться в ідеальному стані).

Висновки до розділу 3

У даному розділі визначено алгоритм, який зосереджується на збільшенні навантаження, щоб задовольнити вимоги користувача для продовження сервісу шляхом розподілу робочого навантаження в балансовому порядку, щоб отримати максимальне використання ресурсів та зменшити час простою системи та запропоновано метод динамічного балансування навантаження в хмарному середовищі на основі методу міграції задач та модернізованого алгоритму Weighted Least Connections, а саме алгоритм динамічного навантаження з урахуванням часу проходження та середнього коефіцієнта використання ресурсів як параметра. Метою даного методу є ефективне використання ресурсів у хмарному середовищі.

Проаналізовано головні переваги даного способу, зосереджуючись на ефективному використанні ресурсів у хмарному середовищі. Також розглянуто міграційні процеси при балансуванні навантаження у хмарному середовищі як особові об’єкти. Тому запропоновано виокремити показник, який визначає вплив міграції на мінімізацію загальної вартості мережі.

РОЗДІЛ 4. ТЕСТУВАННЯ ТА АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ ЗАПРОПОНОВАНОГО СПОСОБУ ДИНАМІЧНОГО БАЛАНСУВАННЯ НАВАНТАЖЕННЯ

4.1. Опис системи під час тестування

Запущена симуляція обробляла дані більше двох годин з різною кількістю завдань (діапазон задач від 10 до 50) з довільною довжиною. Отримуємо результат, використовуючи загальну політику та функції cloudsim. При старті розглядаємо 5 віртуальних машин різної потужності обробки (від 200 до 1000 MIPS), діапазон пропускної спроможності віртуальної машини становив від 100 до 1000 Мбайт/с, кількість процесорів для кожної віртуальної машини становила 1, як показано в таблиці 4.1.1, і враховує від 10 до 50 завдань з довжиною від 20000 MI до 400000 MI, як показано в таблиці 4.1.1.

Таблиця 4.1.1 – Властивості задач

Task ID	**Length**	**File size**	**Output file**	**№ of CPU**
0	116228	300	300	1
1	52118	300	300	1
2	33734	300	300	1
3	67488	300	300	1
4	290543	300	300	1
5	60246	300	300	1
6	366852	300	300	1
7	73891	300	300	1
8	197914	300	300	1
9	54199	300	300	1

Таблиця 4.1.2 - Властивості VM

VM ID	VM MIPS	VM image size	Memory	№ of CPU	VMM
0	1000	1000	512	1	Xen
1	800	1000	512	1	Xen
2	600	1000	256	1	Xen
3	400	1000	512	1	Xen
4	200	1000	256	1	Xen

Результати таблиці 4.1.1 і таблиці 4.1.2 представлені на наступних риснках із результатами роботи при експериментальних запусках та виконанні завдань. Вісь *x* представляє кількість завдань та вісь *y* представляє час виконання завдання.

Завдання виділяється всій віртуальній машині за алгоритмом планування Weighted Least Connections, після чого починається моніторинг віртуальної машини і виявляється, що віртуальна машина 5 (VM ID 4) має перевантажений стан. Наступним кроком проводиться моніторинг віртуальних машин із найменшим показником стану завантаженості.

Завдання перевантаженої віртуальної машини мігрує для балансування віртуальної системи, тому буде перенесене на віртуальну машину 1, а також додано час міграції завдання. Всі завдання будуть виконані за 505 секунд.

4.2. Налаштування системи

В ході роботи при отриманні результатів завдяки проведеним ряду експериментів було визначено, що іноді час алгоритму Max-Min балансування навантаження близький до запропонованого методу з використанням алгоритму (WLC).

Для реалізації вищезгаданих функціональних можливостей CloudSim використовуються базові функції моделювання дискретних подій та моделювання основних компонентів хмарного обчислення. Оскільки об'єкти та компоненти CloudSim спілкуються через операції проходження

повідомлення, основний рівень відповідає за керування події та обробку взаємодій між компонентами. Основні класи ContainerCloudSimare зображені на рисунку 4.2.1.

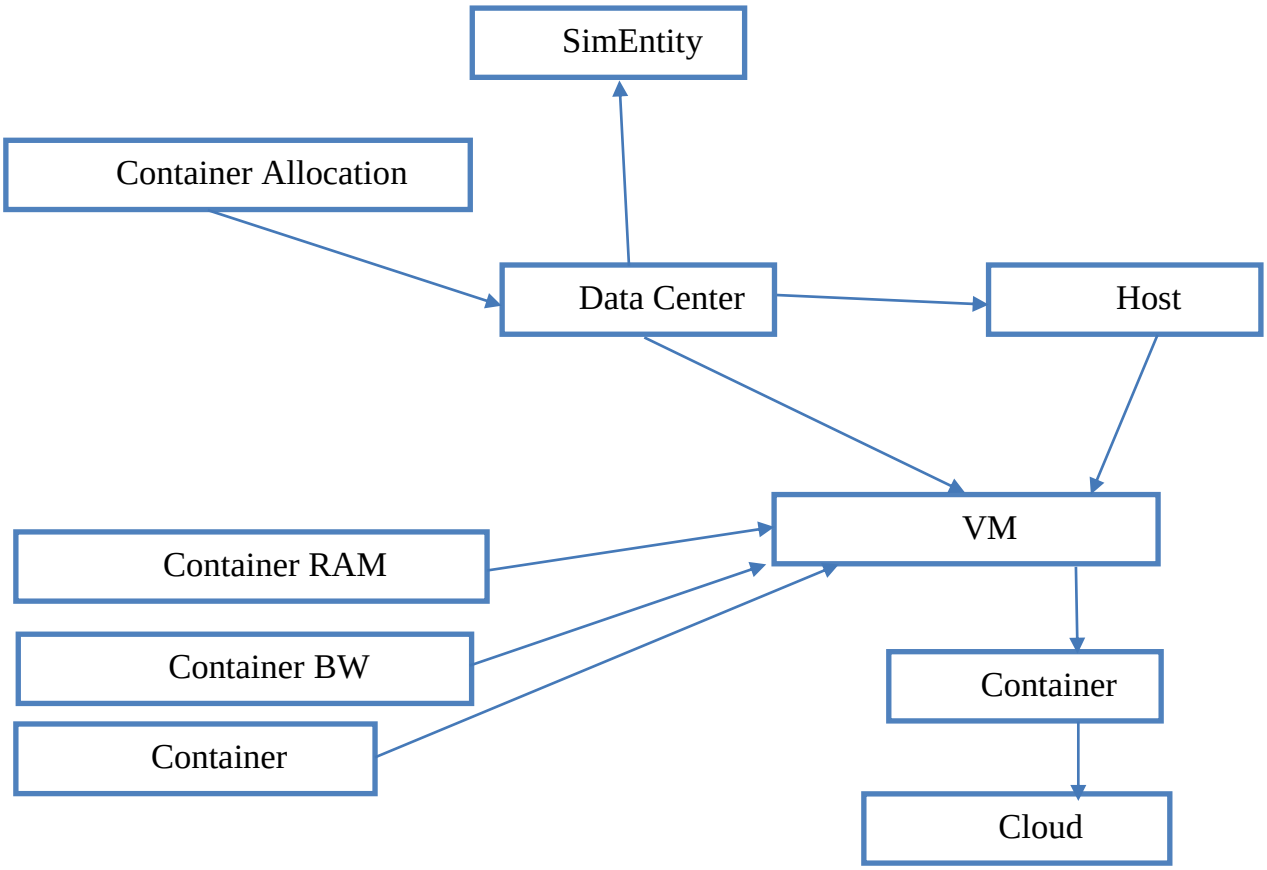


Рисунок 4.2.1 – Діаграма ContainerCloudSimclass

Елементи включають в себе наступне:

- Центр даних: Апаратний рівень хмарних служб моделюється через клас Datacenter.
- Хост: Клас Хосту представляє фізичні обчислювальні ресурси (сервери). Їхні конфігурації визначаються як можливість обробки, яка виражається в MIPS (мільйон інструкцій в секунду), пам'яті та сховища.
- VM: Цей клас моделює віртуальну машину. Віртуальні машини управляються хостом. Атрибути VM - це пам'ять, процесор та розмір його зберігання.

- Контейнер: у цьому класі створюється контейнер, який розміщений у віртуальній машині. Атрибути контейнерів - це доступна пам'ять, процесор та розмір пам'яті.
- Cloudlet: клас Cloudlet моделює додатки, розміщені в контейнері в центрах обробки даних хмар.
- Довжина Cloudlet визначається як Million Instructions (MI), і має функціональні можливості свого попередника в пакеті CloudSim, включаючи StartTime та статус виконання (наприклад, CANCEL, PAUSED і RESUMED).

У якості контейнерного хмарного симулятора ContainerCloudSim масштабується з мінімальним накладними витратами на пам'ять та часом виконання. Для того, щоб дослідити масштабованість розробленого симулятора, проведено експеримент, який був встановлений, розглядаючи алгоритм Weighted Least Connections як політику первинного розміщення контейнера. Цей самий експеримент повторюється для різного числа контейнерів від 50 до 5000.

Для експерименту порівняння числа доступних хостів і віртуальних машин вважається постійним і дорівнює 700 і 1000, відповідно. Для кожного експерименту, що зображені на рисунках 4.2.2-4.2.4, показано час виконання симуляції, який визначається як час, необхідний для закінчення моделювання, та використання пам'яті програми Java (імітація).

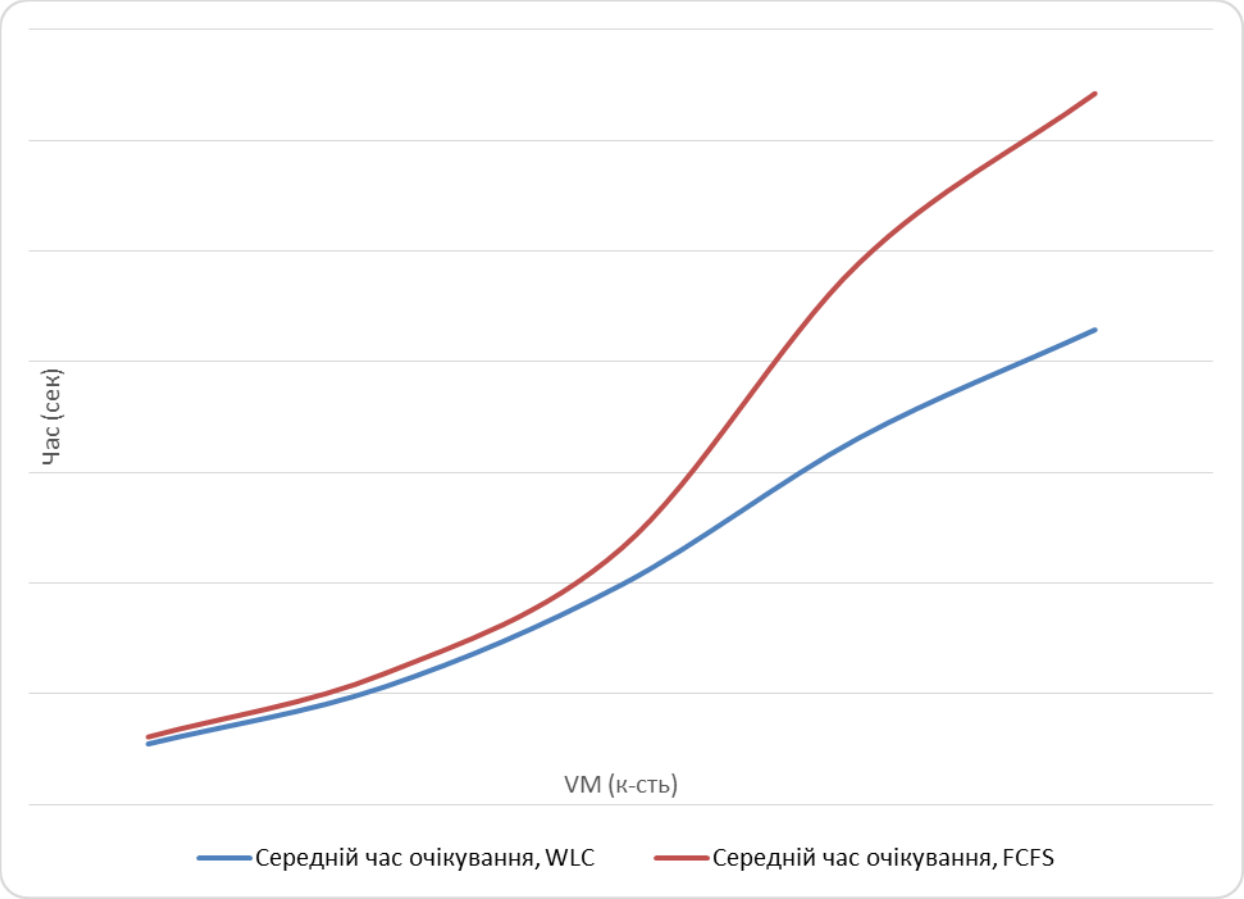


Рисунок 4.2.2 – Використання алгоритмів WLC та FCFS

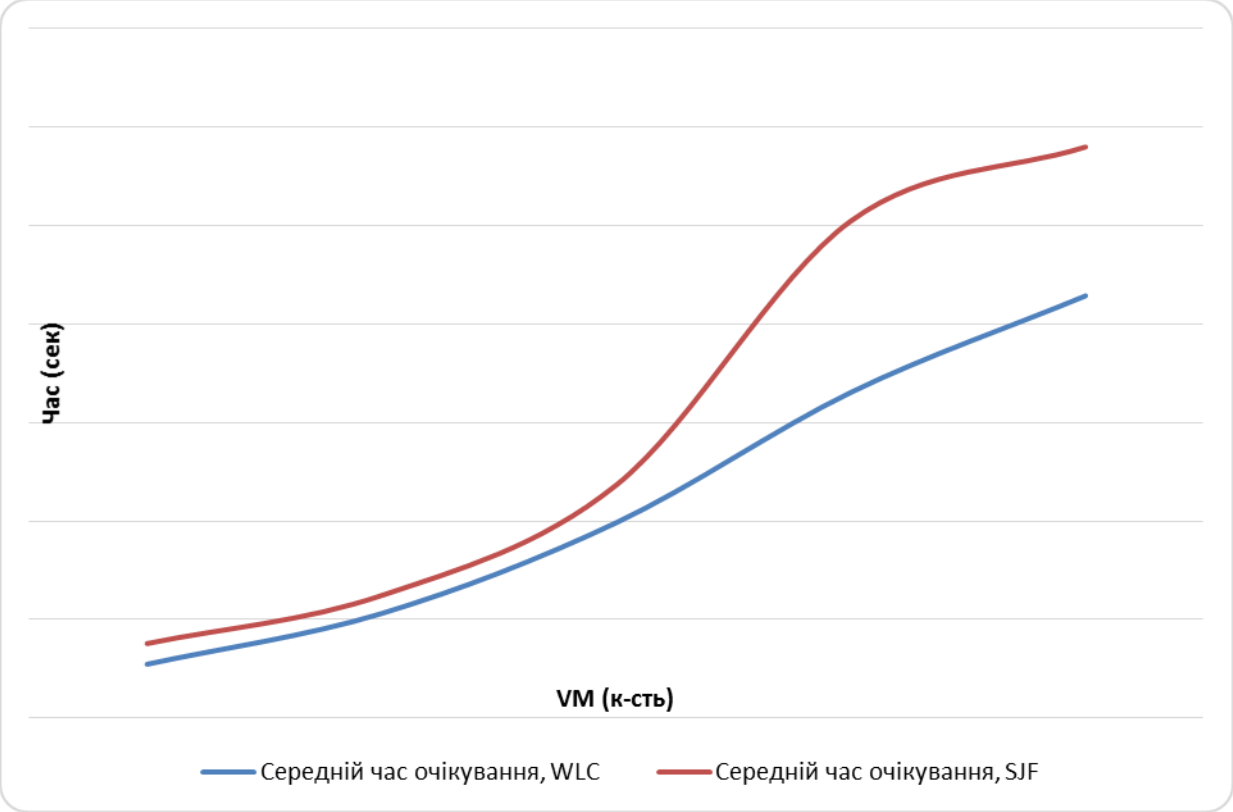


Рисунок 4.2.3 – Використання алгоритмів WLC та SJF

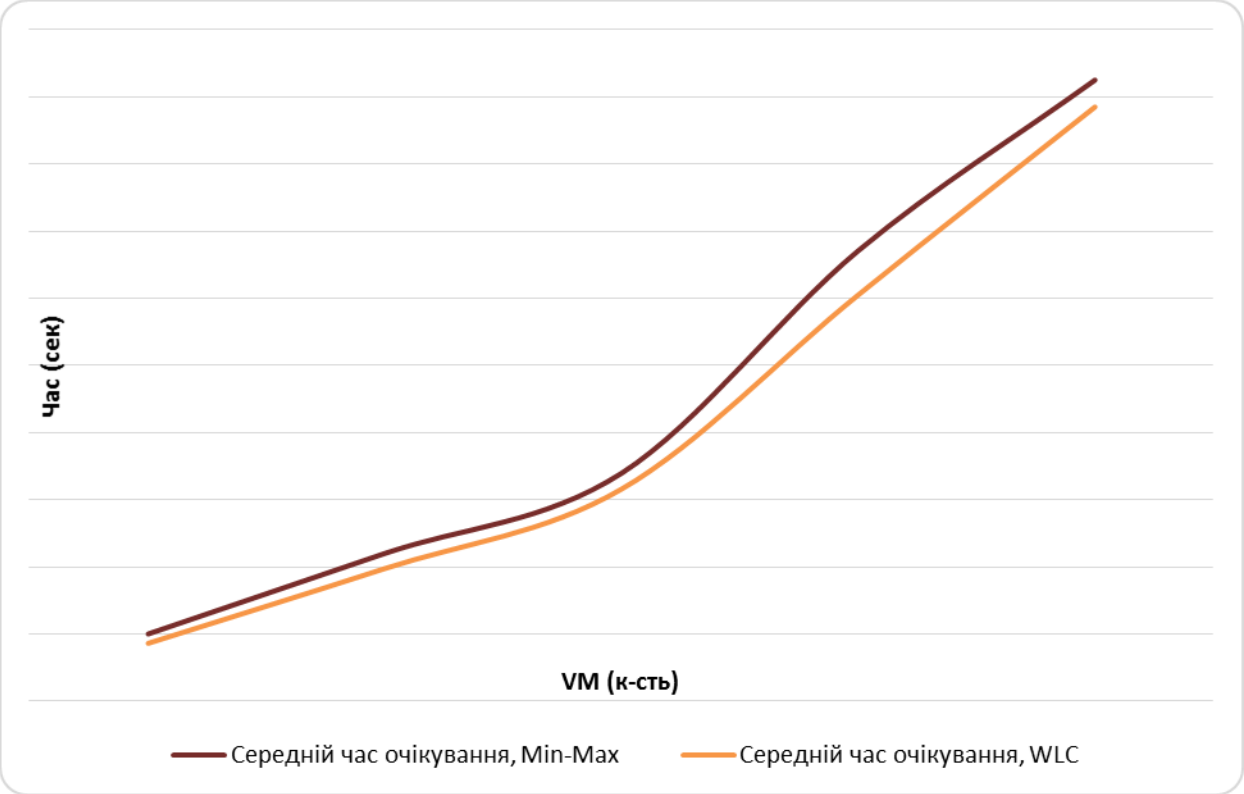


Рисунок 4.2.4 – Використання алгоритмів WLC та Min-Max

Оскільки алгоритм Max-Min дотримується евристичного підходу, а його коефіцієнт використання ресурсів є низьким у порівнянні з запропонованою WLC (завдання не виділяється на віртуальну машину 5). Ми збільшуємо кількість завдань до 50 з різною довжиною і застосовуємо один і той же алгоритм до завдання, результати показані у наступному підрозділі.

Важливою операцією в середовищах хмарних обчислень є інсталяція віртуальних машин. Цей час є незначним і може вплинути на ефективність додатків, що працюють у хмарах. На основі дослідження поточна версія симулятора включає статичну затримку 100 секунд для кожної віртуальної машини. Також важливим компонентом стала затримка запуску контейнерів, оскільки жива міграція контейнерів не застосовна в реальних сценаріях. Тому міграція контейнера виконується шляхом вимкнення контейнера на вихідному хості, як тільки починається той самий контейнер у головному сервері. Аналогічно VM, затримка запуску контейнера може бути встановлена як константа у поточній версії CloudSim.

4.3. Результати запропонованого способу

Технологія віртуалізації є однією з основних концепцій інфраструктури Cloud Computing. CloudSim надзвичайно розгортає технології віртуалізації, щоб імітувати постачання IaaS та PaaS та використовувати її як базу для виконання програм користувачів. З тієї ж точки зору виникає завдання розгортання найкращого розподілу ресурсів та алгоритму планування. Наприклад, один центр обробки даних, який складається з одного хосту з двома процесорами, а хмарний користувач намагається виконати інстанцію двох віртуальних машин з двома обробними одиницями кожна. Далі на рисунку 4.3.1. представлено серію експериментальних даних, які показують залежність часу обробки задач від алгоритму балансування кількість виконаних задач (також представлено результати запропонованого способу).

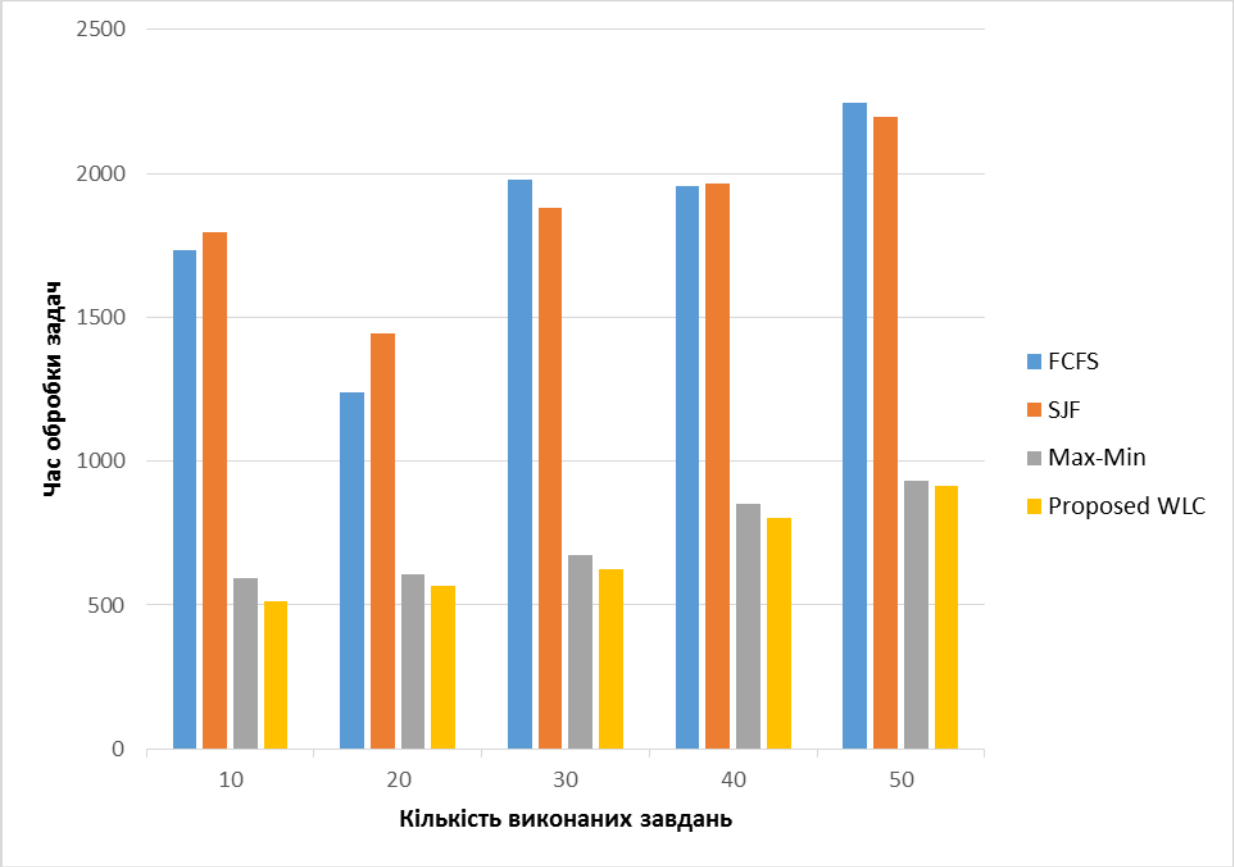


Рисунок 4.3.1 – Результати часового проміжку обробки завдань із застосуванням алгоритмів при VM=5

Збільшуємо кількість завдань від 10 до 20 і випадково вибираємо довжину завдання від 20000 до 200000 MI. Ми зменшуємо довжину завдання,

тому що, якщо ми створимо 20 завдань довжиною від 20K до 400K і виділимо на 5 існуючих віртуальних машин, то вся віртуальна машина буде перевантажена, і планувальник задач облаток потребує завантаження більш віртуальної машини, тут ми працюємо над балансуванням навантаження Проблема, тому ми зменшуємо довжину завдання.

В ході експериментальної роботи було обчислено середнє співвідношення використання ресурсів запропонованого алгоритму за допомогою рівняння 11 та порівняємо результати з існуючими алгоритмами. Також отримані результати були згруповані та занесені до таблиці 2 та таблиці 3 для ресурсу та кількості завдань, після було використано для отримання результатів виконання задач при навантаженні системи при різних умовах.

Було застосовано запропонований алгоритм балансування навантаження для знаходження середнього коефіцієнта використання ресурсів. Графік представляє собою результати запропонованого алгоритму балансування навантаження, що виділяє завдання на віртуальну машину таким чином, що не з'являються віртуальні машини в перевантаженому або завантаженому станах. Розрахований ARUR пропонованого алгоритму балансування навантаження становить 73%. Ми застосовуємо алгоритм Min-Min з однаковим значенням даних і отримуємо 50,8% ARUR, оскільки ресурс R3 та R4 в режимі очікування. Дані кожного з алгоритмів наведені у таблиці 4.3.1 для VM = 5.

Таблиця 4.3.1 – Результати виконання завдань VM = 5

	10	**20**	**30**	**40**	**50**
CFS	1734	1238	1977	1958	2246
SJF	1795	1446	1881	1966	2196
MAX-MIN	594	605	672	854	934
Proposed WLC	515	569	625	803	912

Логічно, що існує розділення між двома віртуальними машинами, однак насправді кожна віртуальна машина обмежується потужністю обробки, яку пропонує фізичний вузол, тому неможливо одночасно ініціалізувати обидві віртуальні машини на тому ж хості без відповідного алгоритму планування. Дані при збільшенні VM для кожного відповідного алгоритму наведені у таблиці 4.3.2.

Таблиця 4.3.2 – Результати виконання завдань VM = 10

	10	**20**	**30**	**40**	**50**
FCFS	2900	2952	5321	6312	7456
SJF	2330	3456	5073	5623	6022
MAX-MIN	1830	1935	1669	1860	1900
Proposed WLC	1190	1017	1145	1234	1389

Експеримент повторювався з різними комбінаціями віртуальних машин (від 5 до 10), з складною математичною операцією, яка виконується для кожної хмари як навантаження, або без неї. Розподілені об'єкти та розподілене виконання контролювалися центром управління. Об'єкти рівномірно розподілені серед доступних екземплярів, споживаючи практично таку ж кількість вхідної пам'яті з кожного екземпляра. Перегородки різних екземплярів однаково виражені або доступні. Це показує ефективний розподіл розподілених об'єктів. Виявлено чотири окремі випадки масштабованості, як описано нижче.

Динамічне масштабування: за умови використання динамічного масштабування, цей випадок представляє найбільше прикладів виконання, оскільки навантаження найвище. Пам'ять, яка використовується програмою у відсотках від загальної використовуваної пам'яті, використовувалася як міра моніторингу стану системи. З адаптивним масштабуванням середовище 5 VM і 10 перевантаження системи збільшено до 3 разів.

Зниження максимального порогу змусило основний шар масштабувати в першу чергу та швидше, залучаючи всі доступні приклади до моделювання.

Час виконання збільшується, оскільки додається більше вузлів до хмарної системи. Отже, введення додаткових вузлів за межі певної максимальної кількості вузлів може бути економічно недоцільним, і в певному пункті це може стати мірою, яка пояснюється нижче як загальна справа. Далі на рисунку 4.3.2. представлено серію експериментальних даних, які показують залежність часу обробки задач від алгоритму балансування кількість виконаних задач (також представлено результати запропонованого способу), але вже на VM=10.

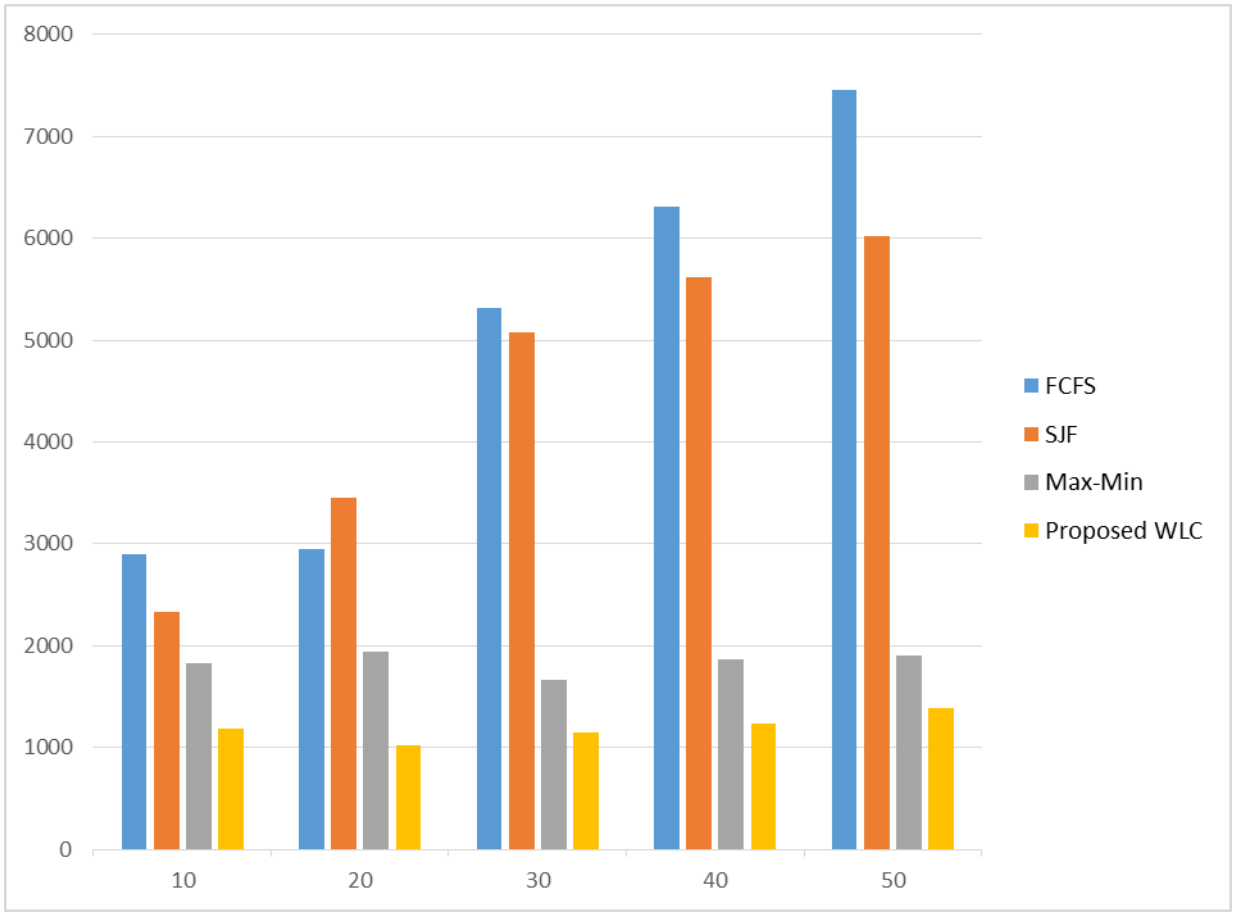


Рисунок 4.3.2 – Результати часового проміжку обробки завдань із застосуванням алгоритмів VM=10

Висновки до розділу 4

Користувачі та потенційні користувачі хмарних обчислень оцінюють хмарні рішення, які допоможуть їм скоротити витрати, підвищити якість надання послуг та збільшувати їхні переваги. Щоб випробувати або розвинути цю нову модель обчислень, як постачальникам, так і користувачам потрібні інструменти, які допоможуть їм прийняти критичне рішення щодо впровадження або переходу до інфраструктури на основі технології хмарних обчислень.

Проведено тестування запропонованого методу динамічного балансування навантаження в хмарному середовищі на основі методу міграції задач та модернізованого алгоритму Weighted Least Connections, а саме алгоритм динамічного навантаження з урахуванням часу проходження та середнього коефіцієнта використання ресурсів як параметра моделі Cloud Computing. Оцінюючи наявні симулятори Cloud, було обрано Cloudsim інструментарій для моделювання Cloud Computing.

Проведено експериментальні дослідження по роботі нового способу, який показав ефективність в порівнянні з інтегрованими CloudSim. У ході експериментальних досліджень було виявлено зменшення часу обробки завдання та збільшення середнього коефіцієнта використання ресурсів. Час обробки завдань на 5-ти віртуальних машин при 10, 20 та 30 завдань зменшився від 30% до 10% (у порівнянні з алгоритмом FCFS та з алгоритмом MAX-MIN відповідно).

ВИСНОВКИ

У даній роботі запропоновано метод динамічного балансування навантаження в хмарному середовищі на основі методу міграції задач та модернізованого алгоритму Weighted Least Connections, а саме алгоритм динамічного навантаження з урахуванням часу проходження та середнього коефіцієнта використання ресурсів як параметра. Метою даного методу є ефективне використання ресурсів у хмарному середовищі. Експериментальні результати показали зменшення часу обробки та збільшення середнього коефіцієнта використання ресурсів.

Було проведено загальний огляд технології хмарних обчислень. Дано визначення хмарним обчисленням та з'ясовані передумови появи хмарних обчислень. Визначені основні види хмарних систем, їх моделі обслуговування та моделі розгортання. Показана важливість систем розподілення навантаження в хмарних обчисленнях. Описані основні способи розподілу навантаження, що використовуються в різних типах хмарних систем. Проаналізовані їх недоліки та переваги, що є передумовою для створення нових рішень в області балансування навантаження. Також було з'ясовано, що однією з основних технологій, що дозволяє створювати хмарні рішення, є віртуалізація серверів. Проведено огляд різних типів віртуалізації, визначені їх недоліки та переваги. Встановлено, що технологія віртуалізації дозволяє виконувати переміщення віртуальних машин без зупинки клієнтських додатків на них, що в свою чергу, забезпечує можливість балансування навантаження на фізичні сервери хмарної системи без порушення угоди про рівень надання послуги.

Запропоновано новий спосіб розподілу навантаження на хмарні системи – обробка статистичних даних про стан системи в реальному часі. Статистичні способи є набагато надійнішими ніж динамічні, які здійснюють розподіл на піку навантаження, так як взагалі не допускають виникнення ситуацій піку навантаження. Статистичні способи зазвичай використовуються в системах з постійною кількістю користувачів,

наприклад, в корпораціях, що використовують власні дата-центри. Модифікований спосіб на основі статистичного підходу дозволяє передбачати піки навантаження в реальному часі, аналізуючи дані декількох останніх результатів моніторингу системи та швидко їх ліквідовувати. Така модифікація вирішує проблему статистичних алгоритмів коли статистичні дані можуть бути викривленими в наслідок постійної зміни кількості користувачів хмарної системи.

Проведено деталізований порівняльний аналіз розповсюджених алгоритмів та методів динамічного балансування навантаження. Зазначено, що балансування навантаження – це оптимізація виконання розподілених або паралельних обчислень за допомогою розподіленої обчислювальної системи. Балансування навантаження передбачає рівномірне навантаження обчислювальних вузлів. При появі нових завдань ПЗ, що реалізує балансування, має прийняти рішення про те, на якому обчислювальному вузлі слід виконувати обчислення, пов’язані з новим завданням. Представлено аналіз способу динамічного балансування навантаження на модулі. Описано модель системи та її роботи під час тестування. Представлено покрокову дію запропонованого методу.

Представлені результати тестування системи за допомогою власного динамічного способу балансування навантаження. Для оцінки запропонованого методу була розроблена модель динамічного балансування навантаження в хмарному середовищі на основі методу міграції задач та модернізованого алгоритму Weighted Least Connections. У сценарії моделювання порівняно продуктивність запропонованої системи з запропонованим алгоритмом балансування навантаження і способом розподілення навантаження та без нього. У ході експериментальних досліджень було виявлено зменшення часу обробки завдання та збільшення середнього коефіцієнта використання ресурсів. Час обробки завдань на 5-ти віртуальних машин при 10, 20 та 30 завдань зменшився від 30% до 10% (у порівнянні з алгоритмом FCFS та з алгоритмом MAX-MIN відповідно).

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Телелейко І.С. Інтеграція сервіс-орієнтованої архітектури з Grid-системами / І.С. Телелейко, Я.М. Клятченко // Восьма наукова конференція магістрантів та аспірантів «Прикладна математика та комп'ютинг (ПМК-2016)», Київ, 20-22 квітня 2016 р.: Збірник тез доповідей. – К.: Просвіта, 2016. – С.60-64.

2. Телелейко І.С. Сумісність підходів хмарних обчислень та грід-технологій / І.С. Телелейко, М.М. Орлова // 19-а Міжнародна науково-технічна конференція «Системний аналіз та інформаційні технології» SAIT 2017, Київ, 22-25 травня 2017 р. / ННК «ІПСА» НТУУ «КПІ ім. Сікорського». – К.: ННК «ІПСА» НТУУ «КПІ ім. Ігоря Сікорського», 2017. – 340 с.

3. Телелейко І. С. Аналіз методів динамічного балансування навантаження в хмарному середовищі / І. С. Телелейко, М. М. Орлова // X конференція молодих вчених «Прикладна математика та комп'ютинг – ПМК'2018», Київ, 21-23 березня 2018р.: Збірник тез доповідей. – К.: Просвіта, 2018.

4. Телелейко І. С. Спосіб динамічного балансування навантаження в хмарному середовищі / І.С. Телелейко, М.М. Орлова // Науковий журнал «Інтернаука. – 2018. – №7. – Електронні дані. – Режим доступу: https://www.inter-nauka.com/issues/2018/7/3687/ .

5. Mell, P. The NIST Definition of Cloud Computing [Electronic resource] / Peter Mell, Timothy Grance. – Recommendations of the National Institute of Standards and Technology NIST. – 2011. – SP 800-145. – Електронні дані. – Режим доступу: https://csrc.nist.gov/publications/detail/sp/800-145/final

6. Bass L. (2003). Software Architecture In Practice, Second Edition / L. Bass, P. Clements, R. Kazman. – Boston: Addison-Wesley. – 2003. – ISBN 0-321-15495-9. – pp. 21–24.

7. Donaldson V. Program Speedup in a Heterogeneous Computing Network / V. Donaldson, F. Berman, R. Paturi. // Journal of Parallel and Distributed Computing. – September 1994. – Vol. 21. – No 3. – pp. 316-322.

8. Leopold C. Parallel and Distributed Computing. A survey of Models, Paradigms, and Approaches Wiley Series on Parallel and Distributed Computing / C. Leopold // Albert Zomaya Series Editor. – 2001.

9. Cloud computing: distributed internet computing for IT and scientific research / [M. D. Dikaiakos, D. Katsaros, P. Mehra and others.] . – Internet Computing, IEEE. – 2009. – №13. – pp. 10–13.

10. Rima B. A Taxonomy and Survey of Cloud Computing Systems / B. P. Rima, E. Choi, and I. Lumb // Proceedings of 5th IEEE International Joint Conference on INC, IMS and IDC, Seoul. – Korea, August 2009. – pp. 44-51

11. Alakeel A. A Guide to dynamic Load balancing in Distributed Computer Systems" / A. M. Alakeel // International Journal of Computer Science and Network Security (IJCSNS) . – Vol. 10. – No. 6. – June 2010. – pp. 153-160.

12. Suresh A. Improving scheduling of backfill algorithms using balanced spiral method for cloud metascheduler / A. Suresh, P. Vijayakarthick // International Conference on Recent Trends in Information Technology. – Chennai. – 2011.

13. Dubey K. A Priority Based Job Scheduling Algorithm Using IBA and EASY Algorithm for Cloud Metaschedular / K. Dubey // International Conference on Advances in Computer Engineering and Applications. Ghaziabad. – India. – 2015.

14. Kumar M. Priority Aware Longest Job First (PA-LJF) algorithm for utilization of the resource in cloud environment / M. Kumar, S.C. Sharma // Computing for Sustainable Global Development (INDIACom). – 3rd International Conference on. IEEE. – 2016.

15. Kumar M. Job Scheduling Algorithm in Cloud Environment Considering the Priority and Cost of Job / M. Kumar, K. Dubey, S.C. Sharma // Proceedings of Sixth International Conference on Soft Computing for Problem Solving. Springer. – India. – 2017.

16. Lakra A. Multi-Objective Tasks Scheduling Algorithm for Cloud Computing Throughput Optimization / A. Lakra, D. Yadav // Procedia Computer Science. – 2015.

17. Ren X. A Dynamic Load Balancing Strategy for Cloud Computing platform based on Exponential Smoothing Forecast / X. Ren // International Conference on Cloud Computing and Intelligence Systems. – China. – 2011.

18. Tziritas N. On minimizing the resource consumption of cloud applications using process migrations / N. Tziritas // Journal of Parallel and Distributed Computing. – 2013.

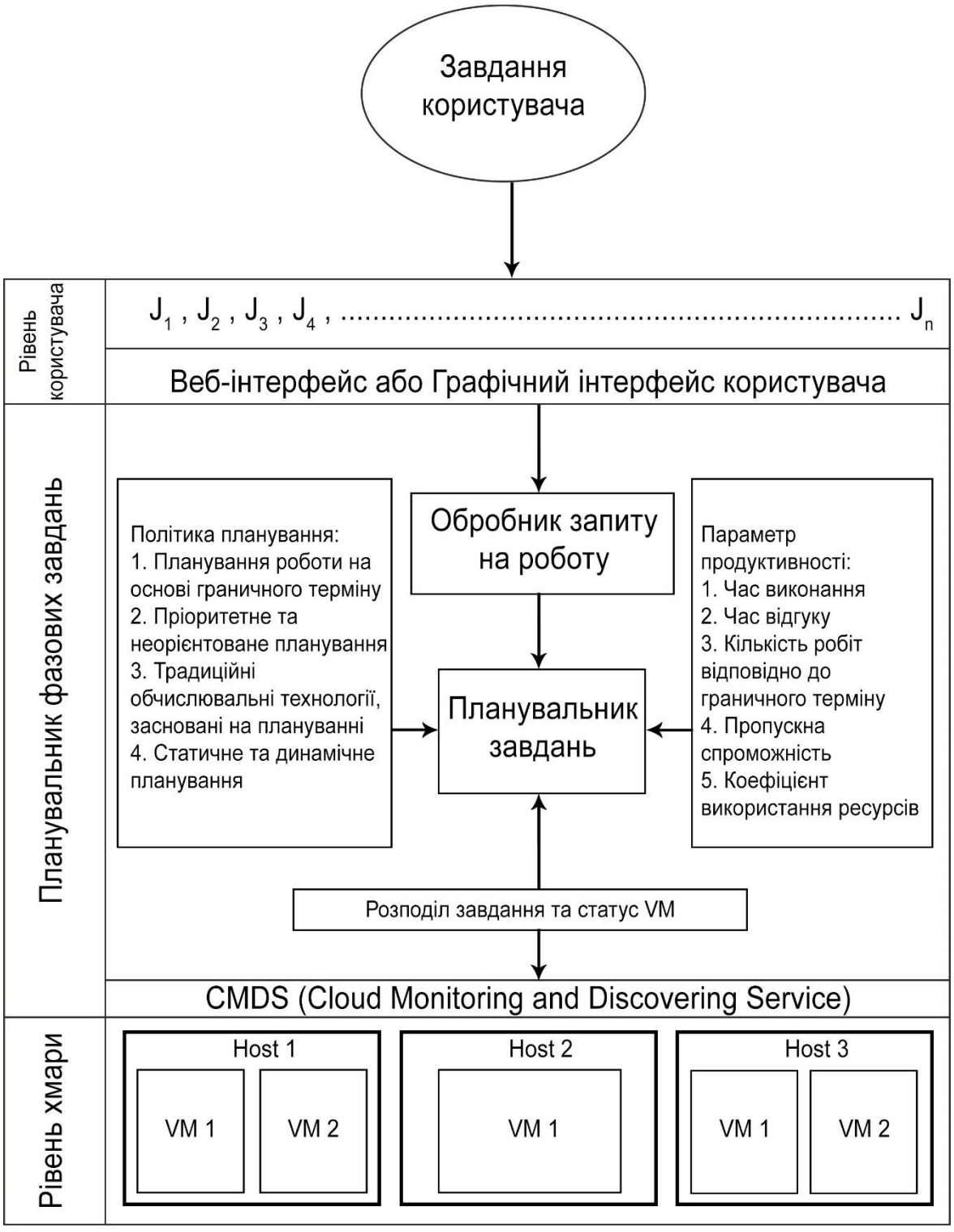
19. Kumar M. Dynamic load balancing algorithm for balancing the workload among virtual machine in cloud computing / M. Kumar, S.C. Sharmab – 2017.

20. NIST Cloud Computing Standards Roadmap. – Special Publication 500-291, Version 2. – U. S. Department of Commerce; National Institute of Standards and Technology. – 2013. – 108 p.

21. The NIST Definition of Cloud Computing : Recommendations of the National Institute of Standards and Technology [Electronic resource]. – Access mode: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

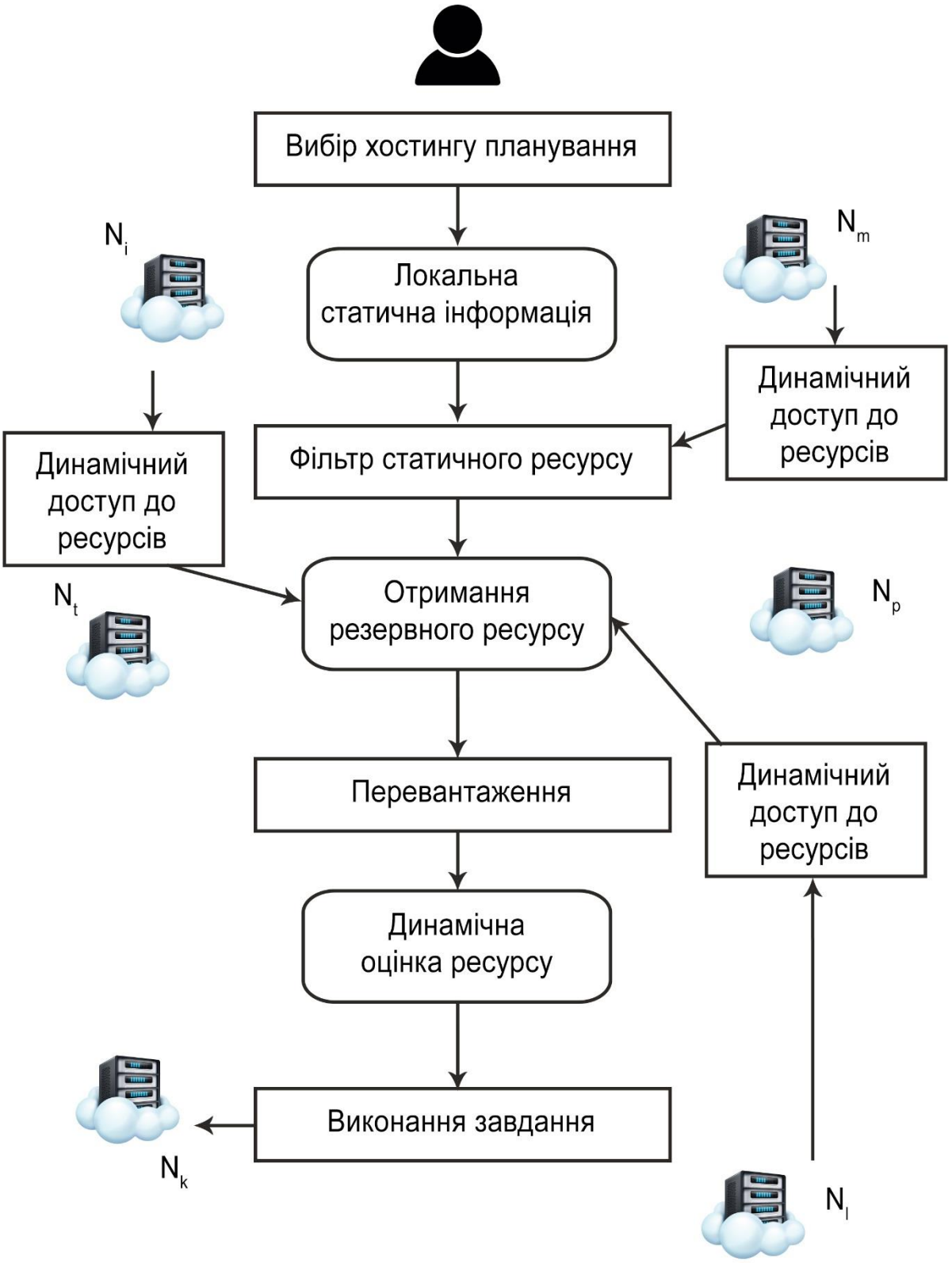
Додаток 1. Копії графічних матеріалів

1. Схема компонентів планування завдань у Cloud Computing



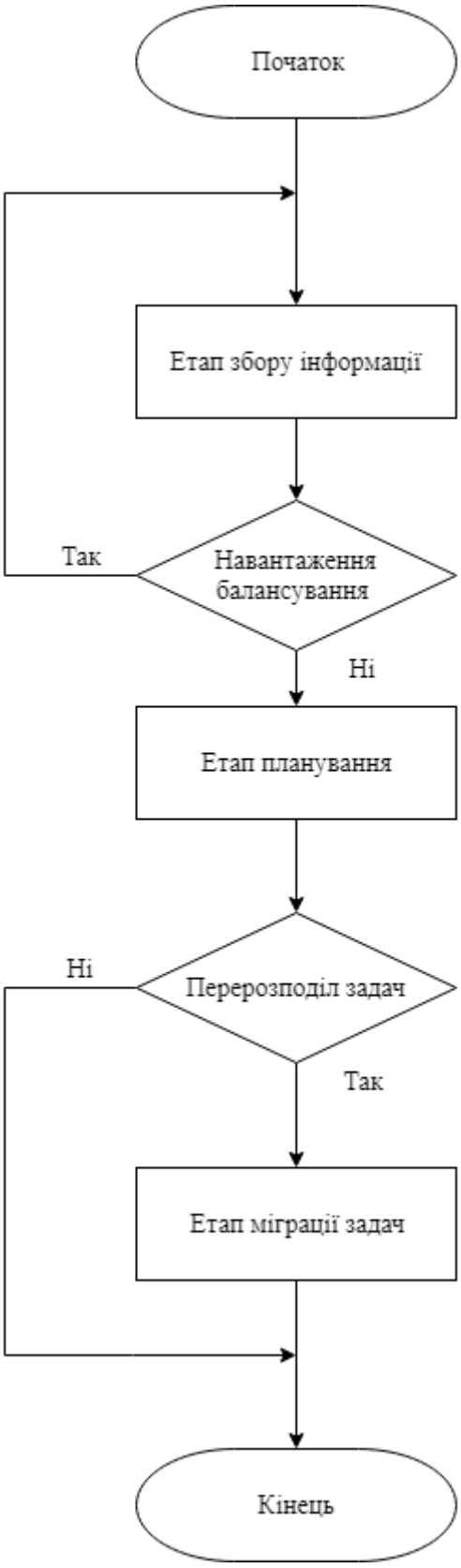
Телелейко І.С.

2. Схема планування завдання



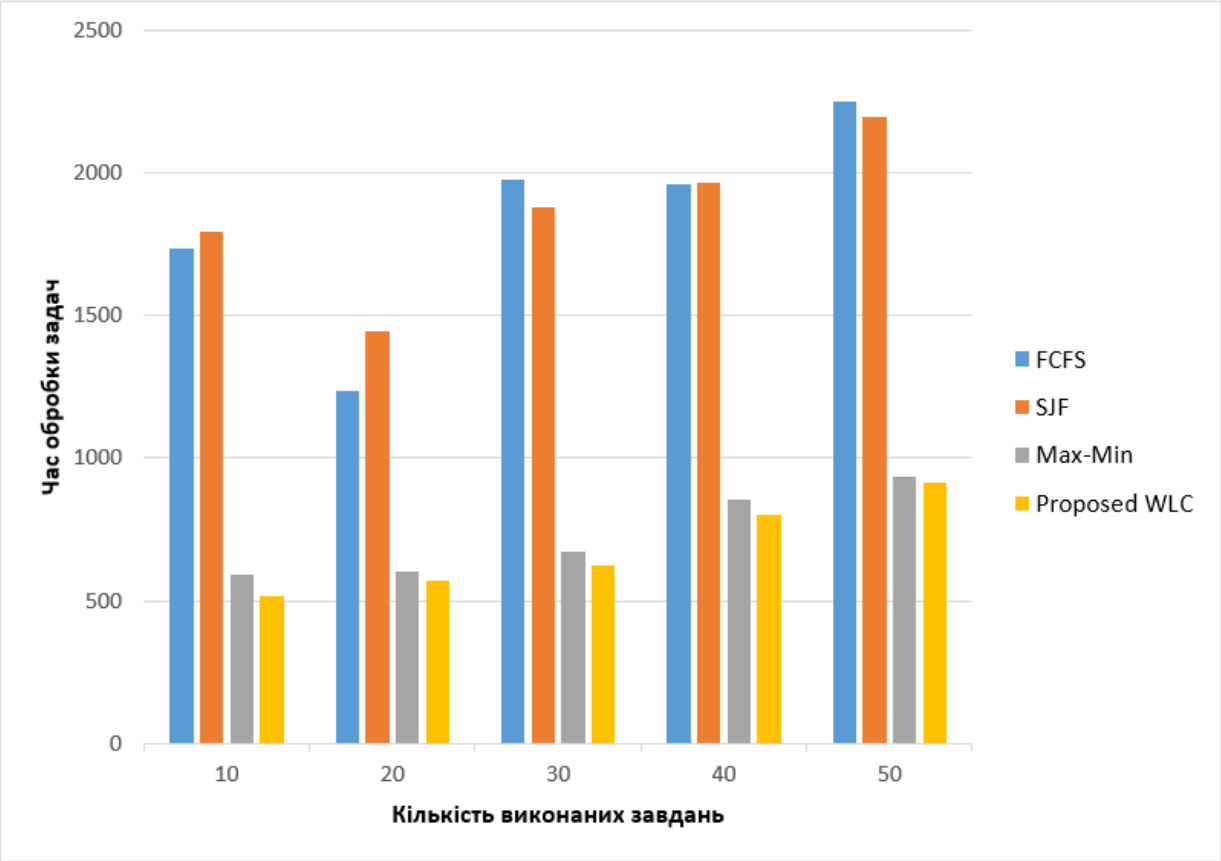
Телелейко І.С.

3. Загальна схема запропонованого способу динамічного балансування навантаження



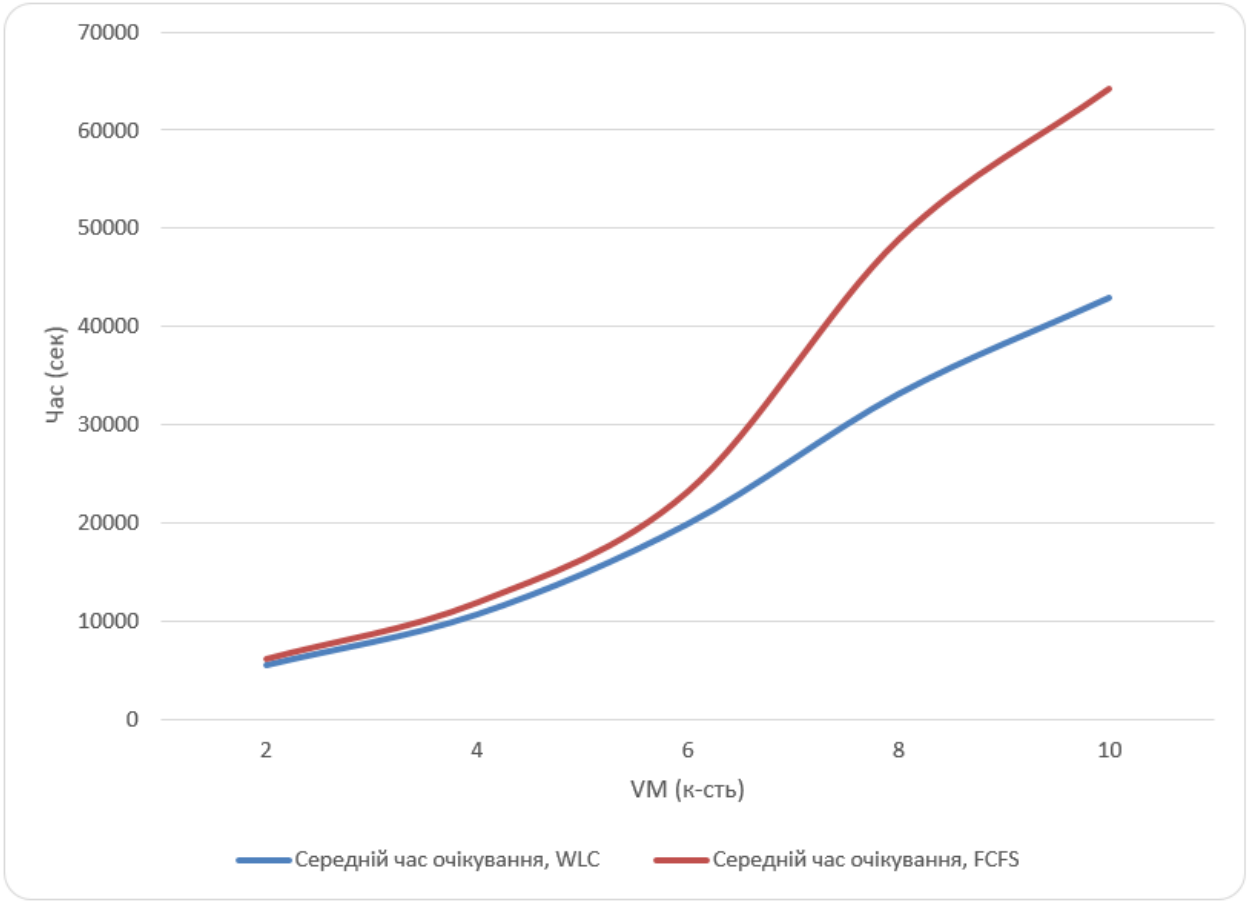
Телелейко І.С.

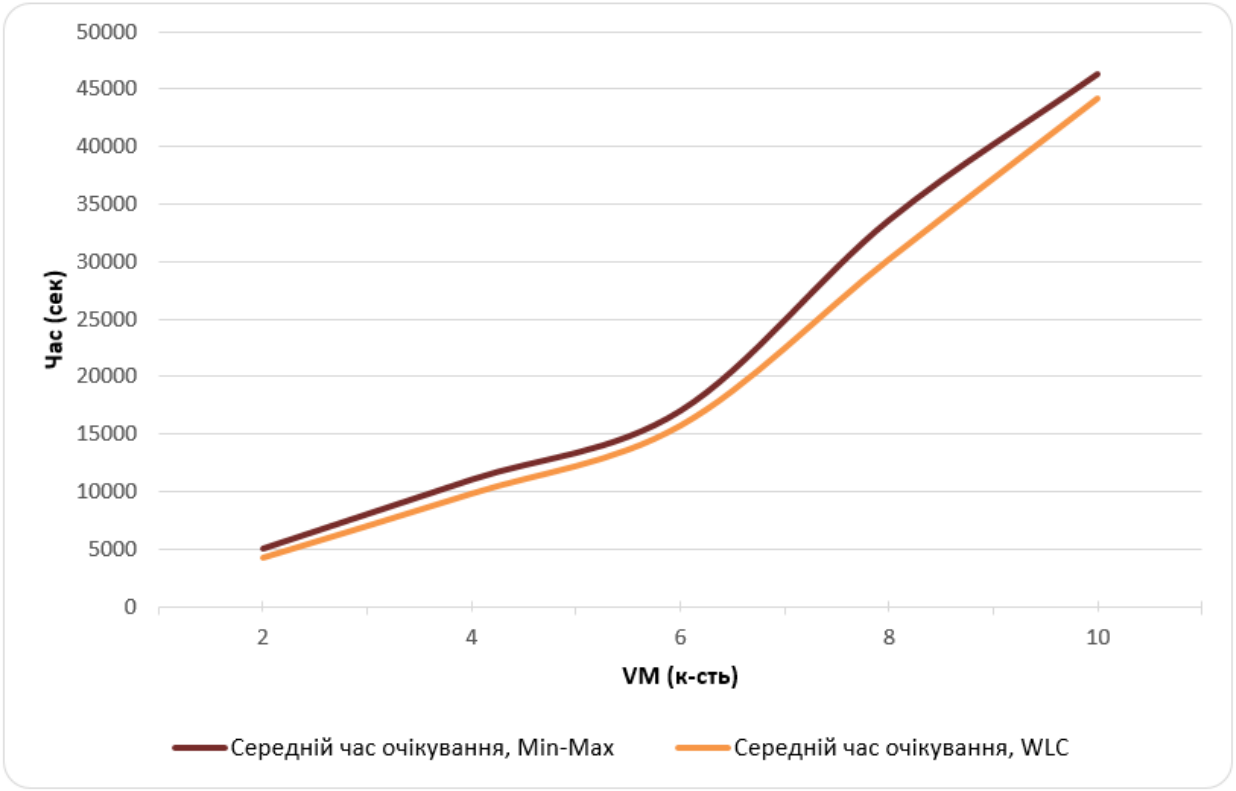
4. Результати проведених досліджень



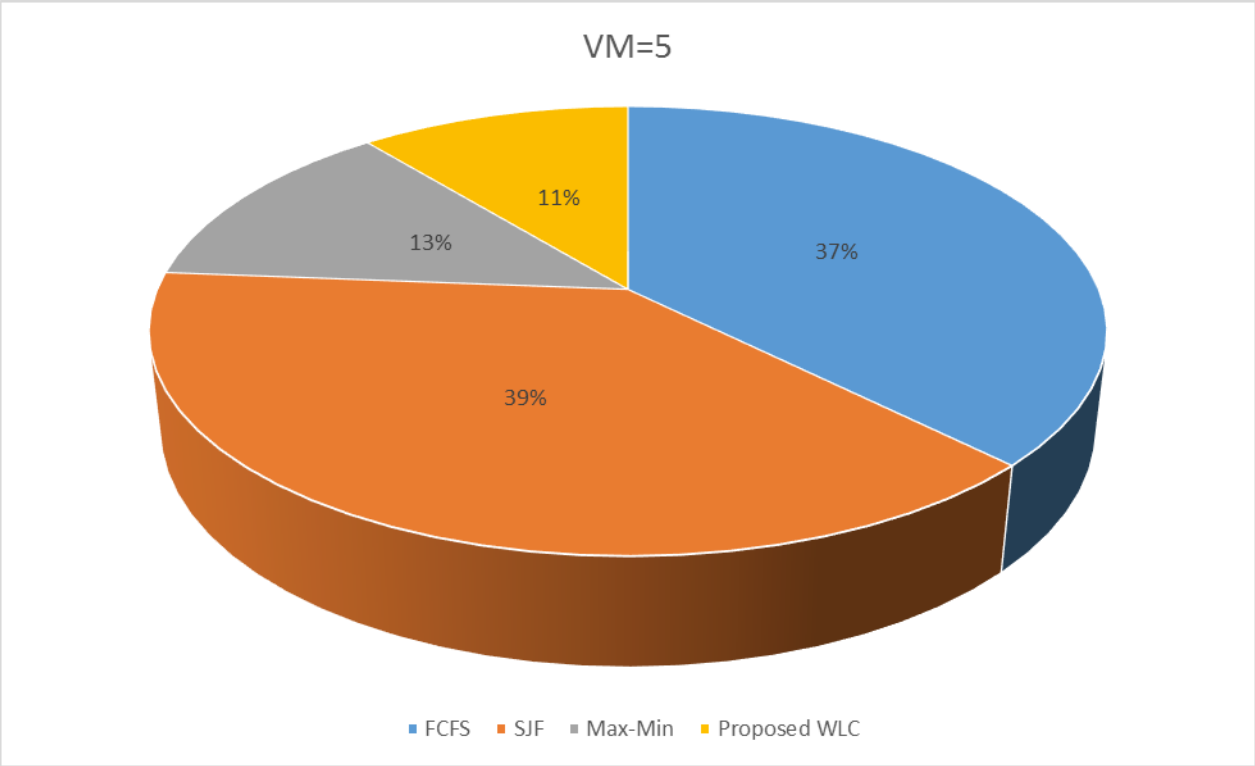
Алгоритм	**Кількість завдань**				
	10	**20**	**30**	**40**	**50**
FCFS	2900	2952	5321	6312	7456
SJF	2330	3456	5073	5623	6022
MAX-MIN	1830	1935	1669	1860	1900
Proposed WLC	1190	1017	1145	1234	1389

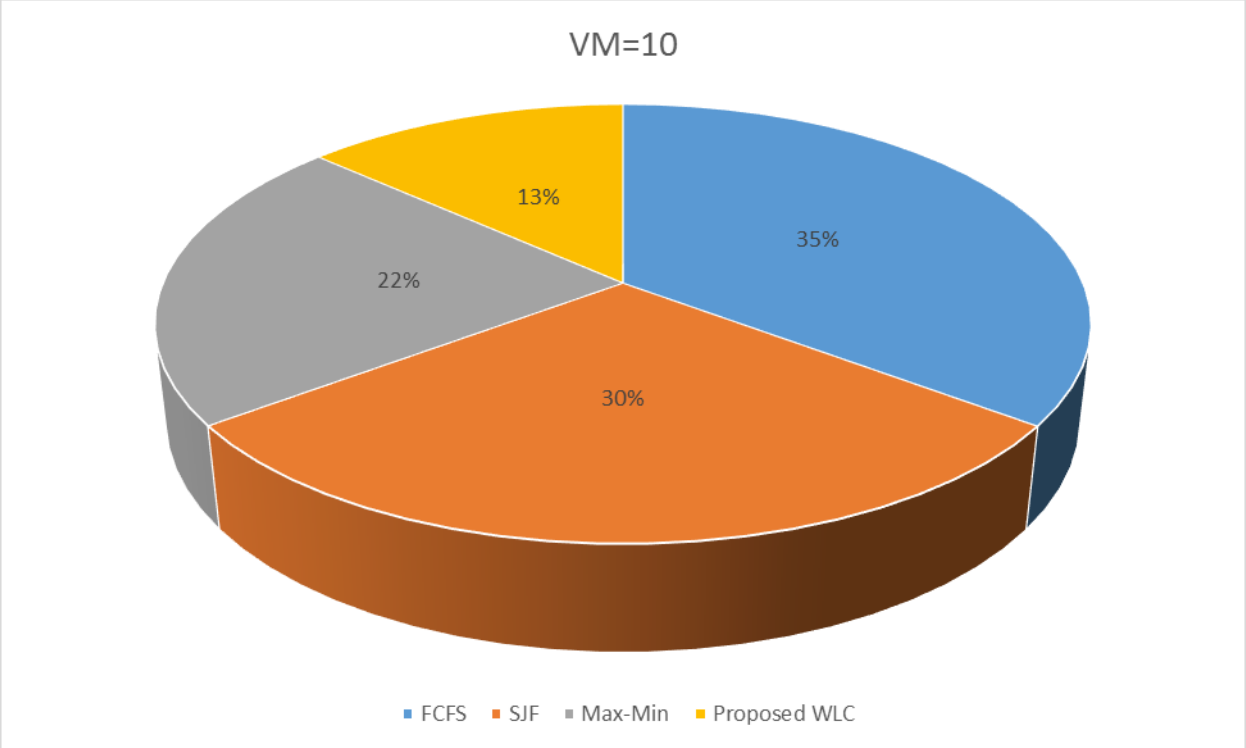
5. Діаграма отриманих результатів порівняння запропонованого способу





6. Діаграма ефективності запропонованого способу





Телелейко І.С.

Додаток 2. Публікації за темою роботи

1. Телелейко І. С. Спосіб динамічного балансування навантаження в хмарному середовищі / І.С. Телелейко, М.М. Орлова // Науковий журнал «Інтернаука. – 2018. – №7. – Електронні дані. – Режим доступу: https://www.inter-nauka.com/issues/2018/7/3687/ .

2. Телелейко І. С. Аналіз методів динамічного балансування навантаження в хмарному середовищі / І. С. Телелейко, М. М. Орлова // X конференція молодих вчених «Прикладна математика та комп'ютинг – ПМК'2018», Київ, 21-23 березня 2018р.: Збірник тез доповідей. – К.: Просвіта, 2018.

3. Телелейко І.С. Сумісність підходів хмарних обчислень та грід-технологій / І.С. Телелейко, М.М. Орлова // 19-а Міжнародна науково-технічна конференція «Системний аналіз та інформаційні технології» SAIT 2017, Київ, 22-25 травня 2017 р. / ННК «ІПСА» НТУУ «КПІ ім. Сікорського». – К.: ННК «ІПСА» НТУУ «КПІ ім. Ігоря Сікорського», 2017. – 340 с.

4. Телелейко І.С. Інтеграція сервіс-орієнтованої архітектури з Grid-системами / І.С. Телелейко, Я.М. Клятченко // Восьма наукова конференція магістрантів та аспірантів «Прикладна математика та комп'ютинг (ПМК-2016)», Київ, 20-22 квітня 2016 р.: Збірник тез доповідей. – К.: Просвіта, 2016. – С.60-64.

Технічні науки

УДК 004.7

Телелейко Інна Сергіївна

магістрант

Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського»

Телелейко Инна Сергеевна

магистрант

Национального технического университета Украины «Киевский политехнический институт имени Игоря Сикорского»

Teleleiko Inna

Graduating Student of the

National Technical University of Ukraine

«Igor Sikorsky Kyiv Polytechnic Institute»

Орлова Марія Миколаївна

кандидат технічних наук, доцент

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Орлова Мария Николаевна

кандидат технических наук, доцент

Национальный технический университет Украины

«Киевский политехнический институт имени Игоря Сикорского»

Orlova Mariia

Candidate of Technical Sciences, Associate Professor

National Technical University of Ukraine

«Igor Sikorsky Kyiv Polytechnic Institute»

СПОСІБ ДИНАМІЧНОГО БАЛАНСУВАННЯ НАВАНТАЖЕННЯ В ХМАРНОМУ СЕРЕДОВИЩІ

Анотація: *У даній статті викладено основні аспекти сучасного стану даної проблеми, описані існуючі підходи до балансування навантаження в*

хмарному середовищі. Запропоновано метод динамічного балансування для оптимізації цих підходів.

Ключові слова: хмарні технології, веб-сервіс, балансування навантаження, алгоритми динамічного балансування.

Аннотация: *В статье изложены основные аспекты современного состояния данной проблемы, описаны существующие подходы балансировки нагрузки в облачной среде. Предложен метод динамической балансировки для оптимизации этих подходов.*

***К*лючевые слова:** облачные технологии, веб-сервис, балансировка нагрузки, алгоритмы динамической балансировки.

Summary: *The article outlined the key aspects of the current state of the problem, describes a study of existing approaches to the technology of load balancing in cloud computing. Dynamic load method was proposed as an optimization for mentioned purpose.*

Key words: cloud technology, web service, load balancing, load balancing algorithm.

Вступ

При побудові розподіленої системи обчислення в зв’язку з розвитком засобів передачі даних все більше використовують підходи розподіленого програмування. Виконання великої задачі розподіляється на менші підзадачі, які можуть виконуватись на різних комп’ютерах зі спільною мережею, а після всіх обчислень результати їх роботи використовуються при обчисленні початкової задачі [2]. При вирішенні деяких задач використання розподіленої системи застосовується для підвищення таких показників ефективності, як зниження вартості, збільшення надійності, досягнення певного рівня продуктивності системи, простоти масштабування тощо.

Важливим аспектом стало застосування механізмів управління ресурсами в локально та глобально розподілених середовищах. Під «ресурсами» маємо на увазі все, що так чи інакше бере участь в обробці даних: обчислювальні кластери, сховища даних, файлові системи, програмне

забезпечення, мережеве устаткування, яке забезпечує з’єднання ресурсів в єдину систему.

При виконанні розподіленої програми, комп’ютери, які приймають участь в обробці даних, з’єднуються з мережею Інтернет. Обмін повідомленнями відбувається за протоколом HTTP, оскільки обмін даними мережею за іншими портами та в іншому форматі без додаткових налаштувань фільтрується більшістю брандмауерами та проксі.

До появи веб-сервісів у світі вже існували технології, що дозволяли додаткам взаємодіяти на відстані, де одна програма могла викликати будь-який інший ресурс, який при цьому міг бути запущений на комп'ютері, розташованому в іншому місті або навіть країні. Такий віддалений виклик процедур називається RPC (Remote Procedure Calling). Прикладом таких технологій є CORBA (Common Object Request Broker Architecture), RMI (Remote Method Invoking). Ідея веб-сервісу полягала в створенні такого RPC, який би взаємодіяв з HTTP пакетом.

Перш ніж викликати віддалену процедуру, необхідно описати виклик в XML файлі формату SOAP (Simple Object Access Protocol, це XML розмітка, яка використовується в веб-сервісах). Все, відправляється через HTTP, спочатку перетворюється в XML опис SOAP, потім застосовується в HTTP пакеті та надсилається на інший комп'ютер в мережі по TCP/IP. WSDL (Web Services Description Language) представляє собою формат XML файлу для опису сервісів мережі як набору кінцевих операцій, які працюють за допомогою повідомлень, які містять документо-орієнтовану інформацію. Документ WSDL повністю описує зовнішній інтерфейс веб-сервісу. Він надає інформацію про послуги, які можна отримати, скориставшись методами сервісу, і способи звернення до цих методів [7].

Далі розглянемо загальний підхід використання веб-сервісу. У веб-сервісах завжди є клієнт і сервер. Сервер – це веб-сервіс (endpoint: кінцева точка, куди доходять SOAP повідомлення від клієнта). Основні кроки реалізації: (i) опис інтерфейсу веб-сервісу; (ii) реалізація інтерфейс; (iii)

запуск веб-сервісу; (iv) створення клієнта і віддалений виклик потрібного методу веб-сервісу.

Веб-сервіси складають єдину концепцію створення таких додатків, функції яких використовують за допомогою стандартних протоколів Інтернет. Реалізація виконується за допомогою технологій, які стандартизовані World Wide Web Consortium (W3C) [7]:

Але все частіше з’являється потреба у користувача в будь-який момент часу мати можливість в нехтуванні ресурсами, яка робоча станція не в змозі забезпечити. Підвищення продуктивності обчислень до прийнятного рівня часто можна досягнути шляхом перерозподілу обчислювальних ресурсів між задачами. Основними характеристиками хмарних обчислень є масштабованість, еластичність, мобільність, необмежений обсяг даних, що обробляються, та можливість нарощувати ресурси [1].

Механізм балансування навантаження

Метою даного дослідження є оцінка можливостей запропонованого методу динамічного балансування навантаження в хмарному середовищі та визначення такого розподілу завдань, який забезпечує приблизно однакове обчислювальне навантаження та мінімальні витрати на передачу даних між ними. Динамічне балансування передбачає перерозподіл обчислювального навантаження на вузли під час виконання програми. Технологія управління розподіленими ресурсами є одною з найважливіших задач, яка направлена на забезпечення керування інформаційної інфраструктури в умовах значного зростання навантаження та збільшення кількості компонентів мережі.

Балансування навантаження в хмарному середовищі відрізняється від класичної моделі архітектури балансування навантаження та її впровадження за допомогою серверів для виконання балансування навантаження, оскільки важко передбачити кількість запитів, які будуть передаватися на сервер. Це забезпечує нові можливості, а також представляє свій унікальний комплекс завдань. Балансування навантаження є однією з центральних проблем в області хмарних обчислень [8]. Це механізм, який рівномірно розподіляє

динамічне локальне навантаження на всіх вузлах по всій хмарі, щоб уникнути ситуації, коли деякі вузли сильно завантажуються, тоді як інші не працюють. Це допомагає досягти підвищення кількості користувачів та використання ресурсів, а отже, покращення загальної ефективності та економічної вигоди ресурсів системи. Він також гарантує, що кожен обчислювальний ресурс розподіляється ефективно та справедливо [9]. Проблема балансування обчислювального навантаження розподіленого додатка виникає з тих причин, що [5]: (i) неоднорідна структура розподіленого додатка: різні логічні процеси вимагають різні обчислювальні потужності; (ii) неоднорідна структура обчислювального комплексу: різні обчислювальні вузли характеризуються різною продуктивністю; (iii) неоднорідна структура міжвузлової взаємодії: лінії зв'язку, що з'єднують вузли, можуть мати різні характеристики пропускної спроможності.

Концептуально схеми балансування навантаження можна розділити на два типи: статичні та динамічні. Статичне балансування виконується на етапі проектування розподіленого додатка. Дуже часто при розподілі логічних процесів на процесори використовується досвід попередніх запусків, застосовуються генетичні алгоритми. Однак попереднє розміщення логічних процесів між процесорами неефективне. Первинною метою оптимізації балансування навантаження є перерозподіл збалансованого навантаження за допомогою завдань та мінімізація потреб між процесами зв'язку з оптимальним використанням ресурсів та часом роботи. Отже, покращення продуктивності обчислювальних вузлів шляхом вирівнювання робочих навантажень елементів обробки є метою балансування навантаження. Балансування навантаження може здійснюватися за допомогою як апаратних, так і програмних інструментів. Застосування динамічного балансування з урахуванням поточного завантаження серверів дозволяє побудувати хмарне середовище, яке оптимально використовує всі наявні ресурси.

Балансування навантаження досягається в середовищі хмари у два етапи: по-перше, це розподіл завдання серед вузлів, другий полягає в тому,

щоб відстежувати віртуальну машину та виконувати операцію балансування навантаження за допомогою міграції завдань або підходу міграції віртуальної машини. Метою планування завдань є створення графіка і присвоєння кожного завдання вузлу (віртуальній машині) за певний період часу, так що всі завдання виконуються за мінімальний проміжок часу.

Три основні етапи потрібні для планування завдання в хмарному середовищі. На етапі користувача подаються робочі місця через графічний інтерфейс користувача або веб-інтерфейс з вимогою на обслуговування в частині якості обслуговування (QoS, Quality of service), апаратного забезпечення, програмного забезпечення тощо. На етапі планувальника фазових завдань виконуються всі завдання планування завдання та операції балансування навантаження. Обробник запиту на роботу переадресовує автентичний запит до планувальника завдань для подальшої обробки, де формується відповідь всім завданням відповідної віртуальної машини та призначеного завдання. Планувальник завдань містить інформацію про стан всіх вузлів (незайняті або зайняті). На останньому етапі в хмарному середовищі формується базова архітектура планування виконання завдань. Такий етап називається фазою на рівні хмари. Центр обробки даних містить хости, і кожен хост містить віртуальну машину.

Статичне балансування навантаження

Статичний алгоритм балансування навантаження потребує додаткової інформації про кількість завдань та інформацію про наявний ресурс. Коли статичний алгоритм працює, немає необхідності постійно стежити за ресурсом. В роботі [10] запропоновано вдосконалений алгоритм заповнення (IBA) за допомогою методу збалансованої спіралі (BS) для зменшення часу обробки задач, алгоритм IBA забезпечує гарантію якості обслуговування в хмарному середовищі, але цей алгоритм неефективний, коли завдання потрапляє у випадковому порядку в систему. Під часом обробки задачі мається на увазі загальний час міграції завдання. Досягнення кращої якості обслуговування з високим використанням ресурсів запропоновано в роботі

[11] завдяки алгоритму IBA з EASY для планування завдання в середовищі хмари. Коли користувач надсилає запит на послугу, він також додає якість параметрів послуг, таких як кінцевий термін, пріоритет, вартість тощо. Інший алгоритм [12] визначає час виконання завдань, враховуючи пріоритет завдання, тобто першорядне завдання буде виконано, по-перше, після виконання непріоритетного завдання.

Динамічне балансування навантаження

Алгоритм динамічного навантаження не передбачає жодних попередніх відомостей про дії в глобальному стані системи, базується виключно на існуючому або поточному стані системи, тобто існує можливість балансування навантаження. В розподіленій системи алгоритм динамічного навантаження виконується всіма вузлами, які присутні в системі, і завдання балансування навантаження розподіляється між ними. Взаємодія між вузлами для здійснення балансування навантаження може мати дві форми: кооперативну та некооперативну. Алгоритми динамічного розподілу навантаження, що мають розподілений характер, часто виробляють більше повідомлень, ніж нерозподілені, оскільки кожний з вузлів у системі повинен взаємодіяти з кожним іншим вузлом. Перевага такого підходу полягає в тому, що навіть якщо один або декілька вузлів не активуються, це не призведе до зупинки всього процесу балансування навантаження [5].

Вищезгадані алгоритми не підходять для середовища в реальному часі, де навантаження на вузол дуже часто змінюються, тобто не можна передбачити майбутнє навантаження, тому будемо використовувати динамічний алгоритм. Немає потреби в попередній інформації про ресурс (віртуальну машину) і завдання в динамічному алгоритмі, оскільки такий тип алгоритму постійно контролює ресурс. А. Лакра та Д. Ядав [13] запропонували алгоритм зменшення оберненого часу, вартості та оптимізації параметру пропускної спроможності. Існують інші типи динамічних алгоритмів, які використовують евристичний підхід, як алгоритм max-min [14] та метаевристичний підхід для вирішення задачі планування задач у

середовищі хмари. Н. Цірітас та ін. [15] запропонував алгоритм зменшення часу виконання та вартості зв'язку за допомогою методу міграції задач. В роботі [16] запропоновано алгоритм динамічного навантаження з урахуванням часу проходження та визначенням середнього коефіцієнта.

В даній роботі запропоновано спосіб динамічного балансування навантаження в хмарному середовищі на основі методу міграції задач та модернізованого алгоритму Weighted Least Connections, а саме алгоритм динамічного навантаження з урахуванням часу обробки завдання та середнього коефіцієнта використання ресурсів як параметра.

Формулювання проблеми:

Планування виконано n завдань в m вузлах (віртуальних машинах) має бути виконано таким чином, щоб користувач хмарного середовища міг виконувати своє завдання за мінімальний час роботи з максимальним використанням ресурсів. Планувальник задач отримує N запитів на завдання (задачі) $T_1, T_2\ T_3\ T_4 \dots\dots\dots\dots T_N$. У даній роботі не розглядається якість параметрів сервісу: пріоритет, вартість тощо. Всі завдання не є пріоритетними та незалежними, кожне завдання має довжину $T\ L_1$, p швидкість обробки, кількість процесорів q, обсяг основної пам’яті r та обов’язково смугу пропускання B. Планувальник задач в хмарному середовищі містить інформацію про віртуальну машину M, а саме швидкість обробки процесора, кількість процесорів, пам’ять, пропускну спроможність $VM_1, VM_2\ VM_3\ VM_4 \dots\dots\dots\dots VM_N$. Для обчислення потенціал окремої віртуальної машини та ємність використовуємо формулу:

$$C_{VM} = p * q \qquad (1)$$

де p - швидкість обробки процесора в мільйонах інструкцій за секунду; q - кількість процесорів зайнятих для виконання завдання.

Потужність всієї віртуальної машини

$$C = \sum_{j=1}^{M} C_{VM} \qquad (2)$$

Завантаження інформації на віртуальну машину: планувальник завдань із хмарного середовища розподіляє завдання на віртуальну машину, кожна віртуальна машина має чергу для зберігання навантаження. Загальна довжина черги у віртуальній машині визначається як навантаження на цю віртуальну машину. Завантаження віртуальної машини можна розрахувати як

$$LVM_{i,t} = \frac{K * TL_{i(t)}}{S(VM_{i,t})} \quad (3)$$

де $K = 1,2,3 \ldots . N$ завдання. $S(VM_{i,t})$ визначається як швидкість обслуговування віртуальної машини в момент часу t, який можна виразити у формі потужності p та кількості процесорів q як $p * x(t)$, де $x = 1,2,3 \ldots . q$.

Навантаження віртуальної машини за часом t обчислюємо як кількість задач на конкретній віртуальній машині, поділену на швидкість обслуговування віртуальної машини. Отже, загальне навантаження на всю віртуальну машину дорівнює

$$L = \sum_{j=1}^{M} LVM_j \quad (4)$$

Якщо заплановане робоче навантаження всієї системи є більшим, ніж її потенціал, то центр обробки даних не зможе впоратися з усім майбутнім запитом, тому або відмовиться від майбутнього запиту на завдання, або збільшить віртуальну машину за допомогою концепції еластичності. Якщо майбутнє робоче навантаження менше, ніж потенціал всієї віртуальної системи, то знаходиться навантаження на кожну окрему віртуальну машину та визначається, чи є віртуальні машини, які перевантажені або завантажені. Якщо всі віртуальні машини перевантажені, то завдання переноситься на завантажену віртуальну машину так, щоб всі завдання могли бути виконані за мінімальний час. Час передачі завдання може бути розрахований TT = довжина завдання (TL) / пропускна спроможність (B).

Знаходимо час виконання завдання T_i на віртуальній машині VM_j

$$T_{exej} = \frac{\sum_i^N E_{ij} * TL_i}{p * q} \quad (5)$$

де $E_{ij} = 1$, якщо завдання T_i призначене віртуальній машині VM_j, інакше значення = 0.

Необхідно визначити час, який складається з часу виконання та передачі завдання (за умови, що будь-яке завдання може переміщуватися з перевантаженої віртуальної машини до завантаженої).

$$MST = \max\left\{\sum_{j=1}^{M} T_{exej}\right\} \quad (6)$$

Опис алгоритму Weighted Least Connections. Запропонований алгоритм є динамічним, адже використовується у кожній перевантаженій станції та контролюється з головної станції. Алгоритм зважених найменших зв'язків використовує «ваговий» компонент на основі відповідних можливостей кожного сервера. Але необхідно заздалегідь вказати або визначити «вагу» кожного сервера.

Балансувальник навантаження, який реалізує алгоритм зважених найменших зв'язків враховує два параметри: вагу (ємність) кожного сервера та поточну кількість клієнтів, які в даний час підключені до кожного сервера. Вибір вузла системи даним способом балансування навантаження виконується на основі кількості активних з'єднань, тобто потужності сервера. Алгоритм Weighted Least Connections динамічний та здатний назначати «вагу» серверам «на льоту».

Аналіз алгоритму Weighted Least Connections. Недоліки алгоритму: немає ніякого сенсу застосовувати цей алгоритм для задач з короткими сесіями, характерними, наприклад, для HTTP протоколу. Для такого виду задач кращим застосуванням буде Round Robin.

Переваги алгоритму: даний алгоритм підійде для задач, пов'язаних з тривалими з'єднаннями. Наприклад, для розподілу навантаження між

серверами бази даних. Якщо на деяких вузлах буде занадто багато активних підключень, нових запитів вони вже не отримають, і навпаки.

Експериментальні результати:

Розрахунок часу обробки: виділяється завдання всій віртуальній машині за алгоритмом планування, після чого розпочинається моніторинг системи віртуальних машин. Завдяки цьому визначаємо стан кожної віртуальної машини, де з'ясовуємо необхідність балансування навантаження – перенесення завдань з перевантаженої віртуальної машини до менш завантаженої. Для досягнення максимальної оптимізації та виявлення достовірних результатів збільшуємо кількість повідомлень та їх довжину. Граничним значенням стає значення перевантаження всієї системи.

Середній коефіцієнт використання ресурсів (Average Resource Utilization Ratio, ARUR): основна мета розподілу навантаження полягає в тому, щоб максимально використовувати ресурси. Середній коефіцієнт використання ресурсів розраховуємо за формулою:

$$ARUR = \left(\frac{\text{середній час}}{\text{час обробки}}\right) * 100 \qquad (7)$$

де середній час = Σ часу, витраченого ресурсом (VM_j), щоб закінчити всю роботу ресурсу, де $j = \{1, 2, 3 \ldots\ldots\ldots M\}$. Діапазон середнього коефіцієнта використання ресурсів становить від 0 до 1, максимальне значення для ARUR становить 1 (використання ресурсу становить 100%), найгірше - 0 (ресурс знаходиться в ідеальному стані).

Для оцінки запропонованого методу була розроблена модель динамічного балансування навантаження в хмарному середовищі на основі методу міграції задач та модернізованого алгоритму Weighted Least Connections. У сценарії моделювання ми порівнюємо продуктивність запропонованої системи з запропонованим алгоритмом балансування навантаження і способом розподілення навантаження та без нього. Під «звичайною» мається на увазі система без використання запропонованого способу. У ході експериментальних досліджень було виявлено зменшення

часу обробки завдання та збільшення середнього коефіцієнта використання ресурсів. Час обробки завдань на 5-ти віртуальних машин при 10, 20 та 30 завдань зменшився від 30% до 10% (у порівнянні з алгоритмом FCFS та з алгоритмом max-min відповідно).

Висновки:

У даній роботі запропоновано метод динамічного балансування навантаження в хмарному середовищі на основі методу міграції задач та модернізованого алгоритму Weighted Least Connections, а саме алгоритм динамічного навантаження з урахуванням часу проходження та середнього коефіцієнта використання ресурсів як параметра. Метою даного методу є ефективне використання ресурсів у хмарному середовищі. Експериментальні результати показали зменшення часу обробки та збільшення середнього коефіцієнта використання ресурсів.

Література

1. Mell, P. The NIST Definition of Cloud Computing [Electronic resource] / Peter Mell, Timothy Grance. – Recommendations of the National Institute of Standards and Technology NIST. – 2011. – SP 800-145. – Електронні дані. – Режим доступу: https://csrc.nist.gov/publications/detail/sp/800-145/final
2. Bass L. Software Architecture In Practice, Second Edition / L. Bass, P. Clements, R. Kazman. – Boston: Addison-Wesley. – 2003. – ISBN 0-321-15495-9. – pp. 21–24.
3. Donaldson V. Program Speedup in a Heterogeneous Computing Network / V. Donaldson, F. Berman, R. Paturi. // Journal of Parallel and Distributed Computing. – September 1994. – Vol. 21. – No 3. – pp. 316-322.
4. Leopold C. Parallel and Distributed Computing. A survey of Models, Paradigms, and Approaches Wiley Series on Parallel and Distributed Computing / C. Leopold // Albert Zomaya Series Editor. – 2001.
5. Телелейко І. Аналіз методів динамічного балансування навантаження в хмарному середовищ / І.С. Телелейко // X Наукова

конференція магістрантів та аспірантів «Прикладна математика та комп'ютинг – ПМК'2018». – 2018.

6. Cloud computing: distributed internet computing for IT and scientific research / [M. D. Dikaiakos, D. Katsaros, P. Mehra and others.] . – Internet Computing, IEEE. – 2009. – №13. – pp. 10–13.

7. Телелейко І. Інтеграція сервіс-орієнтованої архітектури з grid-системами / І.С. Телелейко // VIII Наукова конференція магістрантів та аспірантів «Прикладна математика та комп'ютинг – ПМК'2016. – 2016.

8. Rima B. A Taxonomy and Survey of Cloud Computing Systems / B. P. Rima, E. Choi, and I. Lumb // Proceedings of 5th IEEE International Joint Conference on INC, IMS and IDC, Seoul. – Korea, August 2009. – pp. 44-51

9. Alakeel A. A Guide to dynamic Load balancing in Distributed Computer Systems" / A. M. Alakeel // International Journal of Computer Science and Network Security (IJCSNS) . – Vol. 10. – No. 6. – June 2010. – pp. 153-160.

10. Suresh A. Improving scheduling of backfill algorithms using balanced spiral method for cloud metascheduler / A. Suresh, P. Vijayakarthick // International Conference on Recent Trends in Information Technology. – Chennai. – 2011.

11. Dubey K. A Priority Based Job Scheduling Algorithm Using IBA and EASY Algorithm for Cloud Metaschedular / K. Dubey // International Conference on Advances in Computer Engineering and Applications. Ghaziabad. – India. – 2015.

12. Kumar M. Priority Aware Longest Job First (PA-LJF) algorithm for utilization of the resource in cloud environment / M. Kumar, S.C. Sharma // Computing for Sustainable Global Development (INDIACom). – 3rd International Conference on. IEEE. – 2016.

13. Lakra A. Multi-Objective Tasks Scheduling Algorithm for Cloud Computing Throughput Optimization / A. Lakra, D. Yadav // Procedia Computer Science. – 2015.

14. Ren X. A Dynamic Load Balancing Strategy for Cloud Computing platform based on Exponential Smoothing Forecast / X. Ren // International Conference on Cloud Computing and Intelligence Systems. – China. – 2011.

15. Tziritas N. On minimizing the resource consumption of cloud applications using process migrations / N. Tziritas // Journal of Parallel and Distributed Computing. – 2013.

16. Kumar M. Dynamic load balancing algorithm for balancing the workload among virtual machine in cloud computing / M. Kumar, S.C. Sharmab – 2017.

УДК 519.688

К.т.н., доцент Орлова М.М., студент Телелейко І.С.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

АНАЛІЗ МЕТОДІВ ДИНАМІЧНОГО БАЛАНСУВАННЯ НАВАНТАЖЕННЯ В ХМАРНОМУ СЕРЕДОВИЩІ

Abstract

Maria M. Orlova, assoc. prof., PhD; Inna Teleleiko, student

Analysis of Dynamic load balanced methods in the cloud environment

This paper concerns the task of an actual analysis of existing approaches to dynamic load balancing in cloud environments. This article is included analyze and study of the load balanced of the most dynamic type, methods and algorithms. The purpose of this study is to evaluate the possibilities of using load balancing algorithms and scalar methods in cloud environments.

Вступ

При побудові хмарної інфраструктури важливо вміти розподіляти навантаження між компонентами, проте існуючі рішення з балансування мають ряд обмежень. Потребу в плануванні навантаження необхідно вирішувати на початковій стадії розвитку будь-якого проекту. Проблеми недостатньої продуктивності сервера в зв'язку зі зростанням навантаження можна вирішувати шляхом нарощування потужності сервера або ж за рахунок оптимізації використовуваних алгоритмів, програмних кодів тощо. Але рано чи пізно настає момент, коли і ці заходи виявляються недостатніми та неефективними. Актуальним є аналіз існуючих підходів динамічного балансування навантаження в хмарному середовищі.

Основними характеристиками хмарних обчислень є масштабованість, еластичність, мобільність, необмежений обсяг даних, що оброблюються, та можливість нарощувати ресурси [1]. У даній статті проводиться аналіз та досліджується балансування навантаження саме динамічного виду, методи та алгоритми.

Постановка задачі

Метою даного дослідження є оцінка можливостей використання алгоритмів балансування навантаження на ресурси хмарної інфраструктури та методів масштабування в хмарному середовищі.

У відповідності з поставленою метою слід вирішити наступні завдання: проаналізувати основні переваги та недоліки різних методів та алгоритмів в

хмарі хмарному середовищі; визначити основні вимоги до алгоритмів балансування навантаження.

Балансування у хмарному середовищі

Для визначення ефективності алгоритму балансування навантаження, виділяємо наступні властивості:

- рівномірне завантаження ресурсів системи;
- масштабування: алгоритм повинен зберігати працездатність при збільшенні навантаження;
- передбачуваність: потрібно чітко розуміти, в яких ситуаціях і при яких навантаженнях алгоритм буде ефективним для вирішення поставлених завдань.

Проблема балансування обчислювального навантаження розподіленого додатка виникає з тих причин, що:

- неоднорідна структура розподіленого додатка: різні логічні процеси вимагають різні обчислювальні потужності;
- неоднорідна структура обчислювального комплексу: різні обчислювальні вузли характеризуються різною продуктивністю;
- неоднорідна структура міжвузлової взаємодії: лінії зв'язку, що з'єднують вузли, можуть мати різні характеристики пропускної спроможності.

Концептуально схеми балансування навантаження можна розділити на два типи: статичні та динамічні. Статичне балансування виконується на етапі проектування розподіленого додатка. Дуже часто при розподілі логічних процесів на процесори використовується досвід попередніх запусків, застосовуються генетичні алгоритми. Однак попереднє розміщення логічних процесів між процесорами неефективне.

Динамічне балансування передбачає перерозподіл обчислювального навантаження на вузли під час виконання програми. Програмне забезпечення, що реалізує динамічне балансування, враховує:

- завантаження обчислювальних вузлів;
- пропускну спроможність ліній зв'язку;
- частоту обмінів повідомленнями між логічними процесами розподіленого додатка тощо.

Балансування навантаження може здійснюватися за допомогою як апаратних, так і програмних інструментів. Застосування динамічного балансування з урахуванням поточного завантаження серверів дозволяє побудувати хмару хмарне середовище, яке оптимально використовує всі наявні ресурси.

Алгоритм динамічного навантаження не передбачає жодних попередніх знань про дії в роботі або глобальному стані системи, тобто рішення щодо балансування навантаження. Вони базуються виключно на існуючому або поточному стані системи. У розподіленій системи алгоритм динамічного

навантаження виконується всіма вузлами, які присутні в системі, і завдання балансування навантаження розподіляється між ними. Взаємодія між вузлами для здійснення балансування навантаження може мати дві форми: кооперативну та некооперативну.

У кооперативній формі вузли працюють поруч один за одним, щоб досягти спільної мети, наприклад, для просування загального часу відгуку. У некооперативному варіанті кожен вузол працює незалежно в напрямку локальної для нього мети, наприклад, для просування часу відповіді локального завдання. Алгоритми динамічного розподілу навантаження, що мають розподілений характер, часто виробляють більше повідомлень, ніж нерозподілені, оскільки кожен з вузлів у системі повинен взаємодіяти з кожним іншим вузлом. Перевага такого підходу полягає в тому, що навіть якщо один або декілька вузлів не активуються, це не призведе до зупинки всього процесу балансування навантаження. Така система впливає на продуктивність системи.

У нерозподіленому типі або один вузол, або група вузлів виконують задачу балансування навантаження. Алгоритми динамічного розподілу навантаження нерозподіленого характеру можуть мати дві форми: централізовану та напіврозподілену. У централізованому алгоритмі балансування навантаження розподіляється лише через один вузол у загальній системі: центральний вузол. Він відповідає за балансування навантаження всієї системи. Інші вузли взаємодіють лише з центральним вузлом. Однак у напіврозподіленій формі вузли поділяються на кластери, де балансування навантаження в кожному кластері має централізовану форму. У кожному кластері вибирається центральний вузол за допомогою відповідної методики вибору, яка забезпечує балансування навантаження всередині цього кластера, тобто балансування навантаження повної системи здійснюється через центральні вузли кожного кластера.

Централізоване динамічне навантаження вимагає менше повідомлень для прийняття рішення, оскільки кількість загальних взаємодій у системі різко зменшується в порівнянні з напіврозподіленою формою. Проте централізовані алгоритми можуть створювати вузьке місце в системі на центральному вузлі, а процес балансування навантаження виявляється безнадійним, коли центральний вузол виходить з ладу. Тому цей алгоритм, в основному, підходить для мереж невеликого розміру.

Далі розглянемо алгоритми, для кожного з якого будуть вказані показники ефективності, які в ньому використовуються.

Алгоритми балансування навантаження

Round Robin або алгоритм кругового обслуговування, являє собою перебір по круговому циклу: перший запит передається одному серверу, потім наступний запит передається іншому і так до досягнення останнього сервера, а потім все починається спочатку, процеси розподілені між усіма

процесорами [2]. Найпоширенішою імплементацією цього алгоритму є, звичайно ж, метод балансування Round Robin DNS. Однак розподіл робочого навантаження між процесорами однаковий, а час обробки завдання для різних процесів різний. Тому в будь-який момент часу лише деякі вузли можуть бути дуже завантажені. Round Robin DNS часто використовується для завантаження запитів балансу між кількома веб-серверами. Головні показники ефективності: пропускна спроможність, час відгуку, витрати, використання ресурсів, продуктивність.

Connection Mechanism або алгоритм балансування навантаження також може базуватися на найменшому механізмі зв'язку, який є компонентом алгоритму динамічного планування. Це вимагає підрахунку кількості підключень для кожного сервера динамічно для приблизного навантаження. Цей алгоритм балансування навантаження відстежує номер підключення кожного сервера. Кількість посилань додається, коли до неї надсилається нове з'єднання, і зменшується, коли закінчується з'єднання або відбувається тайм-аут [3]. Головні показники ефективності: пропускна спроможність, відмовостійкість, використання ресурсів, масштабування.

A Task Scheduling Algorithm Based on Load Balancing або алгоритм планування задач, що базується на балансуванні навантаження: дворівневий метод планування виконання завдань, що базується на балансуванні навантаження для сприйняття динамічних вимог користувачів і отримання високого використання ресурсів. Це забезпечує балансування навантаження за допомогою першого планування завдань для віртуальних машин, зберігає розміщення ресурсів, збільшує час відповіді ресурсу, споживання ресурсів та загальної продуктивності середовища обчислень у хмарі.

Алгоритм Randomized має тип статичного характеру. У цьому алгоритмі процес можна обробляти певним вузлом *n* з ймовірністю *p*. Порядок розподілу процесу зберігається для кожного процесора незалежно від розподілу з віддаленого процесора. Цей алгоритм добре забезпечує процес рівномірного завантаження. З іншого боку, проблема виникає, коли навантаження мають різні обчислювальні складності. Рандомізований алгоритм не підтримує детерміністичний підхід.

Active Clustering – це кластерний алгоритм, який вводить поняття кластеризації в хмарні обчислення. У хмарних обчисленнях існує багато алгоритмів балансування навантаження. Кожен алгоритм має свої переваги та недоліки. Залежно від вимоги, використовується один з алгоритмів. Продуктивність алгоритму може бути покращена шляхом створення кластера вузлів. Кожен кластер можна вважати групою. Принцип активної кластеризації – об'єднати подібні вузли, а потім працювати над цими групами. Процес створення кластера обертається навколо концепції вузла збігання. Продуктивність системи підвищується завдяки високій доступності ресурсів, тим самим збільшуючи пропускну спроможність. Це збільшення пропускної спроможності пояснюється ефективним використанням ресурсів. Головні показники ефективності: час міграції, витрати, використання ресурсів.

Висновок

При проектуванні та створенні хмарних рішень важливе місце займає забезпечення надійності та оптимальності використання ресурсів:

- моніторинг – отримання інформації про доступність, завантаження апаратних ресурсів платформ з копіями розподіленого додатка.
- балансування – розподіл призначених для користувача запитів, що надходять між наявними апаратними платформами.

В роботі проаналізовані основні алгоритмів балансування навантаження за різними показниками продуктивності, тобто для кожного алгоритму вказані показники ефективності, які в ньому використовуються. Існуючі способи балансування навантаження та алгоритми зосереджені на скороченні накладних витрат, зменшенні часу міграції та підвищення продуктивності. Це важливі завдання при розробці хмарної платформи, які дозволяють підвищити пропускну спроможність. Для подальших досліджень важливе розуміння всіх розглянутих методів балансування навантаження та підходів. Питання балансування навантаження хмарних сервісів потребує вдосконалення, а в багатьох аспектах – першочергових розробок та напрацювань.

Література

17. Mell, Peter. The NIST Definition of Cloud Computing / Mell, Peter and Grance, Timothy // Recommendations of the National Institute of Standards and Technology. NIST [Електронний ресурс]. – Режим доступу: https://csrc.nist.gov/publications/detail/sp/800-145/final
18. Zhong Xu. Performance Study of Load Balanacing Algorithms in Distributed Web Server Systems / Zhong Xu, Rong Huang // CS213 Parallel and Distributed Processing Project Report. 2009.
19. P.Warstein/ Load balancing in a cluster computer / P.Warstein, H.Situ and Z.Huang // In proceeding of the seventh International Conference on Parallel and Distributed Computing, Applications and Technologies, IEEE. – 2010.
20. https://www.slideshare.net/kuashaknight/load-analysis-and-structural-consideration

Орлова М.М., Телелейко І.С.
Факультет прикладної математики НТУУ «КПІ», Київ, Україна

Інтеграція підходів хмарних обчислень та грід-технологій

Існує безліч обчислювальних систем різного масштабу для обробки інформації, які застосовуються в різноманітних сферах діяльності. Найбільший інтерес представляють системи, які обробляють надвеликі обсяги даних.

Для грід характерна відсутність центру управління оброблюваних ресурсів, використання відкритих стандартів та спеціалізованого рівня обслуговування. Грід передбачає колективний розподілений режим доступу до ресурсів і пов`язаних з ними послуг у межах глобально-розподілених віртуальних організацій. За допомогою грід-обчислень є можливість надавати обчислювальні ресурси як утиліти, які можуть бути включені або вимкнені. Структурою грід є віртуальна організація. Її задача - створення середовища, де фрагменти одного додатку можуть оброблятися на різних комп'ютерах та мають взаємозв'язок між собою. Віртуальна організація утворюється над простором реальних комп'ютерів, мереж, адміністративних зон, центрів обчислювання даних.

Хмарна інфраструктура дуже серйозно спрощує та здешевлює створення грід-додатків. Якщо існують дані, які потребують обробки, необхідно відправити запит на сервер для обробки цих даних. Після завершення обробки цей сервер можна зупинити або можна відправити запит для обробки нової порції даних. Хмарні обчислення можна розглядати у двох напрямах: зі сторони власника обчислювальних ресурсів та користувача. У першому випадку, вони орієнтовані на надання інформаційних ресурсів зовнішнім користувачам. У другому – це отримання інформаційних ресурсів у вигляді послуги у зовнішнього постачальника, розрахунок якої залежить від кількості спожитих ресурсів відповідно до встановленого тарифу.

Обидва типи обчислень включають в себе багатостроковість та багатозадачність, що означає, що безліч користувачів можуть виконувати різні завдання, маючи доступ до одного чи кількох екземплярів зразків. Спільне використання ресурсів серед великого потоку користувачів допомагає в зниженні витрат на інфраструктуру та максимальної пропускної спроможності навантаження. Хмара та грід-обчисленння забезпечують домовленість про рівень обслуговування (SLA) для гарантованої доступності безперебійної роботи, близько 99 відсотків. Якщо обслуговування послуг нижче рівня гарантованого обслуговування безперебійної роботи, користувач отримує кредит на сервіс для прийому даних. Таким чином, можна вважати, що грід та хмарні обчислення доповнюють один одного. Грід обчислення забезпечує об'єднання гетерогенних обчислюваних ресурсів у єдине обчислювальне середовище, забезпечуючи те, з чого починалися та на чому базуються хмарні обчислення: мають більш високий рівень абстракції, надаючи обчислювальні ресурси кінцевим користувачам у вигляді сервісів. Грід та хмари працюють на різних рівнях організації масштабованих обчислювальних процесів та процесів обробки даних. Тому доцільно інтегрувати ці два підходи в межах мінімізації витрат на обчислення та підвищення ефективності використання обладнання.

Обмін контенту між хмарою та призначеним для користувача терміналом сприяє створенню накладних витрат, які часто не компенсуються підвищеною швидкодією хмарних серверів. У якості підстав, які ускладнюють глобальне впровадження виділимо складність конфігурування, концептуальну складність, необхідність у модифікації існуючого чи створення власного, спеціалізованого ПО. Але хмарні обчислення вже на рівні IaaS надають добре масштабовані віртуальні апаратні ресурси у вигляді віртуальних машин та додаткових двох рівнів: PaaS та SaaS. Отже, для збереження масштабованості та застосування всіх апаратних ресурсів є гібридизація підходів типу накладання структур. Прикладом може стати розвертання грід на хмару на рівні інфраструктури, або реалізація хмарної платформи у складі інфраструктури грід.

Література. 1. George Reese, Cloud Application Architectures. O`Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, 2009, **2.** *Ian F., Yong Z., Ioan R., Shiyong L.* Cloud Computing and Grid Computing 360-Degree Compared [электронный ресурс] // Microsoft Academic Search. URL: http://acade-mic.research.microsoft.com/, 2008, http://academic.research.microsoft.com/Publication/50721241 (дата звернення:19.02.2017) **3.** Dynamic Virtual Clustering with Xen and Moab / W. Emeneker, D. Jackson, J. Butikofer [et al.] // ISPA06: Workshop on XEN in HPC Cluster and Grid Computing nvironments (XHPC). – Sorrento, Italy, 2006. – P. 440 – 451.

УДК 004.75

К.т.н. Клятченко Я.М., студентка Телелейко І.С.

Національний технічний університет України
«Київський політехнічний інститут»

ІНТЕГРАЦІЯ СЕРВІС-ОРІЄНТОВАНОЇ АРХІТЕКТУРИ З GRID-СИСТЕМАМИ

Abstract

Yaroslav M. Klyatchenko, prof., PhD; Inna Teleleiko, student
Ability to integrate service-oriented architecture with Grid-systems

This paper describes SOA-technologies and their relationship with modern Grid-systems. Service-oriented architecture (SOA) is the result of needs of large IT companies in the approach for creation of large corporate software products, based on industrial integration platform with multiple reusable functional IT-elements. This approach is adopted for the implementation of distributed programming paradigms, such as Grid-computing.

Вступ

Оскільки більшість Web-сервісів всесвітньої павутини є готовими платформами для створення та використання розподілених машинно-орієнтованих систем, то одним із сучасних підходів для вирішення проблем, що пов’язані з переходом на нові програмні платформи із збереженням старих, є використання сервіс-орієнтованої архітектури. При цьому, Web-технології забезпечують як вільний доступ, так і спільне використання інтернет-ресурсів, а концепція Grid-систем орієнтована на створення інфраструктури нового типу, що має розширені функціональні можливості як, наприклад, спільне використання та агрегування автономних територіально розподілених ресурсів [1].

В даній статті проводиться аналіз та досліджується архітектура узгодження Grid-систем та Web-сервісів. Ця архітектура об’єднує технології Grid з Web-сервісами, що дозволяє створювати основу інфраструктури для інтеграції, візуалізації та управління ресурсами.

Постановка задачі

Створення системи управління розподіленими ресурсами є однією із найважливіших задач, що направлена на забезпечення управління інформаційною інфраструктурою в умовах зростання навантаження та збільшення кількості користувачів мережі.

Для цього вирішуються наступні задачі:

- аналіз принципу роботи Web-сервісу;
- застосування переваг SOA з можливостями Grid-технологій.

Термінологія

Сервіс-орієнтована архітектура (*Service-Oriented Architecture, SOA*) це така архітектура додатків, в якій компоненти або «сервіси», маючи узгоджені спільні інтерфейси, використовують єдині правила для визначення того, як викликати сервіси і як вони будуть взаємодіяти між собою [2].

Сервіс – програмний компонент, до якого можна дистанційно звернутися за допомогою комп'ютерної мережі, і який надає деякі функціональні можливості [2].

Служба – програмна система, що ідентифікується рядком URI, чиї загальнодоступні інтерфейси визначені мовою XML (eXtensible Markup Language).

Web-сервіс – це абстрактне поняття, що реалізовано агентом. Агент – це апаратний або програмний засіб, який відсилає та приймає повідомлення. Під сервісом мається на увазі ресурс, який характеризується абстрактним набором наданих функцій. Таким чином, певний Web-сервіс може бути реалізований деякою кількість агентів, що надають однаковий набір функцій. З технічного боку Web-сервіс – це програмна система, що створена з метою надання можливості взаємодії між обчислювальними засобами по комп'ютерній мережі. Web-сервіс має інтерфейс доступу, який описано в спеціальному форматі (наприклад, за допомогою мови опису Web-сервісів *WebService Description Language, WSDL*). Інші системи взаємодіють з Web-сервісом через обмін повідомленнями по протоколу HTTP [3].

Grid (*Grid-computing*) – позначення універсальної програмно-апаратної інфраструктури, що об'єднує комп'ютери та суперкомп'ютери в територіально розподілену інформаційно-обчислювальну систему [4].

Open Grid Services Architecture (*OGSA*) являє собою набір стандартів, що пристосовують Web-сервіси та сервіс-орієнтовані архітектури до різних компонент Grid-систем. У цьому контексті Grid-система є масштабованою глобальною мережею (*wide area network, WAN*), яка підтримує спільне використання та розподіл ресурсів [5].

Опис систем

Web-сервіси використовують стандартні протоколи Інтернет для створення додатків. Прикладом такої реалізація є World Wide Web Consortium (W3C). На рис. 1 представлено приклад Web-сервісу.

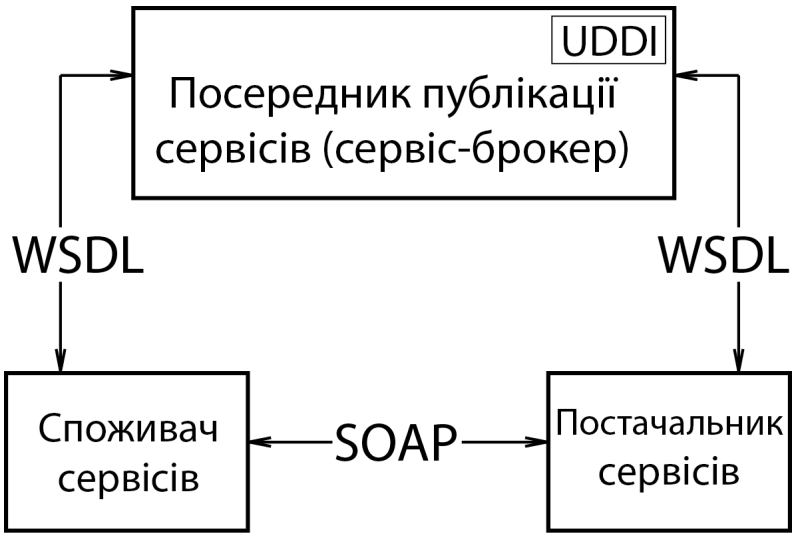


Рис. 1. Приклад Web-сервісу

На першому кроці перед викликом віддаленої процедури, цей виклик подається як XML-файл в одному із форматів SOAP (XML-розмітка, що використовуються в Web-сервісах). Все, що відправляється через протокол HTTP, спочатку перетворюється в XML-опис SOAP, а потім записується в пакет HTTP та пересилається на інший комп'ютер по мережі TCP/IP. WSDL являє собою формат XML-файлу для опису сервісів мережі як набору кінцевих операцій, що функціонують за допомогою повідомлень та містять документо-орієнтовану інформацію. Документ WSDL повністю описує зовнішній інтерфейс Web-сервісу. Він надає інформацію про послуги, які можна отримати, скориставшись методами сервісу, і способи звернення до цих методів.

Технологія Universal Description, Discovery and Integration (UDDI) формує каталог Web-сервісів. Підключившись до цього реєстру, споживач зможе знайти ті Web-сервіси, які найкращим чином задовольняють його вимогам. Технологія UDDI дає можливість пошуку і публікації потрібного сервісу, як людиною, так і програмою-клієнтом.

Web-сервіси позиціонуються як програмне забезпечення так званого проміжного шару. Використовувати їх можуть як клієнтські програми, що безпосередньо працюють з користувачем, так і інші додатки (в тому числі і інші Web-сервіси). Ієрархію вищезгаданих протоколів представлено на рис. 2:

Рис. 2. Протоколи Web-сервісів

Web-сервіси створюють основу для взаємодії між додатками, дозволяючи більш ефективно використовувати можливості Web за рахунок підтримки автоматизованих процесів. Паралельно з цим створюється значна мережева інфраструктура обміну програмами, обчислювальними послугами і даними, яка отримала назву Grid. Ця система підтримує реалізацію інтегрованих обчислювальних середовищ, в яких компанії можуть спільно використовувати дані, програми і обчислювальні ресурси. Grid можна розглядати як розширення Web, що за своїми функціями виходить за рамки спільного використання інформації, даючи можливість спільно використовувати будь-які комп'ютерні ресурси.

Фірми Globus Project і IBM ініціювали проект, в якому поєднали технології Grid з технологіями Web-служб за допомогою відкритої архітектури Open Grid Services Architecture (OGSA). OGSA вирішує поточні питання і проблеми, такі як аутентифікація, авторизація, ведення переговорів з питань політики адміністрування домовленостей про рівень обслуговування клієнтів мережі, управління віртуальними організаціями та інтеграції даних про клієнтів. Програмні компоненти повинні звертатися до служб, що виділяють необхідні ресурси та організують роботу з ними. Така архітектура дозволяє інтегрувати служби та ресурси розподілених, гетерогенних, динамічних середовищ та спільнот. Щоб добитися такого інтегрування, в моделі OGSA використовується мова опису Web-служб WSDL і визначається концепція Grid-служб.

OGSA визначає вимоги до апаратної частини, платформ та програмного забезпечення, що базуються на основі стандартів розподілених обчислень. За своєю сутністю OGSA є розширенням (уточненням) сервіс-орієнтованої архітектури (SOA) [6]. При наявності сотень сервісів користувач може реалізувати централізовану систему адміністрування за допомогою архітектур SOA [6].

Висновки

Поєднання переваг SOA з можливостями Grid-технологій дозволить здійснити інтеграцію не тільки локальних, але і територіально розподілених інформаційних ресурсів. Саме завдяки інтеграції механізму тимчасових сервісів із вже існуючими технологіями Web-сервісів, OGSA значно збагачує можливості останніх, але, в той же час, не вимагає повної модернізації існуючих технологій. Основні принципи створення таких мережевих систем управління завданнями вже існують достатньо давно, але використання технологій Grid дають нові можливості реалізації систем управління обчислювальними розподіленими ресурсами.

В даний час, перебуваючи в стадії постійної розробки, OGSA може забезпечувати міцну основу для майбутніх інфраструктур Grid. Подальші

дослідження необхідно проводити в руслі розробки нових алгоритмів і методів рішення прикладних задач в середовищах розподілених обчислень.

Література

1. *Foster I.* What is the Grid? A Three Point Checklis // Department of Computer Science, University of Chicago, Chicago, IL 60637, July 20, 2002, http://wwwfp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid. pdf.
2. *Богданов А.В., Станкова Е.Н., Мареев В.В.* Сервис-ориентированная архитектура: новые возможности в свете развития Grid-технологий // Всероссийский конкурсный отбор обзорно-аналитических статей по приоритетному направлению "Информационно-телекоммуникационные системы", 2008. – 3-32 с.
3. Web Services Architecture – W3C Working Group Note 11. — The World Wide Web Consortium (W3C), 2004. — URL: http://www.w3.org/TR/ ws-arch/. (дата звернення: 10.03.2016)
4. *Foster I., Kesselman C.* The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, 1998. – 701 p.
5. *Joshy Joseph, Craig Fellenstein* Grid computing. // IBM Press series - information management.; On demand series, Upper Saddle River, NJ : Prentice Hall, 2004. – 117 p.
6. *Лори Маквитти* Архитектура SOA как она есть. URL: http://www.ccc.ru/magazine/depot/06_02/read.html?0104.htm (дата звернення: 10.03.2016)

Додаток 3. Довідка про впровадження

ТОВ «ПРІНТ СІСТЕМ»,
м. Київ, вул. Алма-Атинська, 2/1, оф.33
тел: (044) 502-31-98
www.printsystem.com.ua

Від _____2018 р. № _____
на № _____ від _____

ДОВІДКА

**про впровадження результатів наукових досліджень
з теми магістерської дисертації
Телелейко Інни Сергіївни**

Керівництво ТОВ «ПРІНТ СІСТЕМ» підтверджує впровадження у робочий процес результатів дисертаційної роботи на присвоєння ступеня магістра Телелейко Інни Сергіївни за темою «Спосіб планування та динамічного балансування навантаження на модулі хмарного середовища» зі спеціальністю 123 Комп'ютерна інженерія («Комп'ютерні системи та компоненти»).

Результати, висновки та рекомендації дисертаційного дослідження І.С. Телелейко були використані у робочому процесі ТОВ «ПРІНТ СІСТЕМ».

Використання запропонованого способу планування та балансування навантаження дало позитивні результати і заслуговує як схвальної оцінки, так і широкого застосування.

Директор ТОВ «ПРІНТ СІСТЕМ» Л. В. Станіславська