НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Навчально-науковий комплекс "Інститут прикладного системного аналізу"
(повна назва інституту/факультету)

Кафедра Системного проектування
(повна назва кафедри)

«На правах рукопису»
УДК 004.75

«До захисту допущено»

Завідувач кафедри

_____ _____
(підпис) (ініціали, прізвище)

“___”_____20__ р.

Магістерська дисертація

зі спеціальності 8.05010103 Системне проектування
(код і назва спеціальності)

на тему: Балансування навантаження додатків у "хмарному"середовищі

Виконав: студент 6 курсу, групи ДА-32м
(шифр групи)

Бабенко Олександр Анатолійович _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник к.т.н. Гіоргізова-Гай В.Ш. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант Охорона праці к.б.н., доц. Гусєв А.М. _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент Завідувач кафедри інформаційно-телекомунікаційних мереж д.т.н., проф. Глоба Л.С _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

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

Київ – 2015 року

Національний технічний університет України
«Київський політехнічний інститут»

Факультет (інститут) ННК “Інститут прикладного системного аналізу”
(повна назва)

Кафедра Системного проектування
(повна назва)

Освітньо-кваліфікаційний рівень «Магістр»
(назва ОКР)

Напрям підготовки 6.050101 Комп’ютерні науки
(код і назва)

Спеціальність 8.05010103 Системне проектування
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)

«___» _____2015 р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Бабенку Олександру Анатолійовичу
(прізвище, ім’я, по батькові)

1. Тема дисертації: Балансування навантаження додатків у "хмарному"середовищі
науковий керівник дисертації Гіоргізова-Гай Вікторія Шалвівна, к.т.н., доц. ,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «13» лютого 2015 р. № 19/1-ст

2. Строк подання студентом дисертації 08.06.2015

3. Об’єкт дослідження: Системи балансування навантаження в "хмарних" технологіях

4. Предмет дослідження: Методи і алгоритми балансування навантаження додатків у "хмарному" середовищі

5. Перелік завдань, які потрібно розробити:

1. Провести систематизацію поширених методів і алгоритмів балансування навантаження додатків в "хмарних" системах.
2. Зробити порівняльну характеристику розповсюджених систем балансування навантаження.
3. Надати практичні рекомендації по налаштуванню системи балансування навантаження платформи CloudStack на прикладі додатку дистанційного відео навчання.

4. Експериментально дослідити ефективність створених правил балансування навантаження для вибраного додатку.

6. Орієнтовний перелік ілюстративного матеріалу: презентація по темі «Балансування навантаження "хмарних" додатків»

7. Орієнтовний перелік публікацій

Бабенко О.А. Балансування навантаження "хмарних" додатків / Бабенко О.А., Гіоргізова-Гай В.Ш. // Системний аналіз та інформаційні технології: материалы 17-й Міжнародної науково-технічної конференції SAIT 2015, Київ, 22-25 червня 2015 р. – 2015. – с.184

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	доц., к.б.н. Гусєв А. М		
Основна частина	доц., к.т.н. Гіоргізова-Гай В.Ш.		

9. Дата видачі завдання 30.09.2014

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів магістерської дисертації	Примітка
1	Отримання завдання	30.09.2014	
2	Збір інформації	15.02.2015	
3	Аналіз вимог завдання, вибір методів і засобів розв’язання поставленої задачі	28.02.2015	
4	Систематизація методів і алгоритмів балансування навантаження	10.03.2015	
5	Порівняння можливостей розповсюджених систем балансування навантаження	20.03.2015	
6	Налаштування системи балансування навантаження платформи CloudStack для додатку дистанційного відео навчання	05.04.2015	
7	Навантажувальне тестування створеної системи та надання практичних рекомендацій	25.04.2015	
8	Розробка розділу з охорони праці	10.05.2015	
9	Оформлення дипломної роботи	31.05.2015	
10	Отримання допуску до захисту та подача роботи в ДЕК	10.06.2015	

Студент _____ О.А. Бабенко
(підпис) (ініціали, прізвище)

Керівник роботи _____ В.Ш. Гіоргізова-Гай
(підпис) (ініціали, прізвище)

РЕФЕРАТ

на магістерську дисертацію

виконану на тему: Балансування навантаження "хмарних" додатків

студентом: Бабенком Олександром Анатолійовичем

Робота виконана на 151 сторінках, містить 26 ілюстрацій, 4 таблиці. При підготовці використовувалася література з 30 різних джерел.

Актуальність теми

В зростанні популярності і збільшенні користувачів мережі Інтернет останнім часом важливу роль відіграють «хмарні технології» і Web сервіси, які належать до високонавантажених систем. Тому можливості масштабування та балансування навантаження при побудові "хмарних" систем мають особливе значення. Балансування навантаження напряму впливає на підвищення ефективності роботи "хмарної" системи, а також сприяє підвищенню її відмовостійкості.

Для забезпечення узгодженої роботи вузлів обчислювальної мережі на стороні хмарного провайдера використовується спеціалізоване проміжне програмне забезпечення, що забезпечує моніторинг стану обладнання і програм, балансування навантаження, забезпечення ресурсів для вирішення завдання.

На даний момент інформація про такі аспекти "хмарних" технологій як масштабування та балансування навантаження розрізнена та вимагає систематизації та впорядкування. В цьому полягає актуальність даної роботи. Також актуальною проблемою є вибір засобу балансування навантаження, який би найкращим чином відповідав задачам створення конкретного сервісу в конкретних умовах. Для порівняння і налаштування систем балансування навантаження згідно поставлених вимог важливо мати достатньо повний набір відповідних критеріїв.

Зв’язок роботи з науковими програмами, планами, темами

Роботу виконано у рамках науково-дослідної роботи кафедри СП – Дослідження методів і технологій цифрової науки на базі об’єднання веб, грід і хмарних сервісів; № договора - СП-2/2014р.; Дата - 22.01.2014

Мета і задачі дослідження

Метою магістерської дисертації є проведення систематизації і 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 в платформі CloudStack та надання відповідних практичних рекомендацій. Для цього визначено завдання, які вирішуються в роботі:

1. Провести систематизацію методів і алгоритмів балансування навантаження в "хмарних" системах.
2. Зробити порівняльну характеристику розповсюджених систем балансування навантаження.
3. Надати практичні рекомендації по налаштуванню системи балансування навантаження платформи CloudStack на прикладі додатку дистанційного відео навчання.
4. Експериментально дослідити ефективність створених правил балансування навантаження для вибраного додатку.

Рішення поставлених завдань та досягнуті результати

В даній роботі було проведено систематизацію і порівняльну характеристику основних методів і алгоритмів балансування навантаження та методів масштабування додатків в "хмарному" середовищі, проведено порівняльну характеристику розповсюджених систем балансування навантаження додатків, досліджено налаштування системи балансування навантаження в платформі CloudStack та надано відповідні практичні

рекомендації.

В ході роботи було виконано встановлення системи дистанційного відео навчання BigBlueButton на віртуальні машини платформи CloudStack, обґрунтовано вибір правил балансування навантаження і автомасштабування для створеної розгалуженої інфраструктури та проведено налаштування цих засобів. Експериментальне дослідження ефективності створених правил балансування, яке було проведено шляхом навантажувального тестування системи, підтверджує правильність вибору правил і налаштування системи загалом.

Об'єкт досліджень

Об'єктом даного дослідження є системи балансування навантаження в "хмарних" технологіях і, зокрема, система балансування навантаження платформи CloudStack.

Предмет досліджень

Предметом досліджень в даній роботі є методи і алгоритми балансування навантаження додатків у "хмарному"середовищі, критерії їх порівняння, узагальнена модель "хмарного" додатку, обґрунтування вибору правил балансування навантаження відповідно до вимог створюваного сервісу.

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 платформи CloudStack на прикладі додатку дистанційного відео навчання, експериментальне дослідження ефективності створених правил балансування навантаження для вибраного додатку.

Наукова новизна

Наукова новизна роботи полягає в систематизації і порівняльній характеристиці основних методів і алгоритмів балансування навантаження та методів масштабування додатків в "хмарному" середовищі, порівняльній характеристиці розповсюджених систем балансування навантаження додатків, та наданні рекомендацій по їх вибору у відповідності до обраних критеріїв.

Практичне значення одержаних результатів

В ході роботи було виконано встановлення системи дистанційного відео навчання BigBlueButton на віртуальні машини платформи CloudStack, обґрунтовано вибір правил балансування навантаження і автомасштабування для створеної розгалуженої інфраструктури та проведено налаштування цих засобів. Експериментальне дослідження ефективності створених правил балансування, яке було проведено шляхом навантажувального тестування системи, підтверджує правильність вибору правил і налаштування системи загалом.

Результати даних досліджень можуть бути використані для вибору методів, алгоритмів і систем балансування навантаження при створенні "хмарних" сервісів, в навчальних дисциплінах з "хмарних обчислень", в подальшій роботі кафедри СП по впровадженню приватного «хмарного» сервісу дистанційного навчання.

Апробація результатів дисертації

Результати досліджень оприлюднені на 17-й Міжнародній науково-технічній конференції SAIT 2015.

Публікації

Бабенко О.А. Балансування навантаження "хмарних" додатків / Бабенко О.А., Гіоргізова-Гай В.Ш. // Системний аналіз та інформаційні технології: матеріали

17-й Міжнародної науково-технічної конференції SAIT 2015, Київ, 22-25 червня 2015 р. – 2015. – с.184

Ключові слова

Хмарні технології, хмарні сервіси, CloudStack, балансування навантаження, масштабування, розподілений веб- додаток.

РЕФЕРАТ

на магистерскую диссертацию

выполненную на тему: Балансировка нагрузки "облачных" приложений

студентом: Бабенко Александром Анатольевичем

Работа выполнена на 151 страницах, содержит 26 иллюстраций, 4 таблицы. При подготовке использовалась литература из 30 разных источников.

Актуальность

В росте популярности и увеличении пользователей сети Интернет в последнее время важную роль играют «облачные технологии» и Web сервисы, которые относятся к высоконагруженных систем. Поэтому возможности масштабирования и балансировки нагрузки при построении "облачных" систем имеют особое значение. Балансировка нагрузки напрямую влияет на повышение эффективности работы "облачной" системы, а также способствует повышению ее отказоустойчивости.

Для обеспечения согласованной работы узлов вычислительной сети на стороне облачного провайдера используется специализированное промежуточное программное обеспечение, обеспечивающее мониторинг состояния оборудования и программ, балансировку нагрузки, обеспечение ресурсов для решения задачи.

На данный момент информация о таких аспектах "облачных технологий" как масштабирование и балансировка нагрузки разрозненная и требует систематизации и упорядочения. В этом заключается актуальность данной работы. Также актуальной проблемой является выбор средства балансировки нагрузки, который бы наилучшим образом отвечал задачам создания конкретного сервиса в конкретных условиях. Для сравнения и настройка систем балансировки нагрузки согласно поставленных требований важно иметь достаточно полный набор соответствующих критериев.

Связь работы с научными программами, планами, темами

Работа выполнена в рамках научно-исследовательской работы кафедры СП - исследование методов и технологий цифровой науки на базе объединения

интернет, грид и облачных сервисов; № договора - СП-2/2014 .; Дата - 22.01.2014

Цель и задачи

Целью магистерской диссертации является проведение систематизации и сравнения основных методов и алгоритмов балансировки нагрузки и методов масштабирования приложений в "облачной" среде, проведение сравнения распространенных систем балансировки нагрузки приложений, исследование настройки системы балансировки нагрузки в платформе CloudStack и предоставление соответствующих практических рекомендаций. Для этого определены задачи, которые решаются в работе:

1. Провести классификацию методов и алгоритмов балансировки нагрузки в "облачных" системах.
2. Сделать сравнительную характеристику распространенных систем балансировки нагрузки.
3. Дать рекомендации по настройке системы балансировки нагрузки платформы CloudStack на примере приложения дистанционного видео обучение.
4. Экспериментально исследовать эффективность созданных правил балансировки нагрузки для выбранного приложения.

Решение поставленных задач и достигнутых результатах

В данной работе было проведено систематизацию и сравнительную характеристику основных методов и алгоритмов балансировки нагрузки и методов масштабирования приложений в "облачном" среде, проведена сравнительная характеристика распространенных систем балансировки нагрузки приложений, исследованы настройки системы балансировки нагрузки в платформе CloudStack и предоставлены соответствующие практические рекомендации.

В ходе работы были выполнена установка системы дистанционного видео обучения BigBlueButton на виртуальные машины платформы CloudStack, обоснован выбор правил балансировки нагрузки и автомасштабирования для созданной разветвленной инфраструктуры и проведение настройки этих средств. Экспериментальное исследование эффективности созданных правил

балансировки, проведенное путем нагрузочного тестирования системы, подтверждает правильность выбора правил и настройки системы в целом.

Объект исследований

Объектом данного исследования являются системы балансировки нагрузки в "облачных" технологиях и, в частности, система балансировки нагрузки платформы CloudStack.

Предмет исследований

Предметом исследований в данной работе являются методы и алгоритмы балансировки нагрузки приложений в "облачном" среде, критерии их сравнения, обобщенная модель "облачного» приложения, обоснование выбора правил балансировки нагрузки в соответствии с требованиями создаваемого сервиса.

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 платформы CloudStack на примере приложения дистанционного видео обучения, экспериментальное исследование эффективности созданных правил балансировки нагрузки для выбранного приложения.

Научная новизна

Научная новизна работы заключается в систематизации и сравнительной характеристике основных методов и алгоритмов балансировки нагрузки и методов масштабирования приложений в "облачном" среде, сравнительной характеристике распространенных систем балансировки

нагрузки приложений, и предоставлении рекомендаций по их выбору в соответствии с выбранными критериями.

Практическая ценность

В ходе работы были выполнены установки системы дистанционного видео обучение BigBlueButton на виртуальные машины платформы CloudStack, обоснован выбор правил балансировки нагрузки и автомасштабирования для созданной разветвленной инфраструктуры и проведение настройки этих средств. Экспериментальное исследование эффективности созданных правил балансировки, проведенное путем нагрузочного тестирования системы, подтверждает правильность выбора правил и настройки системы в целом.

Результаты данных исследований могут быть использованы для выбора методов, алгоритмов и систем балансировки нагрузки при создании "облачных" сервисов, в учебных дисциплинах с "облачных вычислений", в дальнейшей работе кафедры СП по внедрению частного «облачного» сервиса дистанционного обучения.

Апробация результатов диссертации

Результаты исследований опубликованы в 17-й Международной научно-технической конференции SAIT 2015.

Публикации

Бабенко О.А. Балансування навантаження "хмарних" додатків / Бабенко О.А., Гіоргізова-Гай В.Ш. // Системний аналіз та інформаційні технології: матеріали 17-й Міжнародної науково-технічної конференції SAIT 2015, Київ, 22-25 червня 2015 р. – 2015. – с.184

Ключевые слова

Облако , облачные вычисления, IaaS, CloudStack, балансировка нагрузки, масштабирование, веб-приложение.

ABSTRACT

on master's thesis

on topic: Load Balancing of Cloud Application

Student: Oleksandr A. Babenko

Work carried out on 151 pages containing 26 figures, 4 tables. The paper was written with references to 30 different sources.

Topicality

Everything changes, the world is not standing still, and most Internet users are also changing their attitude to the world wide web. The main reason for this - "cloud technology" that contribute to an increasing use of the Internet, storing files on the web.

For the "cloud" technologies are of particular importance scaling and load balancing. Scalability - an important aspect of electronic systems, software systems, database systems, routers, networks, and so on. N., If they need to work under high load. Load balancing directly affects the efficiency of computing, fault tolerance.

To ensure the coordinated work of the computer network nodes on the side of the cloud provider uses specialized middleware software that provides status monitoring and programs, load balancing, providing resources for the task.

Currently information about these aspects "cloud technology" as scaling and load balancing fragmented and requires systematization and ranking. This is the relevance of this work. Also actual problem is the choice of load balancing that meet your goals and criteria it set in accordance with the requirements.

Aims and objectives

The purpose of the master's thesis is to systematize and compare basic methods and algorithms for load balancing and scaling methods applications in the "cloud" environment, comparison of distributed load balancing of applications research system setup load balancing in CloudStack platform and provide appropriate practical recommendations. To do this, set tasks are solved in the work:

1. Hold the classification of methods and algorithms for load balancing in the "cloud" systems.

2. Make a comparative description of distributed load balancing.

3. Provide practical advice on setting system load balancing platform CloudStack an example application of remote video learning.

4. Experimentally investigate the effectiveness of load balancing rules created for the selected application.

The solution of the tasks and results

This paper was conducted systematization and comparative description of the basic methods and algorithms for load balancing and scaling methods applications in the "cloud" environment, conducted comparative description of distributed load balancing applications, system settings investigated load balancing in CloudStack platform and provided such practical recommendations.

The work was completed installation of remote video learning BigBlueButton a virtual machine platform CloudStack, reasonable selection rules and auto-load balancing for an extensive infrastructure and organized setting these facilities. Experimental study of the effectiveness of the rules created balancing, conducted by load testing system confirms the correct choice rules and general system settings.

The object of research

Pursuant to the target object is selected research system load balancing platform CloudStack.

Subject of research

The subject of research in this paper selected methods and algorithms for load balancing in "cloud technologies" generic model "cloud" applications rationale for the choice of load balancing rules.

Research Methods

Holding classification methods and algorithms for load balancing in the "cloud" systems, the development of comparative characteristics common load balancing, provide practical recommendations for setting up the system load balancing platform CloudStack an example application of remote video learning, experimental studies of the effectiveness of load balancing rules created for the selected application.

Scientific novelty

Scientific novelty lies in the systematization and comparative characterization methods and basic load balancing algorithms and methods for scaling applications in the "cloud" environment, comparative characteristics of distributed load balancing applications research system setup load balancing in CloudStack platform and providing appropriate recommendations for balancing the load in " Cloud Technology "according to the selected criteria.

The practical value of research

The work was completed installation of remote video learning BigBlueButton a virtual machine platform CloudStack, reasonable selection rules and auto-load balancing for an extensive infrastructure and organized setting these facilities. Experimental study of the effectiveness of the rules created balancing, conducted by load testing system confirms the correct choice rules and general system settings. The results of these studies can be used for load balancing technology selection according to specific requirements in courses on "cloud computing", further work on the introduction of SD private "cloud" service IaaS for distance learning.

Approbation of results of the dissertation

The research results were published in the 17th International Scientific Conference SAIT 2015.

Publications

Бабенко О.А. Балансування навантаження "хмарних" додатків / Бабенко О.А., Гіоргізова-Гай В.Ш. // Системний аналіз та інформаційні технології: матеріали 17-й Міжнародної науково-технічної конференції SAIT 2015, Київ, 22-25 червня 2015 р. – 2015. – с.184

Keywords

Cloud, cloud computing, IaaS, CloudStack, load balancing, scalability, web- based application.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API	application programming interface, прикладний програмний інтерфейс
IaaS	infrastructure as a service, інфраструктура як послуга
ВМ	віртуальна машина
ІТ	інформаційні технології
ОС	операційна система
ПЗ	програмне забезпечення
OSI	open systems interconnection
NAT	Network Address Translation
IP	Internet Protocol
NAPT	Network Address Port Translation
WAPT	Web Application Performance Testing
SNMP	Simple Network Management Protocol

ВСТУП

Все змінюється, світ не стоїть на місці, і більшість користувачів мережі Інтернет також змінюють своє ставлення до світової павутини. Основна причина тому – «хмарні технології», які сприяють все більшому користуванню Інтернетом, і зберіганню файлів в Мережі. Саме «за хмарою» працюють тепер Facebook, Amazon, Twitter і ті ядра, на яких засновані сервіси типу Google Docs і Gmail.

Термін «хмарні обчислення» (cloud computing) став використовуватися на ринку IT з 2008 року. Розробники «хмарних» обчислень визначають їх як інноваційну технологію, яка надає динамічно масштабовані обчислювальні ресурси і програми через Інтернет в якості сервісу під управлінням постачальника послуг.

Кажучи про «хмарні» технології та їх новизну треба казати скоріше, не про інноваційність «хмарних» технологій, а про перехід кількості в якість, тобто про ефект масштабування. Як приклад цього виду послуг можна навести віртуально виділений сервер – сучасну технологію хостингу, що поєднує в собі потужність виділеного сервера з гнучкістю і простотою управління. Причому в деяких застосуваннях «хмарні» обчислення можуть стати альтернативою суперкомп'ютерів.

Суть концепції «хмарних» обчислень полягає в наданні кінцевим користувачам віддаленого динамічного доступу до послуг, обчислювальних ресурсів і додатків (включаючи операційні системи та інфраструктуру) через Інтернет. Розвиток сфери хостингу був обумовлений тим, що виникла потреба в програмному забезпеченні і цифрових послугах, якими можна було б управляти зсередини, але які були б при цьому більш економічними і ефективними за рахунок економії на масштабі.

Не можна не визнати, що технології «хмарних» обчислень мають величезний потенціал, тому що всі сучасні комп'ютерні продукти постійно

збільшують свої вимоги до технічного оснащення комп'ютера користувача, що неминуче веде до значних витрат на апгрейд. Так що дана технологія дозволяє вирішити проблему надмірної вимогливості додатків до ресурсів кінцевого користувача.

Для "хмарних" технологій особливе значення мають масштабування та балансування навантаження. Комп'ютерні технології щодня пропонують все нові і нові варіанти використання персональних комп'ютерів і серверів. Масштабованість - важливий аспект електронних систем, програмних комплексів, систем баз даних, маршрутизаторів, мереж і т. п., якщо для них потрібна можливість працювати під великим навантаженням. В системі з поганою масштабованістю додавання ресурсів призводить лише до незначного підвищення продуктивності, а з деякого «порогового» моменту додавання ресурсів не дає ніякого корисного ефекту.

Для забезпечення узгодженої роботи вузлів обчислювальної мережі на стороні хмарного провайдера використовується спеціалізоване проміжне програмне забезпечення, що забезпечує моніторинг стану обладнання і програм, балансування навантаження, забезпечення ресурсів для вирішення завдання.

Одним з основних рішень для згладжування нерівномірності навантаження на послуги є розміщення шару серверної віртуалізації між шаром програмних послуг та апаратним забезпеченням. В умовах віртуалізації балансування навантаження може здійснюватися за допомогою програмного розподілу віртуальних серверів по реальним.

На даний момент інформація про такі аспекти "хмарних технологій" як масштабування та балансування навантаження розрізнена та вимагає систематизації та впорядкування. В цьому полягає актуальність даної роботи. Також актуальною проблемою є вибір засобу балансування навантаження, що відповідає поставленим задачам та критеріям, його налаштування у відповідності до поставлених вимог.

Метою магістерської дисертації є проведення систематизації і

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 в платформі CloudStack та надання відповідних практичних рекомендацій.

1 ТЕХНОЛОГІЇ «ХМАРНИХ» ОБЧИСЛЕНЬ

«Хмарні» обчислення (англ. cloud computing), в інформатиці – це модель забезпечення повсюдного і зручного мережевого доступу на вимогу до загального пулу (англ. pool) конфігуруємих обчислювальних ресурсів (наприклад, мереж передачі даних, серверів, пристроїв зберігання даних, додатків і сервісів – як разом, так і окремо), які можуть бути оперативно надані і звільнені з мінімальними експлуатаційними витратами і / або зверненнями до провайдера [1]. Діаграма, що дає уявлення про суть «хмарних» обчислень подана на рис. 1.1.

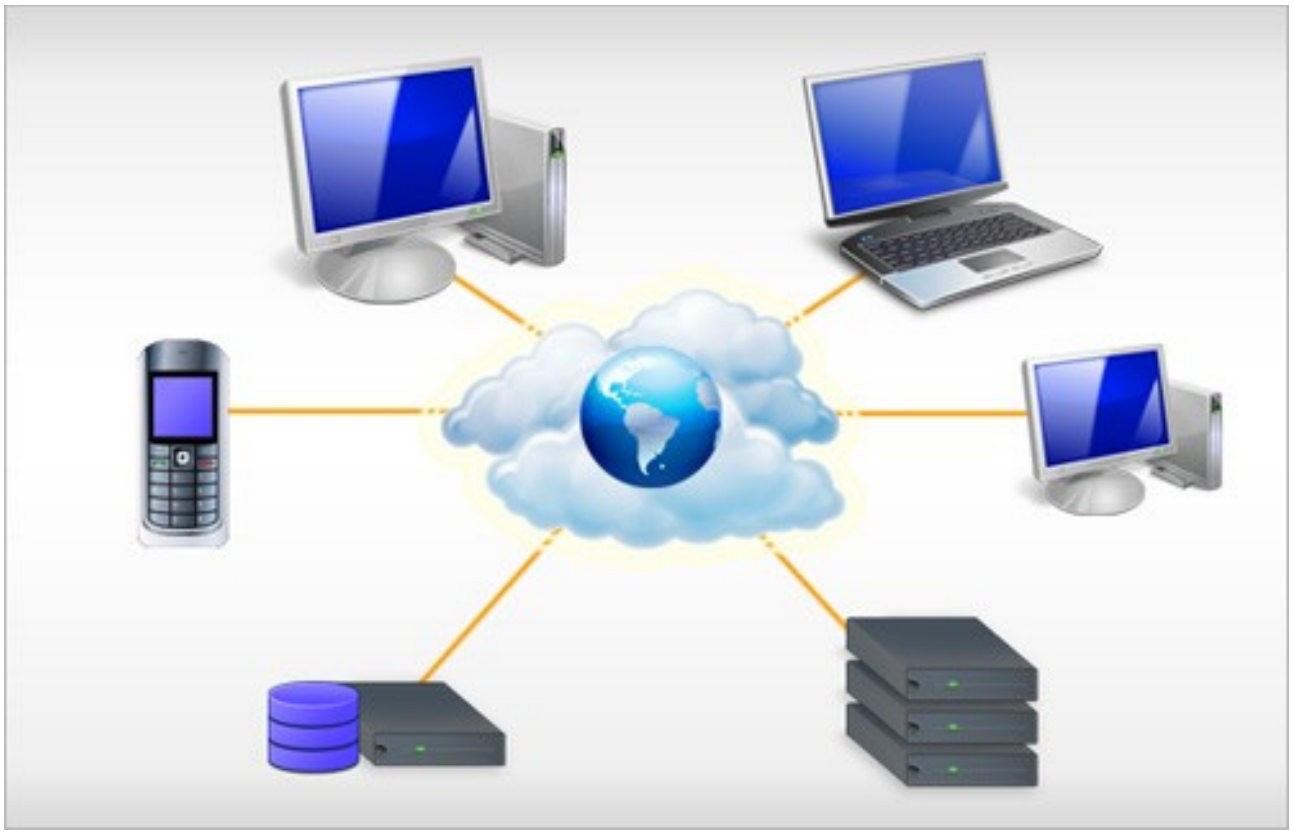

Рисунок 1.1 – Діаграма, що представляє «хмару»

При використанні «хмарних» обчислень програмне забезпечення надається користувачеві як Інтернет-сервіс. Користувач має доступ до власних даних, але не може управляти і не повинен піклуватися про інфраструктуру,

операційну систему і програмне забезпечення, з яким він працює. «Хмарою» метафорично називають Інтернет, який приховує всі технічні деталі. Згідно з документом IEEE (Institute of Electrical and Electronics Engineers), опублікованим у 2008 році, «Хмарні обчислення — це парадигма, в рамках якої інформація постійно зберігається на серверах у мережі Інтернет і тимчасово кешується на клієнтській стороні, наприклад на персональних комп'ютерах, ігрових приставках, ноутбуках, смартфонах тощо» [2].

В основі Cloud computing лежать кілька підходів. Перший - доступність сервісів через Інтернет. Цей підхід не відноситься до закритих інфраструктур, в них мережу Інтернет замінюється локальними мережами, але частина сервісів, як правило, все одно доступні з глобальної мережі.

Другий підхід - віртуалізація. Віртуалізація дає легкість масштабування. Завдяки віртуалізації кожен користувач може отримати необхідну потужність з можливістю її майбутнього розширення або звуження. Усі службові процеси, при цьому, прозорі для користувача. Фізичні ресурси для роботи необхідної системи можуть виділяться на різних серверах знаходяться в різних дата центрах.

Третій підхід - Cloud Computing це послуга. Раніше за машинний час доводилося платити і чекати в черзі вільного години. У Cloud computing використовується схожий, але більш сучасний підхід. Використані ресурси для користувача це набір послуг, які споживаються і в разі надання комерційним постачальником оплачуються. Для прикладу, розглянемо хостінг для даних з доступом до них через HTTP REST API. У користувача є потрібний обсяг даних, який доступний за допомогою зручного інтерфейсу. При цьому дані зберігаються на фізичних серверах, захищаються від втрати за допомогою Raid масивів і територіально розподілені.

Четвертий підхід - простота і стандартність. Дуже важлива властивість для нової, ще не повністю адаптованої технології. Все, що пропонується всередині хмари доступно через прості виклики API і протоколи. Величезну популярність знайшов протокол REST, за допомогою якого всі операції над

даними можна робити через запити. Застосовуються й багато інших рішень, для різних мов програмування вже доступні бібліотеки для написання подібних систем роботи з даними.

1.1 Принцип роботи «хмар»

Щоб зрозуміти принцип роботи «хмарних технологій», потрібно уявити, що «пошарована» «хмара» складається, в основному, з «внутрішніх» (back-end) і «користувацьких» (front-end) шарів. Шари front-end - це те, що бачить, те, з чим має справу користувач. Наприклад, користуючись сервісом Gmail, маємо справу з ПЗ, працюючим на front-end шарі «хмари». Те ж саме відбувається і коли заходимо під своїм обліковим записом у Facebook . Внутрішні ж шари складаються з власне обладнання та програмних сервісів, що забезпечують роботу того інтерфейсу, що видається на «користувацькому» шарі [1].

Оскільки всі комп'ютери «хмари» налаштовані так, щоб працювати спільно, додаткам доступна вся сумарна потужність цих комп'ютерів, начебто це додаток виконувався на конкретному окремо взятому комп'ютері. «Хмарні» технології мають, крім того, і чималу ступінь гнучкості в роботі. Залежно від потреб, можна підняти чи опустити ту планку використовуваних хмарою ресурсів, за якої буде потрібно включення в роботу додаткового обладнання. А можна і зменшити число ресурсів, призначених на роботу, коли вона не є пріоритетною.

1.2 Основні характеристики «хмар»

Провайдери «хмарних» рішень дозволяють орендувати через Інтернет обчислювальні потужності та дисковий простір. Переваги такого підходу — доступність (користувач платить лише за ті ресурси, які йому потрібні) і можливість гнучкого масштабування. Клієнти позбавляються від необхідності створювати і підтримувати власну обчислювальну інфраструктуру.

За оцінками експертів, використання «хмарних» технологій в багатьох

випадках дозволяє скоротити витрати в два-три рази в порівнянні з утриманням власної розвиненої IT-структури.

"Хмара" відкриває новий підхід до обчислень, при якому ані обладнання, ані програмне забезпечення не належать підприємству. Замість цього провайдер надає замовнику вже готовий сервіс.

До допомоги "хмар" часто вдаються молоді компанії-стартапи, які потребують великих обчислювальних ресурсів для обслуговування користувачів, але не можуть дозволити собі створення і експлуатацію власного дата-центру.

Одним з перших широкодоступних «хмарних» Інтернет-сервісів стала електронна пошта з веб-інтерфейсом. У цьому випадку всі дані зберігаються на віддалених серверах, а користувач отримує доступ до своїх листів через браузер з будь-якого комп'ютера або достатньо потужного мобільного пристрою.

Національним інститутом стандартів і технологій США встановлені такі обов'язкові характеристики «хмарних» обчислень [1]:

- Самообслуговування на вимогу (англ. self service on demand), споживач самостійно визначає і змінює обчислювальні потреби, такі як серверний час, швидкості доступу та обробки даних, обсяг збережених даних без взаємодії з представником постачальника послуг;
- Універсальний доступ по мережі, послуги доступні споживачам через мережу передачі даних незалежно від термінального пристрою;
- Об'єднання ресурсів (англ. resource pooling), постачальник послуг об'єднує ресурси для обслуговування великої кількості споживачів в єдиний пул для динамічного перерозподілу потужностей між споживачами в умовах постійної зміни попиту на потужності; при цьому споживачі контролюють тільки основні параметри послуги (наприклад, обсяг даних, швидкість доступу), але фактичний розподіл ресурсів, що надаються споживачеві, здійснює постачальник (в деяких випадках споживачі все ж можуть керувати

деякими фізичними параметрами перерозподілу, наприклад, вказувати бажаний центр обробки даних з міркувань географічної близькості);

- Еластичність, послуги можуть бути надані, розширені, звужені в будь-який момент часу, без додаткових витрат на взаємодію з постачальником, як правило, в автоматичному режимі;
- Облік споживання, постачальник послуг автоматично обчислює спожиті ресурси на певному рівні абстракції (наприклад, обсяг збережених даних, пропускна здатність, кількість користувачів, кількість транзакцій), і на основі цих даних оцінює обсяг наданих споживачам послуг.

З точки зору постачальника, завдяки об'єднанню ресурсів та непостійному характеру споживання з боку споживачів, «хмарні» обчислення дозволяють економити на масштабах, використовуючи менші апаратні ресурси, ніж при виділенні апаратних потужностей для кожного споживача, а за рахунок автоматизації процедур модифікації виділення ресурсів істотно знижуються витрати на абонентське обслуговування.

З точки зору споживача, ці характеристики дозволяють отримати послуги з високим рівнем доступності (англ. high availability) і низькими ризиками непрацездатності, забезпечити швидке масштабування обчислювальної системи завдяки еластичності без необхідності створення, обслуговування і модернізації власної апаратної інфраструктури.

Зручність і універсальність доступу забезпечується широкою доступністю послуг і підтримкою різного класу термінальних пристроїв (персональних комп'ютерів, мобільних телефонів, Інтернет-планшетів).

1.3 Моделі обслуговування та існуючі рішення

Виділяють наступні моделі надання послуг за допомогою хмари.

Програмне забезпечення як послуга (SaaS) – модель, в якій споживачеві надається можливість використання прикладного програмного забезпечення провайдера, що працює в хмарної інфраструктурі і доступного з різних клієнтських пристроїв або за допомогою тонкого клієнта, наприклад, з браузера (наприклад, веб-пошта) або інтерфейс програми. Контроль і управління основним фізичної і віртуальної інфраструктурою хмари, в тому числі мережі, серверів, операційних систем, зберігання, або навіть індивідуальних можливостей додатки (за винятком обмеженого набору налаштувань користувача конфігурації програми) здійснюється хмарним провайдером. Прикладами програмного забезпечення як послуги, що працює на основі обчислювальної хмари, є сервіси Gmail та Google docs.

У рамках моделі SaaS замовники платять не за володіння програмним забезпеченням як таким, а за його оренду (тобто за його використання через веб-інтерфейс). Таким чином, на відміну від класичної схеми ліцензування ПЗ, замовник несе порівняно невеликі періодичні витрати, і йому не потрібно інвестувати значні кошти в придбання ПЗ та апаратної платформи для його розгортання, а потім підтримувати його працездатність. Схема періодичної оплати передбачає, що якщо необхідність в програмному забезпеченні тимчасово відсутня, то замовник може призупинити його використання і заморозити виплати розробнику.

Програмне забезпечення на вимогу володіє наступними ключовими ознаками:

- Доступ до програмного забезпечення, розробленого відповідно до моделі ПЗ як послуга, надається віддалено по мережевих каналах і як правило через веб-інтерфейс, крім того, можуть використовуватися тонкі клієнти і термінальний доступ;
- Програмне забезпечення розгортається в центрі обробки даних у вигляді єдиного програмного ядра, з яким працюють всі замовники;
- Програмне забезпечення надається на умовах сплати періодичних орендних платежів;

- Обслуговування та оновлення програмного забезпечення виконується централізовано на стороні постачальника програми, що надається як послуга (SaaS);
- Вартість технічної підтримки зазвичай включається в орендну плату.

Платформа як послуга (PaaS) – модель, в якій споживачеві надається можливість використання хмарної інфраструктури для розміщення базового програмного забезпечення для подальшого розміщення на ньому нових або існуючих додатків (власних, розроблених на замовлення або придбаних тиражованих додатків). До складу таких платформ входять інструментальні засоби створення, тестування та виконання прикладного програмного забезпечення – системи управління базами даних, зв'язуюче програмне забезпечення, середовища виконання мов програмування – надані хмарним провайдером. Контроль і управління основним фізичної і віртуальної інфраструктурою хмари, в тому числі мережі, серверів, операційних систем, зберігання здійснюється хмарним провайдером, за винятком розроблених або встановлених додатків, а також, по можливості, параметрів конфігурації середовища (платформи). Наприклад, Google Apps надає застосунки для бізнесу в режимі онлайн, доступ до яких відбувається за допомогою Інтернет-браузера тоді як ПЗ і дані зберігаються на серверах Google.

Інфраструктура як послуга (IaaS) надається як можливість використання хмарної інфраструктури для самостійного управління ресурсами обробки, зберігання, мереж та іншими фундаментальними обчислювальними ресурсами, наприклад, споживач може встановлювати і запускати довільне програмне забезпечення, яке може включати в себе операційні системи, платформенне і прикладне програмне забезпечення. Споживач може контролювати операційні системи, віртуальні системи зберігання даних і встановлені програми, а також обмежений контроль набору доступних сервісів (наприклад, міжмережевий екран). Контроль і управління основним фізичної і віртуальної інфраструктурою хмари, в тому числі мережі, серверів, типів використовуваних операційних систем, систем зберігання здійснюється хмарним провайдером.

IaaS складається з трьох основних компонентів: апаратні засоби (сервери, системи зберігання даних, клієнтські системи, мережеве обладнання); операційні системи та системне ПЗ (засоби віртуалізації, автоматизації, основні засоби управління ресурсами); зв'язуюче ПЗ (наприклад, для управління системами).

Технології віртуалізації дозволяють взяти обладнання і розділити його обчислювальні потужності на частини, які відповідають поточним потребам бізнесу, тим самим збільшуючи утилізацію наявних потужностей. В результаті від придбання, управління і амортизації апаратних активів можна перейти до покупки процесорного часу, дискового простору, пропускної здатності мережі, яка необхідна для виконання програми.

У минулому для управління різними типами устаткування було потрібне різне ПЗ управління. Віртуалізація дозволяє реалізувати весь набір функцій управління в одній інтегрованій платформі.

Якщо раніше кожній компанії для реалізації необхідної інфраструктури доводилося "винаходити велосипед" – то зараз маємо готові інфраструктури, реалізовані з урахуванням накопиченого досвіду і знань.

Infrastructure as a Service (IaaS) позбавляє підприємства від необхідності підтримки складних інфраструктур центрів обробки даних, клієнтських і мережевих інфраструктур, а також дозволяє зменшити пов'язані з цим капітальні витрати та поточні витрати. Можлива й додаткова економія, якщо послуги надаються в рамках інфраструктури спільного використання.

Найбільшими гравцями на ринку інфраструктури як послуги є Amazon, Microsoft, VMWare, Rackspace та Red Hat. Хоча деякі з них пропонують більше, ніж просто інфраструктуру, їх об’єднує мета продавати базові обчислювальні ресурси.

Загальною характеристикою компаній, що будують свої продукти на основі хмар, є впевненість у тому, що мережа Інтернет в змозі задовольнити потреби користувачів в обробці даних.

Обчислювальна «хмара» може бути розгорнута як: приватна, публічна,

громадська або гібридна.

Приватна «хмара» (англ. private cloud) - це «хмарна» інфраструктура, яка призначена для використання виключно однією організацією, що включає декілька користувачів (наприклад, підрозділів). Приватна «хмара» може перебувати у власності, керуванні та експлуатації як самої організації, так і третьої сторони (чи деякої їх комбінації). Така «хмара» може фізично знаходитись як в, так і поза юрисдикцією власника.

Публічна «хмара» (англ. public cloud) - це «хмарна» інфраструктура, яка призначена для вільного використання широким загалом. Публічна «хмара» може перебувати у власності, керуванні та експлуатації комерційних, академічних (освітніх та наукових) або державних організацій (чи будь-якої їх комбінації). Публічна «хмара» перебуває в юрисдикції постачальника «хмарних» послуг.

Громадська «хмара» (англ. community cloud) - це «хмарна» інфраструктура, яка призначена для використання конкретною спільнотою споживачів із організацій, що мають спільні цілі (наприклад, місію, вимоги щодо безпеки, політику та відповідність різноманітним вимогам). Громадська «хмара» може перебувати у спільній власності, керуванні та експлуатації однієї чи більше організацій зі спільноти або третьої сторони (чи деякої їх комбінації). Така «хмара» може фізично знаходитись як в, так і поза юрисдикцією власника.

Гібридна «хмара» (англ. hybrid cloud) - це «хмарна» інфраструктура, що складається з двох або більше різних «хмарних» інфраструктур (приватних, громадських або публічних), які залишаються унікальними сутностями, але з’єднанні між собою стандартизованими або приватними технологіями, що уможливлюють переносимість даних та прикладних програм (наприклад, використання ресурсів публічної хмари для балансування навантаження між хмарами).

1.4 Переваги та недоліки «хмарних» обчислень

Технології «хмарних» обчислень мають свої переваги та недоліки.

Спочатку розглянемо переваги «хмарних» обчислень.

По-перше до них можна віднести доступність. Хмари доступні всім, з будь-якої точки, де є Інтернет, з будь-якого комп'ютера, де є браузер. Це дозволяє користувачам (підприємствам) заощаджувати на закупівлі високопродуктивних, дорогих комп'ютерів. Також співробітники компаній стають більш мобільними так, як можуть отримати доступ до свого робочого місця з будь-якої точки земної кулі, використовуючи ноутбук, нетбук, планшет або смартфон. Немає необхідності в купівлі ліцензійного ПЗ, його налаштуванні і відновленні, ви просто заходите на сервіс і користуєтеся його послугами заплативши за фактичне використання.

Наступна перевага – це низька вартість. Основні фактории, що знизили вартість використання хмар, наступні:

- Зниження витрат на обслуговування віртуальної інфраструктури, викликане розвитком технологій віртуалізації, за рахунок чого потрібно менший штат для обслуговування всієї IT інфраструктури підприємства;
- Оплата фактичного використання ресурсів. Користувач хмари платить за фактичне використання обчислювальних потужностей хмари, що дозволяє йому ефективно розподіляти свої грошові ресурси. Це дозволяє користувачам (підприємствам) заощаджувати на покупці ліцензій до ПЗ;
- Використання хмари на правах оренди дозволяє користувачам знизити витрати на закупівлю дорогого устаткування, і зробити акцент на вкладення грошових коштів на налагодження бізнес процесів підприємства, що в свою чергу дозволяє легко почати бізнес;
- Розвиток апаратної частини обчислювальних систем, у зв'язку з чим знижується вартість обладнання.

Також однією з переваг є гнучкість - необмеженість обчислювальних ресурсів (пам'ять, процесор, диски). За рахунок використання систем

віртуалізації, процес масштабування і адміністрування «хмар» стає досить легким завданням, оскільки «хмара» самостійно може надати ресурси, які необхідні, а плата відбувається тільки за фактичне їх використання.

Ще одна з переваг – надійність. Надійність «хмар», особливо тих, що перебувають у спеціально обладнаних ЦОД, дуже висока так, як такі ЦОД мають резервні джерела живлення, охорону, професійних працівників, регулярне резервування даних, високу пропускну здатність Інтернет каналу, високу стійкість до DoS (Denial of Service) атак.

Важливою перевагою є безпека. «Хмарні» сервіси мають досить високу безпеку при належному її забезпеченні, однак при недбалому ставленні ефект може бути повністю протилежним.

Найважливішою перевагою є великі обчислювальні потужності. Користувач «хмарної» системи може використовувати всі її обчислювальні здібності, заплативши тільки за фактичний час використання. Підприємства можуть використовувати дану можливість для аналізу великих обсягів даних.

Незважаючи на всі перераховані вище переваги, «хмари» мають і свої недоліки.

Перший з них – це необхідність постійного з'єднання з мережею. Для отримання доступу до послуг «хмари» необхідно постійне з'єднання з мережею Інтернет. Але у наш час це не такий і великий недолік особливо з приходом технологій стільникового зв'язку 3G і 4G.

Наступним недоліком є програмне забезпечення і його кастомізація. Є обмеження на ПЗ яке можна розгортати на «хмарах» і надавати користувачеві. Користувач ПЗ має обмеження у використовуваному ПЗ і іноді не має можливості налаштувати його під свої власні цілі.

Конфіденційність даних збережених на публічних «хмарах» в даний час викликає багато суперечок, але в більшості випадків експерти сходяться в тому, що не рекомендується зберігати найбільш цінні для компанії документи на публічній "хмарі", так як в даний час немає технології яка б гарантувала 100% конфіденційність збережених даних.

Що стосується надійності інформації, що зберігається, то з упевненістю можна сказати що якщо було втрачено інформацію збережену в "хмарі", то її втрачено назавжди. Це безперечно недолік.

Безпека є також недоліком. "Хмара" сама по собі є достатньо надійною системою, однак при проникненні на неї зловмисник отримує доступ до величезного сховища даних. Ще один мінус це використання систем віртуалізації, в яких як гіпервізор використовуються ядра стандартних ОС таких, як Linux, Windows та ін, що дозволяє використовувати віруси.

Якщо розглядати економічні показники використання «хмарних» технологій, то можна виділити ще один недолік - дороговизна обладнання. Для побудови власної хмари компанії необхідно виділити значні матеріальні ресурси, що не вигідно щойно створеним і малим компаніям.

Чи створюють «хмари» проблеми? Звичайно ж, створюють. Якщо з тієї чи іншої причини втрачено доступ до Інтернету, також буде втрачено і доступ до всіх документів. Не варто забувати і про питання безпеки, і ризик того, що компанія - господар «хмари» зажадає від вас щомісячної плати за використання пропрієтарного формату даних. Варто прострочити платіж - і всі дані буде втрачено безповоротно.

Чим «хмарні» технології можуть допомогти? По-перше, користувачеві не потрібно піклуватися про продуктивність свого ПК, не потрібно переживати про вільне місце на дисковому просторі. З хмарними технологіями це питання автоматично знімається відразу ж, на перших етапах.

По-друге, користувачеві не потрібно витрачатися повністю на весь потрібний йому продукт. Він платить тільки за послугу, надані можливості і тільки за конкретні функції.

По-третє, «хмарні» технології допомагають бізнесу в сезонному сенсі. Наприклад, якщо компанія продає подарунки до міжнародного жіночого дня, який як всі знають тільки раз на рік, у решту часу (взимку, влітку, восени) її послуги нікому будуть не потрібні. І саме в цей час, компанія не буде нести витрати на обслуговування того величезного функціоналу, який потрібен їй

тільки в піковий період.

1.5 Висновки

Отже, «хмарні обчислення», в інформатиці - це модель забезпечення повсюдного і зручного мережевого доступу на вимогу до загального пулу конфігуруємих обчислювальних ресурсів, які можуть бути оперативно надані і звільнені з мінімальними експлуатаційними витратами і / або зверненнями до провайдера.

«Хмари» мають свої переваги і недоліки. До переваг можна віднести:

- доступність з будь-якої місця;
- економія коштів користувача на власній інфраструктурі;
- доступність для користувача великих обчислювальних потужностей;
- легкість процесу масштабування і адміністрування «хмари» ;
- надійність апаратного забезпечення «хмари»;

Серед недоліків хмарних обчислень:

- необхідність постійного з'єднання з мережею;
- ненадійність зберігання даних користувача;
- проблеми інформаційної безпеки;
- висока вартість побудови «хмари».

2 МАСШТАБУВАННЯ В «ХМАРНИХ» ОБЧИСЛЕННЯХ

Основним критерієм при поліпшенні продуктивності є можливість подальшого масштабування. Функції продуктивності оцінюються з точки зору їх ефективності для різних обсягів навантаження - від рівня простого веб-сервера до великої серверної ферми.

Масштабування дозволяє швидко адаптуватися в залежності від ступеня навантаження і коригувати інфраструктуру зростаючої організації. Ключове

місце в концепції «хмарних» обчислень займає масштабування, оскільки тільки завдяки йому можна побудувати по-справжньому ефективну "хмару".

Масштабованість (англ. Scalability) - в електроніці та інформатиці означає здатність системи, мережі або процесу справлятися зі збільшенням робочого навантаження (збільшувати свою продуктивність) при додаванні ресурсів (зазвичай апаратних) [4]. Масштабованість - важливий аспект електронних систем, програмних комплексів, систем баз даних, маршрутизаторів, мереж і т. п., якщо для них потрібна можливість працювати під великим навантаженням. Система називається масштабованою, якщо вона здатна збільшувати продуктивність пропорційно додатковим ресурсам. Масштабованість можна оцінити через відношення приросту продуктивності системи до приросту використовуваних ресурсів. Чим ближче це відношення до одиниці, тим краще. Також під масштабованістю розуміється можливість нарощування додаткових ресурсів без структурних змін центрального вузла системи.

В системі з поганою масштабованістю додавання ресурсів призводить лише до незначного підвищення продуктивності, а з деякого «порогового» моменту додавання ресурсів не дає ніякого корисного ефекту.

В галузі телекомунікацій і програмного забезпечення, масштабованість є бажаною властивістю системи, мережі, або процесу, яка свідчить про здатність системи обробити більший обсяг роботи або бути легко розширеною. Наприклад, масштабованість може позначати здатність системи до збільшення загальної пропускної спроможності відповідно до підвищеного навантаження, коли додано (здебільшого, апаратні) ресурси. Цей термін має аналогічне значення, коли його вживають в галузі комерції, наприклад, масштабованість компанії припускає, що основна бізнес-модель надає можливості для економічного зростання всередині компанії.

Масштабованість, як властивість системи, як правило, важко визначити, і в кожному конкретному випадку потрібно визначити конкретні вимоги до параметрів, які вважаються важливими. Це є дуже важливим питанням у галузі

електронних систем, баз даних, маршрутизаторів і мереж. Систему, що підвищує продуктивність роботи після додавання апаратних засобів пропорційно доданим ресурсам, називають масштабовною. Алгоритм, архітектура, мережевий протокол, програма або інша система називається масштабовними, якщо вони ефективні в застосуванні до великих задач (наприклад, великий набір вхідних даних або велика кількість вузлів у випадку розподіленої системи).

Існують два види масштабування[4]:

- Горизонтальне масштабування;
- Вертикальне масштабування.

Детальніше розглянемо їх далі.

2.1 Горизонтальне масштабування

Горизонтальне масштабування - розбиття системи на більш дрібні структурні компоненти та рознесення їх по окремим фізичним машинам (або їх групам), та (або) збільшення кількості серверів, які паралельно виконують одну й ту ж функцію. Масштабованість в цьому контексті означає можливість додавати до системи нові вузли, сервери, процесори для збільшення загальної продуктивності. Цей спосіб масштабування може вимагати внесення змін до програми, щоб програми могли повною мірою користуватися збільшеною кількістю ресурсів [5].

Горизонтальне масштабування, припускає розширення обчислювальних ресурсів доступних додатку за рахунок збільшення кількості серверів або інстансів додатку, в разі PaaS, на яких розміщено вашу програму. Тобто якщо раніше ваш додаток був розташований на одному сервері, і в якийсь момент він перестав «витягувати» навантаження, можна просто купити другий точно такий же сервер. Поставити на нього ваш додаток і таким чином частина запитів до додатка буде йти на перший сервер, частина - на другий.

Цей принцип і покладено в горизонтальне масштабування додатків

розміщених в «хмарі», тільки замість реальних фізичних серверів і у нас є поняття віртуальна машина. Коли примірника однієї віртуальної машини недостатньо вашого додатком - ви можете збільшити його, таким чином розподіливши навантаження між декількома віртуальними машинами.

Горизонтальне масштабування - створення кластера з пов'язаних між собою (часто вже не дуже потужних) серверів, які разом обслуговують систему. У цьому випадку, використовується балансувальник навантаження (load balancer) - машина або програма, основна функція якої - визначити на який сервер послати запит. Сервера в кластері ділять між собою обслуговування додатки, нічого один про одного не знаючи, таким чином значно збільшуючи пропускну здатність і відмовостійкість системи.

Горизонтальне масштабування значно спрощує управління зростаючої IT-інфраструктурою. Так, наприклад, організації середнього розміру спочатку може бути достатньо двох серверів. З ростом бізнесу, обсягів продажів і збільшенням штату вимоги будуть ускладнюватися, але горизонтальне масштабування дозволить нарощувати інфраструктуру за допомогою додавання комп'ютерів з аналогічною конфігурацією.

Крім збільшення потужності, горизонтальне масштабування додає надійності системі - при виході з ладу одного з серверів, навантаження буде збалансованою між працюючими і додаток буде працездатний.

Таким чином, горизонтальне масштабування додає додатковий рівень абстракції, значно полегшуючи управління обладнанням та нарощування інфраструктури.

2.2 Вертикальне масштабування

Вертикальне масштабування - збільшення продуктивності кожного компонента системи з метою підвищення загальної продуктивності. Масштабованість в цьому контексті означає можливість замінювати в існуючій обчислювальній системі компоненти більш потужними і швидкими в міру зростання вимог і розвитку технологій. Вертикальне масштабування передбачає

збільшення продуктивності додатка при додаванні ресурсів (процесора, пам'яті, диска) в рамках одного вузла (хоста). Це найпростіший спосіб масштабування, так як не вимагає ніяких змін в прикладних програмах, що працюють на таких системах [6].

Зрозуміло, що найпростішим способом буде просте оновлення заліза (процесора, пам'яті, диска) - тобто вертикальне масштабування. Крім того, цей підхід не вимагає ніяких доопрацювань програми. Однак, вертикальне масштабування дуже швидко досягає своєї межі, після чого розробнику та адміністратору нічого не залишається окрім як перейти до горизонтального масштабування додатку.

Всі віртуалізовані середовища - і, зокрема, хмарні середовища, дуже добре масштабуються горизонтально. Але коли мова заходить про вертикальне масштабування, то тут хмарні середовища мають певні сильні сторони і деякі значні недоліки.

Сильною стороною хмарної інфраструктури щодо вертикального масштабування є простота, з якою ви можете протестувати менш потужні і менш спеціалізовані конфігурації і оцінити потреби системи в більш потужних або більше спеціалізованих конфігураціях. Хмарні середовища (а до хмарної середовищі Amazon це відноситься навіть більшою мірою, ніж до її конкурентам) набагато гірше показали себе при роботі зі спеціалізованими системними конфігураціями.

Вертикальне динамічне масштабування складніше, ніж горизонтальне. Якщо бути більш точним, то вертикальне масштабування являє собою просто окремий випадок горизонтального. Воно вимагає наступної комбінації дій:

1. Додавання в хмарне середовище більш потужної системи, використовуючи точно такий же підхід, як при горизонтальному масштабуванні. Єдиною відмінністю тут буде те, що ви будете використовувати більш потужні екземпляри машин замість дублювання існуючих екземплярів.

2. Потім слід дочекатися, коли новий екземпляр почне відповідати на

запити.

3. Після цього з системи можна буде видалити один або більшу кількість малопотужних примірників.

При збільшенні навантаження або відвідуваності проекту, рано чи пізно вертикальне масштабування (збільшення ресурсів сервера, таких як пам'ять, швидкість диска і т.д) впирається в якісь межі і не дає відчутного приросту. У такому випадку використовується горизонтальне масштабування - додавання нових серверів с перерозподілом навантаження між ними.

Якщо ви будете комбінувати вертикальний і горизонтальний підходи до масштабування, ви зможете отримати інфраструктуру, найбільш ефективним чином споживаючу обчислювальні ресурси.

2.2 Висновки

Коли виникає питання підвищення продуктивності програми, то є кілька варіантів. Як відомо можна купити нове «залізо» для сервера, додати кількість оперативної пам'яті і т.д. Цей принцип називається вертикальним масштабуванням. Однак цей спосіб може бути досить дорогим, довгим, та й має межу. Можна, звичайно, купити потужне "залізо", однак його може не вистачити для задоволення всіх вимог застосування.

При збільшенні навантаження або відвідуваності проекту, рано чи пізно вертикальне масштабування (збільшення ресурсів сервера, таких як пам'ять, швидкість диска і т.д) впирається в якісь межі і не дає відчутного приросту. У такому випадку використовується горизонтальне масштабування - додавання нових серверів с перерозподілом навантаження між ними.

Другий спосіб, названий горизонтальним масштабуванням, припускає розширення обчислювальних ресурсів доступних додаткам за рахунок збільшення кількості серверів або екземплярів додатків, в разі PaaS, на яких розміщено програму. Тобто якщо раніше додаток був розташований на одному сервері, і в якийсь момент його ресурсів перестало вистачати для обробки навантаження, можна просто купити другий точно такий же сервер. Поставити

на нього додаток і таким чином частина запитів до додатка буде йти на перший віртуальний сервер, частина - на другий.

3 МЕТОДИ І АЛГОРИТМИ БАЛАНСУВАННЯ НАВАНТАЖЕННЯ В «ХМАРНИХ» СИСТЕМАХ

3.1 Балансування навантаження

У термінології комп'ютерних мереж, балансування навантаження, або вирівнювання навантаження (англ. Load balancing) - метод розподілу завдань між декількома мережевими пристроями (наприклад, серверами) з метою оптимізації використання ресурсів, скорочення часу обслуговування запитів, горизонтального масштабування кластера (динамічне додавання / видалення пристроїв), а також забезпечення відмовостійкості (резервування)[7].

Балансування навантаження може бути використане для розширення можливостей ферми серверів, що складається більш ніж з одного сервера. Воно також може дозволити продовжувати роботу навіть в умовах, коли кілька виконавчих пристроїв (серверів) вийшли з ладу. Завдяки цьому зростає відмовостійкість, і з'являється можливість динамічно регулювати використовувані обчислювальні ресурси, за рахунок додавання / видалення виконавчих пристроїв в кластері.

Балансування навантаження передбачає рівномірне навантаження обчислювальних вузлів. При появі нових завдань програмне забезпечення, що реалізує балансування, має прийняти рішення про те, де (на якому обчислювальному вузлі) слід виконувати обчислення, пов'язані з цим новим завданням. Крім того, балансування передбачає перенос (migration - міграція) частини обчислень з найбільш завантажених обчислювальних вузлів на менш завантажені вузли.

Проблема балансування обчислювального навантаження виникає з тієї причини, що:

- структура розподіленого додатка неоднорідна, різні логічні процеси вимагають різних обчислювальних потужностей;

- структура обчислювального комплексу (наприклад, кластера), також неоднорідна, тобто різні обчислювальні вузли володіють різною продуктивністю;
- структура міжвузлової взаємодії неоднорідна, тому лінії зв'язку, що з'єднують вузли, можуть мати різні характеристики пропускної здатності.

3.2 Види балансування навантаження

Слід розрізняти статичне і динамічне балансування.

Статичне балансування виконується до початку виконання розподіленого додатка. Дуже часто при розподілі логічних процесів по процесорах використовується досвід попередніх виконань, застосовуються генетичні алгоритми[8]. Однак попереднє розміщення логічних процесів по процесорах (комп'ютерів) не дає ефекту.

Це пояснюється тим, що:

- Може змінитися обчислювальна середовище, в якому відбувається виконання програми, який або обчислювальний вузол може вийти з ладу.
- Обчислювальний вузол, на якому виконується розподілене додаток, зайнятий ще й іншими обчисленнями, частка яких з часом може зрости.

Так чи інакше, виграш від розподілу логічних процесів з комп'ютерів з метою виконання паралельної обробки стає неефективним.

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

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

розподіленого додатка та ін.

На підставі зібраних даних (як по розподіленому додатку, так і по обчислювальному середовищу) приймається рішення про перенесення логічних процесів з одного вузла на інший.

Задачу оптимального розподілу обчислювального навантаження на концептуальному рівні можна порівняти із завданням наповнення поїлок - кількох бочок з різними геометричними характеристиками водою, яка надходить через трубу з вентилями. Один раз в секунду дно бочок відкривається, і вся вода з них потрапляє в поїлки. Вода ж, яка переливається через краї бочок, йде в землю безцільно. Для виключення переливу необхідно вибрати такі регулюють значення вентилів, щоб вода в бочках трималася на однаковому рівні. При збільшенні потоку води рівень води у всіх бочках на момент скидання повинен збільшитися на однакову величину, щоб не допустити переливу води в жодній з бочок (необроблені запити). Саме це дозволить досягти максимального обсягу корисно спожитої води (максимальне значення числа оброблених операцій в багатовузлових системах). Даний процес можна представити на графіку залежності числа оброблених запитів від кількості надійшовших до системи, що складається з двох вузлів А і Б. При оптимальному розподілі навантаження - 1/3 запитів на А і 2/3 запитів на Б (рис 3.1, а) - величина граничного сумарного числа оброблених запитів, при якому не відбувається скидання, удвічі вище, ніж для варіанту, представленого на рис 3.1, б (1/3 запитів припадає на Б і 2/3 запитів на А). Причина - швидке досягнення межі можливостей вузла А і деградація продуктивності всієї системи [9].

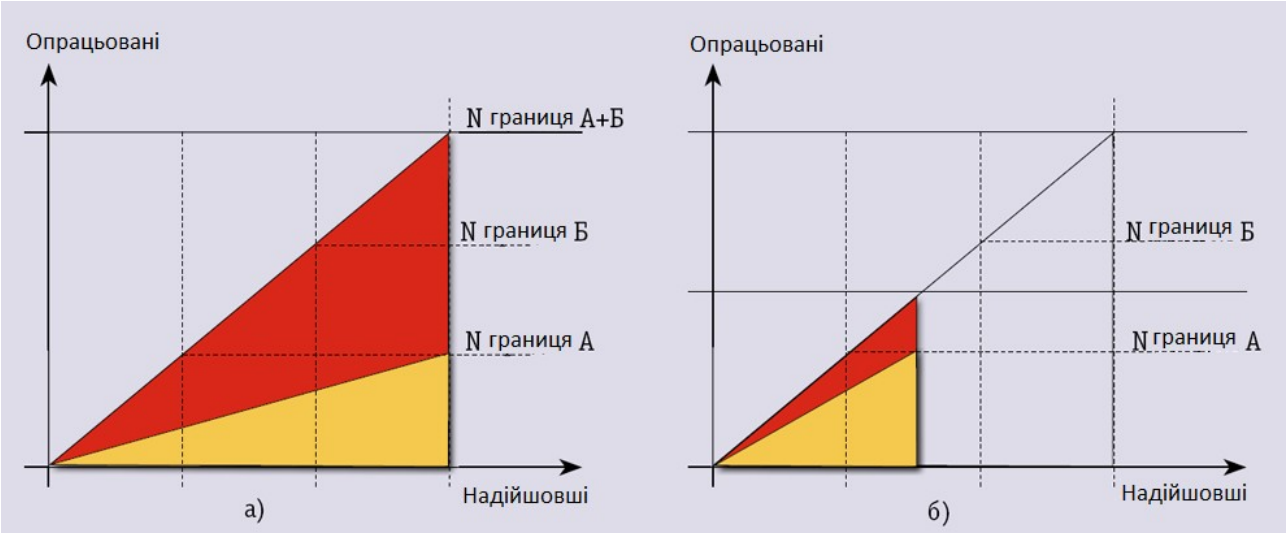

Рисунок 3.1 – Залежність числа оброблених запитів від числа надійшовших запитів

Таким чином, одне з найважливіших завдань балансування - визначення оптимальних пропорцій розподілу надходячих запитів і їх подальше динамічне корегування при зміні умов функціонування системи (зміна числа вузлів, складу встановленого і запущених додатків, модернізація апаратної платформи окремих вузлів і т. д.). Оптимальні пропорції в нашому випадку - це таке співвідношення числа оброблених запитів різними вузлами, при якому рівні завантаження цих вузлів приблизно рівні.

Для досягнення рівномірного розподілу обчислювального навантаження необхідно враховувати насамперед поточне завантаження центрального процесора і ряд характеристик (обсяг доступної оперативної пам'яті, продуктивність шин даних і мережевого інтерфейсу, обсяг доступного дискового простору і т.д.), які можуть використовуватися як індикатори, що вказують на вихід апаратної платформи з допустимого режиму оптимального функціонування.

3.3 Кроки балансування навантаження

Зазвичай практичне і повне рішення задачі балансування завантаження складається з чотирьох кроків[10]:

- Оцінка завантаження обчислювальних вузлів.
- Ініціація балансування завантаження.
- Прийняття рішень про балансування.
- Переміщення об'єктів.

Розглянемо їх докладніше.

Оцінка завантаження

На цьому етапі здійснюється приблизна оцінка завантаження кожного процесора. Отримана інформація про завантаження використовується в якості бази даних для процесу балансування, по-перше, для визначення виникнення дисбалансу, по-друге, для визначення нового розподілу об'єктів імітаційної моделі шляхом обчислення обсягу робіт, необхідного для переміщення об'єктів. Звідси, якість роботи балансування завантаження безпосередньо залежить від точності і повноти інформації в базі даних.

В основному така база даних складається з двох типів даних:

- Дані про роботу процесора (інформація рівня процесора). Ці дані включають: завантаження процесора, час простою процесора, фонове завантаження процесора, швидкість передачі інформації по лініях зв'язку і т.д.
- Дані про роботу розподіленого додатка. Дані включають час виконання окремого завдання, час простою, інтенсивність обміну інформацією і т.д.

Ініціалізація балансування завантаження

Занадто часте виконання балансування завантаження може призвести до того, що виконання імітаційної моделі тільки сповільниться. Витрати на сам процес балансування можуть перевершити можливу вигоду від його проведення. Отже, для продуктивності балансування необхідно якимось чином визначати момент його ініціалізації.

Для цього слід:

- Визначити момент виникнення дисбалансу завантаження.
- Визначити ступінь необхідності балансування шляхом порівняння можливої користі від її проведення та витрат на неї.

Дисбаланс завантаження може визначатися синхронно і асинхронно.

При синхронному визначенні дисбалансу всі процесори (комп'ютери мережі) переривають роботу в певні моменти синхронізації і визначають дисбаланс завантаження шляхом порівняння завантаження окремого процесора з загальної середньої завантаженням.

При асинхронному визначенні дисбалансу кожен процесор зберігає історію своєї завантаження. У цьому випадку момент синхронізації для визначення ступеня дисбалансу відсутня. Обчисленням обсягу дисбалансу займається фоновий процес, що працює паралельно з додатком.

Прийняття рішень в процесі балансування

Більшість стратегій динамічного балансування завантаження можна віднести до класу централізованих або до класу повністю розподілених.

При централізованої стратегії спеціальний комп'ютер збирає глобальну інформацію про стан всієї обчислювальної системи і приймає рішення про переміщення завдань для кожного з комп'ютерів.

При повністю розподіленої стратегії на кожному процесорі виконується алгоритм балансування завантаження, обмінюються інформацією про стан з іншими процесорами. Переміщення відбувається лише між сусідніми процесорами.

Переміщення об'єктів

Після прийняття рішень про балансування відбувається переміщення об'єктів серед процесорів для досягнення нового балансу завантаження. При переміщенні об'єкта повинна забезпечуватися цілісність його стану. Для переміщення даних об'єкта зазвичай використовуються допоміжні функції програми, особливо коли мова йде про складні структурах даних, таких як списки посилань або покажчиків.

Для "хмарних" обчислень останній крок заміняється на переадресацію

запитів, які надходять до системи на екземпляр, який визначено як підходящий.

3.4 Методи балансування навантаження

Нехай ми створили кілька серверів (будь-якого призначення - http, база даних тощо), кожен з яких може обробляти запити. Перед нами постає завдання - як розподілити між ними роботу, як дізнатися, на який сервер відправляти запит?

Робота з сайтом зазвичай не обмежується одним запитом. Тому при проектуванні важливо зрозуміти, чи можуть послідовні запити клієнта бути коректно оброблені різними серверами, або клієнт повинен бути прив'язаний до одного сервера на час роботи з сайтом. Це особливо важливо, якщо на сайті зберігається тимчасова інформація про сесію роботи користувача (у цьому випадку теж можливий вільний розподіл - проте тоді необхідно зберігати сесії в загальному для всіх серверів сховище). «Прив'язати» відвідувача до конкретного сервера можна за його IP-адресою (яка, однак, може мінятися), або по cookie (в яку заздалегідь записаний ідентифікатор сервера), або навіть просто перенаправивши його на потрібний домен.

Процедура балансування здійснюється за допомогою цілого комплексу алгоритмів і методів, відповідним наступним рівням моделі OSI [11]:

- мережевому;
- транспортному;
- прикладному.

Розглянемо ці рівні більш докладно.

Балансування на мережевому рівні передбачає вирішення наступного завдання: потрібно зробити так, щоб за одну конкретну IP-адресу сервера відповідали різні фізичні машини. Таке балансування може здійснюватися за допомогою декількох способів.

- DNS-балансування. На одне доменне ім'я виділяється кілька IP-адрес. Сервер, на який буде направлений клієнтський запит, зазвичай

визначається за допомогою алгоритму Round Robin (про методи і алгоритмах балансування буде детально розказано нижче).

- Побудова NLB-кластеру. При використанні цього способу сервери об'єднуються в кластер, що складається з вхідних і обчислювальних вузлів. Використовується в рішеннях від компанії Microsoft.
- Балансування за IP з використанням маршрутизатора.
- Балансування за територіальною ознакою здійснюється шляхом розміщення однакових сервісів з однаковими адресами в територіально різних регіонах Інтернету.

До переваг балансування на третьому рівні можна віднести:

- незалежність від протоколу високого рівня;
- абсолютна прозорість для серверів;
- незалежність від мережевого розташування серверів.

До недоліків:

- весь трафік серверів повинен проходити через балансувальник, тому він повинен витримувати високе навантаження;
- правила розподілу є статичними, вони не аналізують доступність і поточну завантаженість серверів;
- не підтримує довгострокових сесій між користувачем і сервером.

Балансування на транспортному полягає в тому, що клієнт звертається до балансувальника, той перенаправляє запит одному з серверів, який і буде його обробляти. Вибір сервера, на якому буде оброблятися запит, може здійснюватися відповідно до самих різних алгоритмів (про це ще піде мова нижче): шляхом простого кругового перебору, шляхом вибору найменш завантаженого сервера з пулу і т.п. На транспортному рівні спілкування з клієнтом замикається на балансувальник, який працює як проксі. Він взаємодіє з серверами від свого імені, передаючи інформацію про клієнта в додаткових даних і заголовках. Таким чином працює, наприклад, популярний програмний балансувальник HAProxy.

До переваг балансування на транспортному рівні можна віднести ожливість балансування навантаження незалежно від типу протоколу прикладного рівня до будь-яких TCP- сервісів: HTTPS, SMTP, IMAP, SSH, FTP, SQL і т.д.

При балансуванні на прикладному рівні балансувальник працює в режимі «розумного проксі». Він аналізує клієнтські запити і перенаправляє їх на різні сервери залежно від характеру запитуваного контенту. В такому разі у проксі-сервера може виникнути потреба в переформуванні адрес кожної веб-сторінки (перетворення імен ресурсів доступних зовні, в імена доступні у внутрішній мережі), якщо HTTP протоколу треба передати IP-адресу клієнта, то можна додати заголовок X-Real-IP з IP-адресою клієнта. Так працює, наприклад, веб-сервер Nginx, розподіляючи запити між фронтенда і бекенд. За балансування в Nginx відповідає модуль Upstream.

Завдяки кешуванню відповідей від серверів додатків, у ряді випадків, проксі- сервер може надати запитуваний ресурс взагалі без підключення до якого-небудь сервера. У параметрах конфігурації можна задавати ваги серверам додатків, допустимий час відгуку і допустиму кількість неуспішних запитів. Сервер може підтримувати довгострокові сесії клієнтів з одним і тим самим сервером на основі їх IP- адрес або значень cookie. HTTP проксінг застосовується в проектах mod_backhand для сервера Apache і в сервері nginx, HAProxy. На його базі випускаються також апаратно-програмні платформи, призначені тільки для балансування навантаження.

Спеціалізовані апаратно-програмні рішення, що виконують балансування з використанням трансляції заголовків мережевого, транспортного та рівня додатків, завдяки високій продуктивності мають подолати відповідний недолік балансування з пропуском трафіку через один пристрій. Вони підтримують контроль за встановленими сесіями на окремі сервери, постійно контролюють час відповіді серверів, навантаження на які вони розподіляють, і аналізують кількість запитів, які були їм відправлені. Додатково на ці пристрої можливе винесення функцій формування захищеного

з'єднання SSL і стиснення даних, що дозволяє значно знизити навантаження на сервери додатків.

Переваги балансування на прикладному рівні:

- Робота на рівні протоколу дозволяє проксі-серверу аналізувати і змінювати запити і відповіді, виконувати додавання заголовка, що може використовуватися, наприклад, для підвищення безпеки програми або для перекодування.
- Підтримується прив'язка клієнта до сервера для зберігання його довготривалої сесії.
- Аналіз змісту запитів дозволяє розподіляти їх в залежності від типу по різних серверах (наприклад, до статичних сторінок, до динамічних сторінок і т.д.), що прискорює час відгуку вцілому.
- Є можливість кешування відповідей на проксі-сервері, стискати відправляємі дані самостійно, знижуючи загальне навантаження обслуговуючого сервера.
- У ряді випадків є можливість перемістити обробку SSL з обслуговуючих серверів на проксі;
- Можливий аналіз непрямих показніків завантаження и автоматична перевірка працездатності серверів додатків.

Недоліки балансування на прикладному рівні:

- З усіх розглянутих методів найбільше споживання ресурсів, так як воно зачіпає протоколи більш високого рівня в порівнянні з іншими методами.
- Для кожного прикладного протоколу повинен бути свій тип проксі. Не для всіх протоколів проксінг можна реалізувати таким чином, щоб воно вирішувало потрібні завдання.

Можна виділити наступні класи рішень, використовувані при побудові багатовузлових систем (рис 3.2): балансування з пропуском трафіку через один пристрій балансування; балансування засобами кластера; балансування без пропуску трафіку через один пристрій балансування[9].

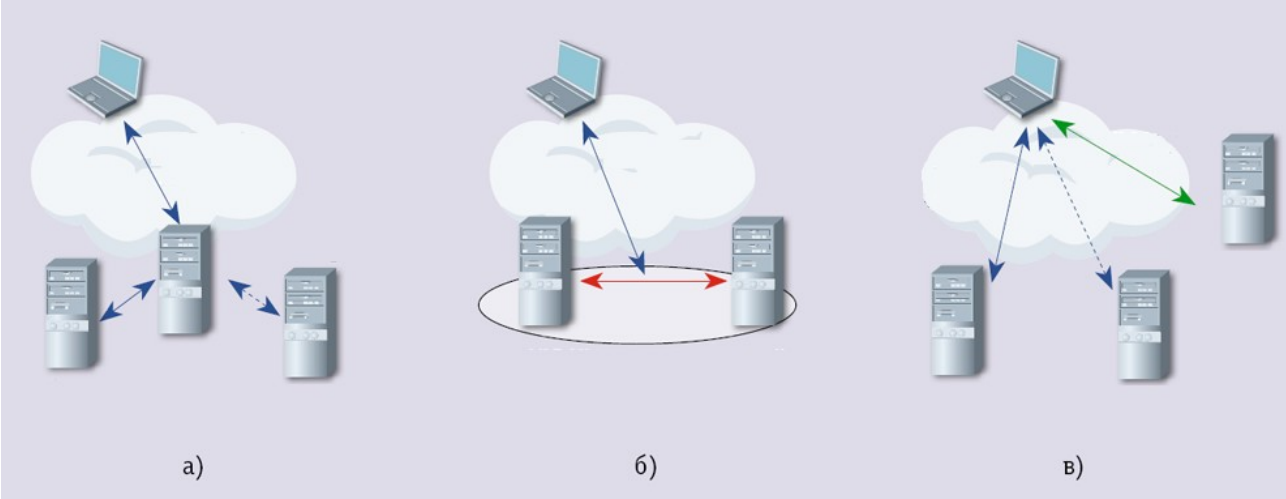

Рисунок 3.2 – Принципи балансування навантаження: а) балансування з пропуском трафіку через один пристрій балансування; б) балансування засобами кластера; в) балансування без пропуску трафіку через один пристрій балансування

Розглянемо ці принципи докладніше.

Балансування з пропуском трафіку через один пристрій балансування

Клієнт шле запит на один фіксований, відомий йому сервер, а той вже перенаправляє запит на один з робочих серверів. Типовий приклад - система з одним frontend і декількома backend-серверами, на які відправляються запити. Однак «клієнт» може знаходитися і всередині нашої системи - наприклад, скрипт може слати запит до проксі-сервера бази даних, який передасть запит одному з серверів СУБД. Сам балансуючий вузол може працювати як на окремому сервері, так і на одному з робочих серверів. Переваги цього підходу в тому, що клієнту нічого не треба знати про внутрішній устрій системи - про кількість серверів, про їх адреси та особливості - всю цю інформацію знає тільки балансувальник. Однак недолік в тому, що балансуючий вузол є єдиною точкою відмови системи - якщо він вийде з ладу, вся система виявиться непрацездатна. Крім того, при великому навантаженні балансувальник може просто перестати справлятися зі своєю роботою, тому такий підхід застосовний

не завжди. Виходом з цієї ситуації є постійне резервування та перевірка стану машин.

У методі перетворення мережевих адрес (Network Address Translation, NAT) система балансування направляє отриманий від клієнта пакет призначеного сервера лише після того, як замінить в пакеті адресу одержувача IP-адресою призначеного сервера і змінить IP-адресу відправника на свою. Це дозволяє приховувати від клієнтів IP-адреси веб-серверів, так що вони можуть використовувати будь-які IP-адреси, в тому числі і приватні. При цьому веб-сервери не обов'язково повинні входити в один і той же сегмент мережі - достатньо, щоб вони могли взаємодіяти з використанням протоколу IP. Даний метод застосовується тільки для балансування тих запитів, де не потрібно створення сесій між користувачем і сервером, існуючих тривалий час, оскільки для кожного користувача потрібно виділити індивідуальну IP-адресу на внутрішньому інтерфейсі системи балансування на весь час сесії.

Відмінністю методу NAPT (Network Address Port Translation) є те, що система балансування змінює в заголовку вже не тільки IP-адреси, як в NAT, але і номери портів транспортного рівня. Це дозволяє для кожного нового надходження запиту від нового унікального користувача вже не виділяти індивідуальну IP-адресу на внутрішньому інтерфейсі системи балансування для сесії, а виділяти унікальний номер порту транспортного рівня. Даний механізм балансування можна реалізувати на універсальних апаратних платформах, на програмних компонентах він входить в стандартні поставки сучасних операційних систем (natd в BSD, iptables у складі дистрибутивів Linux, RRAS у складі серверних збірок Windows і т. д.).

Прикладами продуктів, у яких реалізований підхід NAPT, є пристрої XTM Firewall Appliance компанії WatchGuard, універсальні маршрутизатори Cisco 1900, міжмережевий екран D-Link DFL-860 і т.п. Всі вони при надходженні запитів на зовнішню адресу виконують трансляцію заголовка і перенаправляють запит на одну з внутрішніх адрес, які присвоєні серверам. На період встановленої сесії на пристрої в таблиці зберігається відповідність

зовнішньої IP-адреси, зовнішнього порту транспортного рівня, локальної IP-адреси і локального порту. Пристрої не аналізують можливості серверів, час відгуку, доступність серверів і т.п. Вони здатні виконувати балансування за алгоритмами послідовного перебору або на підставі кількості відкритих сесій TCP на кожен з серверів. Найчастіше дана функціональність служить доповненням до основних функцій маршрутизатора або брандмауера.

Балансування з використанням трансляції заголовків на рівнях вище транспортного також називають проксінг. Проксі-сервер, міняючи поля заголовків транспортного та рівня додатків, перенаправляє запит на сервери, на яких розташовані запитувані ресурси. У ряді випадків він може надати запитуваний ресурс взагалі без підключення до якого-небудь сервера. При цьому проксі-сервер «кешує» відповіді від віддалених серверів і надає збережену інформацію у відповідь на всі наступні запити. Проксі-сервер може розподіляти навантаження між декількома веб-серверами, які можуть обслуговувати різні галузі застосування. В такому разі у проксі-сервера може виникнути потреба в переформування адрес кожної веб-сторінки (перетворення імен ресурсів, доступних зовні, в імена, доступні у внутрішній мережі). Проксі-сервер, який передає запити і відповіді в незмінному вигляді, зазвичай називається шлюзом. Проксінг застосовується в проектах mod_backhand для сервера Apache і в сервері nginx. При надходженні HTTP-запитів сервер перенаправляє їх на заздалегідь сконфігуровані сервери з однаковими ресурсами. У параметрах конфігурації можна задавати ваги сконфігурованим серверам, допустимий час відгуку і допустима кількість неуспішних запитів. Прикладом програмного рішення, що використовує методологію проксінг запитів TCP / HTTP з більш просунутими функціями управління перенаправлення запитів, є HAProxy, що дозволяє також виконувати балансування додатків. HAProxy працює на платформах Linux, Free BSD і Solaris. На його базі випускаються також апаратно-програмні платформи, призначені тільки для балансування навантаження.

Можливості систем балансування на базі програмних продуктів в

частині забезпечення наскрізної продуктивності визначаються потенціалом універсальних апаратних платформ, на яких вони функціонують. Так, для HAProxy була продемонстрована можливість функціонування на швидкостях до 10 Гбіт / с і 40 тис. Одночасних сесій (без контролю системою балансування часу обробки запитів серверами додатків). На базі HAProxy розробляються сервіси балансування навантаження Loadbalancer.org, які використовуються спільно з хмарними сервісами Amazon.

Прикладами спеціалізованих апаратно-програмних рішень, що виконують балансування з використанням трансляції заголовків мережевого, транспортного та рівня додатків, є Brocade Server Iron ADX, xBalancer компанії Net Optics, Cisco ACE 4700 Application Control Engine, Equalizer E650GX компанії Coyote Point і ряд інших продуктів. Такі системи постійно контролюють час відповіді серверів, навантаження на які вони розподіляють, і аналізують кількість запитів, які були їм відправлені. Дані відомості використовуються для подальшого вирівнювання завантаження серверних платформ. Додатково на ці пристрої можливе винесення функцій формування захищеного з'єднання SSL і стиснення даних, що дозволяє значно знизити навантаження на сервери додатків. Такі рішення мають більш високою продуктивністю зі збереженням всієї функціональності. Зокрема, для Server Iron ADX заявлена наскрізна продуктивність може досягати 70 Гбіт / с, 16 млн сесій в секунду. При цьому зберігається вся функціональність з контролю за встановленими сесіями на окремі сервери. Більшість хмарних платформ, що надають сервіси SaaS, використовують подібні апаратно-програмні рішення для балансування навантаження всередині ЦОД спільно з рішеннями для балансування навантаження між ЦОД. Зокрема, Brocade Server Iron ADX використовується в хмарних сервісах Google, Yahoo, Apple, в програмних продуктах корпоративного рівня, додатках IBM, HP, Microsoft і т.д.

Серед основних недоліків рішень, що використовують єдину точку пропуску трафіку, слід відзначити наявність єдиної точки відмови, обмежені можливості по функціональності і масштабуванню продуктивності при

використанні систем балансування на базі універсальних апаратних платформ, а також високу вартість рішень балансування навантаження на базі спеціалізованих апаратно-програмних комплексів, яка обумовлена насамперед вузькою сферою застосування даних рішень і високими вимогами по продуктивності і надійності.

Балансування засобами кластера

Кластеризація дозволяє управляти групою незалежних серверів як єдиною системою, що підвищує відмовостійкість, спрощує управління і дозволяє домогтися більшої масштабованості. Сервіс балансування в кластері забезпечує розподіл потоку IP-даних між вузлами. У кластері кілька апаратних платформ найчастіше розділяють єдину IP-адресу.

Для кожного вузла, який бере участь у розподілі навантаження, адміністратор може вказати його конкретну частку навантаження, або за замовчуванням навантаження буде рівномірно розподілятися між вузлами. Запити клієнтів статистично розподіляються між вузлами, щоб кожен сервер обробляв рівно свою частину, а розподіл навантаження змінюється при включенні або видаленні вузла кластера. Розподіл навантаження не змінюється при зміні параметрів навантаження на апаратні платформи серверів. Для програм з великою кількістю клієнтів і породжуваних ними коротких запитів, таких як веб-сервіси, подібний механізм розподілу навантаження забезпечує ефективне балансування навантаження і швидку реакцію на зміну складу вузлів кластера.

Кожен сервер кластера під управлінням сервісу балансування навантаження періодично розсилає широкомовні або групові повідомлення іншим вузлам і приймає такі ж повідомлення від інших членів кластеру. При збої сервера решта вузли перерозподіляють навантаження, забезпечуючи безперервне обслуговування. Незважаючи на те, що дані існуючих підключень до втраченого вузла губляться, сервіси залишаються доступними. У більшості випадків, як ,наприклад, при балансуванні навантаження веб-серверів, браузер на стороні клієнта автоматично повторює підключення, що закінчилися збоєм, і

збій відбивається на клієнті тільки у вигляді короткої затримки відгуку на запит.

Подібним чином працюють механізми балансування мережного навантаження в кластерах Windows Server 2008 R2, Oracle Solaris Cluster, Open HA Cluster і Linux Virtual Server.

До недоліків рішення на базі кластерів слід віднести необхідність розташування серверних платформ в одному сегменті мережі, неможливість забезпечити мультиплатформенний кластер, необхідність використання на кожному вузлі спеціалізованого ПЗ для функціонування кластера, в завдання якого входитиме контроль стану суміжних вузлів в кластері і диспетчеризація надходять запитів.

Балансування без єдиного пристрою

Якщо ми хочемо уникнути єдиної точки відмови, існує альтернативний варіант - доручити вибір сервера самому клієнтові. У цьому випадку клієнт повинен знати про внутрішній устрій нашої системи, щоб уміти правильно вибирати, до якого сервера звертатися. Безсумнівним плюсом є відсутність точки відмови - при відмові одного з серверів клієнт зможе звернутися до інших. Однак платою за це є ускладнення логіки клієнта і менша гнучкість балансування.

Даний варіант балансування організовується шляхом відправки первісного запиту від користувача (наприклад, запиту IP-адреси для встановлення з'єднання або першого запиту на встановлення HTTP-сесії) на виділений сервер-балансувальник. Після отримання запиту балансувальник вказує, з яким сервером додатків з інфраструктури хмари продовжувати роботу (повертаючи відповідну IP-адресу або перенаправляючи його засобами HTTP Redirect). Подальша взаємодія відбувається без участі балансувальника.

Найбільш відомий метод, який реалізує даний підхід, - використання кругових DNS. У базах даних DNS-серверів зберігаються записи, що визначають відповідність між ім'ям хоста і його IP-адресою. Технологія DNS дозволяє внести в базу кілька записів з одним і тим же доменним ім'ям, але

різними IP-адресами. Таким чином, можна вказати кілька серверних платформ для однієї програми. У цьому випадку перший користувач, який звернувся за даною адресою, буде відправлений на перший комп'ютер, другий - на другу і т. д., а коли сервер «пройдеться» по всіх записах, відповідним одному домену, він знову повернеться на першу. Більшість сучасних DNS-серверів підтримують опцію кругових DNS.

Рідше використовується балансування засобами HTTP Redirect. Рішення, що використовують даний підхід, дозволяють задавати в конфігурації сервера, що виконує балансування, групу IP-адрес серверів додатків. Для кожного з них може бути заданий як параметр вагу, що дозволяє визначити частку HTTP-запитів, які відправляються на кожен з серверів додатків. Прикладом рішень, які використовують засоби балансування із застосуванням механізмів HTTP Redirect, є модулі Mod_backhand для серверів Apache, а також Citrix NetScaler, що оптимізує доставку веб-додатків.

З точки зору простоти, передбачуваності адміністрування і масштабування технологію балансування без пропуску трафіку через єдиний пристрій можна вважати практично ідеальним варіантом - не потрібно купувати, встановлювати і налаштовувати спеціальні пристрої балансування або програмне забезпечення на серверах додатків, а достатньо один раз внести в базу DNS кілька записів . Оскільки пристрій балансування при цьому фактично обробляє тільки початкові запити користувачів, а не весь його трафік, то межі масштабування пристроїв досить високі. Відмовостійкість ж самих пристроїв DNS-серверів або серверів, що виконують HTTP Redirect, цілком можливо забезпечити традиційними засобами резервування DNS.

Разом з тим надійність розподіленої програми, створеної за допомогою кругової DNS або засобів HTTP Redirect, досить низька: якщо один або більше серверів, на яких міститься копія додатку, виходить з ладу або зупиняється на профілактику, то частина запитів «падає в порожнечу». Відсутність зворотного зв'язку не дозволяє виключити сервер додатків зі списку, на який перенаправляються користувачі. Крім того, існуючі рішення балансування

навантаження без пропуску трафіку через один пристрій всього лише ділять запити між усіма платформами, що входять до складу розподіленого сервера, рівномірно або пропорційно наперед заданим ваг. Якщо ж ці платформи мають різні обчислювальні ресурси, різний набір встановлених додатків, то статичний розподіл запитів може привести до того, що частина платформ виявиться перевантажена, а інші, навпаки, будуть простоювати. І навіть якщо всі комп'ютери у складі розподіленого сервера абсолютно однакові, це не означає, що вони будуть обробляти запити з однаковою швидкістю, адже одному користувачеві може знадобитися відкрити статичну веб-сторінку, а інший запустить на виконання скрипти, що вимагають великих системних ресурсів. Крім того, відомо, що при підвищенні часу відгуку від завантаженого сервера користувачі частими натисканнями на Refresh в своїх браузерах починають посилати обвальне кількість запитів, моментально перевантажують машину.

Найбільш придатними до використання при побудові багатовузлових SaaS-систем виглядають рішення без пропуску трафіку через один пристрій, проте відсутність механізмів зворотного зв'язку між серверними платформами, на яких розташовується додаток, і пристроєм, що виконує балансування, не дозволяє повною мірою скористатися перевагами розподілених систем.

3.5 Алгоритми балансування навантаження

Існує багато різних алгоритмів балансування навантаження. Вибираючи конкретний алгоритм, потрібно виходити, по-перше, із специфіки конкретного проекту, а по-друге - з цілей. які ми плануємо досягти.

У числі цілей, для досягнення яких використовується балансування, потрібно виділити наступні [11]:

- справедливість: потрібно гарантувати, щоб на обробку кожного запиту виділялися системні ресурси і не допустити виникнення ситуацій, коли один запит обробляється, а всі інші чекають своєї черги;
- ефективність: всі сервери, які обробляють запити, повинні бути зайняті на 100%; бажано не допускати ситуації, коли один з серверів

простоює в очікуванні запитів на обробку (відразу ж обмовимося, що в реальній практиці ця мета досягається далеко не завжди);

- скорочення часу виконання запиту: потрібно забезпечити мінімальний час між початком обробки запиту (або його постановкою в чергу на обробку) і його завершення;
- скорочення часу відгуку: потрібно мінімізувати час відповіді на запит користувача.

Дуже бажано також, щоб алгоритм балансування володів такими властивостями:

- передбачуваність: потрібно чітко розуміти, в яких ситуаціях і за яких навантаженнях алгоритм буде ефективним для вирішення поставлених завдань;
- рівномірне завантаження ресурсів системи;
- масштабованість: алгоритм повинен зберігати працездатність при збільшенні 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. Найпоширенішою імплементацією цього алгоритму є, звичайно ж, метод балансування Round Robin DNS.

Round robin DNS - один з методів розподілу навантаження, або відмовостійкості за рахунок надмірності кількості серверів, за допомогою управління відповідями DNS-сервера відповідно до якоїсь статистичної моделі. Звичайно застосовується до таких інтернет-протоколів, як веб-сервери, FTP-сервери[12].

У найпростішому випадку Round robin DNS працює, відповідаючи на запити не тільки одною IP-адресою, а списком з декількох адрес серверів, що надають ідентичний сервіс. Порядок, в якому повертаються IP-адреси зі списку, заснований на алгоритмі round-robin. З кожною відповіддю послідовність ip-адрес змінюється. Як правило, прості клієнти намагаються встановлювати

з'єднання з першою адресою зі списку, таким чином різним клієнтам будуть видані адреси різних серверів, що розподілить загальне навантаження між серверами.

Не існує стандартної процедури для визначення того, які адреси будуть використовуватися - деякі сервери намагаються змінити порядок списку, приділяючи пріоритетну увагу чисельно більш «близьким» мережам. Деякі настільні клієнти намагаються отримати альтернативні адреси після того, як не вдалося встановити з'єднання протягом 30-45 секунд.

Кругова система DNS часто використовується для розподілу навантаження територіально розподілених веб-серверів. Наприклад, у компанії є один домен і три ідентичних веб-сайти, розташованих на трьох серверах з трьома різними адресами. Коли один користувач отримує доступ до головної сторінки, він буде направлений на першу адресу IP. Другий користувач, який звертається до головної сторінки, буде відправлений на наступну адресу IP, а третій користувач буде відправлений на третю адресу. У кожному випадку, коли IP-адреса видається, вона відправляється в кінець списку. Четвертий користувач, отже, буде відправлений знову на першу адресу IP, і так далі.

Хоча Round robin DNS (RR DNS) легко реалізувати, все ж цей алгоритм має кілька проблематичних недоліків, пов'язаних з кешуванням записів в ієрархії RR DNS самого себе, а також з кешуванням на стороні клієнта, виданої адреси та її повторного використання, поєднання яких важко керовано . RR DNS не спирається на доступність послуг. Наприклад, якщо сервіс на одній з адрес недоступний, RR DNS буде продовжувати роздавати цю адресу і клієнти будуть як і раніше намагатися з'єднатися з непрацюючим сервером.

Крім того, цей алгоритм не може бути кращим вибором для балансування навантаження на самого себе, оскільки він лише замінює порядок адрес кожен раз, коли ім'я сервера запитується. Не існує урахування відповідності IP-адреси користувача і його географічного розташування, часу виконання, навантаження на сервер, перевантаження мережі і т. д. Кругова система DNS навантаження найкраще підходить для послуг з великою

кількістю рівномірно розподілених з'єднань з серверами еквівалентної потужності. В іншому випадку він просто робить розподіл навантаження.

Існують методи, щоб подолати такі обмеження. Наприклад, модифіковані DNS-сервера (такі, як lbnamed) можуть регулярно опитувати дзеркала серверів для перевірки їх доступності та завантаженості. Якщо сервер не відповідає в міру необхідності, сервер може бути тимчасово вилучений з пулу DNS, поки він не повідомить, що знову працює у відповідності зі специфікацією.

У числі безперечних плюсів цього алгоритму слід назвати, по-перше, незалежність від протоколу високого рівня. Для роботи за алгоритмом Round Robin використовується будь-який протокол, в якому звернення до сервера йде по імені.

Балансування на основі алгоритму Round Robin ніяк не залежить від навантаження на сервер: кешуючі DNS-сервери допоможуть впоратися з будь-яким напливом клієнтів.

Використання алгоритму Round Robin не вимагає зв'язку між серверами, тому він може використовуватися як для локального, так і для глобального балансування.

Нарешті, рішення на базі алгоритму Round Robin відрізняються низькою вартістю: щоб вони почали працювати, досить просто додати кілька записів в DNS.

Алгоритм Round Robin має і цілий ряд істотних недоліків. Щоб розподіл навантаження за цим алгоритмом відповідав згаданим вище критеріями справедливості та ефективності, потрібно, щоб у кожного сервера був в наявності однаковий набір ресурсів. При виконанні всіх операцій також має бути задіяно однакову кількість ресурсів. У реальній практиці ці умови в більшості випадків виявляються нездійсненними.

Також при балансуванні за алгоритмом Round Robin абсолютно не враховується завантаженість того чи іншого сервера в складі кластера. Уявімо собі наступну гіпотетичну ситуацію: один з вузлів завантажений на 100%, в той

час як інші - всього на 10 - 15%. Алгоритм Round Robin можливості виникнення такої ситуації не враховує в принципі, тому перевантажений вузол все одно буде отримувати запити. Ні про яку справедливість, ефективність та передбачуваність в такому випадку не може бути й мови.

Weighted Round Robin - це вдосконалена версія алгоритму Round Robin. Суть удосконалень полягає в наступному: кожного серверу присвоюється ваговий коефіцієнт відповідно до його продуктивністю і потужністю. Це допомагає розподіляти навантаження більш гнучко: сервери з великою вагою обробляють більше запитів. Однак усіх проблем з відмовостійкістю це аж ніяк не вирішує. Більш ефективне балансування забезпечують інші методи, в яких при плануванні та розподілі навантаження враховується більшу кількість параметрів.

Вище ми перерахували основні недоліки алгоритму Round Robin. Назвемо ще один: у ньому абсолютно не враховується кількість активних на даний момент підключень.

Розглянемо практичний приклад. Є два сервера - позначимо їх умовно як А і Б. До сервера А підключено менше користувачів, ніж до сервера Б. При цьому сервер А виявляється більш перевантаженим. Як це можливо? Відповідь досить проста: підключення до сервера А підтримуються протягом більш довгого часу в порівнянні з підключеннями до сервера Б.

Описану проблему можна вирішити за допомогою алгоритму, відомого під назвою least connections (скорочено - leastconn). Він враховує кількість підключень, підтримуваних серверами в поточний момент часу. Кожен наступний запит передається серверу з найменшою кількістю активних підключень.

Існує вдосконалений варіант цього алгоритму, призначений в першу чергу для використання в кластерах, що складаються з серверів з різними технічними характеристиками і різною продуктивністю. Він називається Weighted Least Connections і враховує при розподілі навантаження не тільки кількість активних підключень, а й ваговий коефіцієнт серверів.

У числі інших удосконалених варіантів алгоритму Least Connections слід передусім виділити Locality-Based Least Connection Scheduling і Locality-Based Least Connection Scheduling with Replication Scheduling.

Перший метод був створений спеціально для кешуючих проксі-серверів. Його суть полягає в наступному: найбільша кількість запитів передається серверам з найменшою кількістю активних підключень. За кожним з серверів додатків закріплюється група клієнтських IP. Запити з цих IP направляються на «рідний» сервер, якщо він не завантажений повністю. В іншому випадку запит буде перенаправлений на інший сервер (він повинен бути завантажений менш ніж наполовину).

В алгоритмі Locality-Based Least Connection Scheduling with Replication Scheduling кожна IP-адреса або група IP-адрес закріплюється не за окремим сервером, а за цілою групою серверів. Запит передається найменш завантаженому серверу з групи. Якщо ж всі сервери з «рідної» групи перевантажені, то буде зарезервований новий сервер. Цей новий сервер буде додано до групи, яка обслуговує IP, з якого був відправлений запит. У свою чергу найбільш завантажений сервер з цієї групи буде видалено - це дозволяє уникнути надмірної реплікації.

Алгоритм Destination Hash Scheduling був створений для роботи з кластером кешуючих проксі-серверів, але він часто використовується і в інших випадках. У цьому алгоритмі сервер, що обробляє запит, вибирається з статичної таблиці за IP-адресою одержувача.

Алгоритм Source Hash Scheduling ґрунтується на тих же самих принципах, що і попередній, тільки сервер, який буде обробляти запит, вибирається з таблиці за IP-адресою відправника.

Sticky Sessions - алгоритм розподілу вхідних запитів, при якому з'єднання передаються на один і той же сервер групи. Він використовується, наприклад, в веб-сервері Nginx. Сесії користувача можуть бути закріплені за конкретним сервером за допомогою методу IP hash. За допомогою цього методу запити розподіляються по серверах на основі IP-адреси клієнта. Як зазначено в

документації, «метод гарантує, що запити одного і того ж клієнта буде передаватися на один і той же сервер». Якщо закріплений за конкретною адресою сервер недоступний, запит буде перенаправлений на інший сервер.

Застосування цього методу пов'язане з деякими проблемами. Проблеми з прив'язкою сесій можуть виникнути, якщо клієнт використовує динамічний IP. У ситуації, коли велика кількість запитів проходить через один проксі-сервер, балансування навряд чи можна назвати ефективним і справедливим. Описані проблеми, однак, можна вирішити, використовуючи cookies. У комерційній версії Nginx є спеціальний модуль sticky, який якраз використовує cookies для балансування. Є у нього і безкоштовні аналоги - наприклад, nginx-sticky-module.

3.6 Висновки

Важливою частиною "хмарної" інфраструктури є балансування навантаження. Можна виділити наступні класи рішень, використовувані для балансування: балансування з пропуском трафіку через один пристрій балансування; балансування засобами кластера; балансування без пропуску трафіку через один пристрій балансування.

Балансування навантаження, або вирівнювання навантаження (англ. Load balancing) - метод розподілу завдань між декількома мережевими пристроями (наприклад, серверами) з метою оптимізації використання ресурсів, скорочення часу обслуговування запитів, горизонтального масштабування кластера (динамічне додавання / видалення пристроїв), а також забезпечення відмовостійкості (резервування).

Балансування навантаження передбачає рівномірне навантаження обчислювальних вузлів. При появі нових завдань програмне забезпечення, що реалізує балансування, має прийняти рішення про те, де (на якому обчислювальному вузлі) слід виконувати обчислення, пов'язані з цим новим завданням.

Балансування навантаження може бути використане для розширення

можливостей ферми серверів, що складається більш ніж з одного сервера.

Слід розрізняти статичне і динамічне балансування. Статичне балансування виконується до початку виконання розподіленого додатка. Динамічне балансування передбачає перерозподіл обчислювального навантаження на вузли під час виконання програми.

Програмне забезпечення, що реалізує динамічне балансування, визначає такі показники:

1. Прямі:

- завантаження процесора обчислювальних вузлів;
- завантаження пам'яті обчислювальних вузлів;
- пропускну спроможність ліній зв'язку.

2. Непрямі:

- кількість активних з’єднань;
- час відгуку обчислювального вузла

Зазвичай практичне і повне рішення задачі балансування завантаження складається з чотирьох кроків:

- Оцінка завантаження обчислювальних вузлів.
- Ініціація балансування завантаження.
- Прийняття рішень про балансування.
- Перенаправлення об'єктів.

Для досягнення рівномірного розподілу обчислювального навантаження необхідно враховувати насамперед поточне завантаження центрального процесора і ряд характеристик (обсяг доступної оперативної пам'яті, продуктивність шин даних і мережевого інтерфейсу, обсяг доступного дискового простору і т.д.), які можуть використовуватися як індикатори, що вказують на вихід апаратної платформи з допустимого режиму оптимального функціонування.

Процедура балансування здійснюється за допомогою цілого комплексу алгоритмів і методів, відповідним наступним рівням моделі OSI:

- мережевому;
- транспортному;
- прикладному.

Балансування на мережевому рівні може здійснюватися за допомогою різноманітних способів:

- DNS-балансування;
- Побудова NLB-кластеру;
- Балансування за IP засобами вхідного шлюзу;
- Балансування зазасобами маршрутизаторів глобальної мережі.

Переваги балансування на третьому рівні:

- незалежність від протоколу високого рівня;
- абсолютна прозорість для серверів;
- незалежність від мережевого розташування серверів.

Недоліки балансування на третьому рівні відповідають способу реалізації балансування: з єдиним пристроєм балансування або без єдиного пристрою балансування.

Балансування на транспортному рівні полягає в тому, що перенаправлення запитів клієнтів одному з обробляючих серверів системою балансування здійснюється не тільки на основі IP- адрес, а й на основі номерів портів в першу чергу протоколу TCP. Це надає методам транспортному рівні більшу гнучкість у порівнянні до методів мережевого рівня. До переваг балансування на транспортному рівні можна віднести можливість балансування навантаження незалежно від типу протоколу прикладного рівня до будь-яких TCP- сервісів: HTTPS, SMTP, IMAP, SSH, FTP, SQL і т.д.

Методи, що відносяться до транспортного рівня:

- Метод NAPT (Network Address Port Translation);
- Проксі- сервер 4-го рівня OSI.

При балансуванні на прикладному рівні балансувальник аналізує клієнтські запити і перенаправляє їх на різні сервери залежно від характеру

запитуваної контенту. До методів прикладного рівня можна віднести проксінг та HTTP Redirect

Переваги балансування на прикладному рівні:

- Робота на рівні протоколу дозволяє проксі-серверу аналізувати і змінювати запити і відповіді, виконувати додавання заголовка, що може використовуватися, наприклад, для підвищення безпеки програми або для перекодування.
- Підтримується прив'язка клієнта до сервера для зберігання його довготривалої сесії за cookie, за заголовком HTTP.
- Аналіз змісту запитів дозволяє розподіляти їх в залежності від типу по різних серверах (наприклад, до статичних сторінок, до динамічних сторінок і т.д.), що прискорює час відгуку вцілому.
- Є можливість кешування відповідей на проксі-сервері, стискати відправляємі дані самостійно, знижуючи загальне навантаження обслуговуючого сервера.
- У ряді випадків є можливість перемістити обробку SSL з обслуговуючих серверів на проксі.

Недоліки балансування на прикладному рівні:

- З усіх розглянутих методів найбільше споживання ресурсів, так як воно зачіпає протоколи більш високого рівня в порівнянні з іншими методами.
- Для кожного прикладного протоколу повинен бути свій тип проксі. Не для всіх протоколів проксінг можна реалізувати таким чином, щоб воно вирішувало потрібні завдання.

Для розподілу навантаження використовуються настуні алгоритми:

- Round Robin;
- Weighted Round Robin;
- Least Connections;
- Weighted Least Connections;

- Locality-Based Least Connection Scheduling;
- Locality-Based Least Connection Scheduling with Replication Scheduling;
- Destination Hash Scheduling;
- Source Hash Scheduling;
- Sticky Sessions.

Для розподілу навантаження територіально розподілених веб-серверів дуже часто використовується алгоритм Round robin. Але це не оптимально, так як в цьому методі не існує урахування відповідності IP-адреси користувача і його географічного розташування, часу виконання, навантаження на сервер, перевантаження мережі і т. д. Кругова система найкраще підходить для послуг з великою кількістю рівномірно розподілених з'єднань з серверами еквівалентної потужності.

У випадку, коли необхідно враховувати кількість підключень, які підтримуються сервером на даний момент часу рекомендується використовувати алгоритм балансування навантаження least connections. Він дозволяє уникнути ситуації, коли сервера, до якого підключено менше користувачів, виявляється більш перевантаженим ніж сервер, до якого підключено більше користувачів.

Для географічно розподіленій інфраструктурі доцільно використовувати алгоритми Destination Hash Scheduling та Source Hash Scheduling. У цих алгоритмах сервер, що обробляє запит, вибирається з статичної таблиці за IP-адресою одержувача або відправника відповідно.

Якщо необхідно підтримувати тривалу сесію з одним і тим же сервером групи, то слід використовувати алгоритм Sticky Sessions. В ньму запити розподіляються по серверах на основі IP-адреси клієнта.

У таблиці 3.2 наведено порівняльну характеристику розглянутих методів балансування навантаження за основними критеріями, які характеризують особливості у побудові і роботі різних Web- сервісів і відповідно їх вимоги до розподілу навантаження. Критерії, наведені в таблиці:

- Ефективність розподілу навантаження – рівномірність розподілу при різних умовах (однотипні запити, різні запити, сесії, т.д.);
- Показники завантаження вузлів – бувають прямі (поточне навантаження процесора, пам’яті і т.д.) та непрямі (кількість активних з’єднань, час відгуку);
- Врахування гео-положення вузлів – врахування показників завантаження ліній зв’язку, зниження часу відгуку;
- Можливості маштабування – можливість масштабування системи при балансуванні;
- Контентно залежний аналіз, кешування – балансування за видом контенту та кешування файлів;
- Забезпечення довгострокових сесій – можливість для користувача підтримувати сесію з кількох повторних запитів з одним і тим самим вузлом;
- Різні потужності обчислювальних вузлів – можливість балансування при неоднорідності обчислювальних ресурсів;
- Різні платформи обчислювальних вузлів – можливість балансування при неоднорідності обчислювальних платформ;
- Незалежність від типу мережі – незалежність від того, у локальній чи у глобальній мережі знаходяться обслуговуючі сервери;
- Автоматичне визначення і видалення непрацюючих вузлів – можливості моніторингу стану здоров'я обслуговуючих серверів та автоматичного видалення непрацюючих вузлів з пулу розподілу;
- Незалежність від прикладного протоколу – незалежність методу балансування навантаження від використовуваного протоколу прикладного рівня;
- Затрати ресурсів на балансування – чи витрачаються фізичні ресурси обслуговуючих серверів в процесі балансування навантаження.

Розшифровка позначень: + –є, - – немає, -/+ – частіше немає, чим є, +/- –

навпаки.

Таблиця 3.2 – Порівняння методів балансування навантаження

	Через єдиний пристрій балансування	NLB-кластери	Без єдиного пристрою балансування	2-й рівень OSI	3-й рівень OSI	Проксі 4-й рівень OSI	Проксі 7-й рівень OSI	HTTP Redirect 7-й рівень OSI
Ефективність розподілу навантаження	+	+	-	+	єдин / без єдин.	+	+	+
Показники завантаження вузлів (прямі,непрямі)	прямі, непрямі	прямі, непрямі	-, або непрямі	непрямі	-, рідко непрямі	прямі, непрямі	непрямі	-
Врахування гео-положення вузлів	+	-/+, можливо, у деяких реалізаціях	-/+	-	-/+	-	-	-
Можливості маштабування	-	+	++	+	єдин / без єдин.	-	-	++
Контентно залежний аналіз, кешування	+	-	-	-	-	+	+	+, але без кешування
Забезпечення довгострокових сесій	+	+	-	+	єдин / без єдин.	+	+	-
Різні потужності обчислювальних вузлів	+	+/-	+	+	єдин / без єдин.	+	+	+
Різні платформи обчислювальних вузлів	+	+/-	+	+	+	+	+	+
Незалежність від типу мережі	+	-/+	+	-	+	-	-	-
Автоматичне визначення і видалення непрацюючих вузлів	+	-/+ з НА	-/+	-	-/+рідко	+	+	–
Незалежність від прикладного протоколу	+/-	+/-	+	+	+	+	-	-
Затрати ресурсів на балансування	+	+	+	+	+	+	- -	++

4 СИСТЕМИ БАЛАНСУВАННЯ НАВАНТАЖЕННЯ «ХМАРНИХ» ДОДАТКІВ

4.1 Типові моделі "хмарних" додатків

Більшість веб-додатків сьогодні створюється з використанням триланкового (рис. 4.1) розподіленого підходу[13].

При доступі до ресурсів веб-додатку запит користувача спочатку обробляється DNS-сервером, який видає користувачеві IP-адресу одного з інтерфейсних серверів переднього плану (front-end). Користувач взаємодіє з одним з таких серверів, що забезпечують обробку статичного змісту і формування візуального результату виконання запиту, а також створення захищеного з'єднання. При обробці динамічних запитів, що вимагають обчислювальних дій чи запиту до даних, які динамічно формуються, виконується переадресація запиту на сервер додатків середнього плану (middle-end). Сервер додатків, в свою чергу, взаємодіє з серверами заднього плану, що відповідають за зберігання даних (back-end).

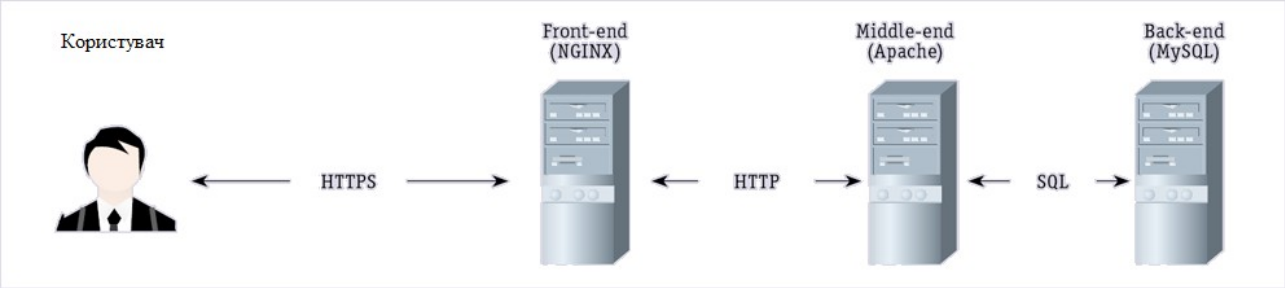

Рисунок 4.1 – Архітектура розподіленого додатка

Для побудови високонавантажених систем в якості зовнішніх серверів найчастіше використовуються сервери nginx, HAProxy, в якості серверів середнього плану - Apache, Microsoft IIS і т.п., а в якості внутрішніх серверів - рішення на основі MySQL, PostgreSQL Oracle, Microsoft SQL Server, і т. д.

Також ці рівні можна описати так:

1. Рівень представлення на стороні клієнта - відповідає за взаємодію додатка з користувачем - забезпечує графічний інтерфейс з додатком, відправку і прийом даних;
2. Рівень бізнес логіки - включає бізнес потоки і логіку, які управляють додатком, а також зовнішні процедури, які відповідають за формування представлення даних для клієнта та перевірку отриманих від нього даних;
3. Управління ресурсами - забезпечує засоби доступу до запитуваних даних і їх подання.

Для досягнення максимальної продуктивності серверів додатків і серверів переднього плану на них можуть розгортатися програмні засоби кешування та оптимізації зберігання статичних і часто використовуваних даних (Varnish Cache, Memcached). Для підвищення продуктивності та впровадження механізмів навантаженого резервування на кожному рівні виконується горизонтальне масштабування апаратних ресурсів за рахунок використання додаткового обладнання аж до створення інфраструктур на географічно рознесених майданчиках. Основна проблема, яка при цьому виникає, - це організація ефективного розподілу навантаження між усіма апаратними платформами, на яких функціонує розподілений додаток. Однак, незважаючи на розподіл обчислювального навантаження між серверними платформами та впровадження засобів кешування, продуктивність програми буде обмежена якимсь граничним значенням, залежним від ресурсів апаратної платформи. Крім того, при збільшенні числа елементів в будь-якій інформаційній системі з послідовною обробкою зростає ймовірність відмов будь-якого з них, що в кінцевому рахунку призводить до зниження загальної надійності програми.

Для рівня front-end балансування може виконуватись сервером DNS, що містить таблицю відповідності конкретного імені сайту і переліку його IP-адрес, відповідно до яких запити будуть потрапляти на різні front-end сервери

На рівні middle-end за балансування навантаження, як правило,

відповідає проксі-сервер, встановлений на рівні front-end, який дозволяє не тільки кешувати частина контенту і створювати ще один рівень безпеки системи, але і розподілити користувачів по серверам додатків. Наприклад, проксі-сервер nginx має в своєму арсеналі модуль upstream, який за допомогою алгоритму зваженого обслуговування розподіляє запити користувачів по різних серверах додатків.

Розподіл навантаження між серверними платформами кешуючих серверів переднього плану, серверів бізнес-логіки (middle-end) може проводитися також з використанням спеціалізованих апаратно-програмних комплексів, подібних продуктам NetScaler ADC від компанії Citrix, BIG-IP від компанії F5 Networks і т.п. спеціалізовані апаратно-програмні комплекси, аналізують непрямі показники завантаження серверних платформ,

На рівні back-end, як правило, балансування навантаження виконується засобами кластерів баз даних і ОС.

4.2 Використання методів балансування навантаження в "хмарних" додатках

На сьогоднішній день найпопулярніші методи розподілу навантаження - це циклічний (алгоритм round robin) і зважене обслуговування (алгоритм Weighted Fair Queuing)[11]. Циклічний метод найчастіше використовується для організації рівномірного розподілу навантаження між адресатами, що входять в один список. Кожен новий запит надходить до наступного одержувача за списком. При досягненні кінця списку відбувається повернення до першої позиції. Зважене обслуговування передбачає розподіл поступаючого навантаження пропорційно заданим «вагам» одержувачів. Ці алгоритми застосовуються зараз майже на всіх рівнях розподіленої архітектури; наприклад, на рівні front-end балансування виконується сервером DNS, що містить таблицю відповідності конкретної адреси (імені) сайту і IP-адреси. При вказівці декількох IP-адрес для одного імені між ними буде виконуватися

балансування за алгоритмом round robin з рівномірним розподілом навантаження. Для того щоб DNS-сервер реалізовував алгоритм зваженого обслуговування в рамках циклічного перебору, можна кілька разів вписати одне і те ж ім'я з однаковим IP-адресою, і тоді в межах одного циклу DNS-сервер видаватиме кілька разів однаковий IP-адресу. Таким чином, користувачі будуть потрапляти на різні front-end сервери пропорційно кількості згадувань IP-адреси даного сервера в конфігурації DNS.

На рівні middle-end за балансування навантаження, як правило, відповідає проксі-сервер, встановлений на рівні front-end, і дозволяє не тільки кешувати частина контенту і створювати ще один рівень безпеки системи, але і розподілити користувачів по серверам додатків. Наприклад, проксі-сервер nginx має в своєму арсеналі модуль upstream, який за допомогою алгоритму зваженого обслуговування розподіляє запити користувачів по різних серверах додатків. Крім вбудованих засобів DNS і nginx розподіл навантаження між серверними платформами кешуючих серверів переднього плану, серверів бізнес-логіки (middle-end) може проводитися також з використанням спеціалізованих апаратно-програмних комплексів, подібних продуктам NetScaler ADC від компанії Citrix, BIG-IP від компанії F5 Networks і т.п.

На рівні back-end, як правило, балансування навантаження виконується засобами кластерів баз даних і ОС.

Більшість з наявних рішень балансування (в тому числі і вбудовані засоби DNS / nginx) мають або статичні фіксовані пропорції розподілу навантаження, або, як спеціалізовані апаратно-програмні комплекси, аналізують непрямі показники завантаження серверних платформ, що може призводити до перекосів у завантаженні серверних платформ і , як наслідок, до неефективного використання наявних ресурсів. Такі перекоси виникають при використанні різнорідних апаратних платформ в межах одного рівня, виході з ладу платформ, появі завдань, додатковому навантаженні на процесор (архівування даних резервного копіювання) і т.п. Дані завдання не можуть ефективно відслідковуватися системами балансування і коригуватися наявними

інструментами в автоматичному режимі.

4.3 Розповсюджені системи балансування навантаження

4.3.1 Балансування в Google Compute Engine

Google Compute Engine пропонує балансування на стороні сервера, так що можна розподіляти вхідний мережевий трафік між декількома екземплярами віртуальних машин. Балансування навантаження забезпечує наступні переваги з балансуванням мережевого навантаження або балансування навантаження HTTP [14]:

- Масштабування додатків;
- Підтримка інтенсивного трафіку;
- Виявлення хворих примірників віртуальних машин;
- Балансування навантаження по регіонах;
- Маршрут трафіку до найближчої віртуальної машини;
- Підтримка маршрутизації на основі контенту.

Балансувальник Google Compute Engine являється програмно-апаратним балансувальником навантаження.

Балансування навантаження Google Compute Engine використовує об'єкти правил пересилання, які порівнюють і направляють певні типи трафіку на балансування. Наприклад, правило пересилання може відповідати за TCP трафік на порт 80 на публічній IP-адресі 192.0.2.1. Будь-який трафік, призначений для цього IP, протокол і порт, що відповідає цьому правилу пересилання буде спрямований на здорові екземпляри віртуальних машин на основі правил перевірки здоров'я, які визначає користувач.

Google пропонує два типи балансування навантаження, які відрізняються за можливостями, сценаріями використання і налаштуваннями. Нижче розглянемо сценарії, що можуть допомогти вирішити, мережеве чи HTTP балансування максимально задовольнить потреби.

Припустимо, що маємо веб-сайт на Apache і починаємо отримувати

настільки високий рівень трафіку і навантаження, що з'являється потреба додати додаткові екземпляри Apache, щоб допомогти опрацювати це навантаження. Можете додати додаткові примірники Google Compute Engine і налаштувати балансування для розподілу навантаження між цими екземплярами. У ціому випадку опрацьовується однаковий контент від кожного з примірників. По мірі того, як сайт стає все більш популярним, продовжується збільшення кількості примірників, які доступні, щоб опрацювати запити.

У цій ситуації, треба вибирати мережеве балансування навантаження для зіставлення вхідних запитів TCP / IP на порт 80 (правила пересилання) з цільовим пулом віртуальних машин в тому ж регіоні. Можна налаштувати об'єкт правил пересилання, цільовий пул, в якому перераховані екземпляри, що отримуватимуть трафік, і правила перевірки здоров'я. У цій ситуації, не має потреби в можливостях або більш складній конфігурації HTTP балансування навантаження.

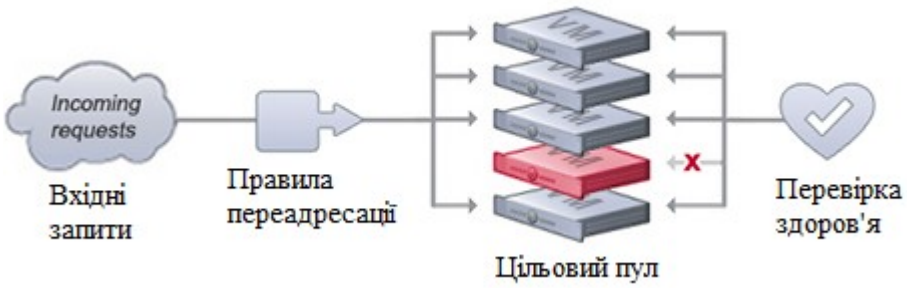

Рисунок 4.2 – Схема роботи мережевого балансування навантаження в Google Compute Engine

Мережеве балансування навантаження складається з операцій на основі протоколу. Цей тип балансування навантаження дозволяє збалансувати навантаження систем, заснованих на вхідних даних протоколу IP, таких як адреса, порт і тип протоколу.

Мережеве балансування навантаження використовує правила

пересилання, які вказують на цільові пули, які містять списки примірників, доступних для балансування навантаження і визначають, який тип перевірки повинен бути виконаний в цих випадках.

Мережеве балансування навантаження пропонує деякі варіанти балансування навантаження, які не доступні з балансуванням навантаження HTTP. Наприклад, ви можете балансувати додаткові протоколи на основі TCP/UDP, такі як SMTP-трафік. Якщо ваш додаток зацікавлений в характеристиках, пов'язаних з TCP-з'єднанням, балансування мережевого навантаження дозволяє вашому додатку перевірку пакетів, яку ви не можете зробити з балансуванням навантаження HTTP.

Також можна використовувати мережеве балансування навантаження для обробки HTTPS трафіку. Для цього потрібно:

- Створити правило брандмауера, яке дозволяє трафік через порт 443, і застосувати його до балансованих примірників.
- Керувати шифруванням / дешифруванням на екземплярах віртуальних машин.

За замовчуванням для розподілу трафіку на інстанси Google Compute Engine вибирає екземпляр, заснований на хеші вихідного IP і порта та IP і порта призначення. Вхідні з'єднання TCP розкидані по інстансах і кожне нове підключення може перейти до іншого інстансу. Всі пакети зі з'єднання спрямовані до одного інстансу, поки з'єднання не буде закрито.

Можна вибрати інший метод хешування, якщо потрібно більше прив'язки сеансу для того, щоб запити конкретного клієнта переходили до конкретного екземпляру віртуальної машини.

Правила пересилання працюють у поєднанні з цільовими пулами і цільовими інстансами для підтримки можливостей балансування навантаження і протоколу перенаправлення. Щоб використовувати балансування навантаження і протокол перенаправлення необхідно створити правила пересилання, які направляють трафік до конкретних цільових пулів (для балансування навантаження) або цільових інстансів (для протоколу

перенаправлення). Не можливо використовувати будь-яку з цих функцій без правил пересилання.

Ресурси правил пересилання знаходяться в колекції правил пересилання. Кожне правило пересилання відповідає окремій IP-адресі, протоколу і, додатково, діапазону портів для одного цільового пулу або цільового інстансу. Коли трафік надсилається на зовнішню IP-адресу, яка обслуговується правилом пересилання, воно спрямовує трафік до відповідного цільового басейну або цільового інстансу. Можна створити до 50 об'єктів правила пересилання на проект.

Цільовий пул визначає групу екземплярів, які повинні отримувати вхідний трафік від правил пересилання. Коли правила пересилання перенаправляють трафік до цільової пулу, Google Compute Engine вибирає інстанс з цих цільових пулів на основі хеш вихідного IP і порт та IP і порт призначення.

Цільові пули можуть бути використані тільки з правилами пересилання, які займаються TCP і UDP трафіком. Для всіх інших протоколів, треба створити цільовий пул, перш ніж ви можете використовувати його з правилами пересилання. Кожен проект може мати до 50 цільових пулів.

Якщо цільовий пул містить один примірник віртуальної машини, треба розглянути можливість використання протоколу перенаправлення.

Мережеве балансування навантаження підтримує Compute Engine Autoscaler, що дозволяє користувачам виконувати авто-масштабування по групах інстансів, наприклад, у цільовому пулі.

Сценарій мережевого балансування навантаження, розглянутий вище добре масштабується для одного регіону, але для розширення сервісів по регіонах, потрібно буде використовувати громіздкі і іноді проблемні рішення. При використанні HTTP балансування навантаження в цій ситуації, ви можете використовувати глобальну IP-адресу, яка є спеціальним IP, який може маршрутизувати користувачів на основі близькості. Можна збільшити продуктивність і надійність системи для глобальної бази користувачів,

встановлюючи просту топологію.

У цій ситуації необхідно визначити глобальні правила пересилання, які відображаються на цільовий HTTP проксі-сервер, який перенаправляє запити до найближчих інстансів в back-end службі. Об'єкти back-end послуг визначають групи інстансів, які здатні обробляти запити.

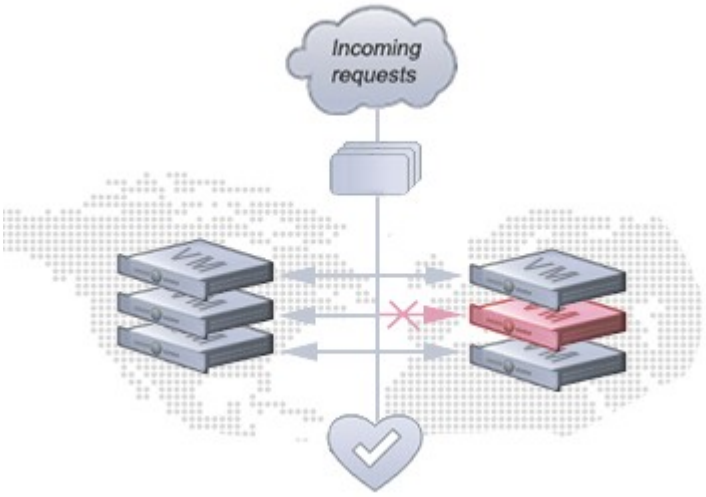

Рисунок 4.3 – Схема роботи міжрегіонального балансування навантаження в Google Compute Engine

Балансування навантаження на основі контенту (Content-based) використовує балансування навантаження HTTP для розподілу трафіку в різні інстанси на основі вхідного HTTP URI (Uniform Resource Identifier). Наприклад, є сайт з статичним контентом (CSS, зображення), динамічним контент і додаванням відео. Можна налаштувати конфігурацію балансування навантаження, щоб обслуговувати трафік з різних наборів інстансів, які оптимізовані для типу контенту, який вони обслуговують.

У цій ситуації, глобальні правила перенаправлення вказують на цільовий HTTP проксі, який перевіряє запити, щоб визначити, яка back-end служба підходить для запиту. Back-end служба потім розподіляє запит до одної зі своїх груп ресурсів, яка має один або кілька примірників віртуальних машин.

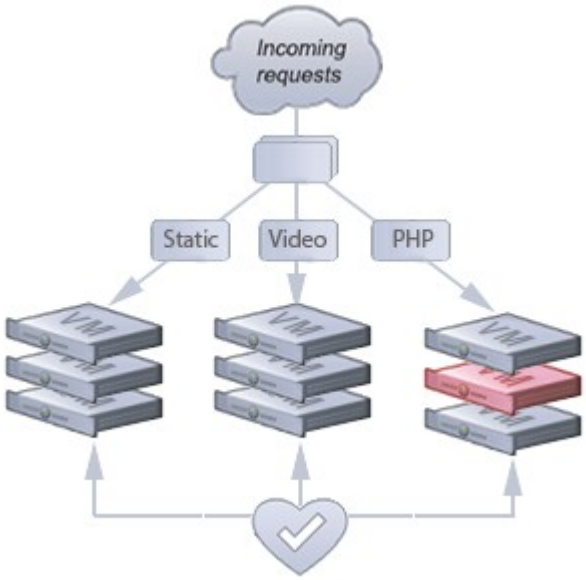

Рисунок 4.4 – Схема роботи балансування навантаження на основі контенту в Google Compute Engine

Балансування навантаження на основі контенту та міжрегіональне балансування можуть працювати разом за допомогою декількох back-end служб і кількох регіонів.

HTTP / HTTPS балансування навантаження забезпечує два способи визначення навантаження інстансу. В об'єкті backend служби, властивість balancingMode вибирає між запитами в секунду (RPS) та використанням процесора. Обидва режими дозволяють встановити максимальне значення; балансування навантаження HTTP буде прагнути до того, щоб навантаження залишається нижче межі, але короткі скачки вище межі можуть статися під час відновлення після збоїв або стрибків навантаження.

Вхідні запити надсилаються в регіон, найближчий до користувача, в якому залишилася ємність. Якщо більш ніж одна зона налаштована з ресурсів в регіоні, трафік розподіляється по групах ресурсів у кожній зоні відповідно до потужності кожної групи. У зоні, запити рівномірно розподіляються по інстансах в кожній групі.

Глобальні правила пересилання маршрутизують трафік по IP адресі,

порту і протоколу конфігурації балансувальника навантаження, що складається з URL карти, і однієї або декількох backend служб.

Глобальне правило пересилання забезпечує єдину глобальний IP-адресу, яка може бути використана в записах DNS для програми. Балансування навантаження на основі DNS не потрібно. Можна або вказати IP адресу, яка буде використовуватися або дозволити Google Compute Engine призначити її самостійно.

4.3.2 Балансувальник LoadMaster для Microsoft Azure

Віртуальний балансувальник LoadMaster для хмарного середовища Azure Cloud ADC - оптимізований для роботи в хмарі Microsoft Azure Cloud, LoadMaster для Azure забезпечує повноцінну балансування навантаження і доставку додатків на рівнях L4-L7 для робочих навантажень в хмарному середовищі Azure. Об'єкти LoadMaster, розміщені в хмарі Microsoft Azure і в середовищах локальної приватної хмари, працюють спільно для забезпечення безперервності доставки послуг на міжхмарному рівні [15].

Віртуальний балансувальник LoadMaster для Azure доповнює додатки, розміщені в інфраструктурі IaaS Microsoft Azure, надаючи повноцінний розподіл трафіку і безперебійність сеансу роботи на рівні L7, перевірку працездатності додатків і прискорення SSL. Додаткові послуги, такі як запобігання вторгнень, кешування і стиснення для опублікованих служб і здатність спільно використовувати окрему крайню точку для публікації множинних віртуальних служб оптимізує потік трафіку для додатків, розгорнутих в Microsoft Azure.

Всі віртуальні машини, створені в Azure можуть автоматично взаємодіяти, використовуючи приватний мережевий канал з іншими віртуальними машинами в одному хмарному сервісі або віртуальній мережі. Всі інші вхідні з'єднання, такі як трафік від інтернет-хостів і віртуальних машин в інших хмарних сервісах і віртуальних мережах, потребують кінцевої точки (endpoint).

Кінцеві точки можуть бути використані для різних цілей. Використання за замовчуванням і конфігурації кінцевих точок на віртуальній машині, що ви створюєте з Azure Management Portal призначені для протоколу віддаленого робочого столу (RDP) і трафіку дистанційних сеансів Windows PowerShell. Ці кінцеві точки дозволяють дистанційно керувати віртуальною машиною через Інтернет.

Ще одне застосування кінцевих точок - настройка балансування навантаження Azure для поширення певного типу трафіку між декількома віртуальними машинами або сервісами. Наприклад, ви можете розподілити навантаження трафіку веб-запиту між декількома веб-серверами і веб-ролями.

Кожній кінцевій точці визначеній для віртуальної машини присвоюється публічний і приватний порт, або TCP або UDP. Інтернет-хости відправляють їх вхідний трафік на публічну IP адресу хмарного сервісу і до публічного порту. Віртуальні машини і сервіси в рамках хмарного сервісу "слухають" приватну IP-адресу та приватний порт. Azure Load Balancer прокладає маршрут із публічної IP-адреси і номеру порту вхідного трафіку до приватної IP-адреси і номеру порту віртуальної машини, і навпаки для трафіку відповіді від віртуальної машини.

При налаштуванні балансування навантаження для трафіку між декількома віртуальними машинами або сервісами, Azure забезпечує випадковий розподіл вхідного трафіку.

Для хмарного сервісу, який містить екземпляри веб-ролей або ролей працівників, можна визначити публічну кінцеву точку у визначенні сервісу. Для хмарного сервісу, який містить віртуальні машини, ви можете додати кінцеву точку на віртуальній машині при її створенні або ви можете додати кінцеву точку пізніше.

Наступний малюнок показує кінцеву точку балансування навантаження для зашифрованого веб-трафіку, який є спільним для трьох віртуальних машин для публічного та приватного порту TCP 443 Ці три віртуальні машини перебувають в збалансованності.

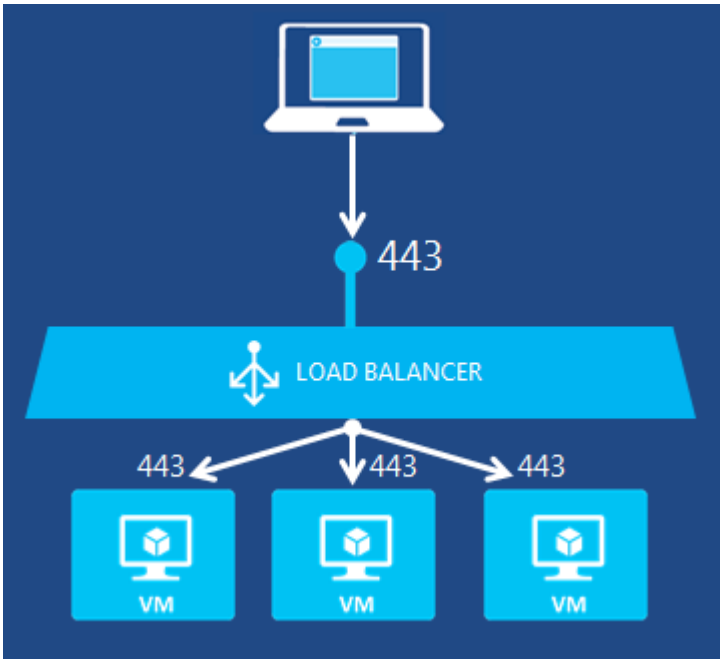

Рисунок 4.5 – Схема роботи Load Balancer

Коли Інтернет-клієнти відправляють запити до веб-сторінки на публічну IP адресу хмарної служби і TCP-порт 443, Azure Load Balancer виконує випадкове балансування цих запитів між трьома віртуальними машинами в наборі.

Azure також підтримує внутрішнє балансування навантаження трафіку між віртуальними машинами в межах хмарного сервісу, між віртуальними машинами в хмарних сервісах що містяться у віртуальній мережі, а також між локальними комп'ютерами і віртуальними машинами в віртуальній мережі.

Azure Load Balancer працює на рівні 4, транспортному рівні моделі OSI. Це означає, що вона працює на окремих потоках TCP або UDP трафіку, як це визначено в їх джерелі і IP-адресами призначення і номерами портів. Балансування навантаження на рівні 4 гарантує, що всі пакети для даного з'єднання TCP або UDP обміну повідомленнями направляються до однієї цілі.

Azure Load Balancer розподіляє навантаження серед декількох доступних серверів (віртуальних машин) шляхом обчислення хеш-функцій на трафіку, отриманого на заданій вхідній кінцевій точці. Azure Load Balancer використовує наступні поля з вхідного пакета для обчислення значення хеш:

- Вихідна IP-адреса;
- IP-адреса призначення;
- Тип протоколу (TCP або UDP);
- Вихідний порт;
- Порт призначення.

Azure Load Balancer потім використовує це значення для направлення трафіку на доступний сервер. Всі пакети з однієї TCP або UDP передачі, направлються на один сервер в наборі. Коли клієнт закриває і знову відкриває з'єднання або починає новий сеанс з того ж IP-адресу джерела, порт джерела, як правило, змінюється. Це створює нове й інше значення хеш та новий маршрут трафіку на доступний сервер.

Результат - випадковий розподіл трафіку по членам зі збалансованої множини. На підставі цього випадкового розподілу, можливо для різних з'єднань отримати направлення на один і той же сервер.

Azure Load Balancer підтримує конфігурацію тайм-ауту простою TCP, після якого Azure Load Balancer припиняє маршрутизацію для відкритого, але не діючого зв'язку TCP. Значення за замовчуванням тайм-ауту простою TCP становить 4 хвилини.

4.3.3 Балансування навантаження в Amazon EC2

Балансування навантаження і автомасштабування є дуже важливими функціями EC2. Ви можете створити правила при яких стане можливо автоматично збільшити кількість серверів, наприклад, якщо один або декілька серверів не справляються з навантаженням. Контроль за здоров'ям серверів веде ще один сервіс AWS - Amazon Cloud Watch. За допомогою цього сервісу можна створювати різного роду перевірки - checks - за допомогою яких контролюються найважливіші показники роботи ОС[16].

Балансувальник в Amazon EC2 - програмний балансувальник.

Функція Elastic Load Balancing автоматично розподіляє вхідний трафік

додатків між декількома інстансами Amazon EC2 в хмарі. Вона дає можливість досягати ще більшої відмовостійкості в додатках, «прозоро» виділяючи обсяг ресурсів, необхідний для розподілу навантаження згідно обсягами вхідного трафіку додатків.

Поділ інстансів Amazon EC2 по різних зонах доступності ще більше підвищує надійність додатків. Функція Elastic Load Balancing ще більше підвищує відмовостійкість і знижує необхідність людського втручання. Відмовостійкість можна підвищити шляхом розташування інстансів EC2 за балансувальником навантаження, так як він автоматично розподіляє трафік між декількома екземплярами EC2 і декількома зонами доступності і стежить за тим, щоб навантаження потрапляло тільки на здорові інстанси Amazon EC2. Elastic Load Balancer дозволяє перерозподіляти вхідний трафік додатків між екземплярами Amazon EC2 в одній або декількох зонах доступності. Elastic Load Balancing стежить за станом кожного підключеного екземпляра Amazon EC2. При погіршенні стану інстанси Amazon EC2 подача трафіку до нього припиняється, а його трафік перенаправляється на більш вільні інстанси. Якщо в одній зоні доступності немає жодного здорового інстанси EC2 і у вас є інстанси в інших зонах доступності, то Elastic Load Balancing виконає маршрутизацію трафіку на них. За відновленні стану цього інстанси на нього знову буде направлений трафік.

Можливості переносу сервісу DNS та моніторингу стану системи Amazon Route 53 ще більше підвищують доступність додатків, що працюють через балансувальник навантаження. Route 53 перехопить роботу балансувальника навантаження, якщо в наявності не залишилося здорових інстанси EC2 або якщо стан самого балансувальника критичний.

Перекидання сервісу DNS за допомогою Route 53 дозволяє підтримувати роботу додатків в різних регіонах AWS і призначати альтернативні балансувальники для перекидання сервісу в інші регіони. У разі відмови додатки Route 53 перенаправить запити з недоступного балансувальника навантаження на альтернативний в іншому регіоні.

Уявімо, що необхідно постійно мати як мінімум два здорових інстанси Amazon EC2 під управлінням Elastic Load Balancer. Виставимо ці умови в системі Auto Scaling, і при невиконанні умов необхідну кількість інстансів Amazon EC2 буде автоматично додано в групу Auto Scaling. Або ж, якщо необхідно, щоб при перевищенні будь-яким з інстанси Amazon EC2 затримки в 4 секунди протягом 15 хвилин додавався новий - досить встановити цю умову в настройках, і система Auto Scaling зробить все інше. Auto Scaling працює однаково добре з інстансами Amazon EC2, незалежно від того, чи використовується Elastic Load Balancing чи ні.

Elastic Load Balancing спрощує процес створення точки входу в VPC з мережі Інтернет для баласування навантаження між рівнями додатків. Можна визначити групи безпеки на ELB і контролювати порти і доступ до них з дозволених ресурсів. Так як Elastic Load Balancing прив'язується до VPC, всі наявні мережеві параметри доступу (ACL) і таблиці маршрутизації продовжують надавати додаткові елементи керування мережею.

При створенні балансувальника навантаження в VPC можна вибрати його тип: зовнішній (за замовчуванням) або внутрішній. При виборі внутрішнього доступ до нього буде здійснюватися без інтернет-шлюзу, а в відповідному записі DNS буде використовуватися його приватний IP.

Можна виділити такі переваги ціого балансувальника:

1. Доступність. Автоматична маршрутизація трафіку с допомогою Elastic Load Balancing між численними екземплярами і зонами доступності дозволяє досягти більш високого рівня відмовостійкості ваших додатків. Виключаючи нездорові інстанси Amazon EC2 і перерозподіляючи трафік додатків між рештою, Elastic Load Balancing забезпечує підтримку додатків тільки на здорових інстанси. Якщо в одній зоні доступності немає жодного здорового інстанси EC2 і у вас є інстанси в інших зонах доступності, то Elastic Load Balancing виконає маршрутизацію трафіку на них.

2. Гнучкість. Elastic Load Balancing автоматично масштабує потужності обробки запитів відповідно до трафіку додатків. Elastic Load

Balancing також підтримує інтеграцію з системою Auto Scaling, що автоматизує масштабування серверних ресурсів для будь-яких потоків трафіку без необхідності безпосереднього втручання.

3. Безпека. Elastic Load Balancing в сукупності з Amazon Virtual Private Cloud (VPC) забезпечують потужні мережеві функції і заходи безпеки. Ви можете створити внутрішній (без виходу в Інтернет) балансувальник навантаження для маршрутизації трафіку по приватних IP всередині закритої віртуальної мережі. Ви можете реалізувати багаторівневу архітектуру, використовуючи внутрішні і зовнішні (з виходом в Інтернет) балансувальники навантаження для маршрутизації трафіку між рівнями додатків. Багаторівнева архітектура інфраструктури додатків підтримує використання груп безпеки і приватних IP - публічні IP можна залишити тільки на тому рівні системи, через який проходить трафік. Elastic Load Balancing також підтримує інтегровану централізовану систему управління сертифікатами та SSL-шифруванням, що дозволяє легко змінювати налаштування шифрування на балансувальник навантаження і знімати високоскладних обчислювальні завдання з поточних інстанси.

4.3.4 Балансувальник навантаження HAProxy

HAProxy це безкоштовне, дуже швидке і надійне рішення, що пропонує високу доступність і балансування навантаження для TCP і HTTP-додатків, за допомогою розподілу вхідних запитів на кілька обслуговуючих серверів. Програма написана на C і має репутацію швидкого, ефективного (в плані використання процесора і оперативної пам'яті) і стабільного рішення[17].

HAProxy є проксі сервером прикладного рівня (7го) для протоколів HTTP і TCP. Відмітна здатність - вміння перевіряти і вбудовувати в сесію "cookie" для браузера користувача, яка згодом буде використана для сталості при направленні з'єднання на один з фронт або бекенда серверів з метою підтримки активних сесій з веб додатками. Вміє вирішувати на яку групу серверів буде направлений користувач залежно від будь-якого з параметрів

HTTP запиту. Здатний працювати з WebSockets (у вигляді TCP з'єднань).

На Лінуксі, може бути налаштований як "прозорий" проксі (передає запити на обслуговуючі сервера залишаючи вихідні адреси пакетів отриманих від користувачів без зміни на свою головну IP адресу як при стандартному або зворотному проксінгу). Перевірка пакетів на сьомому рівні дозволяє фільтрувати несанкціонований трафік і протоколи. Багатошарова архітектура, планувальника на рівні ядра, оптимізовані протоколи та алгоритми балансування дозволяють досягти пропускних спроможностей у кілька терабайт трафіку за день. Відмовостійкість одного проксі сервера забезпечується через використання демона keepalived перевіряючого працездатність самого проксі і синхронізуючись з резервним сервером (протокол VRRP), забезпечуючи дублювання в разі падіння першого. Також можлива паралельна робота обох для балансування, але в такій конфігурації потрібно зовнішній балансер або розкид по roud-robin DNS.

Проект не настільки популярний як LVS, але і не дуже від цього страждає. Пишаються стабільністю та безпекою коду. Встановлення та налаштування тривіальні.

HAProxy використовується в ряді високо-навантажених веб-сайтів, включаючи Твіттер, Інстаграм Github, Stack Overflow, Reddit, Tumblr і OpsWorks product з Amazon Web Services, W3C (W3C Validator), а також є складовою частиною хмарної платформи Red Hat OpenShift і балансувальник по -умолчанію в хмарної платформі OpenStack.

HAProxy є програмою з відкритим вихідним кодом і поширюється відповідно до GNU General Public License (GNU GPL v2).

До можливостей можна віднести:

- Періодична перевірка доступності обслуговуючих (back-end) серверів, на які перенаправляються запити користувачів;
- Декілька алгоритмів визначення доступності сервера: tcp-check, http-check, mysql-check;
- Балансування HTTP / TCP-запитів між "живими" серверами;

- Підтримка: IPv6 і UNIX sockets, HTTP 1.1 стиск (deflate, gzip), SSL-шифрування, повна підтримка HTTP keepalive;
- Веб-інтерфейс зі статистикою роботи програми.

4.3.5 Балансування навантаження з LVS

Linux Virtual Server (LVS) - широко поширений засіб управління кластерних систем для Linux систем. Цей вільний проект почав Wensong Zhang в травні 1998. Мета проекту - побудова високонадійних і високошвидкісного сервера з використанням кластерної технології, яка забезпечує хорошу масштабованість, надійність і працездатність[18].

Linux Virtual Server дозволяє створити кластер з фізичних (або віртуальних) серверів, що розподіляє навантаження між машинами в залежності від їх стану, пріоритету та інших параметрів настройки. Підтримувані протоколи: TCP, UDP. LVS є по суті комутатором протоколів 4го рівня. Компоненти перевірки прикладного стану з'єднань на більш високому рівні протоколів (наприклад по cookies в HTTP) не доопрацьовані.

Linux Virtual Server (LVS) - це набір інтегрованих програмних компонентів для розподілу навантаження між декількома реальними серверами. LVS працює на двох однаково налаштованих комп'ютерах: один з них є активним LVS- балансувальником навантаження, а другий-резервним LVS-балансувальником навантаження. Активний LVS-маршрутизатор виконує два завдання:

- Розподіл навантаження між реальними серверами.
- Перевірка працездатності сервісів на кожному реальному сервері.

Резервний LVS-маршрутизатор відстежує стан активного LVS-маршрутизатора і бере на себе функції останнього в разі виходу його з ладу.

Однією з переваг використання LVS є його здатність виконувати гнучке балансування навантаження рівня IP в пулі реальних серверів. Така гнучкість пояснюється наявністю великої кількості алгоритмів планувальника, які

адміністратор може вибрати на етапі налаштування LVS. Балансування навантаження в LVS значно перевершує такі методи, як Round-Robin DNS, коли ієрархічна структура DNS і кешування на клієнтській машині може привести до розбалансування навантаження. Також, низькорівнева фільтрація, використовувана в LVS- балансувальника навантаження, має ряд переваг перед перенаправленням запитів рівня додатків, так як балансування навантаження на рівні мережевих пакетів вимагає менших обчислювальних витрат і забезпечує кращу масштабованість, на відміну, наприклад, HAProxy або Nginx, які можуть використовуватися як балансувальник навантаження на 7-му рівні моделі OSI (рівні додатків)

Використовуючи планувальник, активний балансувальник навантаження може брати до уваги активність реального сервера, а також, опціонально, призначений адміністратором ваговий коефіцієнт при розподілі запитів до сервісу. Використання призначених вагових коефіцієнтів дозволяє призначити індивідуальним машинам довільні пріоритети. При використанні відповідного режиму планувальника стає можливим побудова пулу реальних серверів на базі різних апаратних і програмних конфігурацій. При цьому активний балансувальник навантаження може рівномірно розподіляти навантаження між реальними серверами.

Механізм планувальника LVS реалізований модулях IP Virtual Server або IPVS. Ці модулі реалізують комутацію на 4 рівні (L4), що дозволяє працювати з декількома серверами, використовуючи всього одну IP-адресу.

Для ефективного відстеження стану і маршрутизації пакетів IPVS будує в ядрі таблицю IPVS. Ця таблиця використовується активним балансувальником навантаження для перенаправлення запитів з адреси віртуального сервера на адреси реальних серверів в пулі, а також для перенаправлення відповідей реальних серверів клієнтам. Таблиця IPVS постійно оновлюється за допомогою утиліти ipvsadm, яка додає і прибирає реальні сервери залежно від їх доступності.

Основна термінологія в LVS наступна:

- Director - власне вузол, що здійснює маршрутизацію.
- Realserver - вузол ферми серверів.
- VIP або Virtual IP - IP віртуального (зібраного з реальних) сервера.
- Відповідно DIP і RIP - IP директора і реальних серверів.

Всі процеси зв'язку кластера з реальними серверами прозорі для кінцевого користувача - йому доступна тільки зовнішня IP адреса кластеру. Фізичні сервери можуть знаходиться як в одному сегменті мережі з самим сервером LVS (т.зв. Директором), так і бути віддалені навіть географічно і зв'язуватися різними методами: тунелювання трафіку всередині IP пакетів, пряма маршрутизація і NAT. У методі прямої маршрутизації фізичні сервери повертають пакети безпосередньо до кінцевого користувача щоб не завантажувати трафіком канал зв'язку кластера і сам сервер Директор, маскуючи при цьому вихідну адресу повернутих пакетів під головну адресу кластера (з метою збереження цілісності з'єднання і TCP сесій). Сервери, що надають сам контент не обов'язково повинні працювати на Лінуксі, є можливість роботи з BSD, Windows і Solaris. Розробки проекту LVS використовуються в інших проектах кластеризації, наприклад демон keepalived, Linux-HA, Ultra Monkey, Piranha (Red Hat Cluster Suite).

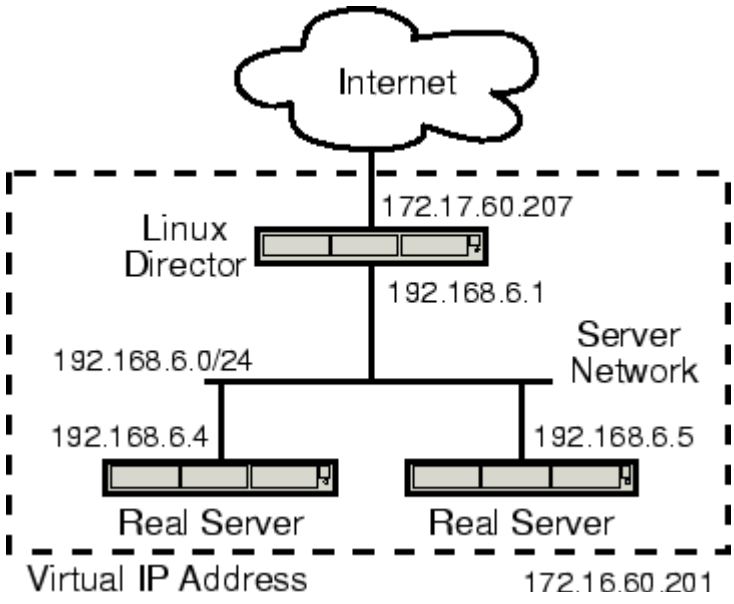

Рисунок 4.6 – Схема роботи LVS

Пакети за замовчуванням перенаправляються методом DR. Взагалі існують наступні варіанти маршрутизації:

- Direct Routing (gatewaying) - пакет направляється безпосередньо на ферму, без змін.
- NAT (masquarading) - механізм NAT.
- IPIP incapsulation (tunneling) - туннелювання.

DR є найбільш простим, але якщо, наприклад, потрібно поміняти порт призначення, то доведеться створювати правила в iptables. Механізм NAT же вимагає, щоб маршрут за замовчуванням у всієї ферми був направлений на директора, що не завжди зручно, особливо якщо реальні сервери мають і зовнішні адреси.

Планувальників в комплекті поставки декілька:

- Round Robin
- Weighted Round Robin
- Least Connection
- Weighted Least Connection

До всього іншого, можна повідомити, що сервіс вимагає persistence, тобто утримування користувача на одному з серверів протягом заданого проміжку часу - всі нові запити з одного і того ж IP будуть направлені на той же сервер.

Проект стабільний, досить простий в установці та налаштуванні, присутній у більшості репозиторіїв великих збірок Лінукса і добре документований. Код ядра проекту (IPVS) внесений до офіційного ядра Лінукса у версіях 2.4 і 2.6.

З великих компаній, що використовують LVS, можна відзначити: linux.com, Siemens (siemens.com), Wikimedia, Drupal (drupal.org), SourceForge (sourceforge.net), Zope (www.zope.org), Real Networks (www.real .com), Tiscali

Group, ibiblio.org, University of Nottingham та багато інших.

4.3.6 Балансування навантаження за допомогою Nginx

Nginx (engine x) - це HTTP-сервер і зворотний проксі-сервер, поштовий проксі-сервер, а також TCP проксі-сервер загального призначення, що спочатку був написаний Ігорем Сисоєвим[19].

Для балансування використовуються дві директиви Nginx:

- upstream - директива, яка поставляється з модулем HttpUpstream і дозволяє балансувати навантаження на декілька серверів;
- proxypass - директива, яка поставляється з модулем HttpProxy. Вона дозволяє коректно відправляти / проксирувати запити на сервери за балансувальником.

При створенні конфігураційного файла можна задати такі параметри:

- weight - визначає значимість сервера в системі. Якщо в системі 1 сервер значно потужніше інших - можна вказати для нього вище значення цього параметра і Nginx буде слати йому більше запитів, ніж іншим. Один із способів більш точно розподілити користувачів серверів - це встановити питому вагу (за кількістю клієнтів) для деяких машин. Nginx дозволяє встановити число, що визначає частку трафіку, яка повинна спрямовуватися на кожен сервер.
- max_fails - визначає кількість невдалих спроб з'єднання з backend сервером. Згідно налаштувань round robin за замовчуванням, Nginx буде продовжувати передавати дані на віртуальні виділені сервери, навіть якщо сервери не відповідають. Директива max_fails (максимальна кількість невдалих спроб) може автоматично запобігти таку поведінку, визнаючи що не відповідають сервери непрацездатними протягом певного часу.
- fail_timeout - проміжок між невдалими з'єднаннями.

Методи балансування навантаження описуються на початку секції upstream:

- ip_hash - відповідно до цього методу запити від одного і того ж клієнта будуть завжди вирушати на один і той же backend сервер на основі інформації про ip адресу клієнта. Не сумісний з параметром weight.
- least_conn - запити будуть відправлятися на сервер з найменшою кількістю активних сполук.
- round-robin - режим за замовчуванням. Тобто якщо не задано жодного з вищезгаданих способів балансування - запити будуть доставлятися по черзі на всі сервери в рівній мірі. Якщо при спробі роботи з сервером відбувається помилка, то запит передається наступного сервера, і так далі до тих пір, поки не будуть випробувані всі працюючі сервери. Якщо не вдасться отримати успішний відповідь від жодного з серверів, то клієнтові буде повернений результат роботи з останнім сервером.

Великою перевагою балансування навантаження за допомогою Nginx є підтримка SSL Termination. Тобто захищені сесії встановлюються з самим балансувальником і від нього ж клієнти отримують відповідь. Є два варіанти настройки:

1. Налаштовуємо балансування https сесій. При цьому ssl хости повинні бути описані і на Apache серверах. У цьому випадку https з'єднання будуть встановлюватися з Nginx, далі він формуватиме новий пакет, встановлювати захищене з'єднання з Apache серверами, отримувати такий же захищений пакет у відповідь, розпаковувати його, формувати новий і відправляти кінцевий результат клієнту в захищеному вигляді. Досить трудомісткий процес. Відносно продуктивності дуже ресурсо-витратний.

2. Налаштовуємо https хост в Nginx, описуємо http (без «S») з'єднання в upstream списку. Багато CMS систем мають вбудовані перевірки наявності безпечних сесій і примусово переадресовують клієнтів на безпечне з'єднання на сторінках введення конфіденційної інформації (авторизації користувачів, сторінки оплати послуг). Як правило це виробляється на основі наявності

HTTPS хедера зі значенням «on» у змінному оточенні SERVER. Можна просто додавати цей хедер в запити до серверів Apache і не створювати зайвого навантаження.

4.3.7 Балансування навантаження в CloudStack

CloudStack реалізує балансування навантаження транспортного рівня.

В CloudStack можна створювати правила балансування навантаження, що балансують трафік, отриманий на публічний IP на одну або декілька віртуальних машин. Користувач CloudStack або адміністратор може створювати правила балансування навантаження, що балансує трафік, отриманий на відкритому IP на одну або декілька віртуальних машин. Користувач створює правило, визначає алгоритм, і присвоює правило до множини віртуальних машин[20].

При створенні правила балансування можна задати такі параметри:

- Ім'я (Name): ім'я правила балансування навантаження.
- Публічний порт (Public Port): порт прийому вхідного трафіку який має балансуватися.
- Приватний Порт (Private Port): порт, який будуть використовувати віртуальні машини, щоб отримати трафік.
- Алгоритм (Algorithm): Алгоритм балансування.
- Липкість (Stickiness): (Необов'язково) Алгоритм для політики липкості.
- Автомасштабування (AutoScale): Налаштування конфігурації AutoScale.
- Перевірка здоров'я (Health Check): (Необов'язково; тільки для балансувальників навантаження NetScaler) Характеристики перевірки здоров'я:
    - Ping path (Необов'язково): Послідовність адрес, до яких слід відправити запити перевірки здоров'я. За замовчуванням: / (all).
    - Response time (Необов'язково): Як довго чекати відповіді від

перевірки здоров'я (2 - 60 секунд). За замовчуванням: 5 секунд.

- Interval time (Необов'язково): Кількість часу між перевірками здоров'я (1 секунда - 5 хвилин). Значення за замовчуванням встановлюється в глобальний параметр конфігурації lbrule_health check_time_interval.
- Healthy threshold (Необов'язково): Число послідовних успіхів перевірок здоров'я, які необхідні перш ніж оголосити екземпляр здоровим. За замовчуванням: 2.
- Unhealthy threshold (Необов'язково): Число послідовних невдач перевірки здоров'я, які необхідні, перш ніж оголосити екземпляр нездоровим. За замовчуванням: 10.

Балансувальник навантаження CloudStack підтримує Sticky сесії для правил балансування. Sticky сесії використовуються в веб-додатків для забезпечення постійної доступності інформації через запити сесії користувача.

Політика складається з імені методу, Sticky і параметрів. Метод Sticky може бути заданий балансуванням навантаженням за cooky або джерелом. У методі джерела IP джерела використовується для ідентифікації користувача і збереження даних користувача. В інших методах, використовуються cooky. Cooky створюються балансувальником навантаження або програмою, якій це потрібно, і користувачу передається URL-адреса, за якою балансувальник сам знає на який віртуальний сервер відправити користувача. Назва cooky може бути вказана адміністратором або автоматично. Різноманітність варіантів надає можливість контролю cooky, наприклад, як вони генеруються і, чи будуть вони кешуватися.

Липкі сесії використовуються в Web-додатках, щоб забезпечити постійну доступність інформації в рамках декількох запитів у сесії користувача. Наприклад, якщо покупець заповнює візок, ви повинні пам'ятати, що було придбано досі. Поняття «липкості» також називають збереження стану.

Будь-яке правило балансування навантаження, що визначається в CloudStack може мати політику липкості. Політика складається з імені методу,

липкості і параметрів. Параметри - пари ім'я-значення або прапори, які визначені постачальником балансування навантаження. Метод липкість може бути cookie, згенерованим балансувальником навантаження, додатком, або джерелом. У методі на основі джерела, IP-адреса джерела використовується для ідентифікації користувача і пошуку збережених даних користувача. В інших методах використовуються cookie. Cookie, згенеровані балансувальником навантаження або програмою включаються до запиту і відповіді. Назва cookie може бути вказана адміністратором або автоматично.

Система CloudStack також підтримує зовнішні балансувальники навантаження. В їх якості можна використовувати як комерційні балансувальники від Google, Microsoft, Amazon, Rackspace, NetScaler, так і відкриті, такі як HAProxy, LVS.

Вбудований балансувальник підтримує такі алгоритми балансування:

- LEASTCONNECTION - базується на тому, який сервіс в даний час має найменшу кількість клієнтських підключень. Це алгоритм балансування навантаження за замовчуванням.
- SOURCEIPHASH. З'єднання розподіляються внутрішнім серверами на базі вихідної IP-адреси. Поки всі сервери працюють з урахуванням IP-адреси клієнта запити завжди будуть йти до того ж веб-сервера.
- ROUNDROBIN. Бере адресу у верхній частині списку. Після вибору адреси для зв'язку, вона переміщається в низ списку.

CloudStack також підтримує автомасштабування (AutoScaling). AutoScaling дозволяє масштабувати back-end служби або програмні ВМ вгору або вниз легко і автоматично відповідно до певних умов. При включенні AutoScaling, можна гарантувати, що кількість віртуальних машин, які використовуються, будуть масштабуватися вгору при підвищенні попиту, і автоматично вниз при спаді попит. Таким чином, це допоможе заощадити обчислювальні витрати за рахунок зупинки віртуальних машин, які недостатньо використовуються, автоматично і запуску нових віртуальних машин, коли є

потреба в них, без необхідності ручного втручання[21].

NetScaler AutoScaling призначений для запуску або припинення віртуальних машин на основі умов. Умови, які викликають scaleup або scaledown, можуть варіюватися від простого моніторингу використання ЦП сервера до моніторингу поєднання різних факторів. Наприклад, можна налаштувати автоматичне масштабування так, щоб додаткова ВМ запускалася коли завантаження процесора перевищує 80 відсотків протягом 15 хвилин, або так, щоб ВМ видалялась, коли завантаження процесора становить менше 20 відсотків протягом 30 хвилин.

CloudStack використовує балансувальник навантаження NetScaler, щоб контролювати всі аспекти здоров'я системи і працювати в унісон з CloudStack, щоб ініціювати масштабування вгору або вниз.

Для конфігурації автомасштабування необхідно вказати наступне:

- Шаблон: Шаблон складається з базового зображення ОС і додатку. Шаблон використовується для надання нового екземпляра додатка при масштабуванні вгору. Коли ВМ розгортається з шаблону, віртуальна машина може почати приймати трафік від балансувальника навантаження без будь-якого втручання адміністратора.
- Обчислювальний ресурс: визначений набір віртуальних атрибутів обладнання, включаючи швидкість процесора, кількість процесорів і розмір оперативної пам'яті, що користувач може вибрати при створенні нового екземпляра віртуальної машини. Виберіть одну з обчислювальних пропозицій, які будуть використовуватися під час ініціалізації екземпляра ВМ в рамках масштабування вгору.
- Мінімум екземплярів: мінімальна кількість активних екземплярів ВМ, які присвоюєні правилу балансування навантаження. Активними екземплярами ВМ є екземпляри додатків, які обслуговують трафік, і в даний час балансуються. Цей параметр гарантує, що правило балансування навантаження має принаймні вказане число активних примірників ВМ, щоб обслуговувати трафік.

- Максимум екземплярів: Максимальна кількість активних екземплярів ВМ, які повинні бути приписані правилу балансування навантаження. Цей параметр визначає верхню межу активних екземплярів ВМ, які можуть бути віднесені до правила балансування навантаження.

Є такі налаштування політики масштабування вгору або вниз:

- Тривалість: тривалість, в секундах, протягом якого визначені умови повинні бути правдою, щоб викликати масштабування.
- Лічильник: Лічильники продуктивності показують стан контрольованих параметрів. За замовчуванням, CloudStack пропонує чотири лічильники продуктивності: три лічильники SNMP (Simple Network Management Protocol) і один NetScaler лічильник. Лічильники SNMP - це процесор користувача Linux, процесор системи Linux і Linux CPU Idle. NetScaler лічильник - ResponseTime. Адміністратор можете додати додаткові лічильники в CloudStack за допомогою API CloudStack.
- Оператор: Наступні п'ять реляційних операторів підтримуються у функції автоматичного масштабування: більше, менше, менше або дорівнює, більше або дорівнює, і дорівнює.
- Поріг: Граничне значення для лічильника. Після того, як лічильник порушує порогове значення, функція AutoScale ініціює масштабування.
- Додати: Натисніть кнопку "Додати", щоб додати умову.

Крім того, якщо необхідно налаштувати додаткові параметри, треба натиснути кнопку "Показати додаткові налаштування", і вказати наступне:

- Інтервал опитування: Частота оцінки лічильників, за якої вони опитуються перш ніж масштабувати. Інтервал опитування за замовчуванням становить 30 секунд.
- Тиха година: Період охолодження після ініціювання масштабування. Цей час включає в себе час, необхідний для завершення ініціалізації екземпляра ВМ з її шаблоном і час, витрачений додатками

для підготовки до обслуговування трафік. Цей час дозволяє перейти в стабільний стан, перш ніж будь-яка дія може мати місце. За замовчуванням становить 300 секунд.

- Період знищення ВМ: час в секундах після масштабування вниз, для очікування перш ніж ВМ буде знищена. Це необхідно для забезпечення закінчення сесій або транзакцій, що обслуговуються в ВМ і її безпечного знищення. За замовчуванням становить 120 секунд.
- Групи безпеки: групи безпеки забезпечують можливість ізолювати трафік екземплярів ВМ. Група безпеки - рупа віртуальних машин, що фільтрує вхідний і вихідний трафік відповідно до набору правил, названих правилами входу і виходу. Ці правила фільтрації мережевого трафіку відповідно до IP-адреси, яка намагається спілкуватися з ВМ.
- Дискові пропозиції: визначений набір розміру диска для зберігання первинних даних.
- SNMP Community: рядок SNMP, який буде використовуватися пристроями NetScaler для запросу заданих значень лічильника з підготовлених екземплярів VM. За замовчуванням є відкритою.
- Порт SNMP: Номер порту, на якому агент SNMP, який працює на підготовлених віртуальних машинах, слухає. За замовчуванням порт 161.

Якщо треба виконати будь-яку операцію з технічного обслуговування екземплярів ВМ, треба відключити конфігурацію AutoScale. При відключеній конфігурації AutoScale, масштабування не відбувається. Можна використовувати цей простой для робіт з технічного обслуговування. Щоб відключити конфігурацію AutoScale, треба натиснути кнопку "Відключити AutoScale". Після операції з технічного обслуговування можна включити назад конфігурацію AutoScale. Щоб включити треба відкрийте сторінку конфігурації AutoScale, а потім натиснути кнопку "Включити AutoScale". Існує можливість оновити різні параметри і додати або видалити умови в правила.

4.3.8 Порівняльна таблиця розповсюджених систем балансування навантаження

Розглянута інформація про системи балансування навантаження додатків в "хмарному середовищі" дозволяє створити порівняльну таблицю. Порівняння систем балансування навантаження за основними критеріями наведено в таблиці 4.1. Критерії, наведені в таблиці:

- Ефективність розподілу навантаження – рівномірність розподілу при різних умовах (однотипні запити, різні запити, сесії, т.д.);
- Врахування гео-положення вузлів – врахування показників завантаження ліній зв’язку, зниження часу відгуку;
- Можливості автомаштабування – можливість автоматичного масштабування в системі при балансуванні;
- Контентно залежний аналіз, кешування – балансування за видом контенту та кешування файлів;
- Забезпечення довгострокових сесій – можливість для користувача підтримувати сесію з кількох повторних запитів з одним і тим самим вузлом;
- Незалежність від типу мережі – незалежність від того, у локальній чи у глобальній мережі знаходяться обслуговуючі сервери;
- Автоматичне визначення і видалення непрацюючих вузлів – можливості моніторингу стану здоров'я обслуговуючих серверів та автоматичного видалення непрацюючих вузлів з пулу розподілу;
- Незалежність від прикладного протоколу – незалежність методу балансування навантаження від використовуваного протоколу прикладного рівня;

Таблиця 4.1 – Порівняльні характеристики розповсюджених систем балансування навантаження

	Google Compute Engine	Microsoft Azure LoadMaster	Amazon EC2 Elastic Load Balancing	HAProxy	LVS	Nginx	CloudStack
Ефективність розподілу навантаження	+	-+	+	+	+	+	+
Врахування гео-положення вузлів	+	-	+	-	-	-	-
Можливості автомасштабування	+	+	+	-	-	-	+
Контентно залежне балансування, аналіз, кешування	+	+	+	+	+	+	+
Забезпечення довгострокових сесій	+	+	+	+	+	+	+
Незалежність від типу мережі	+	+	+	+	+	-	-
Автоматичне визначення і видалення непрацюючих вузлів	+	+	+	+	-	+	+
Незалежність від прикладного протоколу	+	+	+	+	+	-	+

4.4 Висновки

Більшість веб-додатків сьогодні створюється з використанням триланкового розподіленого підходу. При доступі до ресурсів веб-додатки запит користувача спочатку обробляється DNS-сервером, який видає користувачеві IP-адресу одного з інтерфейсних серверів переднього плану (front-end). Користувач взаємодіє з одним з таких серверів, що забезпечують обробку статичного змісту і формування візуального результату виконання запиту, а також створення захищеного з'єднання. При обробці динамічних запитів, що вимагають обчислювальних дій чи запиту до динамічно формується

даними, виконується переадресація запиту на сервер додатків середнього плану. Сервер додатків, в свою чергу, взаємодіє з серверами заднього плану, що відповідають за зберігання даних (back-end).

Алгоритми Destination Hash Scheduling та Source Hash Scheduling використовуються в "хмарному" сервісі Google Compute Engine. Мережеве балансування навантаження в ньому підтримує Compute Engine Autoscaler, що дозволяє користувачам виконувати авто-масштабування по групах інстансів, наприклад, у цільовому пулі. Мережеве балансування навантаження Google Compute Engine добре масштабується для одного регіону.

Для балансування навантаження територіально розподіленої інфраструктури в Google Compute Engine можна використовувати HTTP балансування навантаження, в якому за рахунок використання глобальної IP-адреси та використання алгоритмів Destination Hash Scheduling та Source Hash Scheduling можна маршрутизувати користувачів на основі близькості.

Також Google Compute Engine пропонує балансування навантаження на основі контенту, що є дуже зручним при необхідності розподілення вхідного трафіку по набору інстансів, які оптимізовані для типу контенту, який вони обслуговують.

Azure Load Balancer Azure Load Balancer доцільно використовувати при комплексному балансуванні продуктів Microsoft через підтримку та взаємну вбудованість сервісів.

"Хмарний" балансувальник Elastic Load Balancing, який використовується в Amazon EC2 є найбільш функціональним та надійним з розглянутих балансувальників в ньму є все те, що зазвичай вимагається від балансувальника: автомасштабування, відмовостійкість, контроль за "здоров'ям" інстансів, простота використання, гнучкість, безпека, міграція, явні параметри завантаження. За рахунок великого рівня автоматизації, людське втручання в роботу балансувальника зведено до мінімуму.

HAProxy це безкоштовне, дуже швидке і надійне рішення, що пропонує високу доступність і балансування навантаження для TCP і HTTP-додатків, за

допомогою розподілу вхідних запитів на кілька обслуговуючих серверів. Програма написана на C і має репутацію швидкого, ефективного (в плані використання процесора і оперативної пам'яті) і стабільного рішення.

Linux Virtual Server (LVS) - широко поширений засіб управління кластерних систем для Linux систем. Linux Virtual Server дозволяє створити кластер з фізичних (або віртуальних) серверів, що розподіляє навантаження між машинами в залежності від їх стану, пріоритету та інших параметрів настройки. Підтримувані протоколи: TCP, UDP. LVS є по суті комутатором протоколів 4го рівня.

Балансувальники HAProxy та Linux Virtual Server доцільно використовувати при побудові власної "хмари" з "нуля". Це потужні, функціональні та гнучкі системи з відкритим кодом, які можна використовувати для балансування навантаження в "хмарному середовищі".

Nginx (engine x) - це HTTP-сервер і зворотний проксі-сервер, поштовий проксі-сервер, а також TCP проксі-сервер загального призначення. Великою перевагою балансування навантаження за допомогою Nginx є підтримка SSL Termination. Використовувати доцільно при необхідності детальної настройки всіх, навіть найдрібніших аспектів балансування. Але для TCP балансування більше рекомендується HAProxy, а Nginx - для HTTP балансування.

CloudStack реалізує балансування навантаження TCP рівня, яке підтримує такі алгоритми або політику: Round-robin, Least Connection, and Source IP. Система CloudStack також підтримує зовнішні балансувальники навантаження. В їх якості можна використовувати як комерційні балансувальники від Google, Microsoft, Amazon, Rackspace, NetScaler, так і відкриті, такі як HAProxy, LVS. Це досить потужна та надійна платформа для побудови "хмар". Її балансувальник включає досить повний набір налаштувань, що робить його одним з кращих при побудові власної "хмари".

5 НАЛАШТУВАННЯ БАЛАНСУВАННЯ НАВАНТАЖЕННЯ В СИСТЕМІ CLOUDSTACK ДЛЯ ДОДАТКУ ДИСТАНЦІЙНОГО ВІДЕО НАВЧАННЯ

5.1 Створення віртуальної машини з BigBlueButton

Можливості балансування навантаження системи CloudStack було вирішено тестувати на прикладі віртуальної машини з системою дистанційного навчання BigBlueButton.

BigBlueButton - відкрите програмне забезпечення для проведення веб-конференції. Система розроблена в першу чергу для дистанційного навчання. Назва BigBlueButton походить від початкової концепції, що, початок веб-конференції повинно бути максимально простим, як натискання метафоричної великої синьої кнопки. [22]

BigBlueButton підтримує наявність декількох аудіодоріжок і обмін відео, можливість показу презентацій, документів Microsoft Office і OpenOffice, зображень, PDF документів. Так само підтримуються розширені можливості дошки - такі, як покажчик, масштабування і малювання, доступ до робочого столу. Для зворотного зв'язку зі слухачами веб-конференції існують публічні та приватні чати. Інтегрована VoIP на базі FreeSWITCH. Крім того, користувач може увійти в конференцію або як глядач або як модератор. Як глядач, користувач може приєднатися до голосової конференції, використовувати web-камеру, підняти руку (попросити слово), і спілкуватися з іншими людьми. В якості модератора, користувач має можливість відключити / включити мікрофон будь-якого глядача, видалити будь-якого глядача з веб-конференції, а так само передати слово будь-якому глядачеві для виступу (зробити будь-якого користувача провідним). Ведучий може завантажувати презентації, документи, використовувати дошку. Хоча компоненти мають відкритий вихідний код, клієнт BigBlueButton залежить від розширення для браузера для Adobe Flash

платформи. Сервер BigBlueButton працює на Ubuntu 14.04 64-бітної версії і може бути встановлений як з вихідного коду, так і з пакетів Ubuntu. Сервер BigBlueButton також може працювати і в хмарному середовищі, як Amazon EC2, при його установці на Ubuntu 14.04 64-бітної версії.

Перейдемо до встановлення системи BigBlueButton.

Спочатку необхідно створити шаблон віртуальної машини для нашого сервера BigBlueButton. Для його створення в користувацькому інтерфейсі CloudStack вибираємо вкладку "Шаблони" та натискаємо кнопку "Регістрація шаблону".

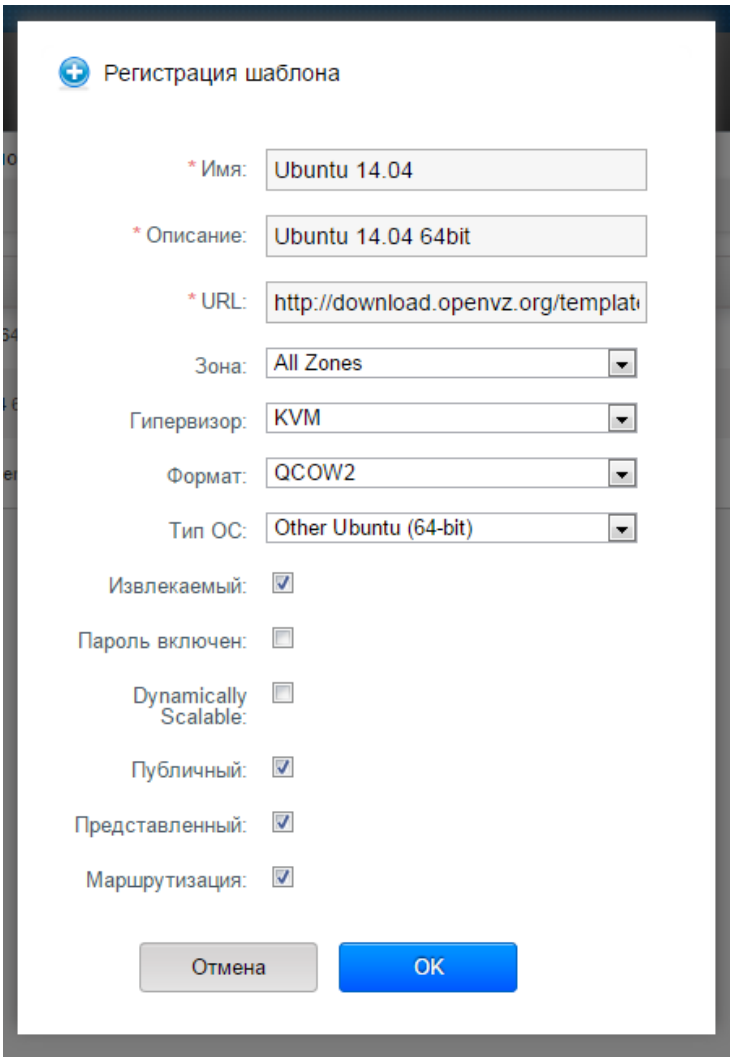

Рисунок 5.1 – Вікно регістрації шаблону CloudStack

Заповняємо поля, які можна побачити на рис. 5.1, натискаємо кнопку

"OK" та чекаємо поки шаблон скачається та встановиться. Коли шаблон буде готовий до використання, ми побачимо його у списку доступних, що показано на рис. 5.2.

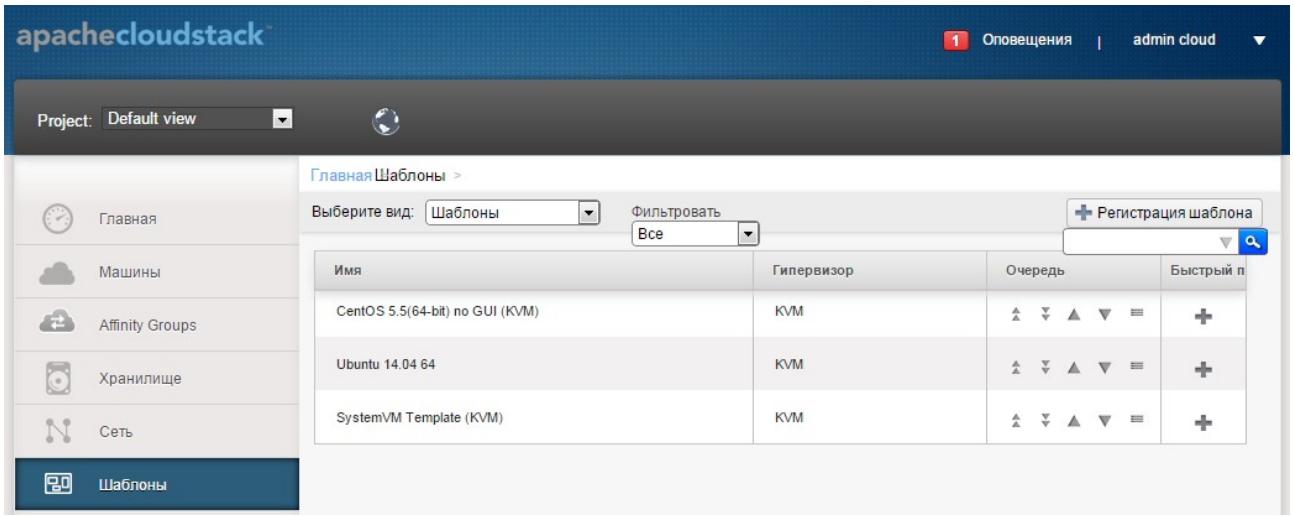

Рисунок 5.2 – Вкладка з доступними шаблонами CloudStack

Далі необхідно створити віртуальну машину з нашого шаблону. Для цього переходимо на вкладку "Машини" та натискаємо кнопку "Додати машини". Вибираємо такі налаштування для віртуальної машини (рис. 5.3):

Зона - Zone1;

Використовувати шаблон чи ISO - Шаблон;

Шаблон - Ubuntu 14.04 64;

Обчислювальний ресурс - Medium Instance (2 GHz, 2 GB)

Дисковий ресурс - Custom (40 GB)

Мережа - default

Ім'я - BigBlueButton

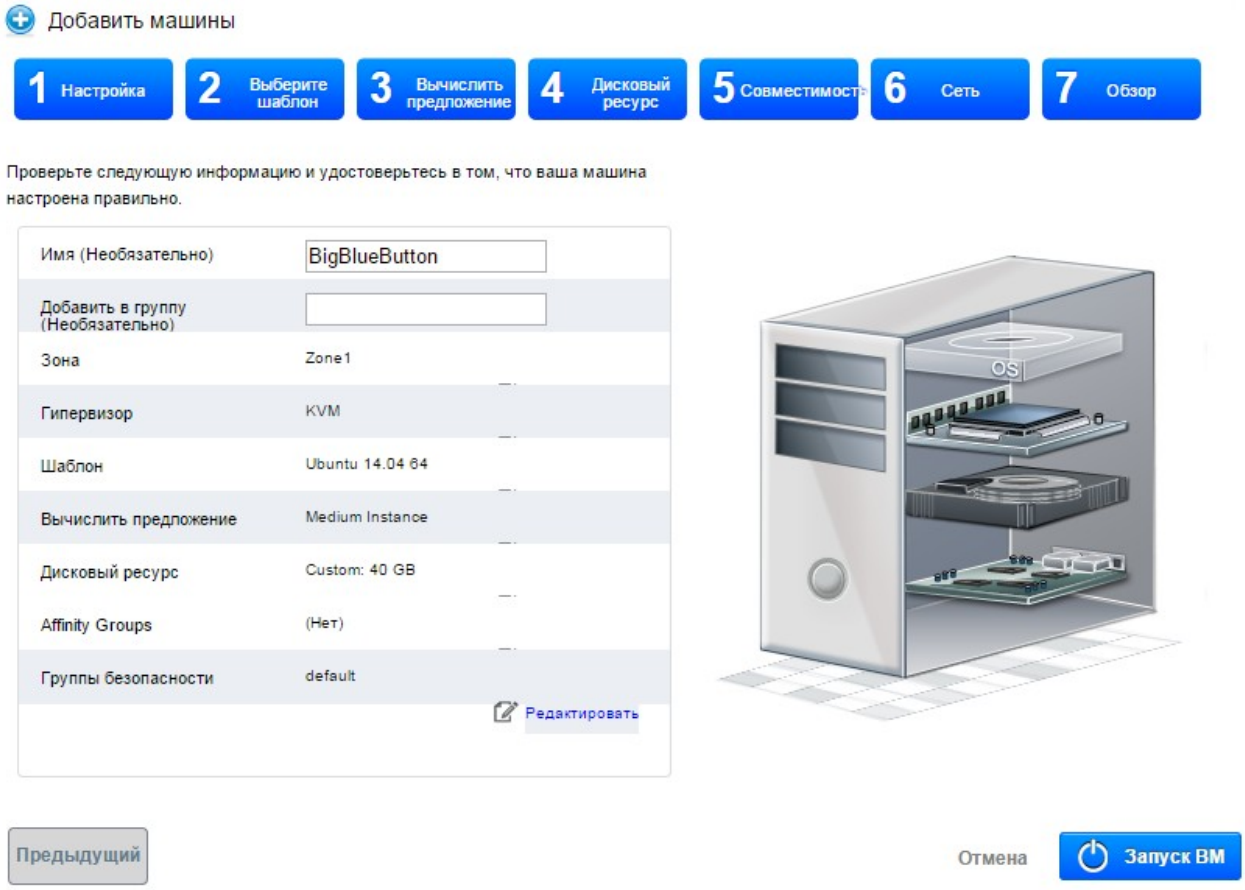

Рисунок 5.3 – Вікно створення віртуальної машини CloudStack

Натискаємо кнопку "Запуск ВМ" та очікуємо створення та встановлення ВМ.

Після встановлення та налаштування ВМ можна переходити до встановлення BigBlueButton.

Перевірка перед установкою складається з наступного:

Локаль сервера повинна бути en_US.UTF-8. Перевірити можна командою:

cat /etc/default/locale

Якщо вивід відрізняється від такого: LANG = "en_US.UTF-8"

То потрібно зробити наступне:

sudo apt-get install language-pack-en sudo update-locale LANG=en_US.UTF-8

Після цього, перезаходимо, і перевіряємо ще раз cat / etc / default / locale.

Перевіряємо чи 64-бітний сервер:

uname -m x86_64

Також потрібно переконається, що версія Ubuntu 14.04. На іншій версії BBB працювати не буде.

cat /etc/lsb-release

DISTRIB_ID=Ubuntu

DISTRIB_RELEASE=14.04

DISTRIB_CODENAME=trusty

DISTRIB_DESCRIPTION="Ubuntu 14.04.1 LTS"

Для встановлення BigBlueButton 0.9.0 робимо наступні кроки:

1. Оновлення сервера

Потрібно переконатися в тому, що multiverse репозиторій підключений:

grep "multiverse" /etc/apt/sources.list

Необхідно переконатися, що зазначений нижче рядок не за закоментований і взагалі є.

deb http://ru.archive.ubuntu.com/ubuntu/ trusty multiverse

Далі, оновлюємо сервер:

sudo apt-get update && sudo apt-get dist-upgrade

Після оновлення перезавантажуємо його.

2. Додаємо репозиторій BigBlueButton

Додаємо ключ репозиторія, сам репозиторій і оновлюємо список пакетів.

wget http://ubuntu.bigbluebutton.org/bigbluebutton.asc -O- | sudo apt-key add -

echo "deb http://ubuntu.bigbluebutton.org/trusty-090/ bigbluebutton-trusty main" | sudo tee /etc/apt/sources.list.d/bigbluebutton.list

sudo apt-get update

3. Встановлюємо ffmpeg

BigBlueButton 0.9.0-beta використовує ffmpeg для запису і відтворення трансляцій.

FFmpeg — це комплекс вільних комп'ютерних програм та програмних бібліотек для маніпуляцій з цифровими відео- та аудіо-матеріалами — запис, конвертація та пакування у різні формати контейнерів. Він включає libavcodec - бібліотеку кодування і декодування аудіо і відео і libavformat - бібліотеку мультиплексування і демультиплексування в медіаконтейнера. Назва походить від назви експертної групи MPEG і FF, що означає fast forward.

FFmpeg розроблений під ОС на основі Linux, однак може бути скомпільований під багато інші операційні системи. Поширюється під ліцензіями GNU LGPL або GNU GPL.

Для того, щоб встановити ffmpeg, потрібно створити файл install-ffmpeg.sh:

nano install-ffmpeg.sh

і вставити в нього наступний код:

sudo apt-get install build-essential git-core checkinstall yasm texi2html libvorbis-dev libx11-dev libvpx-dev libxfixes-dev zlib1g-dev pkg-config netcat libncurses5-dev

FFMPEG_VERSION = 2.3.3

cd / usr / local / src

if [! -d "/ usr / local / src / ffmpeg - $ {FFMPEG_VERSION}"]; then

sudo wget "http://ffmpeg.org/releases/ffmpeg-$ {FFMPEG_VERSION}.tar.bz2"

sudo tar -xjf "ffmpeg - $ {FFMPEG_VERSION} .tar.bz2"

fi

cd "ffmpeg - $ {FFMPEG_VERSION}"

sudo ./configure --enable-version3 --enable-postproc --enable-libvorbis --enable-libvpx

sudo make

sudo checkinstall --pkgname = ffmpeg --pkgversion = "5: $ {FFMPEG_VERSION}" --backup = no --deldoc = yes --default

Зберігаємо, і виконуємо наступні команди, встановлюємо флаг запуску на файл, і запускаємо його:

chmod + x install-ffmpeg.sh

./install-ffmpeg.sh

Після цього потрібно переконатися встановився чи ffmpeg, для чого запускаємо команду ffmpeg -version, вивід повинен бути приблизно таким:

ffmpeg -version

ffmpeg version 2.3.3 Copyright (c) 2000-2014 the FFmpeg developers

built on Feb 8 2015 13:38:27 with gcc 4.8 (Ubuntu 4.8.2-19ubuntu1)

configuration: --enable-version3 --enable-postproc --enable-libvorbis --enable-libvpx

libavutil 52. 92.100 / 52. 92.100

libavcodec 55. 69.100 / 55. 69.100

libavformat 55. 48.100 / 55. 48.100

libavdevice 55. 13.102 / 55. 13.102

libavfilter 4. 11.100 / 4. 11.100

libswscale 2. 6.100 / 2. 6.100

libswresample 0. 19.100 / 0. 19.100

4. Встановлення BBB

Для установки самого BBB потрібно виконати наступну команду:

sudo apt-get update

sudo apt-get install bigbluebutton

Це мета-пакет за допомогою якого встановляться всі компоненти BBB і всі залежності.

Якщо з'явилося таке повідомлення про помилку:

..... Error: FreeSWITCH did not start

Її можна проігнорувати. Запустимо його на 7 кроці.

Якщо з'явилося таке повідомлення про помилку:

Setting up bbb-playback-presentation (0.9.0-1ubuntu5) ...

chown: invalid user: 'tomcat7: tomcat7'

То потрібно повторно запустити sudo apt-get install bigbluebutton, і установка повинна буде пройти без помилок.

Також в кінці установки з'явилося повідомлення про помилку:

Errors were encountered while processing:

bbb-record-core

bbb-playback-presentation

bbb-config

bigbluebutton

E: Sub-process / usr / bin / dpkg returned an error code (1)

Повторно запускаємо sudo apt-get install bigbluebutton щоб установка завершилася коректно.

5. Встановлення API Demos

Щоб протестувати BBB сервер, можна встановити пакет bbb-demo, с допомогою якого можна подивитися можливості надані BBB API.

sudo apt-get install bbb-demo

6. Включення WebRTC

Годинник реального часу (англ. Real Time Clock, RTC) - електронна схема, призначена для обліку хронометричних даних (поточний час, дата, день тижня та ін.), Являє собою систему з автономного джерела живлення і враховує пристрою. Найчастіше годинник реального часу зустрічаються в обчислювальних машинах, хоча насправді RTC присутній практично у всіх електронних пристроях, які повинні зберігати час.

WebRTC (англ. Real-time communications - комунікації в реальному часі) - проект з відкритим вихідним кодом, призначений для організації передачі потокових даних між браузерами або іншими підтримуючими його додатками за технологією точка-точка.

Застосування технології виходить за рамки peer-to-peer між браузерами і широко використовується на серверній стороні, головним чином для того щоб забезпечити сумісність з іншими сигнальними і комунікаційними протоколами і кодеками. За допомогою сервера можна організовувати WebRTC трансляції не тільки з браузерів, а й зі стаціонарних IP-камер, що використовують протокол RTSP / RTP і відеокодек H.264.

Щоб включити WebRTC audio, потрібно виконати команду:

sudo bbb-conf --enablewebrtc

7. Перезавантаження BBB

Для перезавантаження BBB потрібно виконати

sudo bbb-conf --clean

sudo bbb-conf --check

--clean -очищає всі log файли BBB;

--check -просматрівает все log файли на предмет помилок;

Також sudo bbb-conf --check виводить поточні настройки BBB.

На даному уже можна спробувати сервер BigBlueButton, та зайти в його веб-інтерфейс (рис. 5.4).

Для балансування створимо ще одну таку ж машину. Також, для можливості автомасштабування створимо шаблон з віртуальної машини.

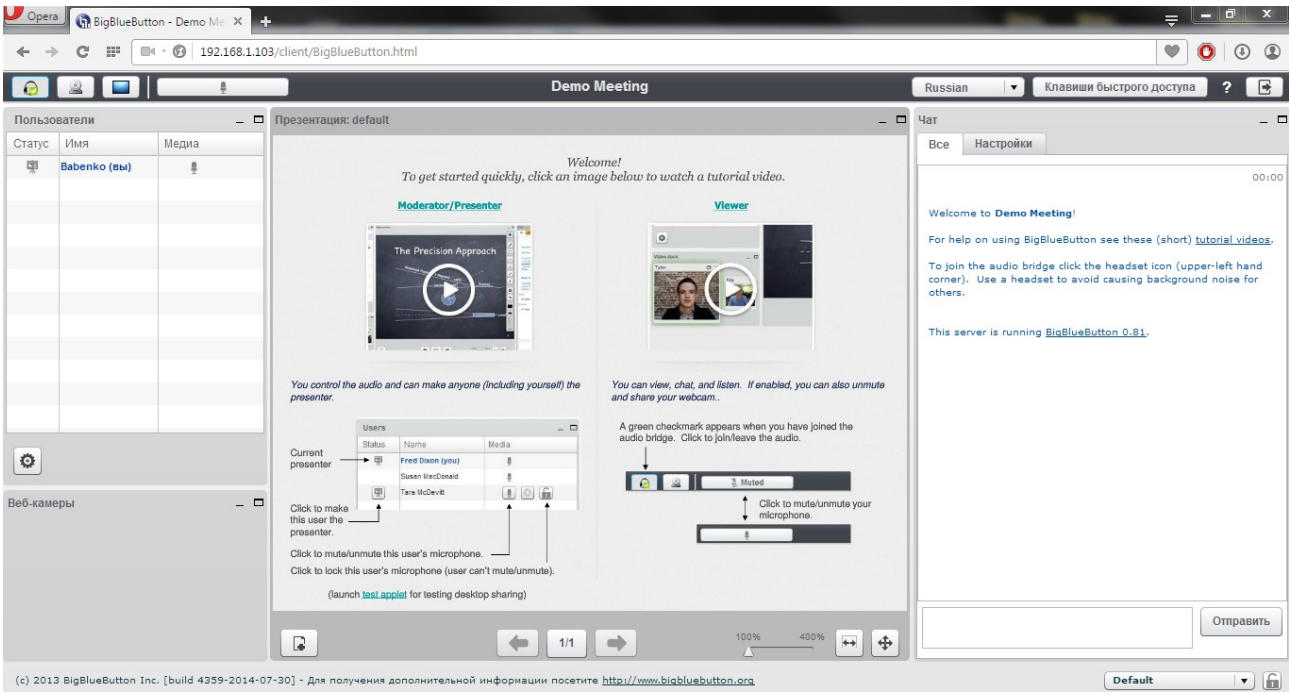

Рисунок 5.4 – Веб інтерфейс системи BigBlueButton

5.2 Створення правил балансування

На основі вимог додатку сформулюємо правила для балансувальника навантаження:

- Алгоритм: з трьох доступних в CloudStack в даному випадку доцільніше використовувати алгоритм Least Connection, так як при роботі з даним додатком передбачаються тривалі сессії, що може призвести до нерівномірного навантаження при використанні інших алгоритмів.
- Липкість: потрібна, щоб один користувач мав можливість приєднуватись до одної й тої ж машини. Підійде політика липкості за IP адресою користувача.
- Перевірка здоров'я: потрібна. Задамо опитування кожні 10 секунд.
- Автомасштабування: потрібне. Масштабування вгору будемо здійснювати при використанні CPU або RAM більше ніж на 80%, а вниз

при використанні CPU або RAM менше ніж на 20%.

Для додавання правила балансування навантаження виконаємо наступні кроки:

1. У лівій панелі навігації вибираємо вкладку "Мережа".

2. Натискаємо на назву нашої мережі, де ми хочемо балансувати трафік. Назва - default.

3. Натискаємо кнопку "Перегляд IP-адрес".

4. Натискаємо IP-адресу, для якої хочемо створити правило (192.168.1.125), вибираємо вкладку "Конфігурація".

5. У вузлі балансування навантаження, натискаємо "Показати всі".

У базовій зоні, також можна створити правило балансування навантаження без вибору IP-адреси. CloudStack всередині призначить IP при створенні правила балансування навантаження, який буде вказаний на сторінці IP-адреси коли правило створиться. Щоб зробити це, треба вибрати ім'я мережі, а потім натиснути вкладку "Додати балансувальник навантаження".

6. Заповняємо наступне:

Рисунок 5.5 – Налаштування правила балансування CloudStack

- Ім'я (Name): web
- Публічний порт (Public Port): 80
- Приватний Порт (Private Port): 80
- Алгоритм (Algorithm): Least connection
- Липкість (Stickiness): Метод SourceBased (рис 5.5)

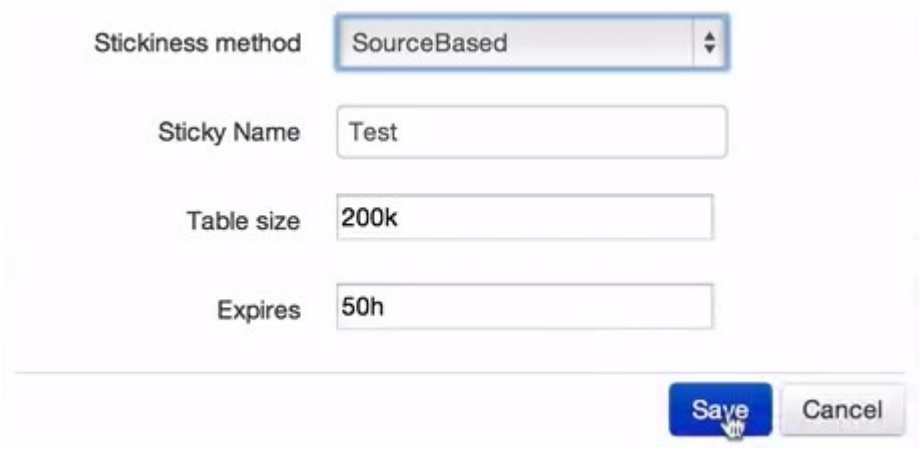

Рисунок 5.6 – Вікно конфігурації Stickiness CloudStack

- Перевірка здоров'я (Health Check): Натискаємо кнопку "Налаштування" (рис. 5.7) і заповняємо характеристики перевірки здоров'я:

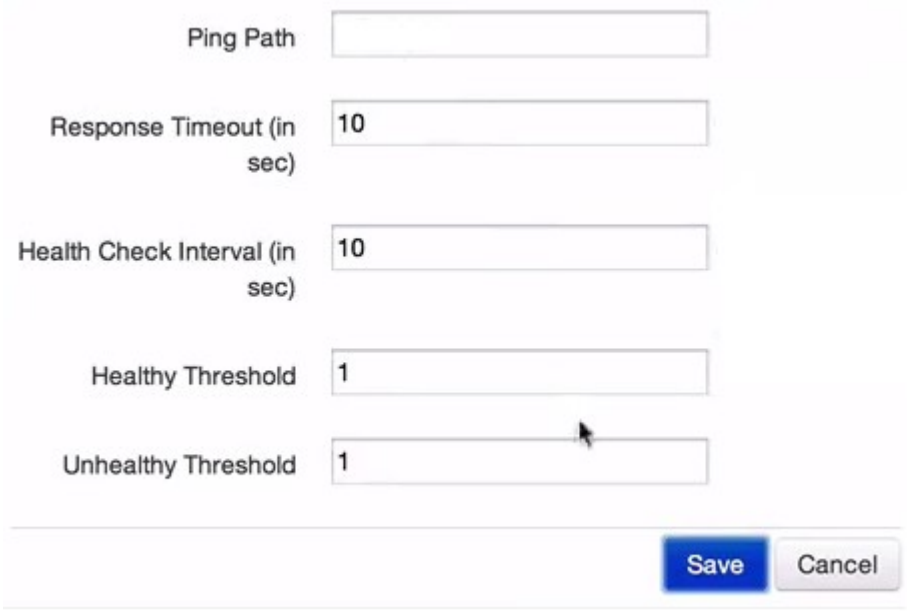

Рисунок 5.7 – Вікно конфігурації перевірки здоров'я CloudStack

o Ping path: Залишаємо значення за замовчуванням. За замовчуванням воно: / (all).

o Response time: 10 секунд.

o Interval time: 10 секунд.

- Healthy threshold: 1.
- Unhealthy threshold: 1.

7. Створимо правило автомасштабування.

Налаштуємо масштабування вгору при використанні CPU або RAM більше 80% та масштабування вниз при використанні CPU або RAM менше ніж на 20% (рис. 5.8).

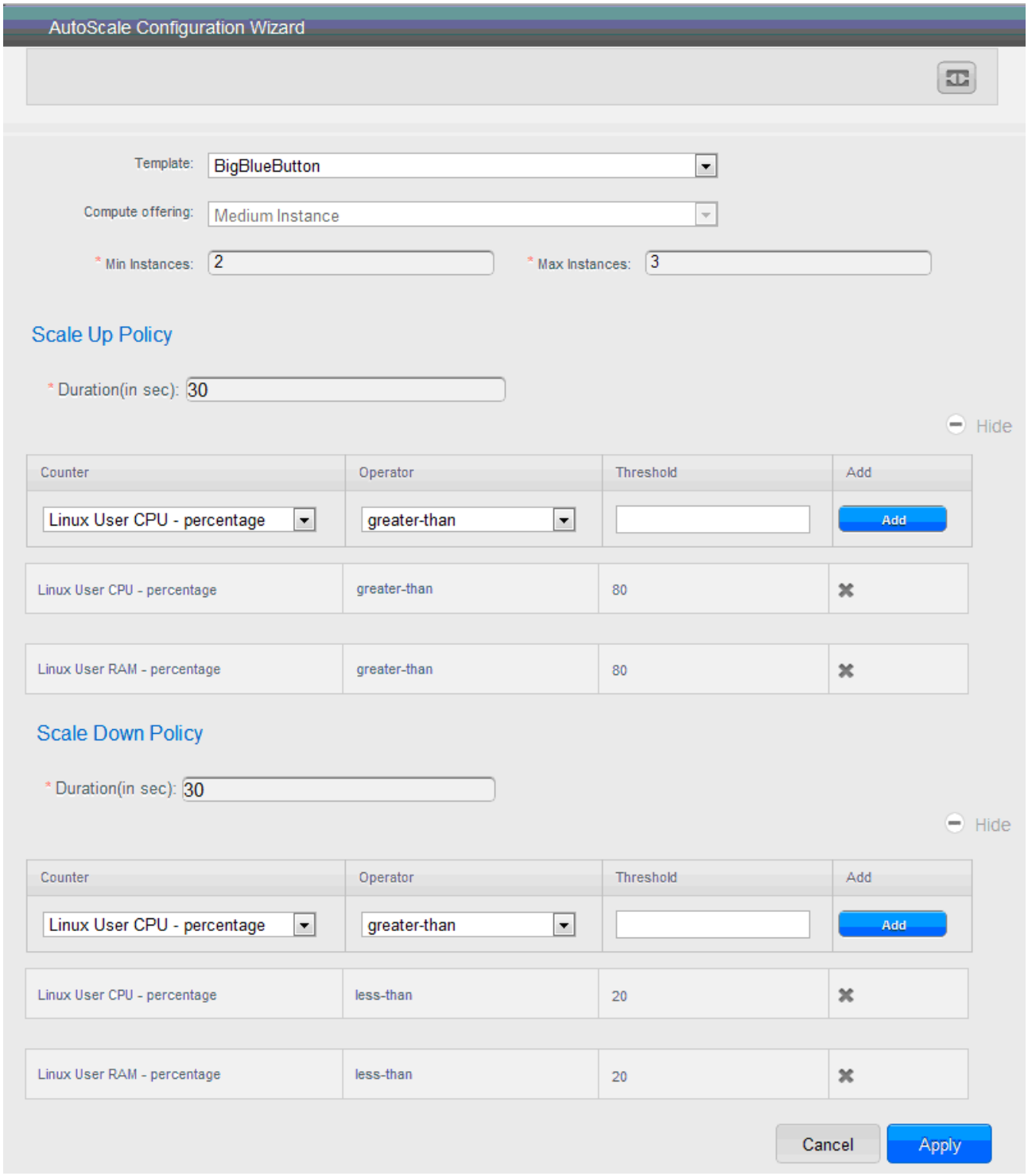

Рисунок 5.8 – Правила автомасштабування CloudStack

8. Натискаємо кнопку "Додати віртуальні машини", а потім вибираємо наші дві віртуальні машини зі встановленим BigBlueButton, які будуть розділяти навантаження вхідного трафіку, і натискаємо кнопку "Застосувати".

Нове правило балансування навантаження (рис. 5.9) з'явилося в списку.

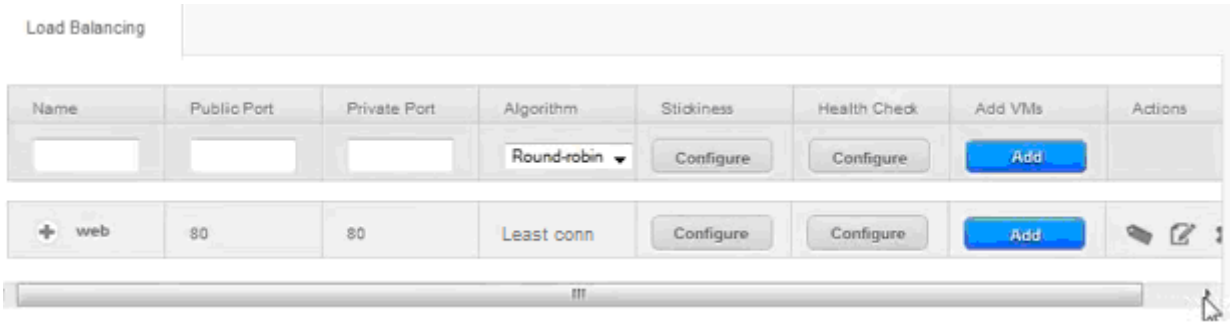

Рисунок 5.9 – Список створених правил балансування CloudStack

5.3 Навантажувальне тестування

Навантажувальне тестування (англ. Load testing) - підвид тестування продуктивності, збір показників і визначення продуктивності та часу відгуку програмно-технічної системи або пристрою у відповідь на зовнішній запит з метою встановлення відповідності вимогам, що пред'являються до даної системи (пристрою).[23]

Для дослідження часу відгуку системи на високих або пікових навантаженнях проводиться стрес-тестування, при якому створювана на систему навантаження перевищує нормальні сценарії її використання. Не існує чіткої межі між навантажувальним і стрес-тестуванням, однак ці поняття не варто змішувати, так як ці види тестування відповідають на різні бізнес-питання і використовують різну методологію.

Основна мета навантажувального тестування полягає в тому, щоб, створивши певне очікуване в системі навантаження (наприклад, за допомогою віртуальних користувачів) і, звичайно, використавши ідентичне програмне і апаратне забезпечення, спостерігати за показниками продуктивності системи.

В якості засобу для навантажувального тестування було обрано WAPT.

WAPT (Web Application Performance Testing) є засобом для тестування продуктивності, навантаження та стресового тестування веб-сайтів і Інтернет додатків з веб-інтерфейсом.[24]

WAPT є засобом навантажувального і стресового тестування, що дає можливості для детального та ефективного тестування веб-сайтів і Інтернет додатків з веб-інтерфейсом. Використовуючи WAPT можна тестувати і аналізувати характеристики продуктивності та вузькі місця веб-сайтів при різних умовах навантаження. Продукт створює навантаження на тестований сайт шляхом емуляції типової активності сотень або навіть тисяч користувачів, що працюють з сайтом одночасно. У процесі тесту сайт поводиться таким же чином, як і при реальному навантаженні того ж рівня, видаючи ті ж параметри продуктивності. WAPT вимірює і протоколює ці параметри. При цьому досить важливо, щоб поведінка віртуальних користувачів сайту, створюваних під час тесту, якнайкраще відповідала поведінці реальних користувачів. Це робить результати тестування надійними. Поступово збільшуючи число віртуальних користувачів у процесі тестування, можна визначити максимальне навантаження, яке витримує сайт, зберігаючи прийнятні параметри продуктивності.

WAPT наділений низкою спеціальних можливостей, які дозволяють йому тестувати RIA-додатки з динамічними даними, змінюючимися прямо під час тесту. Він підтримує всі типи авторизації на сервері. Результати тестування представляються у наочній формі у вигляді звітів і графіків, які дозволять проаналізувати характеристики продуктивності сайту під навантаженням різних типів і рівнів, виявити і усунути вузькі місця, які здатні створити проблеми при повсякденній роботі сайту, і оптимізувати конфігурацію обладнання та програмного забезпечення.

Для моніторингу продуктивності наших машин за допомогою WAPT необхідний SNMP (Simple Network Management Protocol) - протокол, який використовується для управління мережевими пристроями. За допомогою

протоколу SNMP, програмне забезпечення для управління мережевими пристроями може отримувати доступ до інформації, яка зберігається на керованих пристроях (наприклад, на комутаторі)[25].

Для його встановлення та налаштування зробимо наступне:

1. Встановимо: sudo apt-get install snmpd

2. Зробимо копію /etc/snmp/snmpd.conf в файл /etc/snmp/snmpd.conf.org.

mv /etc/snmp/snmpd.conf /etc/snmp/snmpd.conf.org

3. Створимо новий файл /etc/snmp/snmpd.conf, в який запишемо:

rocommunity public

syslocation "cloudtest"

syscontact Babenko

4. В файлі /etc/default/snmpd замінимо

snmpd options (use syslog, close stdin/out/err).

SNMPDOPTS='-Lsd -Lf /dev/null -u snmp -I -smux -p /var/run/snmpd.pid 127.0.0.1'

на

snmpd options (use syslog, close stdin/out/err).

#SNMPDOPTS='-Lsd -Lf /dev/null -u snmp -I -smux -p /var/run/snmpd.pid 127.0.0.1'

SNMPDOPTS='-Lsd -Lf /dev/null -u snmp -I -smux -p /var/run/snmpd.pid -c /etc/snmp/snmpd.conf'

5. Перезапустимо snmpd

/etc/init.d/snmpd restart

Далі налаштуємо тестування в WAPT.

1. Створимо сценарій тесту. Він створюється за допомогою внутрішнього браузера програми, який записує дії користувача в веб-додатку.

2. Налаштуємо кількість користувачів та час роботи тестування.

3. Додамо наші дві машини в засіб моніторингу(рис. 5.10).

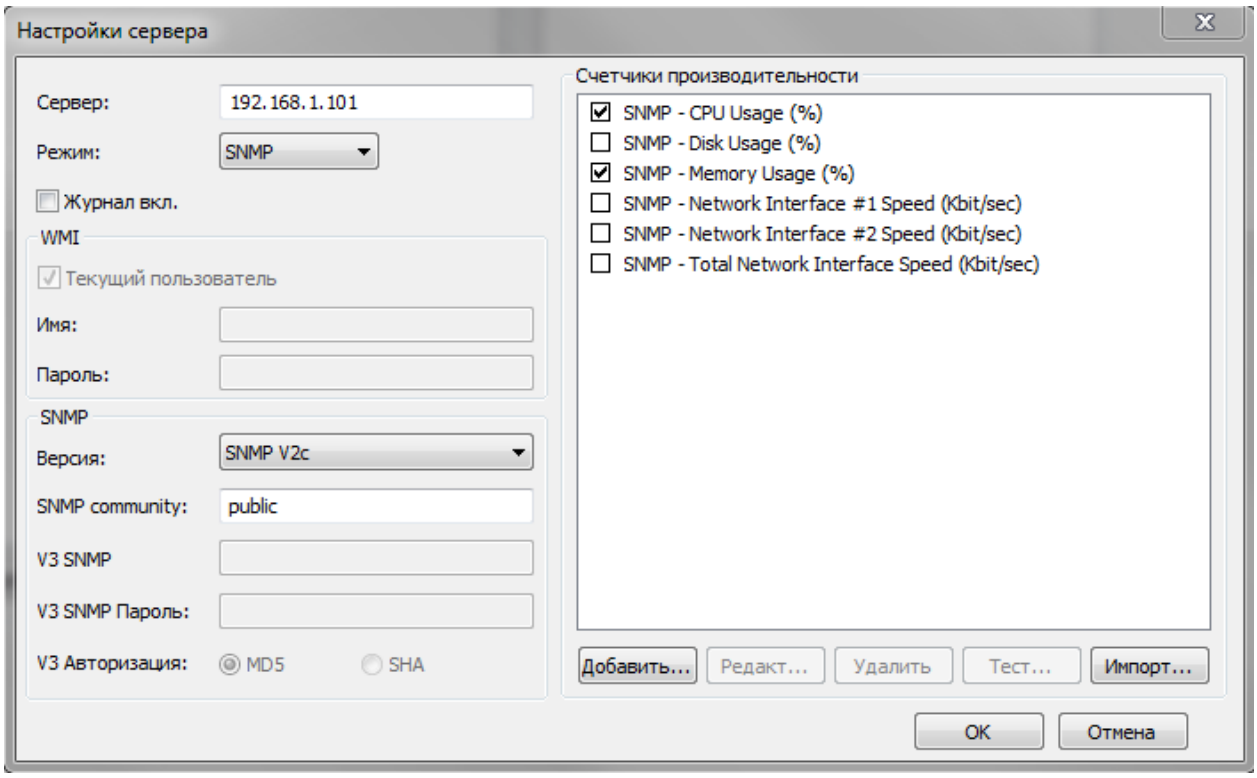

Рисунок 5.10 – Вікно налаштування засобу моніторингу продуктивності WAPT

4. Додамо агент для тестування (localhost:9475).

Далі можна переходити до тестування.

При кожному тестуванні будемо імітувати роботу користувача (логін, переміщення в інтерфейсі системи) з використанням веб-камери і мікрофона.

В першому тесті перевіримо ефективність балансування. Для нього задамо в налаштуваннях тестування кількість користувачів, що збільшується від 0 до 40 з додаванням по 2 користувача кожні 10 секунд. Результати можна побачити на рис. 5.11 (для першої віртуальної машини), рис. 5.12 (для першої віртуальної машини).

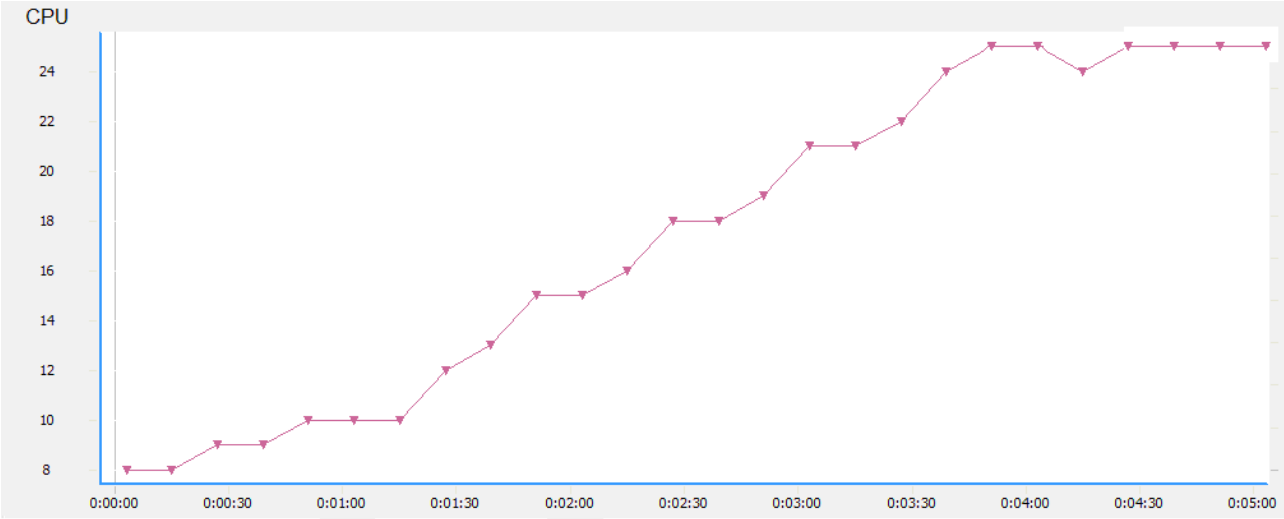

Рисунок 5.11 – Графік навантаження на CPU для першої ВМ

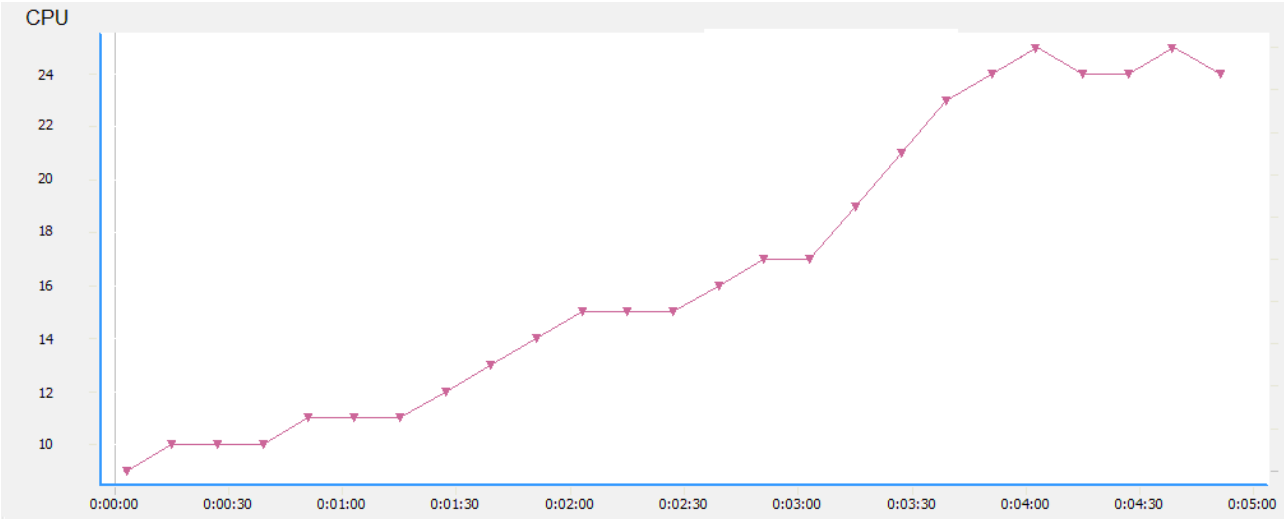

Рисунок 5.12 – Графік навантаження на CPU для другої ВМ

З результатів тестування видно, що балансувальник справляється з навантаженням. Навантаження на обидві ВМ приблизно рівне. В момент часу 0:30:20 надходять останні користувачі, після чого навантаження вирівнюється. Продивившись логи CloudStack, бачимо, що на кожну машину прийшло по 20 користувачів.

Далі проведемо тестування роботи засобу автомасштабування CloudStack. Для цього зробимо два сценарії тесту, які будуть виконуватись послідовно. В першому сценарії кількість користувачів буде збільшуватися від

0 до 160 з додаванням 2 користувачів кожні 6 секунд. В другому - ті ж користувачі, створені в першому сценарії будуть завершувати сесію і їх кількість користувачів буде зменшуватись від 160 до 0 по 2 користувача кожні 6 секунд. Результати наведені на рис. 5.13 (для першого сценарію), рис. 5.14 (для другого сценарію).

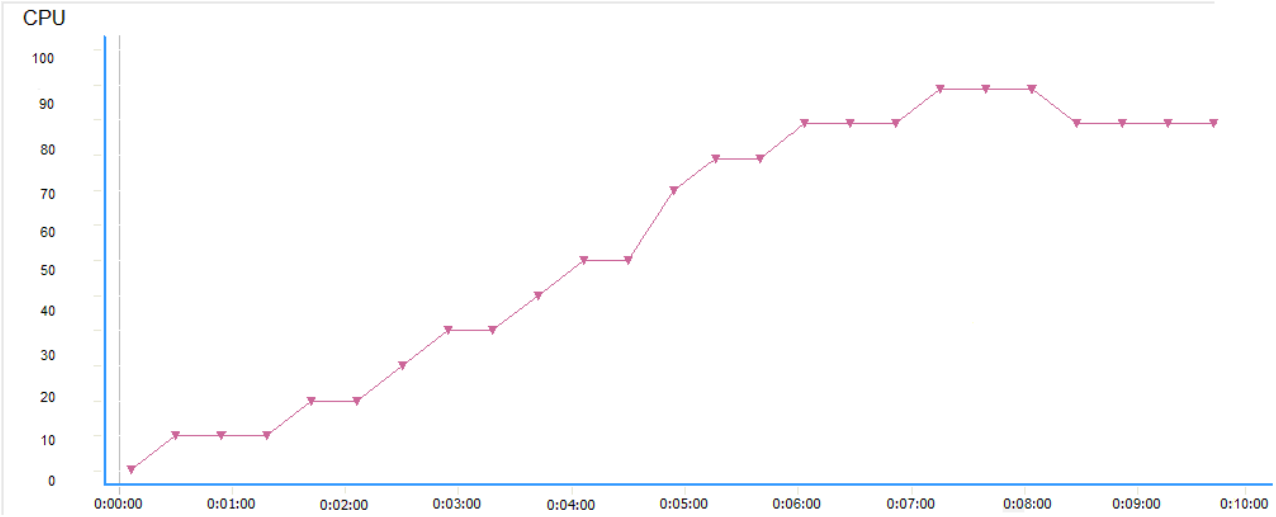

Рисунок 5.13 – Графік навантаження на CPU першої ВМ при виконанні першої половини сценарію

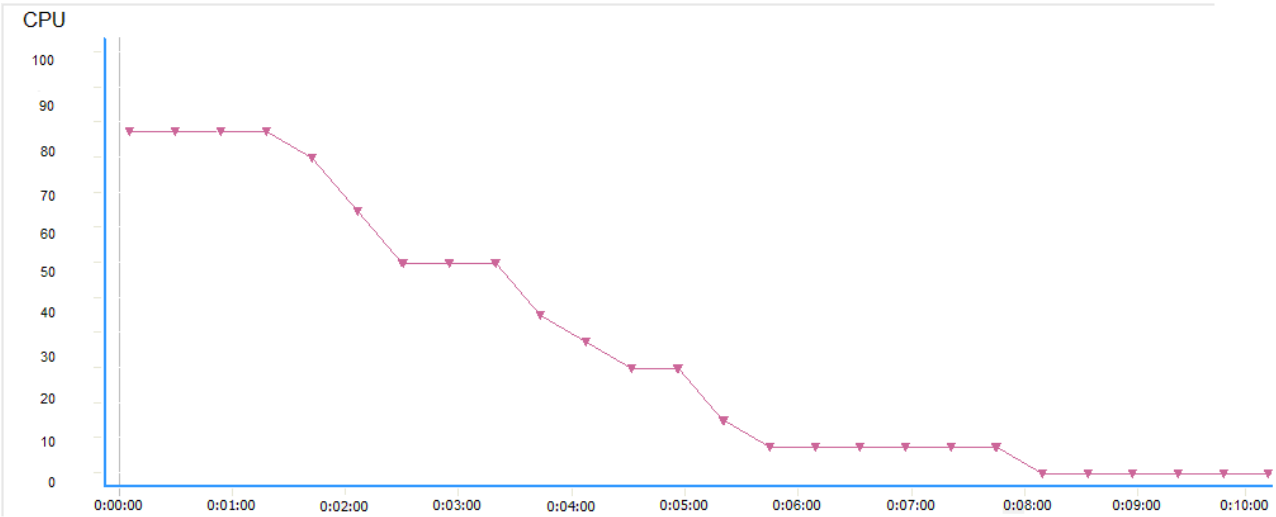

Рисунок 5.14 – Графік навантаження на CPU першої ВМ при виконанні другої половини сценарію

Продивившись логи CloudStack, бачимо, що на наші дві машини прийшло по 69 користувачів. В момент часу 00:05:43 навантаження на CPU перевищило 80%, після чого через 30 секунд в CloudStack було ініційоване створення нової ВМ, яка почала працювати в момент часу 00:06:35. Список ВМ зі створеною засобами автомасштабування ВМ можна побачити на рис. 5.15.

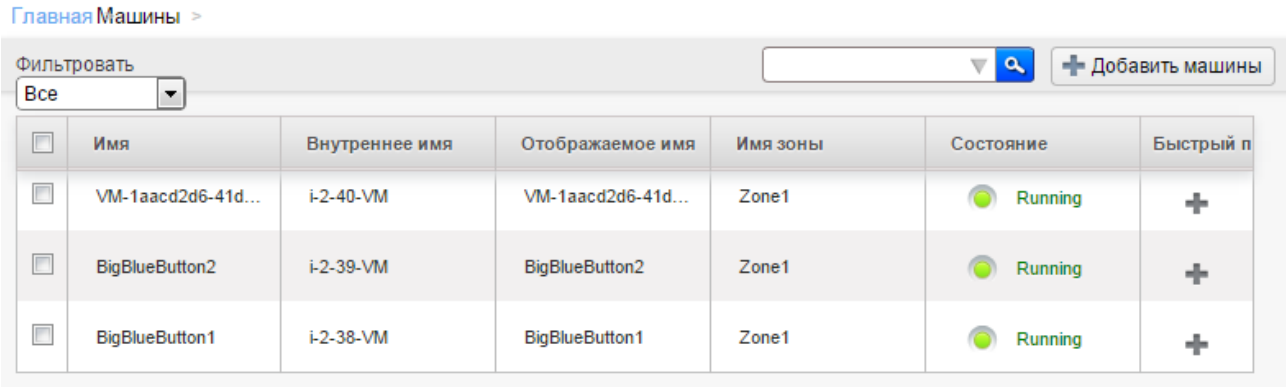

Рисунок 5.15 – Список ВМ зі створеною засобами автомасштабування ВМ в CloudStack

Всі наступні користувачі, відповідно до алгоритму Least Connections, балансувальник відправляв на нову машину. Кількість користувачів на ній досягнула 22. Як можна побачити з графіка (рис. 5.13), після створення нової ВМ, навантаження на неї вирівнюється.

Наступна фаза тестування почалась після моменту часу 00:10:00 (з обнуленням таймера). Під час неї навантаження спадає, поки всі користувацькі сесії не будуть завершені, а потім вирівнюється. В момент часу 00:05:16 навантаження на CPU стало нижче 20% після чого через 30 секунд було ініційоване знищення третьої ВМ (всі сесії її користувачів були вже завершені). Список ВМ після завершення тестування знову складається з двох ВМ, створених вручну (рис. 5.16).

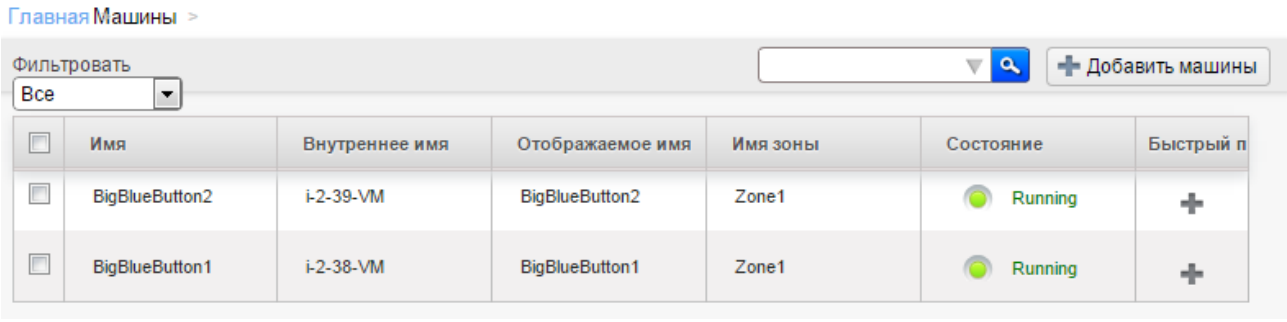

Рисунок 5.16 – Список ВМ CloudStack після завершення тестування

5.4 Розрахунок необхідної пропускної спроможності каналу

Відносно передбачуваності швидкості трафіку додатки діляться на два великі класи: потоковий трафік (stream) і пульсуючий (burst).

Додатки з потоковим трафіком породжують рівномірний потік даних, який надходить в мережу з постійною бітовою швидкістю (Constant Bit Rate, CBR). При пакетному методі комутації трафік таких додатків виглядає як послідовність пакетів однакового розміру, рівного У біт, наступних один за одним через одні і той же інтервал часу Т.

Бітова швидкість, бітність, бітрейт (англ. bitrate) — швидкість проходження бітів інформації за секунду. Бітову швидкість прийнято використовувати до вимірювання ефективної швидкості передачі інформації по каналу, тобто швидкості передачі «корисної інформації» (адже крім такої є ще службова інформація).

Алгоритми обробки черг використовуються в будь-якому мережевому пристрої, який працює на основі механізму комутації пакетів, - в маршрутизаторі, в комутаторі локальної або глобальної мережі, в кінцевому вузлі.

Чергу потрібна для періодів тимчасових перевантажень, коли мережевий пристрій не може передавати пакети на вихідні порти в тому темпі, в якому вони надходять на вхідні порти.

Головним за ступенем впливу на виникнення черг фактором є

коефіцієнт навантаження пристрої (utilization) - відношення середньої інтенсивності вхідного трафіку пристрою до середньої інтенсивності просування пакетів на вихідний інтерфейс. Коефіцієнт навантаження завжди повинен бути <1.

Навантаження системи обчислюється за формулою: P1=T_adopt_t/T_mt, де T_adopt_t - середня тривалість обслуговування, T_mt - середній час між надходженням заявок.

Коефіцієнт використання системи масового обслуговування U=min(T_b/T,1), де T_b - загальний час зайнятості обслуговуючого пристрою, T - загальний час.

Пропускна спроможність системи масового обслуговування LU = min (L1, (L) * (M1)), де L1 - інтенсивність надходження заявок, M1 - інтенсивність обслуговування.

Розрахуємо необхідну пропускну спроможність каналу до сервісу з K віртуальних машин при умові їх пікового навантаження 80%. В нашому випадку K=3.

Як видно із дослідів, на одній віртуальній машині при завантаженості 80% приблизна кількість користувачів N = 65.

Для одного відепотоку середньої якості (720x480), бітрейт B = 0,4 Мбіт/с.

Приймемо необхідну завантаженість каналу M = 75%.

Можемо порахувати необхідну пропускну спроможність каналу:

C = K * N * B / M = 3 * 65 * 0,4 / 0,75 = 104 Мбіт/с.

Отже, для сервісу з 3 віртуальних машин при умові їх завантаженості до 80% необхідну пропускну спроможність каналу складає 104 Мбіт/с.

5.5 Висновки

В даному розділі було проведено дослідження можливостей балансування навантаження системи CloudStack на прикладі додатку дистанційного відео навчання BigBlueButton.

BigBlueButton - відкрите програмне забезпечення для проведення веб-конференції. Система розроблена в першу чергу для дистанційного навчання.

В платформі CloudStack було створено дві віртуальні машини і встановлено на них систему дистанційного відео навчання BigBlueButton. На основі вимог додатку було визначено необхідні правила для балансувальника навантаження, та створено їх в платформі CloudStack.

Після цього було проведено навантажувальне тестування створеної інфраструктури. В результаті нього було показано ефективність створених правил балансувальника та протестовано засіб автомасштабування.

Система CloudStack показала себе як досить надійна та потужна платформа для створення приватної "хмари" корпоративного рівня та балансування навантаження всередині неї.

6 ОХОРОНА ПРАЦІ

6.1 Вступ

У даному розділі проводиться аналіз середовища, в якому проводилося дослідження магістерської дисертації на основі санітарних норм України.

Проводилося дослідження балансування навантаження додатків в хмарному середовищі. Ці дослідження напряму пов'язані з роботою на комп'ютері.

При роботі з персональною комп'ютерною технікою змінюються фізичні і хімічні чинники середовища: електромагнітні випромінювання, статична електрика, температура і вологість повітря, вміст кисню і озону. Повітря забруднюється шкідливими хімічними речовинами антропогенного походження за рахунок деструкції полімерних матеріалів, що використовуються для обробки приміщень та обладнання.

Перш за все, треба обумовити ряд шкідливих факторів, що можуть впливати на якість виконання роботи, а також на здоров’я виконуючого.

Шкідливі фактори, які можуть впливати на роботу програміста з ПОЕМ, можна поділити на дві категорії:

- Фізичні
- Психофізіологічні

Обидві ці групи можуть спричиняти дискомфорт при роботі, або ж навіть призвести до захворювань. Дотримання правил роботи з ПОЕМ, а також санітарних норм забезпечить безпеку працюючого.

Визначаючи основні загрози при роботі, розподілимо їх, як було зазначено вище на дві категорії:

До фізичних віднесемо:

- Підвищений рівень освітленості;
- Нерівномірність розподілу яскравості в полі зору;

- Підвищена яскравість світлового зображення;
- Підвищене значення напруги в електричному ланцюзі;
- Підвищений рівень статичної електрики;

До психофізіологічних віднесемо:

- Напруга зору;
- Напруга уваги;
- Інтелектуальні навантаження;
- Великий обсяг інформації, що обробляється в одиницю часу;
- Нераціональна організація робочого місця.

Неправильна організація робочого місця сприяє загальній та локальній напрузі м'язів шиї, тулуба, верхніх кінцівок, викривлення хребта і розвитку остеохондрозу.

6.2 Характеристика приміщення

Дослідження, що проводилося в даній магістерській дисертації, проводилося в приміщенні, план якого приведено на рис. 6.1.

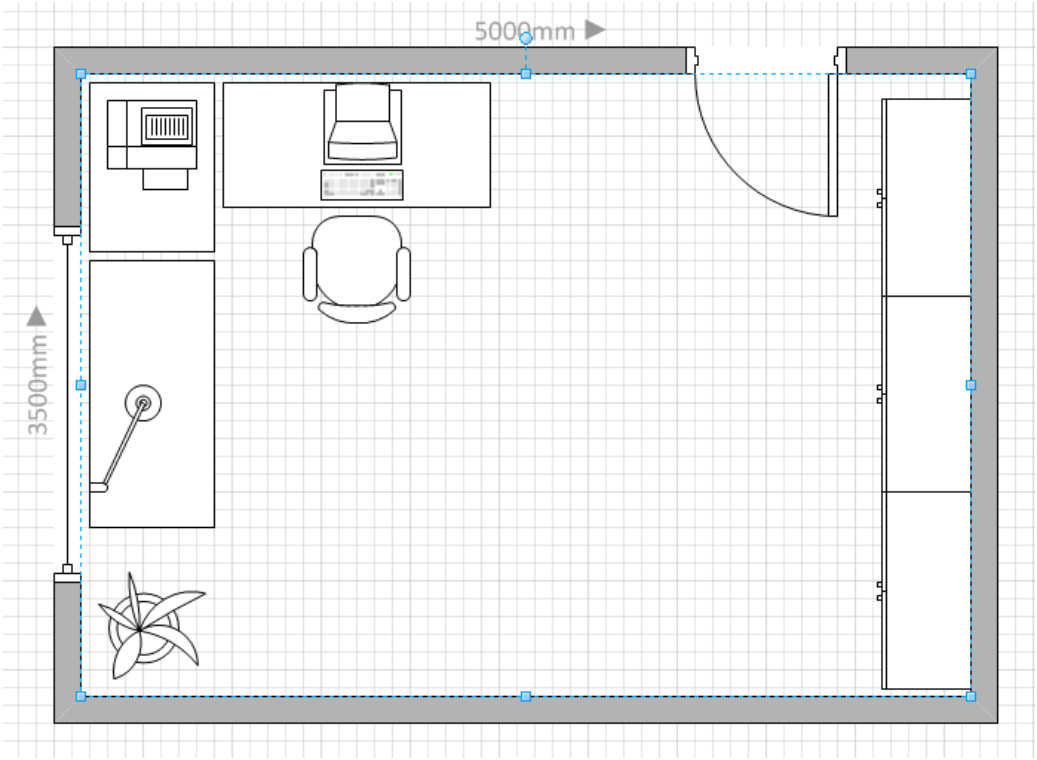

Рисунок 6.1 – План приміщення

Приміщення має одностороннє природне освітлення і загальне штучне освітлення. Стіни і стеля обклеєні світлими шпалерами, підлога вкрита світлим паркетом. У приміщенні відсутні сильні вібрації та шкідливі речовини. Склад повітря в нормі.

У кімнаті знаходиться ПК з двоядерним мікропроцесором Intel Core i5 та 19 дюймовим TFT монітором, принтер, а також меблі.

Приміщення має довжину 5 м, ширину 3.5 м, висоту стелі 2,7 м. Кількість робочих місць - одне. Приміщення знаходиться на другому поверсі п’ятиповерхової цегляної будівлі. Площа – 17,5 м^2, об'єм – 47,25 м^3. Виходячи з цього, отримаємо дані, наведені в таблиці 6.1. Нормативні значення згідно [26].

Таблиця 6.1 – Фактичні та нормативні значення параметрів приміщення

Параметр	Норма	Реальні параметри
Площа, S	не менше 6 м^2	17,5 м^2
Об’єм , V	не менше 15 м^3	47,25 м^3

Можна зробити висновок, що отримані показники відповідають існуючим нормам та вимогам.

6.3 Мікрокліматичні умови

Згідно [27] цю роботу можна віднести до категорії легка 1а. Джерелами тепла в цьому приміщенні є люди, електроустаткування, освітлювальні прилади в темний час доби і система опалювання зимою. Оператором виділяється до 120 Ккал теплової енергії за годину. Оптимальні та фактичні значення параметрів мікроклімату приведені в таблиці 6.2.

Таблиця 6.2 – Значення мікроклімату

Період року	Параметр	Оптимальний	Фактичний
Теплий	Температура	23 – 25 °С	24 °С
	Вологість	40 – 60 %	50 %
	Швидкість повітря	≤ 0.1 м/с	
Холодний	Температура	22 – 24 °С	23 °С
	Вологість	40 – 60 %	55 %
	Швидкість повітря	≤ 0.1 м/с	

Всі показники задовольняють вимогам зазначеним в [27] для робіт категорії легка 1а і є задовільними для здоров'я людини.

6.4 Шкідливі речовини в повітрі робочої зони

У внутрішньому оздобленні інтер’єру приміщень з ПЕОМ забороняється використання полімерних матеріалів, що не дозволені для застосування органами і установами Державного санітарно епідеміологічного нагляду.

Для захисту від впливу шкідливих речовин в повітрі робочої зони слід використовувати кондиціювання повітря робочого приміщення.

Для нормалізації повітря робочої зони необхідно:

- Проводити систематичне провітрювання з використанням кондиціонеру, або вікна, якщо рівень забруднення повітря за вікном не перевищує допустимої норми.
- Проводити вологе прибирання робочої кімнати не рідше двох разів на тиждень, або використовувати зволожувачі повітря.
- Необхідно слідкувати за температурою повітря.

6.5 Освітлення

Освітленість на поверхні столу в зоні розміщення робочого документу повинна бути 300-500 люкс, допускається установка світильника місцевого освітлення для підсвічування документів.[28]

Природне світло повинно проникати через бічні світлопрорізи, зорієнтовані, як правило, на північ або північний схід, і забезпечувати коефіцієнт природної освітленості (КПО) не нижче 1,5%.

Вікна приміщень, повинні мати регулювальні пристрої для відкривання, а також жалюзі.

Штучне освітлення приміщення з робочими місцями, обладнаними відеотерміналами ЕОМ персонального користування, має бути обладнане системою загального рівномірного освітлення.

Загальне освітлення має бути виконане у вигляді суцільних або переривчастих ліній світильників, що розміщуються збоку від робочих місць (переважно зліва) паралельно лінії зору працівників.

Як джерело світла при штучному освітленні застосовуються, люмінесцентні лампи типу ЛБ.

Яскравість світильників загального освітлення в зоні кутів випромінювання від 50 град. до 90 град. відносно вертикалі в подовжній і поперечній площинах повинна складати не більше 200 кд / кв. м, а захисний кут світильників повинен бути не більше 40 град.

Коефіцієнт пульсації повинен не перевищувати 5% і забезпечуватися застосуванням газорозрядних ламп у світильниках загального та місцевого освітлення.

Світильники місцевого освітлення повинні мати напівпрозорий відбивач світла з захисним кутом не менше 40 град.

Необхідно передбачити обмеження прямого блиску від джерела природного та штучного освітлення, при цьому яскравість поверхонь, які світяться (вікна, джерела штучного світла) і перебувають у полі зору, повинна бути не більше 200 кд / кв. м.

Необхідно обмежувати нерівномірність розподілу яскравості в полі зору

осіб, які працюють з відеотерміналом, при цьому відношення значень яскравості робочих поверхонь не повинно перевищувати 3:1, а робочих поверхонь і навколишніх предметів (стіни, обладнання) - 5:1.

Необхідно використовувати систему вимикачів, що дозволяє регулювати інтенсивність штучного освітлення залежно від інтенсивності природного, а також дозволяє освітлювати тільки потрібні для роботи зони приміщення.

Для забезпечення нормованих значень освітлення в приміщеннях з відеотерміналами ЕОМ загального та персонального користування необхідно очищати віконне скло та світильники не рідше ніж 2 рази на рік, і своєчасно проводити заміну ламп, що перегоріли.

Згідно [28] ця робота відноситься до V_a розряду зорових робіт. Передбачається використання природного, штучного і змішаного освітлення.

Природне освітлення здійснюється за допомогою вікна, площа якого складає $S' = 2,0*1,6 = 6,4м^2$ та являється боковим освітленням.

У світильниках місцевого і загального освітлення використовуються лампи розжарювання потужністю 75Вт із світловим потоком лампи = 940 лм.

Можна зробити висновок, що освітлення задовольняє нормам.

6.6 Шум і вібрація

Джерелами шуму в приміщенні є комп'ютер і принтер.

Кулери комп'ютера є сучасними і мають низький рівень шуму, так само як і принтер. Згідно технічній документації шум обумовлений кулером в блоці живлення складає 25 дБ, кулером процесора - 30 дБ, загальний, - 34 дБ. Враховуючи незначний рівень шуму від персонального комп'ютера, малий рівень шуму від принтера, незначний рівень фонового шуму від іншого устаткування - сумарний рівень шумового забруднення приміщення не перевищує максимально допустимий рівень коригованої звукової потужності і складає не більше 50 дБ.

При роботі з персональним комп'ютером в робочому приміщенні

значення характеристик вібрації на робочих місцях не повинна перевищувати допустимих значень.

6.7 Випромінювання

У приміщенні відсутні інфрачервоні, ультрафіолетові та та електромагнітні випромінювання, бо усі монітори ПК вироблені на основі рідко-кристалічної матриці, підсвітка якої здійснюється неоновою лампою, що не має сильного електромагнітного випромінювання і сертифіковані в Україні.

6.8 Електробезпека

Мережеве електроживлення пристроїв ПЕОМ повинно здійснюватися тільки від розеток типу "Європа" з заземлюючими контактами.

Всі електричні розетки, призначені для підключення до них пристроїв ПЕОМ, повинні мати маркування за напругою.

Значення номінальної напруги мережі (220 В) необхідно наносити яскравою фарбою, великими символами (висотою не менше 50 мм) на стіні або щиті, біля або над розеткою.[29]

Заземлюючі контакти розеток повинні мати з'єднання з заземлюючим контуром приміщення або повинні бути занулені. При зануленні необхідно звернути особливу увагу на створення надійного контакту нульового проводу з нульовою шиною мережі електроживлення.

Забороняється використовувати як заземлення радіатори опалення, водопровідні труби.

Аналіз приміщення показав, що воно задовольняє норми електробезпеки.

6.9 Пожежна безпека

Приміщення, яке розглядається належить до категорії В, так як в даному приміщенні є горючі та важкогорючі матеріали (документація, меблі,

презентаційний матеріал тощо). Клас приміщення з пожежонебезпеки — П-IIа, бо в приміщенні є тверді горючі речовини і матеріали[30].

Приміщення, обладнане первинними засобами пожежогасіння: вуглекислотний вогнегасник типу ВВ-2 (ємність 2 л), та забезпечено інструкціями щодо його застосування. Кожен вогнегасник має паспорт.

Пристрої ПЕОМ встановлені вдалині опалювальних і нагрівальних приладів (відстань не менше 1 м і в місцях де не утруднена їх вентиляція).

6.10 Правила роботи з ПЕОМ

Перед початком роботи оператор зобов'язаний

- Оглянути і привести в порядок робоче місце;
- Відрегулювати освітленість на робочому місці, переконатися в достатності освітлення, відсутності відображень на екрані, відсутності зустрічного світлового потоку;
- Перевірити правильність підключення обладнання в електромережу;
- Протерти спеціальною серветкою поверхню екрана і захисного фільтра;
- Перевірити правильність установки столу, стільця, підставки для ніг, пюпітра, положення обладнання, кута нахилу екрану, положення клавіатури і, при необхідності, провести регулювання робочого столу і крісла, а також розташування елементів комп'ютера відповідно до вимог ергономіки та з метою виключення незручних поз і тривалих напруг тіла.

При включенні комп'ютера оператор зобов'язаний дотримуватися наступної послідовності включення обладнання

- Включити блок живлення;

- Включити периферійні пристрої (принтер, монітор, сканер тощо)
- Включити системний блок.

Оператору забороняється приступати до роботи при

- Відсутності інформації про результати атестації умов праці на даному робочому місці або при наявності інформації про невідповідність параметрів даного обладнання вимогам санітарних норм;
- Виявленні несправності обладнання;
- Відсутності захисного заземлення пристроїв ПЕОМ і ВДТ;
- Відсутності вуглекислотного або порошкового вогнегасника та аптечки першої допомоги;
- Порушення гігієнічних норм розміщення ВДТ (при однорядному розташуванні менше 1 м від стін, при розташуванні робочих місць в колону на відстані менше 1,5 м, при розміщенні на площі менше 6 кв.м на одне робоче місце, при рядном розміщенні дисплеїв екранами один до одного).

6.11 Ергономіка робочого місця

Обладнання і організація робочого місця працюючих з ПЕОМ мають забезпечувати відповідність конструкцій всіх елементів робочого місця.

Висота робочої поверхні складає 750 мм, має простір для ніг 700 мм, довжина столу 1500 мм, ширина – 700 мм. Робочий стілець – обертаючийся стілець з регулятором висоти.

Робоче місце відповідає вимогам [26].

Для захисту користувача від небезпечної дії випромінювань відстань від екрану монітора до користувача складає 600-700 мм, робота за комп'ютером виконується не більше 4 годин в день і проводяться періодичні перерви (через

кожних 2 години на 10-15 хвилин). Також періодично виконується комплекс вправ для очей.

До таких вправ можна віднести наступний комплекс:

1. Часто-часто моргати очима одну-дві хвилини.

2. Прикласти долоні до чола і зробити кілька рухів вниз до підборіддя. Проробити цю вправу 10 разів. Закінчивши його, не поспішати відкривати очі. Сидячи з закритими очима, зосередити свій внутрішній погляд на очних яблуках, на живильних їх судинах. Відкривати очі повільно.

3. Гарненько розігріти долоні, потерши їх одна об одну, скласти їх човниками і потримати напроти очей, не торкаючись до повік, як би прогріваючи очі теплом долонь.

4. При зімкнутих повіках обертати очима за годинниковою стрілкою 20 разів, стільки ж - проти годинникової; потім - 20 разів по горизонталі і стільки ж по вертикалі.

5. Повторити попередню вправу, але вже з відкритими очима.

6. Сфокусувати погляд на кінчику носа, а потім перевести його на який-небудь об'єкт, що знаходиться в двох-трьох метрах від вас, потім - на дальній об'єкт десь у лінії горизонту. Повторити 20 разів.

7. Подушечками трьох пальців - вказівного, середнього та безіменного - дуже-дуже легко натискати на закриті повіки 8-10 разів. Виконувати відразу двома руками.

6.12 Висновки

Аналіз умов праці в розглянутому робочому приміщенні показав, що умови праці з ПЕОМ відповідають вимогам , оскільки площа та об’єм не менше нормативних значень, рівні шуму, вібрації і загазованості не перевищують нормативних обмежень.

Для підтримання параметрів мікроклімату в приміщенні встановлено радіатор центральної водяної системи опалення, що складається з 7 секцій.

Аналіз приміщення показав, що воно задовольняє норми електробезпеки. Вимоги пожежної безпеки також виконані.

Ергономіка робочого місця і режим зорової роботи задовольняють вимогам і сприяють зниженню втоми.

ВИСНОВКИ

На даний момент йде активна розробка і вдосконалення технології «хмарних обчислень». Переваги від впровадження даної технології очевидні. Адже це економія для споживачів, боротьба з піратством для розробників, мінімізація витрат в IT сфері для бізнесу, уніфікація мережевих стандартів для всіх користувачів. Також важливо відзначити, що «хмара» з тисяч машин здатна вирішувати дуже важкі завдання, які необхідні сучасним ученим всіх галузей. Так що технології «хмарних обчислень» є перспективною складової IT сфери, що швидко розвивається .

На даний момент інформація про такі аспекти "хмарних технологій" як масштабування та балансування навантаження розрізнена та вимагає систематизації та впорядкування. Для цього важливим являється вибір критеріїв.

В даній роботі було проведено систематизацію і порівняльну характеристику основних методів і алгоритмів балансування навантаження та методів масштабування додатків в "хмарному" середовищі, проведено порівняльну характеристику розповсюджених систем балансування навантаження додатків, досліджено налаштування системи балансування навантаження в платформі CloudStack та надано відповідні практичні рекомендації.

При балансуванні навантаження «хмарних» додатків слід враховувати особливості роботи додатків, особливості інфраструктури обчислювальних комплексів і додаткових вимог при організації сервісів:

- Неоднорідність структури розподіленого сервісу, різні запроси вимагають різних обчислювальних потужностей і звернення до різних виконуючих модулів (наприклад, статичні Web- сторінки, динамічні Web- сторінки, відео і т.п.);

- Необхідність підтримувати безперервність довгострокової (складається з багатьох запитів) сесії користувача, тобто закріплення клієнта за конкретним з однотипних серверів на час роботи з сайтом;
- Неоднорідність структури обчислювального комплексу (наприклад, кластера), з точки зору продуктивності обчислювальних вузлів, що виконують однакові функції;
- Неоднорідність структури обчислювального комплексу з точки зору апаратних і програмних платформ обслуговуючих вузлів;
- Необхідність забезпечення відмовостійкості компонентів сервісу;
- Неоднорідність характеристик пропускної здатності лінії зв'язку, що з'єднують однотипні обчислювальні вузли (всі вузли в одній локальній мережі чи географічно розподілені).
- Можливість автоматичного (або автоматизованого) масштабування додатків.

Можна виділити такі види балансування навантаження:

- статичне - виконується до початку виконання додатка;
- динамічне - перерозподіл обчислювального навантаження на вузли під час виконання програми за прямими (поточне завантаження центрального процесора і ряд характеристик (обсяг доступної оперативної пам'яті, обсяг доступного дискового простору тощо), швидкість передачі інформації по лініях зв'язку і т.д.) і непрямими (кількість активних підключень і час відгуку сервера) показниками.

Можна виділити наступні класи рішень, використовувані при побудові багатовузлових систем: балансування з пропуском трафіку через один пристрій балансування; балансування засобами NLB (Load Balanced) кластера; балансування без пропуску трафіку через один пристрій балансування.

Процедура балансування здійснюється за допомогою цілого комплексу алгоритмів і методів, відповідним наступним рівням моделі OSI:

- мережевому;

- транспортному;
- прикладному.

Балансування на мережевому рівні може здійснюватися за допомогою різноманітних способів:

- DNS-балансування;
- Побудова NLB-кластеру;
- Балансування за IP засобами вхідного шлюзу;
- Балансування зазасобами маршрутизаторів глобальної мережі.

Переваги балансування на третьому рівні:

- незалежність від протоколу високого рівня;
- абсолютна прозорість для серверів;
- незалежність від мережевого розташування серверів.

Недоліки балансування на третьому рівні відповідають способу реалізації балансування: з єдиним пристроєм балансування або без єдиного пристрою балансування.

Балансування на транспортному рівні полягає в тому, що перенаправлення запитів клієнтів одному з обробляючих серверів системою балансування здійснюється не тільки на основі IP- адрес, а й на основі номерів портів в першу чергу протоколу TCP. Це надає методам транспортному рівні більшу гнучкість у порівнянні до методів мережевого рівня. До переваг балансування на транспортному рівні можна віднести можливість балансування навантаження незалежно від типу протоколу прикладного рівня до будь-яких TCP- сервісів: HTTPS, SMTP, IMAP, SSH, FTP, SQL і т.д.

Методи, що відносяться до транспортного рівня:

- Метод NAPT (Network Address Port Translation);
- Проксі- сервер 4-го рівня OSI.

При балансуванні на прикладному рівні балансувальник аналізує клієнтські запити і перенаправляє їх на різні сервери залежно від характеру запитуваної контенту. До методів прикладного рівня можна віднести проксінг

та HTTP Redirect

Переваги балансування на прикладному рівні:

- Робота на рівні протоколу дозволяє проксі-серверу аналізувати і змінювати запити і відповіді, виконувати додавання заголовка, що може використовуватися, наприклад, для підвищення безпеки програми або для перекодування.
- Підтримується прив'язка клієнта до сервера для зберігання його довготривалої сесії за cookie, за заголовком HTTP.
- Аналіз змісту запитів дозволяє розподіляти їх в залежності від типу по різних серверах (наприклад, до статичних сторінок, до динамічних сторінок і т.д.), що прискорює час відгуку вцілому.
- Є можливість кешування відповідей на проксі-сервері, стискати відправляємі дані самостійно, знижуючи загальне навантаження обслуговуючого сервера.
- У ряді випадків є можливість перемістити обробку SSL з обслуговуючих серверів на проксі.

Недоліки балансування на прикладному рівні:

- З усіх розглянутих методів найбільше споживання ресурсів, так як воно зачіпає протоколи більш високого рівня в порівнянні з іншими методами.
- Для кожного прикладного протоколу повинен бути свій тип проксі. Не для всіх протоколів проксінг можна реалізувати таким чином, щоб воно вирішувало потрібні завдання.

Для розподілу навантаження використовуються настуні алгоритми:

- Round Robin;
- Weighted Round Robin;
- Least Connections;
- Weighted Least Connections;
- Locality-Based Least Connection Scheduling;

- Destination Hash Scheduling;
- Source Hash Scheduling;
- Sticky Sessions.

Більшість веб-додатків сьогодні створюється з використанням триланкового розподіленого підходу, що складається з таких рівнів: рівень представлення на стороні клієнта (front-end), рівень бізнес логіки (middle-end), управління ресурсами (back-end).

Для рівня front-end балансування може виконуватись сервером DNS, що містить таблицю відповідності конкретного імені сайту і переліку його IP-адрес, відповідно до яких запити будуть потрапляти на різні front-end сервери. Також можна використовувати методи 3 та 4 рівня з єдиною точкою управління.

На рівні бізнес логіки за балансування навантаження, як правило, відповідає проксі-сервер, встановлений на рівні front-end, який дозволяє не тільки кешувати частина контенту і створювати ще один рівень безпеки системи, але і розподілити користувачів по серверам додатків. Наприклад, проксі-сервер nginx має в своєму арсеналі модуль upstream, який за допомогою алгоритму зваженого обслуговування розподіляє запити користувачів по різних серверах додатків.

Розподіл навантаження між серверними платформами кешуючих серверів переднього плану, серверів бізнес-логіки може проводитися також з використанням спеціалізованих апаратно-програмних комплексів, подібних продуктам NetScaler ADC від компанії Citrix, BIG-IP від компанії F5 Networks і т.п. спеціалізовані апаратно-програмні комплекси, аналізують непрямі показники завантаження серверних платформ,

На рівні back-end, як правило, балансування навантаження виконується засобами кластерів баз даних і ОС.

Алгоритми Destination Hash Scheduling та Source Hash Scheduling використовуються в "хмарному" сервісі Google Compute Engine. Мережеве балансування навантаження в ньому підтримує Compute Engine Autoscaler, що

дозволяє користувачам виконувати авто-масштабування по групах інстансів, наприклад, у цільовому пулі. Мережеве балансування навантаження Google Compute Engine добре масштабується для одного регіону.

Для балансування навантаження територіально розподіленої інфраструктури в Google Compute Engine можна використовувати HTTP балансування навантаження, в якому за рахунок використання глобальної IP-адреси та використання алгоритмів Destination Hash Scheduling та Source Hash Scheduling можна маршрутизувати користувачів на основі близькості.

Також Google Compute Engine пропонує балансування навантаження на основі контенту, що є дуже зручним при необхідності розподілення вхідного трафіку по набору інстансів, які оптимізовані для типу контенту, який вони обслуговують.

Azure Load Balancer Azure Load Balancer доцільно використовувати при комплексному балансуванні продуктів Microsoft через підтримку та взаємну вбудованість сервісів.

"Хмарний" балансувальник Elastic Load Balancing, який використовується в Amazon EC2 є найбільш функціональним та надійним з розглянутих балансувальників в ньму є все те, що зазвичай вимагається від балансувальника: автомасштабування, відмовостійкість, контроль за "здоров'ям" інстансів, простота використання, гнучкість, безпека, міграція, явні параметри завантаження. За рахунок великого рівня автоматизації, людське втручання в роботу балансувальника зведено до мінімуму.

HAProxy це безкоштовне, дуже швидке і надійне рішення, що пропонує високу доступність і балансування навантаження для TCP і HTTP-додатків, за допомогою розподілу вхідних запитів на кілька обслуговуючих серверів. Програма написана на C і має репутацію швидкого, ефективного (в плані використання процесора і оперативної пам'яті) і стабільного рішення.

Linux Virtual Server (LVS) - широко поширений засіб управління кластерних систем для Linux систем. Linux Virtual Server дозволяє створити кластер з фізичних (або віртуальних) серверів, що розподіляє навантаження між

машинами в залежності від їх стану, пріоритету та інших параметрів настройки. Підтримувані протоколи: TCP, UDP. LVS є по суті комутатором протоколів 4го рівня.

Балансувальники HAProxy та Linux Virtual Server доцільно використовувати при побудові власної "хмари" з "нуля". Це потужні, функціональні та гнучкі системи з відкритим кодом, які можна використовувати для балансування навантаження в "хмарному середовищі".

Nginx (engine x) - це HTTP-сервер і зворотний проксі-сервер, поштовий проксі-сервер, а також TCP проксі-сервер загального призначення. Великою перевагою балансування навантаження за допомогою Nginx є підтримка SSL Termination. Використовувати доцільно при необхідності детальної настройки всіх, навіть найдрібніших аспектів балансування. Але для TCP балансування більше рекомендується HAProxy, а Nginx - для HTTP балансування.

CloudStack реалізує балансування навантаження TCP рівня, яке підтримує такі алгоритми або політику: Round-robin, Least Connection, and Source IP. Система CloudStack також підтримує зовнішні балансувальники навантаження. В їх якості можна використовувати як комерційні балансувальники від Google, Microsoft, Amazon, Rackspace, NetScaler, так і відкриті, такі як HAProxy, LVS. Це досить потужна та надійна платформа для побудови "хмар". Її балансувальник включає досить повний набір налаштувань, що робить його одним з кращих при побудові власної "хмари".

В ході роботи було виконано встановлення системи дистанційного відео навчання BigBlueButton на віртуальні машини платформи CloudStack, обґрунтовано вибір правил балансування навантаження і автомасштабування для створеної розгалуженої інфраструктури та проведено налаштування цих засобів. Експериментальне дослідження ефективності створених правил балансування, яке було проведено шляхом навантажувального тестування системи, підтверджує правильність вибору правил і налаштування системи загалом.

Результати даних досліджень можуть бути використані для вибору

методів, алгоритмів і систем балансування навантаження при створенні "хмарних" сервісів, в навчальних дисциплінах з "хмарних обчислень", в подальшій роботі кафедри СП по впровадженню приватного «хмарного» сервісу дистанційного навчання.

ПЕРЕЛІК ПОСИЛАНЬ

1. Gillam, Lee. Cloud Computing: Principles, Systems and Applications / Nick Antonopoulos, Lee Gillam. — L.: Springer, 2010. — 379 с.

2. Осколков И. Ещё раз о облачных вычислениях / И.Осколков // КомпьютерраOnline. 2009. №6. С. 9-11.

3. Риз Дж., Облачные вычисления: Пер. с англ. — СПб.: БХВ-Петербург, 2011. — 288 с.: ил.

4. Стив Александер. Масштабируемость / Стив Александер // – Режим доступу: http://citforum.ck.ua/hardware/articles/mashtab.shtml - Дата доступу : 03.03.2015

5. Горизонтальное и вертикальное масштабирование. Взгляд со стороны бизнес приложений. – Режим доступу: http://blog.vadmin.ru/2013/06/blog-post.html - Дата доступу : 07.03.2015

6. Масштабирование нагрузки web-приложений. – Режим доступу: http://habrahabr.ru/post/113992/ - Дата доступу : 07.03.2015

7. Системы балансировки нагрузки Web-серверов. – Режим доступу: http://citforum.ru/internet/webservers/websbal.shtml - Дата доступу : 15.03.2015

8. Балансировка нагрузки в распределенных системах. – Режим доступу: http://www.intuit.ru/studies/courses/1146/238/lecture/6153 - Дата доступу : 27.03.2015

9. Балансировка в облаках. – Режим доступу: http://www.osp.ru/os/2011/09/13011562/ - Дата доступу : 02.04.2015

10. Балансировка нагрузки в распределенных системах. – Режим доступу: http://www.intuit.ru/studies/courses/1146/238/lecture/6153?page=2 - Дата доступу : 05.04.2015

11. Балансировка нагрузки: основные алгоритмы и методы. – Режим доступу: http://habrahabr.ru/company/selectel/blog/250201/ - Дата доступу : 28.04.2015

12. DNS балансировка. – Режим доступу: http://ruhighload.com/post/DNS+балансировака+ - Дата доступу : 03.05.2015

13. Балансировка масштабируемых приложений. – Режим доступу: http://www.osp.ru/os/2012/08/13019244/ - Дата доступу : 03.05.2015

14. Load Balancing - Compute Engine - Google Cloud Platform. – Режим доступу: https://cloud.google.com/compute/docs/load-balancing/ - Дата доступу : 04.05.2015

15. LoadMaster For Azure. – Режим доступу: http://kemptechnologies.com/solutions/microsoft-load-balancing/loadmaster-azure/ - Дата доступу : 04.05.2015

16. AWS | Elastic Load Balancing– сетевой балансировщик нагрузки в облаке. – Режим доступу: http://aws.amazon.com/ru/elasticloadbalancing/ - Дата доступу : 04.05.2015

17. Обзор балансировщиков нагрузки для *nix. – Режим доступу: https://xakep.ru/2014/03/15/62207/ - Дата доступу : 05.05.2015

18. LVS Introduction - Load Balancing Server Cluster. – Режим доступу: http://www.linuxvirtualserver.org/whatis.html - Дата доступу : 05.05.2015

19. Load Balancing - High Availability and Performance - NGINX. – Режим доступу: http://nginx.com/solutions/load-balancing/ - Дата доступу : 10.05.2015

20. CloudStack: Configure Load Balancer. – Режим доступу: https://support.getcloudservices.com/entries/22002577-CloudStack-Configure-Load-Balancer - Дата доступу : 25.05.2015

21. Managing Networks and Traffic - Apache CloudStack Administration Documentation 4.5.0 documentation. – Режим доступу: docs.cloudstack.apache.org/projects/cloudstack-administration/en/4.5/networking_and_traffic.html - Дата доступу : 25.05.2015

22. BigBlueButton. – Режим доступу: http://docs.bigbluebutton.org - Дата доступу: 02.05.2015

23. Нагрузочное тестирование. – Режим доступу: http://www.protesting.ru/automation/performance.html - Дата доступу : 01.06.2015

24. WAPT - Инструмент для Нагрузочного Тестирования Сайтов и Веб-приложений. – Режим доступу: http://www.loadtesting.ru/product.shtml - Дата доступу : 26.05.2015

25. SNMP. – Режим доступу: http://xgu.ru/wiki/SNMP - Дата доступу : 26.05.2015
26. ДСанПіН 3.3.2.007-98 Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин. – К.: Постанова Головного державного санітарного лікаря України від 10.12.1998 р. № 7.
27. ДСН 3.3.6.042-99 Санітарні норми мікроклімату виробничих приміщень. – К., 2000.- 16 с.
28. ДБН В.2.5-28:2006 Природнє і штучне освітлення. – К. : Міністерство будівництва, архітектури та житлово-комунального господарства України, 2006. – 68 с.
29. НПАОП 0.00-1.28-10 Правила охорони праці під час експлуатації ЕОМ. – Держгірпромнагляд, № 65 від 26 березня 2010 р.
30. НАПБ Б.03.002-2007 Норми визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою