Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інфокомунікацій
(повна назва)

Кафедра Інформаційно-мережної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіті другий (магістерський)

Дослідження методів оптимізації кластерних обчислень на базі Hadoop

(тема)

Виконав:
здобувач 2 курсу, групи ІМІм-23-1
Руденко Д.І.
(прізвище, ініціали)

Спеціальність 172 Електронні комунікації та радіотехніка
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Інформаційно-мережна інженерія
(повна назва освітньої програми)

Керівник: проф. Тихонов В.А.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Безрук В.М.
(підпис) (прізвище, ініціали)

2025 р.

Не містить відомостей, заборонених до відкритого публікування

Здобувач _____ / Руденко Д.І. /
(підпис) (прізвище та ініціали)

Керівник _____ / Тихонов В.А. /
(підпис) (прізвище та ініціали)

Харківський національний університет радіоелектроніки

Факультет Інфокомунікацій

Кафедра Інформаційно-мережної інженерії

Рівень вищої освіти другий (магістерський)

Спеціальність 172 Електронні комунікації та радіотехніка

(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформаційно-мережна інженерія

(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
« 28 » жовтня 2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Руденко Денису Ігоровечу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів оптимізації кластерних обчислень на базі Hadoop

затверджена наказом по університету від« 28» жовтня 2024 р. № 1148 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 21 січня 2025 р.

3. Вхідні дані до роботи Дослідити підходи до побуду на налаштування обчислювального кластеру у контексті Map/Reduce на базі Hadoop. Побудувати кластер, що включає у себе 3 вузла. Розглянути процес оптимізації окремих вузлів кластера. Розрахувати масштаб розширення кластеру протягом року з щоденним об'ємом обчислень 100ГБ, тенденцією щомісячного росту 5% та коефіцієнтом реплікації 3. Обсяг файлосховища вузла – 4 ТБ, резерв на обчислення Map/Reduce - 25%, резерв на власні потреби – 30%. Дослідити процес оптимізації функіонування кластеру, як єдиної структури.

4. Перелік питань, що потрібно опрацювати у роботі Вступ

1 . Обчислювальний кластер, як специфічна обчислювальна система

2. Розгортання обчислювального кластеру на базі Hadoop

3. Загальні підходи до налаштування вузлів кластеру Hadoop у контексті принципу mapreduce

4. Оптимізація hadoop для розгорнутого кластера

Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) слайди презентації в форматі Power Point (назва та мета роботи, сутність та специфіка кластерних обчислень, розгортання кластеру Hadoop, оптимізація вузлів Hadoop, налаштування Hadoop-кластеру, висновки)

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Вступ		виконано
2	Обчислювальний кластер, як специфічна обчислювальна система		виконано
3	Розгортання обчислювального кластеру на базі hadoop		виконано
4	Загальні підходи до налаштування вузлів кластеру hadoop у контексті принципу mapreduce		виконано
5	Оптимізація hadoop для розгорнутого кластера		виконано
6	Висновки		виконано
7	Оформлення пояснювальної записки		виконано

Дата видачі завдання 28 жовтня 2024 р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. Тихонов В.А.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 77 с., 8 рис., 20 джерел, 2 додатка.

Об'єкт дослідження – обчислювальні клатсери на базі Apache Hadoop.

Мета роботи – дослідження методів та засобів збільшення продуктивності кластерних обчислень.

Розглядається специфіка розгортання таподальшого налаштування обчислювальних кластерів Apache Hadoop. Досліджуються підходи до підвищення продуктивності роботи кластеру шляхом оптимального налаштування як окремих вузлів кластеру, так і оптимізації взаємодії між вузлами кластеру. Розраховується план розширення кластеру. Виконується дослідження параметрів, що є критичними для функціонування кластеру.

HDFS, MAP/REDUCE, TASK TRACKER, APACHE HADOOP, DATA NODE, NAME NODE, JVM, ОБЧИСЛЮВАЛЬНИЙ КЛАСТЕР.

THE ABSTRACT

Explanatory note: 77 p., 8 fig., 20 sources, 2 app.

The object of research is computing clusters based on Apache Hadoop.

The purpose of the work is to research methods and means of increasing the productivity of cluster computing.

The specifics of deployment and subsequent configuration of Apache Hadoop computing clusters are considered. Approaches to increase the productivity of cluster work by optimally setting both individual cluster nodes and optimizing the interaction between cluster nodes are studied. A cluster expansion plan is being calculated. A study of the parameters that are critical for the functioning of the cluster is being carried out.

HDFS, MAP/REDUCE, TASK TRACKER, APACHE HADOOP, DATA NODE, NAME NODE, JVM, COMPUTING CLUSTER.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ 9
ВСТУП 10
1 ОБЧИСЛЮВАЛЬНИЙ КЛАСТЕР, ЯК СПЕЦИФІЧНА ОБЧИСЛЮВАЛЬНА СИСТЕМА 12
1.1 Принцип Map/Reduce для обробки Big Data 12
1.2 Поняття кластерної обчислювальної системи 13
1.3 Обчислювальний кластер 14
1.3.1 Компоненти кластеру 15
1.4 Види обчислювальних кластерів 16
1.4.1 Кластери високого рівня доступності 16
1.4.2 Кластери з балансуванням навантаження 17
1.4.3 Кластери високої продуктивності 20
2 РОЗГОРТАННЯ ОБЧИСЛЮВАЛЬНОГО КЛАСТЕРУ НА БАЗІ HADOOP 22
2.1 Поняття кластерної обчислювальної системи 22
2.2 Підготовка операційної системи 22
2.2.1 Оновлення пакетів 23
2.2.2 Конфігурування брандмауера 23
2.2.3 Налаштування імен у кластері 24
2.2.4 Інсталяція середовища Java 24
2.3 Інсталювання Apache Hadoop 25
2.3.1 Завантаження вихідного файлу 25
2.3.2 Інсталяція та налаштування середовища 26
2.3.3 Перевірка попереднього налаштування середовища 28
2.3.4 Створення сертифікатів безпеки 29
2.4 Налаштування та запуск Hadoop 31
2.5 Тестовий запуск кластеру 34
2.6 Налаштування автозапуску Hadoop 35
3 ЗАГАЛЬНІ ПІДХОДИ ДО НАЛАШТУВАННЯ ВУЗЛІВ КЛАСТЕРУ HADOOP У КОНТЕКСТІ ПРИНЦИПУ MAPREDUCE 36
3.1 Загальна стратегія налаштування кластеру 36
3.2 Визначення необхідного розміру кластера Hadoop 36
3.2.1 Огляд архітектури кластеру Hadoop 36

3.2.2 Формування плану розширення кластеру 38
3.2.3 Обчислення необхідного об’єму оперативної пам'яті для вузлів 41
3.3 Оптимізація вузлів кластеру для завдань Map і Reduce..... 42
4 ОПТИМІЗАЦІЯ HADOOP ДЛЯ РОЗГОРНУТОГО КЛАСТЕРА 47
4.1 Налаштування конфігураційних файлів Hadoop 47
4.2 Тестування Hadoop..... 50
4.3 Підхід до підвищення продуктивності кластеру на базі повторного використання віртуальних машин Java..... 51
4.4 Аналіз та оптимізація продуктивності кластерів. Моніторинг та налаштування продуктивності кластерів Hadoop 52
4.4.1 Інтеграція та налаштування інструментарію моніторингу продуктивності 53
4.4.2 Рекомендації з оптимізації параметрів Hadoop..... 54
ВИСНОВКИ..... 59
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ..... 61
ДОДАТОК А ТЕЗИ КОНФЕРЕНЦІЇ..... 63
ДОДАТОК Б СЛАЙДИ ПРЕЗЕНТАЦІЇ..... 69

ПЕРЕЛІК СКОРОЧЕНЬ

СЗД – система зберігання даних;

JVM (Java Virtual Machine) – віртуальна машина Java;

HDFS (Hadoop Distributed File System) – розподілена файлова система Hadoop;

NN (Name Node) – компонента HDFS, що забезпечує коректну реплікацію блоків даних у кластері;

SNN (Secondary Name Node) – компонента HDFS, що забезпечує коректну реплікацію блоків даних у кластері;

DN (Data Node) – компонента HDFS, що здійснює управління станом вузла HDFS, а також взаємодіє з його блоками даних;

TT (Task Tracker) – компонента кластеру, що виконує запуск завдання на кожному вузлі;

JT (Job Tracker) – компонента, що відстежує завдання, які виконуються у кластері;

HAC (High-availability clusters) – кластери високої доступності;

HPC (High performance computing clusters) – кластери високої продуктивності.

ВСТУП

Одним з напрямків розвитку сучасного технологічного базису є збільшення ролі Big Data у багатьох галузях діяльності, що сприяє оптимізації їх роботи. Зокрема, збір та наступна обробка великих масивів даних дозволяє [1-5]:

− збільшити ефективність вже існуючих технологічних механізмів;

− заощадити витрати на залучення клієнтів, розширення продажів та впровадження багатьох мережевих сервісів;

− виконувати прогнози аварійних станів систем, росту/падіння вартості акцій або товарів;

− виявляти аномалії у процесі функціонування технологічних систем, що дозволяє виявити неочевидні помилки налаштування, а також загрози інформаційній безпеці тощо.

Все це забезпечується за рахунок побудови моделей машинного навчання для класифікації та кластеризації даних, виявлення неочевидних статистичних залежностей, прогнозування поведінки системи за певних умов і т.д.

Разом з тим, в умовах росту об’ємів даних, які необхідно обробляти, нерідко потужності окремих комп’ютерів недостатньо, що потребує побудови обчислювальних кластерів на базі кількох окремих вузлів. У таких кластерах обчислювальні процеси виконуються розподіллено – тобто, єдину задачу вирішують кілька вузлів, що суттєво скорочує час обчислень [1].

При цьому, основу кластерних обчислень зараз складає запропонований компанією Google принцип Map/Reduce [6], який дозволяє збільшити ефективність обробки великих даних.

У свою чергу, підсумкова ефективність роботи кластеру визначається багатьма факторами, серед яких:

− апаратні ресурси вузлів кластеру;

− пропускна здатність мережі, що поєднує між собою вузли;

− тип програмного забезпечення для управління процесами у кластері;

− налаштування ПЗ управління кластером;

− налаштування завдань Map/Reduce.

Безумовно, роль апаратних ресурсів, а також параметри мережі, є визначальними факторами для підсумкової продуктивності кластеру. Разом з тим навіть в умовах, коли на кожен з вузлів виділено значну кількість

процесорних ресурсів та ресурсів пам’яті, задіяно дискові накопичувачі надвисокої швидкості та забезпечено мережеві канали не нижче 10 Гб/сек, оптимальну роботу кластеру не може бути гарантовано. Майже завжди це є наслідками некоректного розподілу ресурсів, ігнорування специфіки конкретних завдань Map/Reduce та неоптимального конфігурування ПЗ управління кластером.

У зазначений умовах питання, пов’язані з дослідженням архітектури кластерних обчислювальних систем, інструментів моніторингу продуктивності, а також підходами до оптимізації кластерних обчислень у цілому є актуальними.

1 ОБЧИСЛЮВАЛЬНИЙ КЛАСТЕР, ЯК СПЕЦИФІЧНА ОБЧИСЛЮВАЛЬНА СИСТЕМА

1.1 Принцип Map/Reduce для обробки Big Data

Обробка великих даних традиційними способами нерідко є неефективною з причини значних часових витрах, які є неприйнятними для сервісу/додатку, у межах якого здійснюється дана процедура. У таких умовах багаторазово знизити час обчислень можливо з використанням принципу Map/Reduce [6, 7].

Згідно даному принципу, обробка вихідного масиву даних включає у себе такі кроки, як:

1. Перед обробка даних на базі обробника Map, функціонал якого відповідно до конкретної задачі встановлює користувач. У результаті даної стадії один вхідний запис отримує пару (ключ, значення).

2. Розбиття результату, отриманого на стадії Map, на окремі сегменти. При цьому, кожен сегмент відповідає одному ключу виведення стадії Map. Інакше кажучи, виконується сортування даних за ключем. Надалі ці сегменти спрямовуються до обробників Reduce.

3. Обробка кортежів даних, отриманих з попередньої стадії, на базі обробника Reduce.

4. Узагальнення даних, отриманих від різних модулів Reduce та формування результату.

Загальну схему роботи принципу Map/Reduce наведено на рисунку 1.1.

Такий принцип обробки дозволяє, по-перше, зменшити час обчислення як за рахунок упорядкування самого процесу ведення розрахунків, так і за рахунок паралельної обробки усього масиву даних на одному вузлі. По-друге, даний принцип дозволяє виконувати обробку окремих сплітів (початкових фрагментів) вихідного масиву даних на окремих ПК, що, зі свого боку, також дозволяє суттєво збільшити швидкість обчислення навіть в умовах, коли окремі вузли не матимуть значної обчислювальної потужності – наприклад, з використанням ПК офісного формату [7].

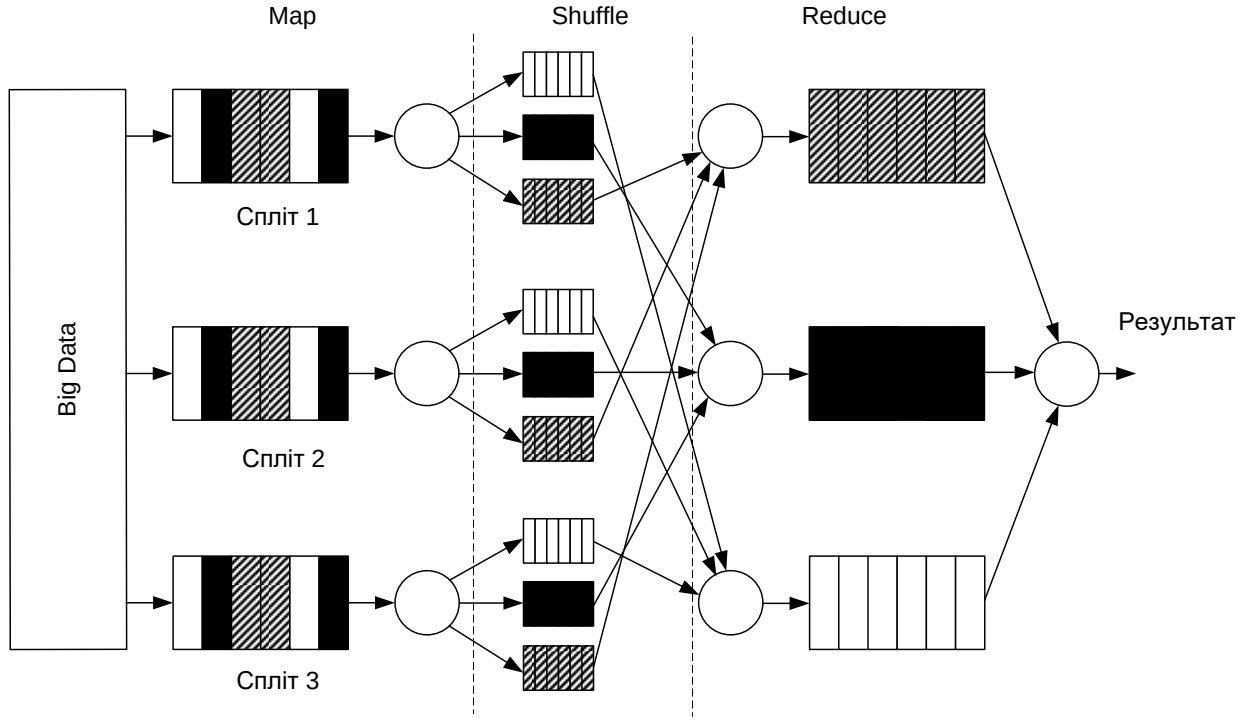

Рисунок 1.1 – Загальна схема обробки даних за принципом Map/Reduce

1.2 Поняття кластерної обчислювальної системи

Кластер являє собою локальну (тобто, таку, що розміщується географічно на обмеженій території) обчислювальну систему [8-12], до складу якої входить множина взаємно незалежних обчислювальних терміналів (комп'ютерів) та мережа, яка їх поєднує між собою.

Також кластер вважається суто локальною системою з тієї причини, що керування ним здійснюється виключно у рамках окремого адміністративного домену, як єдиної комп'ютерної системи.

Головний сенс розгортання обчислювальних кластерів – забезпечення більшої продуктивності та доступності системи ніж у випадку, коли використовується лише один вузол, хоча і дуже продуктивний. Нерідко таке рішення є також суттєво економічним, ніж використання окремих комп'ютерів.

У свою чергу, комп'ютерні вузли, що входять до складу кластеру, є стандартними, універсальними ПК, які широко застосовуються для рішення різноманітних занять у різних галузях з використанням широкого набору додатків.

Окремі обчислювальні вузли кластеру можуть містити у собі як один мікропроцесор, так і кілька, утворюючи, в останньому випадку, симетричну (SMP-) конфігурацію.

При цьому, мережева складова кластеру може бути реалізована на базі:

− звичайної локальної мережі [8];

− спеціалізованих мережевих технологій, які дозволяють забезпечити можливість над швидкого обміну даними між вузлами, що входять до кластеру.

Так як головне завдання мережі кластеру зводиться до інтеграції вузлів кластера та обміну даними між ними, тому зазвичай така мережа відокремлюється від зовнішньої опорної мережі, яка забезпечує доступ користувачів до кластеру.

Водночас, усі процеси у кластері реалізуються під управлінням спеціалізованого програмного забезпечення, що має у своєму складі 2 компонент, а саме [6-9, 12]:

− засоби розробки/програмування;

− засобів забезпечення керування ресурсами.

Так, до засобів розробки належать:

− компілятори для мов;

− бібліотеки функцій різного призначення;

− засоби вимірювання продуктивності;

− налагоджувачі тощо.

Усе вище перелічене дає змогу будувати додатки паралельних/розподілених обчислень.

При цьому, ПЗ управління ресурсами – це:

− засоби інсталяції;

− засоби адміністрування;

− засоби планування потоків робіт.

1.3 Обчислювальний кластер

Отже, як було зазначено вище, обчислювальний кластер, у сутності - набір з'єднаних між собою терміналів (комп'ютерів, серверів), що функціонують синхронно та можуть розглядатися у вигляді єдиної системи. Ключовою відмінністю кластерних обчислювальних систем від GRID-систем є те, що [6-9, 12]:

− управління усіма вузлами кластеру здійснюється на базі однієї системи управління;

− вузли кластеру виконують одну й ту саму задачу.

З’єднання серверів кластеру між собою зазвичай забезпечуються ресурсами локальної мережі. При цьому, у межах кожного з серверів функціонує окремих екземпляр ОС [10].

Переважна більшість кластерів будується таким чином, щоб обчислювальні вузли мали однакову апаратну комплектацію, а також здійснювали функціонування під управлінням однієї й тієї ж самої операційної системи.

Водночас, у ряді специфічних випадків, зокрема, коли використовується платформа додатків для побудови кластерів OSCAR (Open Source Cluster Application Resources), можуть використовуватись різні серверні апаратні складові а також різня операційні системи.

1.3.1 Компоненти кластеру

Загальну схему типового кластеру показано на рис.1.2 [10].

Рисунок 1.2 – Ключові компоненти кластеру

Як видно з даного рисунку, до складу обчислювального кластеру частіше за все входять такі наступні компоненти, як:

− вузол доступу;

− обчислювальні вузли;

− файловий сервер;

− файлова або об'єктна СЗД, що має загальний доступ;

− локальна мережа.

1.4 Види обчислювальних кластерів

Сьогодні існують такі базові види кластерів, як [10-12]:

− кластери високого рівня доступності (High-availability clusters, HA);

− кластери з можливістю балансування навантаження (Load balancing clusters);

− кластери високої продуктивності (High performance computing clusters, HPC).

1.4.1 Кластери високого рівня доступності

Основою кластерів високого рівня доступності, що також відомі як відмово стійкі (failover) кластери, є схема мережі з великою надмірністю (redundancy) [10-12].

Такі кластери частіше за все використовуються для забезпечення стабільності критичного серверного ПЗ. Наприклад - серверів баз даних.

При цьому, кластер може вважатися, таким, що належить до класу HA у тому випадку, коли він здатен забезпечити доступність додатків не менше, ніж на рівні «п’ять дев’яток».

Іншими словами, будь-який додаток у кластері HA-типу повинен бути доступним (мати статус «uptime») не менш, ніж 99,999 відсотків часу протягом року.

Такий надвисокий рівень доступності кластерів HA-типу забезпечується шляхом використання спеціалізованого ПЗ та апаратних систем, ураховуючи імплементацію схем локалізації відмов, а також за рахунок підготовчих робіт з протидії відмов.

У свою чергу, програмне забезпечення для кластерів HA-типу частіше за все заздалегідь будує конфігурацію вузла у межах резервного серверу. Після

цього виконує запуск на ньому додатку у фоновому режимі таким чином, щоб надати можливість оперативного переходу основному екземпляру програмного засобу на свою репліку у межах резервного комп'ютеру у разі відмови основного.

Окрім серверів баз даних, кластери HA-типу частіше за все розгортаються для побудови термінальних серверів, файлових серверів з загальним доступом та поштових серверів.

Водночас, такі сервери може бути розгорнуто як локально (у т.з. «серверній фермі» - поєднавши кілька апаратних модулів, наприклад у межах однієї стійки), так і у варіанті географічного рознесення окремих, або усіх вузлів.

Разом з тим, технологія HA-кластерів, як, власне, кластеризація взагалі, не може розглядатися у якості заміни резервного копіювання, або backup'у), або у якості рішення для забезпечення катастрофостійкості, або disaster recovery.

1.4.2 Кластери з балансуванням навантаження

Процедура балансування навантаження полягає у розподілі вхідного трафіку мережі на рівні групи серверів, тобто – кластеру [10, 11].

Сам процес балансування має бути ефективним – тобто, у наслідок виконання процедур балансування мають виключатися ситуації, коли одна частина серверів кластеру знаходиться у стані простою, тоді як інші – перевантажені.

Наприклад, сучасні веб-сайти мають забезпечити одночасне обслуговування досить великої кількості клієнтських запитів – від сотень тисяч до мільйонів за одиницю часу. При цьому, необхідно забезпечити обслуговування з мінімально можливим рівнем затримки завантаження контенту незалежно від його типу – як тексту, так і аудіо, відео чи даних додатків.

За таких умов недостатньо лише, щоб запити обслуговувала велика кількість окремих серверів. Необхідно забезпечити балансування навантаження.

У свою чергу, для рішення цього завдання використовується балансувальник навантаження, що виконує рівномірний розподіл запитів клієнтів між усіма існуючими серверами кластеру, які можуть обробляти ті чи інші запити.

Інакше кажучи, балансувальник забезпечує максимізацію частки використання обчислювальної потужності, та створює умови, коли жоден сервер не є перевантаженим, що веде до виникнення загального падіння продуктивності кластеру.

Так, у разі відмови того чи іншого серверу, балансувальник виконує перенаправлення трафіку на інші сервери, які мають резерв обчислювальних можливостей.

Водночас, у разі появи у кластері нового серверу, балансувальник автоматично перерозподіляє навантаження між усіма серверами, ураховуючи той, який щойно розпочав роботу.

Отже, у підсумку балансувальник навантаження реалізує такі функції, як (рис.1.3):

− розподіл клієнтських запитів та навантаження мережі у межах усього кластеру серверів ефективним чином;

− створює умови для забезпечення високого рівня доступності і надійності надсиланням запитів лише на сервери, що знаходяться в онлайн-режимі;

− забезпечує високий рівень гнучкості усієї системи за рахунок додавання, чи, навпаки, вилучення серверів залежно від поточної потреби.

Рисунок 1.3 – Загальна схема функціонування балансувальника навантаження

Алгоритми балансування навантаження

Сьогодні використовуються різні алгоритми балансування. При цьому, кожен з них забезпечують виконання різних задач, а також отримання різного роду переваг.

До найбільш поширених сьогодні алгоритмів балансування можна віднести наступні:

− Round Robin. Даний алгоритм передбачає послідовний розподіл запитів за вузлами кластеру;

− Least Connections. У кластерах, де використовується даний алгоритм, у ході балансування навантаження відбувається спрямування нового запиту на сервер, який у поточний час має найнижчу кількість підключень клієнтів; водночас, береться також до уваги рівень обчислювальної потужності кожного серверу;

− Least Time – за цим алгоритмом виконується спрямування запитів до серверу, який обирається за показниками швидкості відповіді, а також найменшої кількості активних запитів користувачів;

− Hash – розподіл запитів здійснюється на базі ключа, який, у свою чергу, визначається користувацькою інформацією; наприклад, це може бути його IP-адреса, URL запитуваного сайту тощо;

− Random with Two Choices – на першому етапі роботи здійснюється вибір пари серверів на базі випадкового закону, після чого запит спрямовується одному з серверів, який, у свою чергу має меншу кількості клієнтських підключень.

Програмне та апаратне балансування навантаження

У загальному випадку, існуючі балансувальники навантаження може бути поділено на програмні та апаратні.

При цьому, балансувальники програмного типу може бути інстальовано на будь-який сервер, який має відповідні параметри.

На відміну від них, апаратні балансувальники являють собою сервери зі спеціалізованими процесорними системами, на які встановлено програмне забезпечення балансування навантаження.

Програмні балансувальники характеризуються меншою вартістю порівняно з апаратними, та є більш гнучкими. Разом з тим, для проектів великого масштабу, що включає у себе сотні серверів, доцільно використовувати апаратні рішення.

Окремо слід розглядати хмарні сервіси балансування навантаження. Зокрема, такі, як AWS EC2, та подібні.

1.4.3 Кластери високої продуктивності

Обчислення високої продуктивності, або HPC (High-performance computing) – це здатність виконувати обробку даних, а також здійснювати розрахунки підвищеної складності з високою швидкістю [10-12]. Очевидно, що дане поняття є досить відносним.

Для прикладу візьмемо звичайний офісний ПК, що має тактову частоту процесору, рівну 3 ГГц. Такий пристрій, у свою чергу, здатен виконувати 3 мільярди обчислень протягом секунди. Отже, цілком зрозуміло, що для будь-якої звичайної людини це буде надто велика швидкість виконання обчислень.

Водночас, така швидкість суттєво поступається швидкості, яку забезпечують рішення HPC, що здатні виконувати квадрильйони обчислень за секунду.

Типовим, та найвідомішим рішенням HPC є т.з. «суперкомп'ютер». Суперкомп'ютер являє собою систему, що може містити у своєму складі тисячі обчислювальних вузлів. Такі вузли, хоча, апаратно, є окремими обчислювальними модулями, працюють спільно над рішенням одного або одразу кількох завдань.

HPC-рішення відіграють надзвичайну роль для прогресу у галузях науки, промисловості та громадській.

Достатньо згадати про те, що ряд технологій – зокрема - інтернет речей IoT (Internet of Things), штучний інтелект AI (Artificial Intelligence) а також адитивне виробництво (3D imaging), вимагають дуже високих обсягів обробки даних. При цьому, об’єми таких даних зростають з часом за експоненційним законом.

Водночас, здатність виконувати оперативну обробку досить великих обсягів даних є критичною для багатьох додатків, наприклад - стрімінг спортивних подій у форматі 4К і вище, відстежування та прогнозування тайфунів, комплексне тестування нових продуктів, глибокий аналіз фінансових ринків і т.д.

Для створення HPC-кластеру виконується поєднання великої кількості окремих вузлів з високою обчислювальною спроможністю на базі мережі з високою пропускною здатністю.

При цьому, для того, щоб підтримувати високу швидкість виконання обчислень, кожна з компонент мережі повинна функціонувати синхронно з іншими.

Наприклад, операції запису та зчитування інформації, які реалізуються на базі компоненти системи зберігання даних, мають виконуватися таким чином, щоб не створювати при цьому затримки для обчислювальних вузлів і т.д.

Аналогічним чином мережа має виконувати швидку передачу дані між усіма компонентами кластеру HPC-типу.

У тому випадку, коли якась з компонент породжує затримку, це, у свою чергу, здатна помітно знизити загальну продуктивність роботи усього кластеру.

Зараз HPC-кластер може бути побудовано на базі багатьох технічних рішень, залежно від конкретних умов та завдань, які для нього мають бути першочерговими. Водночас HPC-кластер має типову архітектуру, як показано рис.1.4.

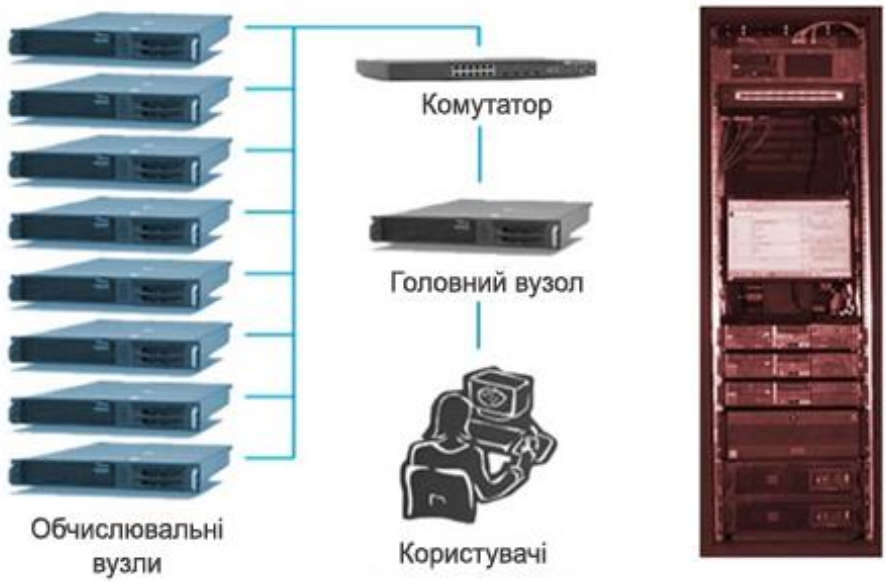

Рисунок 1.4 – Типова архітектура HPC-кластеру

2 РОЗГОРТАННЯ ОБЧИСЛЮВАЛЬНОГО КЛАСТЕРУ НА БАЗІ HADOOP

2.1 Поняття кластерної обчислювальної системи

За умовами завдання, кластер Hadoop має бути побудовано на базі Linux Ubuntu. До складу кластеру повинно входити 3 обчислювальних вузла.

При цьому, вимог до мережевого середовища, а також способу спільного розміщення окремих вузлів висунуто не було.

Відтак, далі буде розглядатися приклад розгортання Hadoop-кластеру на базі мережі локальної мережі 100 Base-T, де обчислювальні вузли локалізовано у межах одного сегменту.

У нашому випадку, оскільки вузлів 3, один з них, а саме - 192.168.1.15, буде конфігуруватися як керуючий, або master.

Тоді 2 інші вузли - 192.168.1.20 та 192.168.1.25 будуть керованими вузлами (slave).

Як для кластеру Hadoop з трьох вузлів, так і для взагалі будь-якого масштабу, процес розгортання складається з ряду типових технологічних етапів, серед яких [13]:

1. Підготовка ОС
2. Інсталяція Hadoop, що включає у себе:

− завантаження архіву;
− налаштування середовища оточення;
− перевірка роботи середовища оточення;
− створення сертифікатів для SSH.

3. Налаштування та запуск Hadoop.
4. Тестовий запуск.
5. Налаштування автозапуску.

Виконаємо далі детальний огляд кожного з зазначених технологічних етапів.

2.2 Підготовка операційної системи

Етап підготовки операційної системи виконується за аналогічним сценарієм на усіх вузлах, які планується інтегрувати у кластер Hadoop.

2.2.1 Оновлення пакетів

Даний етап є стандартним та обов’язковим взагалі для будь-якого прикладного або системного ПЗ у середовищі Linux. Це зумовлено тим, що не оновлений перелік пакетів далі може спричинити виникнення помилок у ході інсталяції програмного забезпечення.

Процедура оновлення списку пакетів використовується командою [9]:

```
apt update
```

Водночас, рекомендовано оновити встановлені пакети (актуально для чистих систем), що може бути виконано командою:

```
apt upgrade
```

Для можливості взаємодії вузлів між собою, далі виконується налаштування брандмауеру.

2.2.2 Конфігурування брандмауера

У першу чергу, коректна робота кластеру, як системи взаємозв’язаних вузлів, потребує відкриття портів як для обміну даними, так і для управління. Це такі порти, як [14]:

− 9870 - веб-інтерфейс для управління;

− 8020 - RPC адреса для клієнтських підключень;

− 9866 - DataNode (передача даних);

− 9864 - DataNode (http-сервіс);

− 9867 - DataNode (IPC-сервіс).

Відкриття портів, при цьому, виконується шляхом конфігурування правил iptables.

У нашому випадку для цього необхідно виконати наступні команди:

```
iptables -I INPUT -p tcp --dport 9870-j ACCEPT
iptables -I INPUT -p tcp --dport 8020-j ACCEPT
iptables -I INPUT -p tcp --match multiport --dports 9866,9864,9867 -j ACCEPT
```

2.2.3 Налаштування імен у кластері

Вузли створюваного кластера повинні вміти звертатися один до одного за ім'ям. У продуктивному середовищі для цього зазвичай використовується DNS [13, 15].

Разом з тим, оскільки за замовчуванням DNS у нашому випадку відсутній, необхідно на кожен вузол додати файл hosts, що міститиме інформацію про усі вузли кластера. Стосовно нашому випадку це наступний запис:

```
vi /etc/hosts
#127.0.1.1 hadoop1
192.168.1.15 hadoop1 192.168.1.20 hadoop2 192.168.1.25 hadoop3
```

Як бачимо, у прикладі, розміщеному вище вище, рядок 127.0.1.1 закоментовано. Якщо цього не зробити, сервер буде запускатися на локальній адресі 127.0.1.1, тому вторинні сервери не зможуть підключитися до головного (майстра).

2.2.4 Інсталяція середовища Java

Hadoop створено на базі мови програмування Java, з цієї причини на усіх вузлах кластеру має бути встановлено саме цю платформу. Для інсталяції Java у нашому випадку виконується наступна команда [9]:

```
apt install default-jdk
```

Далі, для перевірки коректності виконаної інсталяції Java, може бути виконано запит версії встановленої платформи, для чого слугує команда:

```
java-version
```

При цьому, якщо інсталяція була виконана успішно, на консоль буде виведено інформацію виду:

```
openjdk version "11.0.13" 2021-10-19
OpenJDK Runtime Environment (build 11.0.13+8-Ubuntu-
0ubuntu1.20.04)
OpenJDK 64-Bit Server VM (build 11.0.13+8-Ubuntu-
0ubuntu1.20.04, mixed mode, sharing)
```

2.3 Інсталювання Apache Hadoop

Процес інсталяції здійснюється у ручному режимі. Так, у першу чергу виконується процедура завантаження файлу із сайту розробника. Далі інсталяційний файл розміщується на сервері, після чого створюються файли зі змінними оточення. Наступним кроком виконується налаштування автозапуску, використовуючи при цьому systemd.

Означений перелік дій виконується у межах кожного з серверів майбутнього кластера [11, 13].

Окрім цього, необхідно забезпечити можливість з'єднання на базі ssh до усіх без винятку серверів кластеру.

2.3.1 Завантаження вихідного файлу

З міркувань безпеки рекомендується виконувати завантаження дистрибутиву з офіційного сайті розробника (https://hadoop.apache.org/releases.html), де у першу чергу слід обрати відповідну версію програмного забезпечення (за замовчуванням вибирається найновіша, рис.2.1).

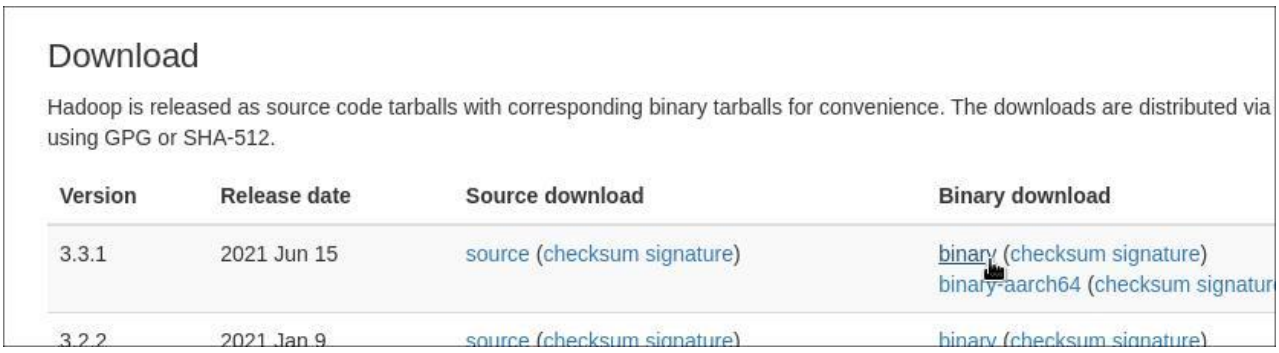

Рисунок 2.1 – Вибір версії дистрибутиву для завантаження

Далі потрібно скопіювати посилання на завантаження необхідного у конкретному випадку архіву (рис.2.2).

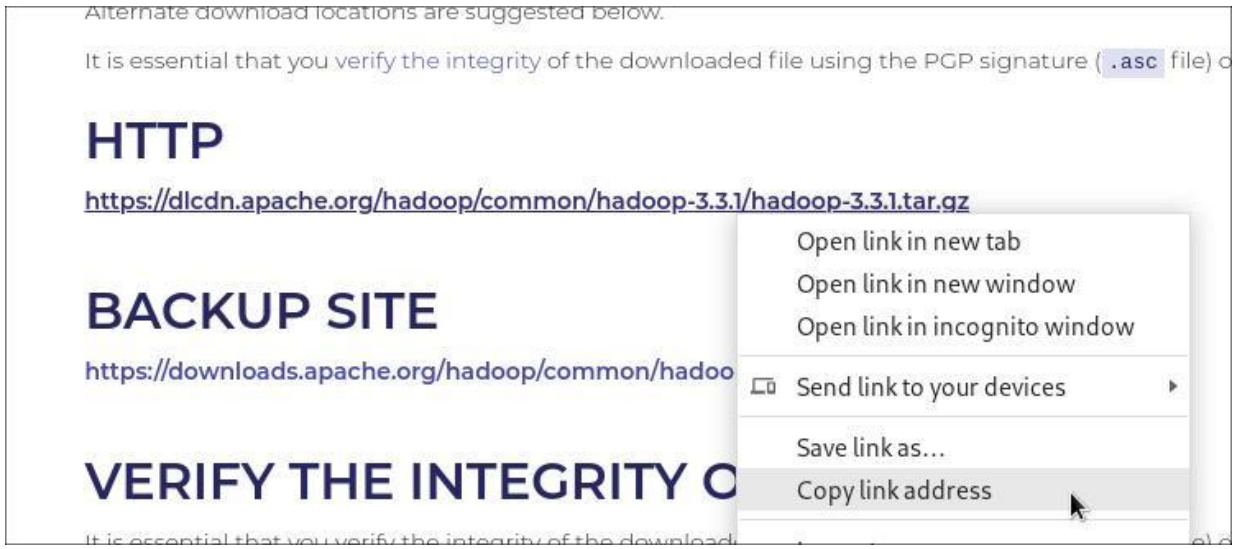

Рисунок 2.2 – Копіювання посилання на завантаження

Після цього, використовуючи скопійоване раніше посилання, виконується завантаження архіву на сервери, використовуючи наступну команду (для нашого випадку):

2.3.2 Інсталяція та налаштування середовища

Перш ніж виконувати будь-які дії з завантаженими файлами, необхідно виконати створення каталогу, де буде розміщено файли. Для цього використаємо наступну команду:

```
mkdir /usr/local/hadoop
```

У наслідок виконання наступної команди вміст завантаженого архіву буде розпаковано до попередньо створеної директорії:

```
tar -zxf hadoop-*.tar.gz -C /usr/local/hadoop --strip-components 1
```

Подальший крок розгортання системи потребує створення користувача hadoop. У свою чергу, щоб створити користувача, достатньо виконати наступну команду:

```
useradd hadoop –m
```

Після цього виконується встановлення паролю для створеного користувача за допомогою наступної команди:

```
passwd hadoop
```

Далі щойно створений користувач призначається власником директорії hadoop, що може бути виконано наступним чином [9]:

```
chown -R hadoop:hadoop /usr/local/Hadoop
```

На наступному кроці налаштування необхідно створити файл із профілем для поточної конфігурації hadoop. Типовий приклад файлу профілю наводиться далі:

```
vi /etc/profile.d/hadoop.sh
export HADOOP_HOME=/usr/local/hadoopexport
HADOOP_HDFS_HOME=$HADOOP_HOMEexport
HADOOP_MAPRED_HOME=$HADOOP_HOMEexport
HADOOP_COMMON_HOME=$HADOOP_HOMEexport
HADOOP_COMMON_LIB_NATIVE_DIR=$HADO TS -
Djava.library.path=$HADOOP_HOME/lib/native"export YARN_HOME
=$HADOOP_HOMEexport
PATH="$PATH:${HADOOP_HOME}/bin:${HADOOP_HOME}/sbin"
```

Далі виконаємо огляд складу файлу профілю, який наведено вище.

У першу чергу, тут задається ряд системних змінних, які необхідні для функціонування hadoop, а саме [13]:

– HADOOP_HOME вказує шлях, де розміщено файли hadoop;

– HADOOP_HDFS_HOME встановлює директорію, де розміщено розподілену файлову систему HDFS;

– HADOOP_MAPRED_HOME локація, яка використовується для відправлення завдання MapReduce за допомогою MapReduce v2 YARN;

– HADOOP_COMMON_HOME - шлях зберігання файлів для модулю common;

– HADOOP_COMMON_LIB_NATIVE_DIR – локація для розташування бібліотеки native-hadoop;

– HADOOP_OPTS – встановлює додаткові опції запуску системи;

– YARN_HOME – містить шлях розміщення файлів модулю YARN;

– PATH - доповнює загальну змінну PATH, де зберігаються шляхи до бінарних файлів, що використовуються для запуску виконуваних файлів.

Після того, як файл профілю налаштовано, встановити актуальний шлях файлів openjdk. У нашому випадку це шлях /usr/lib/jvm/java-11-openjdk-amd64. Для того, щоб мати можливість виконати дану операцію, попередньо необхідно відкрити файл hadoop-оточення за допомогою наступної команди:

```
vi /usr/local/hadoop/etc/hadoop/hadoop-env.sh
```

Далі у файлі hadoop-env.sh необхідно знайти рядок, у якому вказується шлях до openjdk. За замовчуванням це – рядок виду # export JAVA_HOME=. Запис в означеному рядку необхідно змінити до вигляду, який відображає реальне розміщення openjdk. У нашому випадку це:

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
```

2.3.3 Перевірка попереднього налаштування середовища

Операцію перевірки середовища, налаштованого раніше, може бути виконано виключно з облікового запису раніше створеного користувачем hadoop, відтак, спочатку виконується вхід, для чого слугує наступна команда:

```
su – hadoop
```

Потім вже з користувацького облікового запису виконується команда:

```
$env | grep -i -E "hadoop|yarn"
```

При цьому, якщо усі передуючі кроки налаштування було виконано вірно, на консоль буде виведено інформацію наступного вигляду [13]:

```
MAIL=/var/mail/hadoopUSER=hadoopHADOOP_COMMON_HOME=/usr/local
/hadoopHOME=/home/hadoopHADOOP_COMMON_LIB_NATIVE_DIR=/usr/local/ha
doop/lib/nativeLOGNAME=hadoopPATH=/usr/local/sbin
/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bi
```

```
n:/usr/local/hadoop/bin:/usr/local/hadoop/sbinYARN_HOME
=/usr/local/hadoopHADOOP_MAPRED_HOME=/usr/local/hadoopHADOOP_HDFS_
HOME=/usr/local/hadoopHADOOP_HOME=/usr/local/Hadoop
```

Окрім перевірки попереднього налаштування у зазначений спосіб, може бути, наприклад, виконано запит поточної версії hadoop, для чого слугує наступна команда:

```
$ hadoop version
```

У нашому випадку отримуємо наступний результат виконання команди, зазначеної вище [13]:

```
Hadoop 3.3.1Source code repository
https://github.com/apache/hadoop.git -r
a3b9c37a397ad4188041dd80621bdeefc46885f2
a4ddb2299aca054416d6b7f81ca55This command was run using /usr/local/
hadoop/share/hadoop/common/hadoop-common-3.3.1.jar
```

Виконання усіх подальших дій, повязаних з конфігуруванням системи, також потребує перебування під обліковим записом користувача hadoop.

2.3.4 Створення сертифікатів безпеки

Функціонування кластеру hadoop неможливе без використання сертифікатів базпеки.

Це зумовлено тим, що усі внутрішні звернення здійснюються на базі запитів ssh.

У свою чергу, генерування сертифікату виконується на одному з серверів, після чого його необхідно скопіювати на інші вузли.

Для цього, відповідно, на головному сервері кластеру (майстер-сервері) створюються ключі безпеки. Для цього використовується наступна команда:

```
$ssh-keygen
```

Після цього публічний ключ копіюється на локальний комп'ютер, для чого виконується команда, як показано далі:

```
$ssh-copy-id localhost
```

Далі, у ході першого звернення за SSH, буде виконуватися запит на прийняття сертифіката:

```
Would you like to get connecting (yes/no/[fingerprint])? (Yes)
```

Після цього системою буде видано запит на введення пароля поточного користувача hadoop.

Наступний крок налаштування передбачає процедуру копіювання потрібних ключів на інші вузли кластеру, виконавши для цього наступні команди:

```
$ scp -r .ssh hadoop@haddop2:~
$ scp -r .ssh hadoop@haddop3:~
```

Розглянемо лістинг команд, наданий вище. Тут директорія .ssh копіюється до вузлів haddop2 та haddop3, які у нашому випадку є керованими серверами (slave).

Коли процедуру копіювання директорії .ssh буде завершено, необхідно виконати перевірку можливості входу до системи через ssh на усі сервери кластеру.

У цьому разі, підключення до вузлів повинно здійснюватися без запиту пароля. Найпростіше це зробити, виконавши команду на рівні локального вузла:

```
$ssh localhost
```

Дані з'єднання завершується:

```
$ exit
```

У свою чергу, перевірити підключення до двох інших двох серверів можливо аналогічним чином, тобто:

```
$ ssh haddop2 (haddop3)
$ exit
```

Розглянуті операції є фінальними у ході інсталяції та первинного налаштування Hadoop. У свою чергу, для того, щоб мати можливість виконувати подальші дії, попередньо слід повернутися до консолі первинного користувача, виконавши команду виходу (exit).

2.4 Налаштування та запуск Hadoop

Перш, ніж виконувати тестовий запуск hadoop, та далі налаштовувати сервіс для автозапуску, необхідно виконати редагування ряду конфігураційних файлів у межах усіх вузлів кластера. Дана міра є обов'язковою, так як передбачає виконання ряду критичних опцій, ігнорування яких унеможливлює функціонування системи у цілому. Виконаємо детальний огляд цих налаштувань.

Налаштування Hadoop, які передують запуску

Даний етап конфігурування передбачає внесення параметрів, від яких залежить запуск та функціонування кластеру. При цьому, конфігуруються такі файли, як [13]:

– core-site.xml, де містяться загальні налаштування;

– hdfs-site.xml, у якому знаходиться конфігурація файлової системи HDFS;

– mapred-site.xml для налаштування MapReduce;

– yarn-site.xml, для налаштування фреймворку управління кластером за замовчуванням.

У першу чергу, на даному етапі налаштувань конфігуруванню підлягає файл загальних налаштувань core-site.xml, що за замовчуванням знаходиться у наступній локації:

```
/usr/local/hadoop/etc/hadoop/core-site.xml
```

Спершу до файлу вноситься запис про вузол та порт звернення до внутрішньої файлової системи, як показує наступний фрагмент змісту файлу:

```
...<!-- Put site-specific property overrides in this file. -->
```

```
<configuration>  <property>  <name>fs.default.name</name>
<value>hdfs://hadoop1:9000</value> </property></configuration>
```

У наведеному лістингу вузол та порт звернення до внутрішньої файлової системи задається параметром fs.default.name. У нашому прикладі для майстер-серверу (localhost) встановлюється порт 9000. Аналогічне налаштування має бути на усіх інших вузлах кластеру.

На наступному кроці налаштування hadoop виконується редагування файлу hdfs-site.xml. Як і у випадку редагування файлу core-site.xml, тут також доцільно використати текстовий редактор за замовчуванням (наприклад, vi):

```
vi /usr/local/hadoop/etc/hadoop/hdfs-site.xml
```

У файлі hdfs-site.xml необхідно налаштувати такі параметри, як:

– dfs.replication - кількість реплік; не може бути більше вузлів кластеру;

– dfs.name.dir - шлях зберігання таблиці імен fsimage;

– dfs.data.dir – директорія, де мають зберігатися блоки файлової системи HDFS.

Результат налаштування файлу у нашому випадку маємо наступний:

```
<configuration> <property> <name>dfs.replication</name>
<value>3</value> </property> <property>
<name>dfs.name.dir</name> <value>file://
/hadoop/hdfs/namenode</value> </property> <property>
<name>dfs.data.dir</name>
<value>file:///hadoop/hdfs/datanode</value>
</property></configuration>
```

Далі для редагування відкривається файл налаштування MapReduce, тобто - mapred-site.xml. Дана операція реалізується аналогічно до попередньої [9, 11, 13]:

```
vi /usr/local/hadoop/etc/hadoop/mapred-site.xml
```

У зазначеному файлі вказується фреймворк, який буде використано для управління кластером. За це відповідає параметр mapreduce.framework.name. У

нашому випадку використовується Yarn, як засіб за замовчуванням, у результаті маємо:

```
    <configuration>                                          <property>
<name>mapreduce.framework.name</name>          <value>yarn</value>
</property></configuration>
```

Оскільки для управління кластером обрано Yarn, далі виконується налаштування зазначеного фреймворку. Для цього редагується конфігураційний файл yarn-site.xml, де слід встановити тип обслуговування кластеру, за що відповідає параметр yarn.nodemanager.aux-services. Оскільки кластер конфігурується для рішення універсальних завдань, доцільно обрати тип mapreduce_shuffle. Відповідний запис у файлі yarn-site.xml буде наступним:

```
    <name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value> </property></configuration>
```

Після налаштування конфігураційних файлів, створюються директорії, виділені для файлової системи HDFS. Ураховуючи параметри, актуальні у нашому випадку, команда для даної операції буде наступною:

```
mkdir -p /hadoop/hdfs/{namenode,datanode}
```

Врешті решт, раніше створеному користувачеві hadoop встановлюється роль власнику для каталогу /hadoop. Для цього слугує така команда:

```
chown -R hadoop:hadoop /hadoop
```

Дана операція є фінальною для первинного налаштування серверів кластеру. Після цього залишається лише відкрити файл workers, де має бути перелічено slave-вузли, та зробити відповідний запис:

```
vi /usr/local/hadoop/etc/hadoop/workers
haddop2
haddop3
```

2.5 Тестовий запуск кластеру

Дану операцію необхідно виконувати з облікового запису користувача hadoop. Перш, ніж виконати запуск, має бути створено файлову систему HDFS, що виконується командою [9, 11]:

```
$ /usr/local/hadoop/bin/hdfs namenode –format
```

Безпосередньо запуск кластера ініціюються наступними командами:

```
$ /usr/local/hadoop/sbin/start-dfs.sh
$ /usr/local/hadoop/sbin/start-yarn.sh
```

Залежно від параметрів мережі та кожного окремого вузла, повне завантаження java-програми може тривати 10-20 секунд. Після цього може бути виконано звернення до головного сервера кластеру через веб-інтерфейс, використовуючи звичайний браузер. У нашому випадку використовується наступна адреса:

```
http://192.168.1.15:9870
```

У разі успішного запуску кластера, до браузеру буде завантажено веб-інтерфейс, як показано рис.2.3.

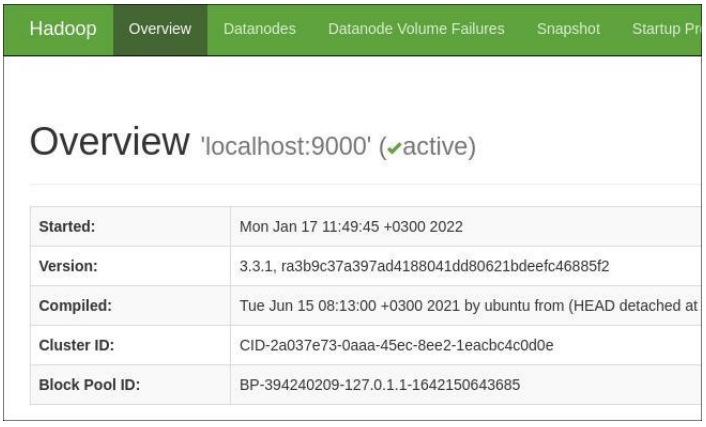

Рисунок 2.3 – Веб-інтерфейс управління кластером Hadoop

При цьому, вкладка Datanodes міститиме усі slave-вузли.

2.6 Налаштування автозапуску Hadoop

Для автоматичного запуску кластеру, необхідно забезпечити автозавантаження Hadoop, як сервісу. Ця процедура передбачає створення та подальше налаштування конфігураційного файлу hadoop.service на головному сервері.

Даний файл необхідно створити у локації /etc/systemd/system/hadoop.service, після чого внести стандартну конфігурацію, рекомендовану Apache, що відповідає за автозавантаження:

```
[Unit]Description=Hdfs serviceAfter=network.target

[Service]Type=forkingUser=hadoopGroup=hadoopExecStart=
/usr/local/hadoop/sbin/start-
all.shExecStop=/usr/local/hadoop/sbin/stop-all.sh
ExecReload=/bin/kill -HUP $MAINPIDRestart=on-failure

[Install]
WantedBy=multi-user.target
```

Далі необхідно оновити конфігурацію підсистеми ініціалізації та управління службами systemd, для чого використовується стандартна команда:

```
systemctl daemon-reload
```

Після цього встановлюється дозвіл автозапуску:

```
systemctl enable hadoop
```

3 ЗАГАЛЬНІ ПІДХОДИ ДО НАЛАШТУВАННЯ ВУЗЛІВ КЛАСТЕРУ HADOOP У КОНТЕКСТІ ПРИНЦИПУ MAPREDUCE

3.1 Загальна стратегія налаштування кластеру

Головним завданням налаштування вже розгорнутого обчислювального кластеру є забезпечення його максимальної продуктивності. При цьому, якщо мова йде про кластер, який має вирішувати будь-які завдання, його налаштування має виконуватися з оглядом на забезпечення найбільшої швидкості операцій Map/Reduce. Як вже зазначалося, продуктивність Hadoop залежить від багатьох факторів, головними з яких є [15, 16]:

– коректне налаштування ПЗ управління кластером;

– апаратні ресурси вузлів кластеру та специфіка налаштування використання кожного вузла;

– пропускна здатність мережі.

Частіше за все, визначити доцільні параметри кластеру можливо на базі приблизного обсягу обчислень, які планується виконати протягом деякого часу. У цьому випадку визначається:

– розмір кластеру;

– перелік параметрів, що можуть створити умови для його найбільшої продуктивності.

Планування кластера Hadoop є складним завданням, яке, у цілому, окрім коректного рішення щодо вибору розміру кластера, має вирішити такі завдання з планування, як, наприклад:

– планування сховищ даних;

– процесорні ресурси кластера;

– оперативна пам'ять вузлів кластера.

Розглянемо, яким чином визначається коректний розмір кластера.

3.2 Визначення необхідного розміру кластера Hadoop

3.2.1 Огляд архітектури кластеру Hadoop

У ході визначення розміру кластеру Hadoop слід ураховувати обсяг даних, які будуть обробляти кінцеві користувачі. Знаючи обсяг таких даних, можна визначити, скільки саме вузлів має бути інтегровано до кластеру для ефективної

обробки вхідних даних, а також розрахувати ємність файлосховища та оперативної пам'яті кожного з них [13].

У свою чергу, Hadoop являє собою платформу, побудовану за архітектурою Master/Slave, яка потребує великої кількості пам'яті та ресурсів ЦП. Архітектура Hadoop складається з таких основних компонентів, як [12, 13] :

– Job Tracker; дана компонента відстежує завдання, які виконуються у кластері;

– Task Tracker; завданням зазначеної компоненти є запуск завдання на кожному вузлі кластера.

Разом з тим, для ефективної роботи системи HDFS, необхідно використовувати жорсткі диски з високою пропускною здатністю, а також базовою файловою системою, здатною підтримувати шаблон читання та запису HDFS (з блоком великого розміру). У рамках такого шаблону передбачається можливість виконання однієї великої операції читання (або запису) одночасно з блоком розміром 64 МБ, 128 МБ і до 256 МБ.

При цьому, система HDFS також має в основі архітектуру Master/Slave, та складається з двох основних компонент, а саме - NameNode/Secondary NameNode та DataNode.

Тут компонента NameNode забезпечує коректну реплікацію блоків даних у кластері, тоді як компонента DataNode здійснює управління станом вузла HDFS, а також взаємодіє з його блоками даних. Для обробки та передачі даних потрібен великий обсяг операцій введення-виведення.

Кожна з зазначених компонент є критично важливими. Вони потребують багато пам'яті для зберігання метаінформації файлу, такої, як атрибути і локалізація файлу, структура каталогів, імена, а також для обробки даних.

Зазвичай, для забезпечення продуктивності обробки даних за принципом Map/Reduce мають виконуватися такі основні вимоги [15]:

– вхідні набори даних повинні мати достатньо великий розмір для того, щоб заповнити блок даних, а також далі розбиватися на дрібніші та незалежні між собою фрагменти даних (наприклад, текстовий файл розміром 10 ГБ можна розділити на 40 960 блоків по 256 МБ кожен; кожен рядок тексту у будь-якому блоці даних може оброблятися незалежно);

– має враховувати локальність даних, тобто, код обробника MapReduce необхідно переміщувати туди, де знаходяться дані, а не навпаки (ефективніше перемістити кілька мегабайт коду так, щоб він був ближче до даних, що

обробляються), переміщувати велику кількість блоків даних по мережі або диску). Необхідність виконання даної вимоги зумовлює наявність розподіленої системи зберігання, що забезпечує локальність даних, а також дає змогу виконувати код на будь-якому вузлі зберігання.

Якщо ж говорити про пропускну здатність мережі, зазначимо, що вона має критичне значення виключно у двох випадках, зокрема:

– у процесі реплікації, а також після запису файлу;

– у ході реалізації процедури балансування коефіцієнта реплікації, яка, у свою чергу, виконується у разі збою вузла.

При цьому, частіше за все використовується підхід до визначення розміру кластеру Hadoop, який базується на величині необхідного обсягу сховища. Згідно даного підходу, чим більше даних надходить у систему, тим більше вузлів необхідно задіяти у кластері. Відтак, кожного разу, коли до кластеру інтегрується новий вузол, кластер отримує більше обчислювальних ресурсів на додаток до нової ємності сховища.

Розглянемо приклад плану розширення кластеру з урахуванням загального обсягу сховища даних та дослідимо існуючий метод розрахунку необхідного розміру сховища, обсягу пам'яті та кількість вузлів даних у межах кластеру. (табл.3.1).

3.2.2 Формування плану розширення кластеру

Усі процеси у галузі обробки Big Data взагалі і, зокрема, у галузі кластерних обчислень, повністю корелюються за багатьма показниками з процесами, які мають місце у глобальному мережевому середовищі. У першу чергу – це стосується постійного росту даних, які підлягають обробці.

Таким чином, обчислювальний кластер, у загальному випадку, має розглядатися, як динамічна структура, масштаби якої, а також склад та параметри кожного вузла, можуть змінюватися з часом [16].

При цьому, у будь-якому випадку необхідно першочергово виконати оцінку об'єму V^{*}_{daily} пам'яті, яка використовується для щоденного введення даних, беручи до уваги реплікацію (якщо її передбачено), для цього може бути використано наступний вираз [10, 11]:

$$V^{*}_{daily} = V_{daily} \times r, \quad (3.1)$$

де r - коефіцієнт реплікації.

У свою чергу, використовуючи результати обчислень за виразом (3.1), щомісячний обсяг V^{*}_{mon} обсяг даних, ураховуючи наявне їх зростання ΔV, може бути розраховано наступним чином:

$$V^{*}_{mon} = V_{daily} \times n + \Delta V \times \left(\frac{V_{daily} \times n}{100}\right), \quad (3.2)$$

де n - кількість днів у місяці.

За аналогією з виразом (3.2), річний об'єм даних V^{*}_{yr} можемо розрахувати як:

$$V^{*}_{yr} = V^{*}_{mon} \times \left(1 + \frac{r}{100}\right) \times 12. \quad (3.3)$$

Очевидно, що у будь-якому випадку на вузлі кластеру, який має у розпорядженні загальну ємність V_{full} файлосховища, для виконання кластерних обчислень буде виділено лише деяку її частину V_{clst}. При цьому, для власних потреб за вузлом буде зарезервовано певний об'єм дискового простору V_{free} без HDFS. Водночас, для зберігання проміжних даних MapReduce на кожному вузлі також необхідно виділити деякий об'єм V_{mr} дискового простору. З урахуванням цього, ємність V_{clst} вузла, яку задіяно безпосередньо для виконання кластерних обчислень, обчислюється за наступним виразом:

$$V_{clst} = V_{full} \times (1 - V_{free} - V_{mr}). \quad (3.4)$$

Тепер, виконавши попередні розрахунки, можемо обчислити кількість DataNodes, які необхідна для обробки даних протягом різних періодів часу.

Так, для обробки даних за перший місяць необхідно $N^{(1)}_{dn}$ вузлів DataNode, що розраховується за формулою:

$$N_{dn}^{(1)} = ceil(\frac{V_{mon}^{*}}{V_{clst}}). \quad (3.5)$$

Відповідно, на базі даних, отриманих з виразу (3.5), далі може бути обчислено об'єм $N_{dn}^{(12)}$ даних за 12-й місяць, як відношення річного об'єму даних до ємності вузла, яку виділено для кластерних обчислень, тобто:

$$N_{dn}^{(12)} = \frac{V_{yr}^{*}}{V_{clst}}. \quad (3.6)$$

За умовами завдання, у ході роботи кластеру, щоденно має оброблятися обсяг даних V_{daily}, рівний 100 ГБ. При цьому, коефіцієнт r реплікації дорівнює 3.

У свою чергу, передбачається щомісячне зростання обсягу ΔV оброблюваних даних на 5%.

Разом з тим, також було зазначено, що проміжні дані V_{mr} обчислень MapReduce можуть займати до 25% дискового обсягу.

Водночас, 30% об'єму V_{free} дискового пристрою не буде містити системи HDFS.

Отже, за виразом (3.1), для можливості щоденної обробки даних з урахуванням реплікації маємо $V_{daily}^{*} = 100 \times 3 = 300$ ГБ.

Відтак, урахувавши отримані дані, та відомості щодо темпів росту обсягу оброблюваної інформації, згідно формули (3.2) отримуємо значення щомісячного обсягу даних (при $n = 30$): $V_{mon}^{*} = 300 \times 30 + 5 \times (\frac{300 \times 30}{100}) = 9450$ ГБ.

Далі, маючи відомості про місячний об'єм, розраховуємо річний об'єм даних, які підлягають обробці, за формулою (3.3). Тут маємо $V_{yr}^{*} = 9450 \times (1 + \frac{5}{100}) \times 12 = 16971$ ГБ.

Тепер, щоб обчислити необхідну кількість DataNodes у різні періоди часу, попередньо виконаємо оцінку дискового простору одного вузла, виділеного для

кластерних обчислень, за виразом (3.4). Отримуємо $V_{clst} = 4 \times (1 - 0{,}25 - 0{,}3) = 1{,}8$ ТБ.

Отже, у зазначених умовах за перший місяць буде необхідно $N_{dn}^{(1)} = \text{ceil}(\frac{9450}{1800}) = 6$ вузлів DataNode.

Врешті-решт, на 12-й місяць для підтримки функціональності кластеру в існуючій ситуації буде необхідно розгорнути $N_{dn}^{(1)} = \text{ceil}(\frac{16971}{1800}) = 10$ вузлів DataNode.

3.2.3 Обчислення необхідного об’єму оперативної пам'яті для вузлів

Виконаємо розрахунок обсягу $B_{NameNode}$ оперативної пам'яті, який є необхідним як NameNode, так і для Secondary NameNode. Зазвичай обсяг пам'яті, потрібний для Secondary NameNode, має бути ідентичним і для NameNode. Дану величину може бути обчислено на базі наступного виразу:

$$B_{NameNode} = B_{cmd} + B_{sys} + B_{ip}, \quad (3.7)$$

де B_{cmd} - пам'ять, необхідна NameNode для керування метаданими кластеру;

B_{sys} - пам'ять, яку необхідно виділити для ОС;

B_{ip} - внутрішня пам'ять NameNode.

Важливо зазначити, що об’єм пам'яті B_{sys}, яку необхідно виділити NameNode для ОС, має дорівнювати сумі обсягів пам’яті B_{cmd} та внутрішньої пам'яті NameNode, тобто:

$$B_{sys} = B_{cmd} + B_{ip}. \quad (3.8)$$

Тоді ураховуючи, що за умовами завдання $B_{cmd} = B_{ip} = 2$ ГБ, за виразами (3.7) та (3.8) отримуємо $B_{NameNode} = 2 + 4 + 2 = 8$ ГБ.

Тепер виконаємо розрахунок обсягу пам'яті, необхідного для DataNode. У даному разі на обсяг пам'яті впливає також кількості ядер фізичного процесору

встановленого на кожному вузлі даних. Загальний вираз, на базі якого можемо обчислити об'єм $B_{DataNode}$ пам'яті, наводиться далі:

$$B_{DataNode} = B_{core} \times \eta + B_{pr} + B_{tt} + B_{sys}, \quad (3.9)$$

де η - кількість ядер процесору;

B_{core} - обсяг пам'яті, виділений на одне ядро;

B_{pr} - об'єм пам'яті, необхідний для обслуговування процесів DataNode;

B_{tt} - пам'ять, що виділяється для DataNode TaskTracker;

B_{sys} - пам'ять, яку необхідно виділити для ОС.

Таким чином, для випадку 4-ядерного процесору, коли, наприклад, $B_{core} = B_{pr} = B_{tt} = B_{sys} = 2$ ГБ, за виразом (3.9) отримуємо $B_{DataNode} = 4 \times 4 + 4 + 4 + 4 = 28$ ГБ.

Що стосується розрахунку потужності ЦП, на сьогодні уніфікованого методу не існує. Натомість існуючі рекомендації зводяться до необхідності використання на вузлах DataNode сучасних багатоядерних ЦП, які мають, щонайменше, чотири фізичні ядра на ЦП. Чим більше фізичних ядер ЦП, тим більше може бути підвищена продуктивність функціонування вузла.

3.3 Оптимізація вузлів кластеру для завдань Map і Reduce

Для того, щоб отримати максимальну продуктивність кластеру Hadoop, потрібно його коректно налаштувати.

Разом з тим, на сьогодні не існує універсального сценарію конфігурування кластеру для того, щоб отримати оптимальну швидкодію. У той же час, 2 існуючі сьогодні підходи до оптимізації передбачають:

- адаптацію платформи Hadoop до існуючого кластера;
- адаптацію платформи Hadoop для конкретного завдання.

Так, для оптимального налаштування кластеру, спочатку рекомендується виконати запуск завдання Hadoop з конфігурацією за замовчуванням, що дозволяє отримати базові показники продуктивності.

Далі виконується аналіз файлів журналів історії завдань, а також кількісні показники процесу (насамперед час, витрачений на виконання завдань). Після

цього виконується ітеративне налаштування конфігурації Hadoop з наступним повторним запуском завдання до того моменту, поки не буде отримано конфігурацію, що забезпечує найвищу продуктивність.

При цьому, важливим є співвідношення кількості завдань Map та Reduce, які мають виконуватися під час вирішення тієї чи іншої задачі. Зазвичай, кількість завдань типу Reduce має бути меншою, ніж кількісті завдань Map. Так, Google вважає оптимальним зіставляти одне завдання Reduce з 20 завданнями Map, хоча зараз існують і інші рекомендації. Таким чином, загальну стратегію розподілу модулів Map та Reduce може бути пояснено наступним виразом [7]:

$$\begin{gathered} N_{Map} \rightarrow M_{Reduce}; \\ M > N. \end{gathered} \tag{3.10}$$

У свою чергу, завдання Map найчастіше пов'язані з обробкою великих масивів даних, результати обробки яких передаються Reduce.

У той же час, нерідко завдання Reduce зводиться до виконання агрегатної функції, яка обробляє меншу частину даних порівняно з завданнями Map. Крім того, необхідно забезпечити коректну кількість модулів Reduce.

Зазначимо, що, кількість модулів Map і Reduce залежить від кількості фізичних ядер у DataNode, що, у свою чергу, визначає найбільшу кількість завдань, які може бути паралельно виконано DataNode.

Архітектура кластеру Hadoop передбачає, що головні вузли (master) зазвичай складаються з комп'ютерів, де один вузол налаштовано як NameNode, а інший - як JobTracker. Інші вузли у кластері є підлеглими (slave), які функціонують як DataNodes і TaskTracker.

Відтак, у ході запуску кластера починається запуск демонів HDFS на головному вузлі, а також демонів DataNode на усіх вузлах даних.

Після цього виконується запуск демонів MapReduce. А саме - JobTracker на головному вузлі, а запуск демонів TaskTracker виконується на усіх slave-вузлах.

Отже, при налаштуванні кластеру необхідно враховувати як ядра ЦП, так і ресурси пам'яті, які необхідно виділити цим демонам, що видно з аналізу псевдоформули демону Hadoop, як показано на рисунку 3.1

Так, у загальному випадку, за потреби обробки умовно істотних обсягів даних, рекомендується зарезервувати 2 ядра ЦП на кожному вузлі даних для демонів HDFS і MapReduce [6, 7, 12].

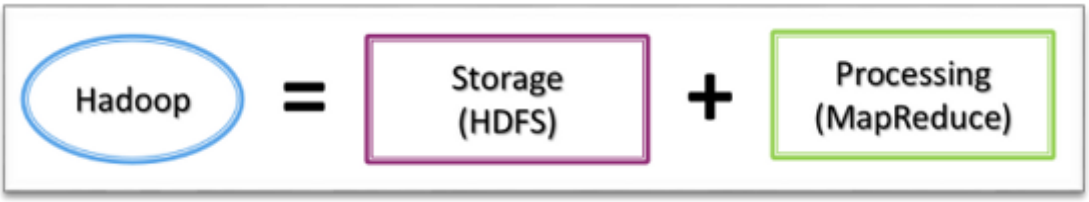

Рисунок 3.1 – Псевдоформула демону Hadoop

З іншого боку, у контексті малих та середніх даних рекомендується зарезервувати лише одне ядро ЦП на кожному з вузлів даних.

Далі після того, як було визначено максимальну кількість слотів меперів (модулів Map), потрібно визначити найбільшу кількість слотів Reduce.

Також можна зазначити, що існує розподіл між завданнями Map і Reduce на вузлах даних, за якого досягається висока продуктивність в умовах, коли

$$H_{Map} = H_{Reduce}$$ чи, принаймні, при $$H_{Map} = \frac{2H_{Reduce}}{3}.$$

Розглянемо процес коректного вибору кількості модулів Map і Reduce, при цьому, для прикладу, візьмемо кластер, параметри якого подано у табл. 3.1.

Таблиця 3.1 – Приклад конфігурації кластеру

Кластерна машина	Nb	Середній розмір даних	Великий розмір даних
Ядра ЦП DataNode	8	Зарезервувано 1 ядро ЦП	Зарезервувано 2 ядра ЦП
Демон DataNode TaskTracker	1	1	1
Демон DataNode HDFS	1	1	1
Кластерна машина	Nb	Середній розмір даних	Великий розмір даних
Розмір блоку даних		128 МБ	256 МБ
Відсоток використання ЦП DataNode		від 95% до 120%	від 95% до 150%
Кількість вузлів		20	40
Коефіцієнт реплікації		2	3

За великим рахунком для того, щоб забезпечити максимальну продуктивність, необхідно використовувати ресурси процесору щонайменше на 95 %.

Разом з тим, використання технології Hyper-Threading дає змогу одному ядру процесора обробляти більше, ніж одне завдання одночасно. Це дозволяє встановити діапазон коефіцієнта Hyper-Threading у діапазоні від 120 до 170 відсотків. З урахуванням цього, розглянемо метод розрахунку кількості модулів Map і Reduce у кластері.

Розрахунок необхідної кількості модулів Map і Reduce у кластері

Максимальну кількість слотів Map на одному вузлі у контексті великих даних може бути розраховано за наступною формулою:

$$\begin{gathered} H_{Map} = \text{round}(\eta_{fc} - \eta_{rc}) \times \gamma; \\ \gamma = \overline{0{,}95; 1{,}5}, \end{gathered} \tag{3.11}$$

де η_{fc} - кількість фізичних ядер процесору;

η_{rc} - кількість зарезервованих ядер процесору.

У прикладі табл. 3.1 зарезервовано 1 ядро для TaskTracker і одне - для HDFS. Також припустимо, що з використанням Hyper-Threading для процесора на вузлі буде встановлено коефіцієнт 120 %, тобто, $\gamma = 1{,}2$.

Таким чином, за формулою (3.11) максимальна кількість слотів Map $H_{Map} = \text{round}(8-2) \times 1{,}2 = 7$.

Далі, якщо буде використано техніку 2/3 Map/Reduce, тоді $H_{Reduce} = 7 \times \frac{2}{3} = 5$.

Далі, оскільки розрахунок кількості слотів виконується у контексті великих даних, сумарна кількість модулів H_{Map}^{clst} Map та модулів H_{Reduce}^{clst} дорівнюватиме $H_{Map}^{clst} = 7 \times 40 = 280$ та $H_{Reduce}^{clst} = 5 \times 40 = 200$.

Разом з тим, розмір блоку даних також впливає на підвищення продуктивності. Так, у конфігурації Hadoop за замовчуванням використовуються блоки розміром 64 МБ, у той час як для оптимізації обчислень часто пропонується використовувати в конфігурації 128 МБ для середніх даних і 256

МБ - для контексту дуже великих даних. Це означає, що завданням модулю Map може бути обробка одного блоку даних (наприклад, 128 МБ). У конфігурації Hadoop за замовчанням для обробки одного і того ж обсягу даних необхідно виконати дві задачі Map. Це можна розглядати як недолік, оскільки ініціалізація ще одного завдання Map і відкриття ще одного файлу займають більше часу.

4 ОПТИМІЗАЦІЯ HADOOP ДЛЯ РОЗГОРНУТОГО КЛАСТЕРА

4.1 Налаштування конфігураційних файлів Hadoop

Налаштування Hadoop виконуються шляхом маніпулювання параметрами конфігурації [15-20].

У свою чергу, параметри конфігурації можуть змінюватися у межах трьох файлів налаштувань, а саме:

1. conf/core-site.xml. У цьому файлі містяться опції, які є загальними для усього дистрибутиву Hadoop.

2. conf/hdfs-site.xml. Тут знаходиться конфігурація для HDFS.

3. conf/mapred-site.xml. Даний файл містить конфігурації для MapReduce.

При цьому, кожен файл конфігурації має формат XML, та містить у собі пари ім'я-значення, які визначають роботу різних аспектів Hadoop.

Розглянемо приклад фрагменту коду файлу конфігурації [15]:

```
<configuration><property><name>mapred.reduce.parallel.
copies</name><value>20</value></property>...
</configuration>
```

У даному фрагменті тег <configuration> є XML-контейнером верхнього рівня, а теги <property>, що визначають окремі властивості, є дочірніми елементами тега <configuration>.

Темер визначимо, яким чином можна налаштувати максимальну кількість завдань Map і Reduce, а також - змінити директорію, у якій будуть записуватися логи Hadoop.

Для рішення зазначеного завдання необхідно виконати такі технологічні кроки, як:

1. Створення директорії, де будуть зберігатися лог-файли. Наприклад, нехай це буде /root/hadoop_logs.

2. Зняття коментарю з рядоку, який містить параметр HADOOP_LOG_DIR, у файлі HADOOP_HOME/conf/hadoop-env.sh. Тут також необхідно зазначити ім’я нової директорії та вказати її локацію.

3. Додавання параметрів, які стосуються завдань Map і Reduce. Такі параметри являють собою відповідні рядки коду, приклад якого наведено далі. Даний код додається до файлу HADOOP_HOME/conf/mapred-site.xml, наприклад [15-18]:

```
<property><name>mapred.tasktracker.map.tasks.
maximum</name><value>2</value></property><property>
<name>mapred.tasktracker.reduce.tasks.maximum</name>
<value>2</value></property>
```

4. Перезавантаження кластеру Hadoop, для чого виконуються команди bin/stop-mapred.sh та bin/start-mapred.sh з каталогу HADOOP_HOME.

5. Перевірка обсягу створених процесів, керуючись при цьому інструментами моніторингу ОС.

У середовищі Linux це можливо, виконавши команду watch ps –ef|grep Hadoop. На випадок Windows, чи MacOS, може бути використано системний диспетчер завдань.

Тут параметр HADOOP_LOG_DIR встановлює локацію, де Hadoop веде запис лог-файлів.

При цьому, властивості mapred.tasktracker.map.tasks.maximum та mapred.tasktracker.reduce.tasks.maximum вказують найбільшу кількість завдань типів Map та Reduce, які може бути виконано у межах одного TaskTracker у поточний момент часу.

Усі перелічені вище, а також інші параметри на боці сервера, визначено у xml-файлах HADOOP_HOME/conf/*.

У свою чергу, Hadoop виконує перезавантаження даних конфігурацій після перезапуску системи.

Разом з тим, Hadoop дозволяє налаштовувати велику кількість параметрів конфігурації.

Деякі з таких параметрів буде наведено далі.

Наприклад, ключові параметри конфігурації, які знаходяться у файлі conf/core-site.xml, зазначено у таблиці 4.1.

Таблиця 4.1 – Головні параметри конфігурації у файлі conf/core-site.xml

Ім'я	Значення за замовчуванням	Опис
fs.inmemory.size.mb	100	Це обсяг пам'яті, виділений у файловій системі, який використовується для об'єднання вихідних даних Map у Reduce-модулях, МБ.
io.sort.factor	100	Максимальна кількість потоків, об'єднаних під час сортування файлів.
io.file.buffer.size	131072	Розмір буфера читання/запису

У свою чергу, параметри конфігурації, які розміщено у файлі conf/mapred-site.xml, наведено у табл. 4.2.

Таблиця 4.2 – Параметри конфігурації, які знаходяться у файлі mapred-site.xml

Ім'я	Значення за замовчуванням	Опис
mapred.reduce.parallel.copies	5	Найбільша кількість паралельних копій кроку reduce, які будуть виконуватися для отримання вихідних даних множини паралельних завдань.
mapred.map.child.java.opts	-Xmx200M	Для передачі параметрів Java до Map JVM
mapred.reduce.child.java.opts	-Xmx200M	Для передачі параметрів Java у Reduce JVM
io.sort.mb	200	Обмеження пам'яті при сортуванні даних, МБ.

Далі, параметри конфігурації для conf/hdfs-site.xml зазначено у табл. 4.3.

Таблиця 4.3 – Параметри конфігурації для conf/hdfs-site.xml

Ім'я	Значення за замовчуванням	Опис
dfs.block.size	67108864	Це розмір блоку HDFS.
dfs.namenode.handler. рахувати	40	Це кількість серверних потоків для обробки викликів RPC у NameNode.

4.2 Тестування Hadoop

До складу Hadoop включено кілька тестів, які може бути використано як як для перевірки коректності встановлення системи, так і для оцінки її продуктивності. Розглянемо деякі з зазначених тестів [19, 20].

У першу чергу, виконаємо тест сортування. Зазначений тест містить у собі два завдання. Так, спершу генеруються деякі випадкові дані, використовуючи завдання Hadoop Randomwriter, далі виконується їх сортування на базі вибірки.

Сама процедура містить ряд кроків, а саме:

1. Перехід до каталогу HADOOP_HOME.

2. Запуск завдання Hadoop Randomwriter, для чого використовється команда, зазначена далі:

```
bin/hadoop jar Hadoop-examples-1.0.0.jarrandomwriter-
Dtest.randomwrite.bytes_per_map=100-
Dtest.randomwriter.maps_per_host=10 /data/unsorted-data
```

У зазначеному фрагменті коду параметри test.randomwrite.bytes_per_map та test. randomwriter.maps_per_host визначають розмір даних, які генеруються Map, а також кількість модулів Map відповідно.

3. Запуск процесу сортування, що може бути виконано командою:

```
>bin/hadoop jar Hadoop-examples-1.0.0.jar sort /data/unsorted-
data/data/sorted-data
```

4. Перевірка остаточних результатів, для чого слугує команда:

```
bin/hadoop jar Hadoop-test-1.0.0.jar testmapredsort -sortInput
/data/unsorted-data -sortOutput /data/sorted-data
```

Далі у випадку, якщо усе було виконано успішно, на консоль буде виведено повідомлення вигляду:

```
The job took 66 seconds.SUCCESS! Task mapReduce framework's
'sort' success.
```

У розглянутому прикладі метод Randomwriter виконує запуск завдання Hadoop для генерації випадкових даних, які далі може бути використано програмою сортування. Після цього здійснюється перевірка результу, використовуючи для цього завдання testmapredsort. Якщо, у даному випадку, ПК, у межах якого виконується тестування, має велику ємність дискового накопичувача, жалі може бути ініційовано початковий етап довільного запису зі збільшеним розміром виводу.

Окрім вже зазначеного, до складу Hadoop входять кілька інших тестів, а саме [20]:

– TestDFSIO. Даний тест виконує перевірку продуктивності вводу-виводу (I/O) HDFS;

– Nnbench. Тут виконується дослідження обладнання NameNode;

– Mrbench. Тест виконує велику кількість дрібних завдань;

– TeraSort. Виконується сортування одного терабайту даних.

4.3 Підхід до підвищення продуктивності кластеру на базі повторного використання віртуальних машин Java

Конфігурація Hadoop за замовчуванням передбачає, що для кожного завдання Map або Reduce ініціюється нова JVM. Разом з тим, запуск кількох завдань у рамках однієї JVM може сприяти суттєвому збільшенню швидкодії.

Ділі розглянемо, яким чином може бути реалізовано означений підхід. Для реалізації необхідно виконати такі кроки, як:

1. Запуск процесу. Для прикладу візьмемо WordCount, передавши при цьому у якості аргумену наступний параметр:

```
bin/hadoop jar Hadoop-examples-1.0.0.jar wordcount
-Dmapred.job.reuse.jvm.num.tasks=-1 /data/input1 /data/output1
```

2. Відстежування кількості процесів, які створено Hadoop (за одним з методів, зазначених вище). За замовчуванням Hadoop виконує запуск лише однієї JVM для кожного слота завдання. Далі її використовує для необмеженої кількості підзавдань у завданні.

Равзом з тим, передача аргументів через параметр –D може функціонувати лише тоді, коли завдання виконується через інтерфейс org.apache.hadoop.util.tools. В усіх інших випадках дану опцію може бути встановлено за допомогою методу JobConf.setNumTasksToExecutePerJvm(-1).

У випадку, якщо встановлено властивість конфігурації завдання через mapred.job.reuse.jvm.num.tasks, отримуємо можливість контролю кількості завдань для JVM, які ініціює Hadoop. При цьому, якщо значення параметру для означеного методу встановлено рівним -1, далі Hadoop буде виконувати завдання в одній і тій же JVM [19, 20].

4.4 Аналіз та оптимізація продуктивності кластерів. Моніторинг та налаштування продуктивності кластерів Hadoop

Платформа Hadoop може забезпечити дуже високий рівень доступність комп'ютерних ресурсів з мінімальними витратами. Наприклад, Hadoop може виконувати завдання, залучивши тисячі вузлів та обробляючи петабайти даних, здійснювати автоматизацію планування завдань та виконувати баланс навантаження.

Усе зазначене можливе, у т.ч. у наслідок того, що окрім архітектурних особливостей, у рамках Hadoop можлива оптимізація як роботи усього кластеру, так і окремих вузлів - процесорів, пам'яті, пристрої вводу-виводу (дискових пристроїв і мережі). Hadoop автоматично підвищує продуктивність, надаючи користувачам інтерфейс, що дозволяє оптимізувати швидкодію власних додатків. Розглянемо найважливіші параметри Hadoop, що можуть налаштовуватися, а також методи аналізу та опції продуктивності.

4.4.1 Інтеграція та налаштування інструментарію моніторингу продуктивності

У Linux-середовищі одним з комплексних засобів управління, налаштування та порівняльного аналізу системи є Nmon. Даний інструмент дозволяє легко відстежувати широкий спектр даних щодо продуктивності.

Нехай у нашому випадку буде розглядатися початковий кластер, де вузол host001 отримує роль NN (NameNode), у свою чергу, host002 - JT (JobTracker), а host003 - SNN (тобто, Secondary NameNode).

Спочатку пакет Nmon завантажується з офіційного сайту розробника (http://nmon.sourceforge.net/pmwiki.php?n=Site.Download), після чого зробити копії на усі вузли Hadoop-кластера [11, 19, 20].

При цьому, слід урахувати те, що вузли з ролями NN, JT а також SNN можуть мати можливість доступу до інших вузлів з ssh без пароля. У свою чергу, усі завдання Map/Reduce надсилатимуться до JT. З огляду на це, саме JT має розглядатися у якості центрального вузла, що виконуватиме збір усіх даних nmon. Тому спершу налаштування Nmon виконується на вузлі JT та, у загальному випадку, включає у себе три ключові технологічні етапи, а саме:

1. Створення відкритої директорії (наприклад, perf_share), та надання до неї доступу через NFS, для чого виконується:

− Безпосередньо створення самої директорії (команда $mkdir /home/hadoop/perf_share);

− додавання до файлу /etc/exports рядку, що встановлює відповідні дозволи (/home/hadoop/perf_share *(rw, sync));

− перезавантаження служби NFS ($/etc/rc.d/init.d/nfs restart;

− створення директорії на інших вузлах кластеру з її наступним монтуванням:

```
$mkdir/home/hadoop/perf_share
$mount host002: /home/hadoop/perf_share
```

Тут розглянуто приклад для вузла host002. Для інших вузлів операція є аналогічною.

2. Створення скрипта, який виконує запуску Nmon одночасно на усіх вузлах кластеру.

Припустимо, що у нашому випадку необхідно виконати запуск Nmon для трьох вузлів, і при цьому:

– зберігати отримувані дані до файлу;

– забезпечити фіксацію даних з певною періодичністю;

– слід виконати певну кількість замірів.

Приклад скрипту, який може бути використано з урахуванням зазначених вимог, наводиться далі:

```
hosts=( host001 host002 host003)
# виконати запуск nmon на усіх вузлах
for host in ${hosts[@]}
do
ssh $host " /usr/bin/nmon -f -m /home/hadoop/perf_share -s 30 -c
360"
done
```

Тут у команді запуску Nmon параметр –f вказує на те, що виконується збереження даних у файл. У свою чергу, –m вказує локацію для зберігання даних (попередньо створена директорія perf_share). Для встановлення періодичності збору даних використовується параметр –s, його величина, рівна 30, вказує на те, збір даних буде виконуватися кожні 30 секунд. Кількість точок фіксації даних встановлює параметр –c, для якого встановлено значення 360. Звідси виходить, що підсумковий період моніторингу складає 30x360 секунд, або 3 години.

3. Завантаження та інсталяція аналітичного інструментарію Nmon - Nmonanalyser [http://www.ibm.com/developerworks/wikis/display/Wikiptype/nmonanalyser].

4.4.2 Рекомендації з оптимізації параметрів Hadoop

У рамках Hadoop як зазначалося раніше, загальну продуктивність може бути забезпечено шляхом оптимізації налаштувань процесорів, пам'яті, дискової система а також мережі. Розглянемо параметри, що відповідають чотирьом переліченим напрямкам [16, 18, 19].

Параметри процесору mapred.tasktracker.map та reduce.tasks.maximum

Означені параметри встановлюють найбільшу кількість завдань Map/Reduce, які може виконати Task Tracker. Як перший, так і другий параметри є найважливішим для регулювання завантаження ЦП. Для обох параметрів за замовчуванням приймається значення, рівне 2.

Відповідно, збільшення значень цих параметрів, ураховуючи особливості кластеру, сприяє росту завантаження ЦП а отже - збільшує його продуктивність.

Розглянемо приклад кластеру, де кожен вузол має 4 ЦП, при цьому, кожен ЦП має два ядра, а також здатен підтримувати паралельну багатопоточність. Для цього випадку результуюча кількість демонів має перевищувати 4x2x2=16.

Беручи до уваги те, що і DataNode, і TaskTracker займають два параметри, завдання Map/Reduce може зайняти до 14 процесів, тому найбільше значення для цих двох параметрів - 7.

Параметр пам'яті mapred.child.java.opts

Mapred.child.java.opts є основним параметром JVM. За замовчуванням він має значення Xmx200м. Це означає, що виділяється до 200 МБ пам'яті для кожного потоку підзадачі. При цьому, якщо завдання велике, даний параметр можна збільшити. Разом з тим, надмірна величина цього парамтеру може викликати появу свопів, що може суттєво погіршити продуктивність.

Для прикладу припустимо, що найбільша кількість завдань Map/Reduce дорівнює 7.

При цьому, використовується значення mapred.child.java.opts за замовчуванням. На цей випадок витрати пам'яті для завдання дорівнюватимуть 2x7x200 МБ = 2800 МБ. Водночас, коли кожен обчислювальний вузол містить демонів DN і TT (які за замовчуванням займають 1 ГБ пам'яті кожен), загальний об’єм виділеної пам'яті буде рівним близько 4,8 ГБ.

Параметри дискових пристроїв

У Hadoop одними з головних параметрів, що регулюють роботу дискових пристроїв, є mapred.compress.map.output, mapred.output.compress та mapred.map.output.compression.codec.

У цілому, перелічені параметри контролюють процес стиснення виводу даних. Так, параметр mapred.compress.map.output застосовується для налаштування стиснення виводу Map. У свою чергу, від mapred.output.compress залежить стиснення вихідних даних завдання, а mapred.map.output.compression.code слугує для стиснення коду.

За замовчуванням усі зазначені параметри вимкнено.

Разом з тим, у разі активації режиму стиснення даних забезпечується:

– прискорення запису на диск (як на локальний пристрій, так і до розподіленої файлової системи HDFS);

– зменшення загального часу передачі (на етапах перемішування та запису HDFS).

Проте, з іншого боку, у випадку активації режиму стиснення зростають накладні обчислювальні витрати за рахунок необхідності виконання операцій компресії та декомпресії.

З практичної точки зору неприпустимо застосовувати режим стиснення для завдань з обробки даних, де передбачається використання використовуються випадкових ключів або значень. Одночасно з цим, активний режим стиснення дозволяє суттєво прискорити перебіг усіх операцій, пов’язаних з обробкою великих обсягів організованих даних, у першу чергу – даних, згенерованих природною мовою.

Параметр буферу

Для Hadoop розмір буфера для сортування на рівні Map встановлює параметр Io.sort.mb. За замовчуванням він має значення 100 МБ.

При цьому, збільшення значення Io.sort.mb веде до скорочення часу введення-виводу за рахунок зменшення переповнення диску.

З іншого боку, збільшення величини Io.sort.mb спричинює ріст необхідного об’єму пам'яті, що виділяється для кожного завдання Map.

Попри все зазначене, збільшувати значення параметру Io.sort.mb доцільно в умовах, коли вивід даних Map є досить великим і, при цьому, дуже часто виконуються операції введення-виводу на боці Map.

Параметр, що визначає кількість вхідних потоків задачі Map/Reduce

За кількість вхідних потоків (файлів), які може бути одночасно поєднано у рамках однієї задачі Map/Reduce, відповідає параметр Io.sort.factor.

У разі збільшення величини цього параметру буде забезпечено зменшення переповнення диску. Як наслідок цього – створюються умови для скорочення часу введення-виводу у ході виконання завдань Map/Reduce.

Таким чином, в умовах, коли спостерігається значний рівень переповнення дискового простору, та/або процедури введення-виведення на етапах сортування та перемішування займають значні часові відрізки, рекомендується збільшувати значення даного параметру.

Параметр вибору обсягу пам'яті для збереження виводу Map на етапі Reduce

Для вузла Hadoop може бути зарезервовано деякий відсоток пам'яті (відносно максимального розміру), який може бути використано для збереження виводу даних Map на етапі Reduce. За це відповідає параметр mapred.job.reduce.input.buffer.percent, який за замовчуванням має значення 0.

Так, коли процедура Shuffle (перемішування) закінчується, вивід даних Map, які залишилися у пам'яті, має бути менше, ніж величина цього порога, до того, ніж може буде ініційовано фазу Reduce. Водночас, чим більшою буде ця величина, тим меншим буде об’єм злиття на диску. За цих умов забезпечується скорочення часу вводу-виведення даних на локальному диску у ході етапу Reduce.

Разом з тим якщо обсягу пам'яті, який виділено для рішення кожного завдання, є недостатньо, збільшення значення цього параметру може спричинити появу «сміттєвих» файлів на диску.

Тобто, в умовах, коли обсяг вихідних даних Map є значним, а операції вводу-виведення на локальному дисковому пристрої виконуються надто часто, має сенс збільшити величину параметру mapred.job.reduce.input.buffer.percent.

Параметри розміщення даних у Hadoop

Вказати, куди саме слід розмістити дані у Hadoop, можна за допомогою параметрів mapred.local.dir та dfs.data.dir

Тут параметр mapred.local.dir вказує локацію, у якій можуть міститися проміжні дані MapReduce (які є вихідними даними Map).

У свою чергу, параметр dfs.data.dir вказує на місце, де будуть зберігатися дані HDFS.

Таким чином, маніпулюючи значеннями даних параметрів, може бути забезпечено баланс дискового вводу-виводу, розподіливши зазначені вище локації у межах усіх дисків на кожному з вузлів, що веде до росту рівня продуктивності процедур вводу-виведення.

Дані параметри можна встановити у конфігураційних файлах mapred-site.xml та hdfs-site.xml.

Мережевий Параметр topology.script.file.name

Означений параметр вказує на сценарій, створений користувачем, який, у свою чергу, виконує процедуру співставлення стійки і окремого хоста для налаштування рівня доступності стійки. Цей параметр може бути налаштовано у файлі core-site.xml.

Параметр визначення кількості потоків копіювання виводу Map на сегмент Reduce

Встановити необхідну кількість потоків копіювання виводу Map на сегмент Reduce дозволяє параметр Mapred.reduce.parallel.copies.

Значення даного параметру за замовчуванням дорівнює 5. При цьому, зі збільшенням величини даного параметру, по-перше, створюються умови для росту швидкості передачі даних мережею а по-друге - прискорюється копіювання вихідних даних Map. На тлі цього збільшується завантаження ЦП.

Значення параметру Mapred.reduce.parallel.copies доцільно збільшувати в умовах, якщо вихідні дані Map мають великі розміри.

ВИСНОВКИ

За технічним завданням до кваліфікаційної роботи було виконано:

- огляд існуючих архітектур, на базі яких виконується побудова обчислювальних кластерів;

- дослідження процесу розгортання та первинного конфігурування кластеру, до складу якого входить 3 окремих вузла, на базі платформи Apache Hadoop;

- висвітлено загальні підходи до налаштування вузлів кластеру Apache Hadoop у контексті принципу Map/Reduce;

- дослідження підходів до оптимізації кластерних обчислень, розглядаючи кластер у вигляді єдиної структури.

При цьому, виявлено, що продуктивність кластерних обчислень залежить від:

- налаштування завдання Map/Reduce, зокрема – вибору моделі розподілу кількості модулів Map та Reduce;

- управління пам’яттю, виділеною для виконання завдань Map/Reduce;

- розподілу процесорних ресурсів та ресурсів оперативної пам’яті між вузлами кластеру залежно від їхньої ролі;

- оптимізації кількості процедур зчитування/запису на рівні дискових пристроїв;

- налаштування взаємодії вузлів у кластері.

Окрім цього, досліджено метод планування та розширення кластеру на базі:

- існуючих обсягів даних, що обробляються;

- наявності реплікації даних та її параметрів;

- темпів росту обсягів даних.

Дане дослідження було проведено в умовах, коли параметри окремого вузла залишалися статичними.

Також було розглянуто метод моніторингу продуктивності вузлів кластеру на базі Nmon, досліджено порядок його розгортання та налаштування на вузлах.

Виявлено та досліджено перелік параметрів, які може бути використано для оптимізації функціонування кластеру. Дані параметри може бути використано незалежно від попереднього конфігурування вузлів, тим самим це

потенційно сприяє росту обчислювального потенціалу кластеру та створює умови для його оптимального функціонування.

Отже, усі завдання виконано у повному обсязі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Telecommunications Industry HPC Cluster Introduction [Електронний ресурс] – Режим доступу: https://www.naddod.com/solution/telecommunications-industry-hpc-cluster-introduction.

2. Zhang, D. Zhang T., Cui Y., Liu X., and Mao G. "New Multi-Hop Clustering Algorithm for Vehicular Ad Hoc Networks," in IEEE Transactions on Intelligent Transportation Systems. Vol. 20, no. 4. pp. 1517-1530, April 2019. doi: 10.1109/TITS.2018.2853165.

3. Zhang D., Liu S., Zhang T., Liang Z. Novel unequal clustering routing protocol considering energy balancing based on network partition & distance for mobile education. Journal of Network and Computer Applications. Volume 88, 2017. Pages 1-9. ISSN 1084-8045.

4. Implementation of Ultra-Low Latency and High-Speed Communication Channels for an FPGA-Based HPC Cluster [Електронний ресурс] – Режим доступу: https://publications.polymtl.ca/2521/1/2017_RobertoSanchezCorrea.pdf.

5. A Virtual Clustering Approach for Routing Problems in telecommunication Networks [Електронний ресурс] – Режим доступу: https://dl.acm.org/doi/10.5555/2700591.2700593.

6. MapReduce: Simplified Data Processing on Large Clusters [Електронний ресурс] – Режим доступу: https://research.google/pubs/mapreduce-simplified-data-processing-on-large-clusters/

7. 7. Principles and Programming of MapReduce (Java realizes Map and Reduce) [Електронний ресурс] – Режим доступу: https://programmersought.com/article/41236050108/

8. 8. Xiang W. Data Clustering: Theory, Algorithms, and Applications (ASA-SIAM Series on Statistics and Applied Probability, Series Number 20) — Society for Industrial and Applied Mathematics, 2007. — 184 p.

9. Van Vugt S. Pro Linux High Availability Clustering 1st ed. Edition — Apress, 2014. — 182 p.

10. Xu R. Wunsch D. Clustering. — Wiley-IEEE Press, 2008. — 368 p.

11. Bookman C. Linux Clustering: Building and Maintaining Linux Clusters. — Sams, 2022 .— 288 c.

12. Mirkin B. Clustering for Data Mining: A Data Recovery Approach (Chapman & Hall/CRC Computer Science & Data Analysis). Chapman and Hall/CRC, 2005. – 295 p.

13. Apache Hadoop 3.3.6 &#x2013 - Hadoop Cluster Setup [Електронний ресурс] – Режим доступу: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/ClusterSetup.html

14. Common Ports And Protocols List [Електронний ресурс] – Режим доступу: https://www.lddgo.net/en/network/port/

15. Hadoop Map Reduce Performance Tuning Best Practices [Електронний ресурс] – Режим доступу: https://data-flair.training/blogs/mapreduce-performance-tuning/

16. Hadoop Optimization | Job Optimization & Performance Tuning [Електронний ресурс] – Режим доступу: https://data-flair.training/blogs/hadoop-optimization/

17. Building a Hadoop distributed storage high-performance cluster [Електронний ресурс] – Режим доступу: https://programmersought.com/article/39499181555/

18. Hadoop high performance cluster - Programmer Sought [Електронний ресурс] – Режим доступу: https://programmersought.com/article/60451326641/

19. Analyze and optimize cloud cluster performance Monitor and tune the performance of Hadoop clusters with configurable parameters [Електронний ресурс] – Режим доступу: https://programmersought.com/article/5120154002/

20. Hadoop cluster performance test [Електронний ресурс] – Режим доступу: https://programmersought.com/article/13786232351/