МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

В. І. Таран,
Ю. Г. Гордієнко,
С. Г. Стіренко

ТЕХНОЛОГІЇ BIG DATA

Практикум

Навчальний посібник

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня магістра
за освітньою програмою «Комп’ютерні системи та мережі»
спеціальності 123 Комп’ютерна інженерія

Електронне мережне навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2022

Рецензент *Ролік О. І.*, д.т.н., професор, завідувач кафедри ІСТ ФІОТ

Відповідальний редактор *Новотарський М. А.*, д.т.н., професор, професор кафедри ОТ ФІОТ

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 1 від 02.09.2022 р.)
за поданням Вченої ради факультету інформатики та обчислювальної техніки
(протокол № 11 від 11.07.2022 р.)

Навчальний посібник розроблено для ознайомлення здобувачів з інфраструктурою для розподілених обчислень на базі кластеру Hadoop та парадигми MapReduce для обробки великих обсягів даних. Навчальний посібник містить інформацію щодо розробки та використання програмного забезпечення для обробки даних на кластері Hadoop, розгортанню та налаштуванню обчислювального кластеру Hadoop, а також містить необхідні теоретичні відомості.

Навчальний посібник призначений для здобувачів ступеня магістра напряму підготовки за освітньою програмою “Комп’ютерні системи та мережі” спеціальності 123 - “Комп’ютерна інженерія”.

Реєстр. № НП 22/23-024. Обсяг 2,3 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Перемоги, 37, м. Київ, 03056
https://kpi.ua

Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

ЗМІСТ

ВСТУП

Дисципліна “Технологія BigData” спрямована на вивчення підходів, методів і механізмів функціонування та використання інфраструктури для розподілених обчислень на базі кластеру *Hadoop* та парадигми *MapReduce*. Необхідність в використанні нових підходів обумовлена тим, що сучасні підходи до вирішення складних завдань, які потребують обробки надзвичайно великого обсягу даних, вимагають великої кількості обчислювальних ресурсів. Вивчення даної дисципліни майбутніми фахівцями дозволить їм набути важливих компетенцій в плані розвитку існуючих і використання нових підходів для організації розподілених обчислень.

Практична частина курсу складається з п’яти лабораторних робіт і призначена для отримання практичних навичок використання існуючих технологій для обробки великих обсягів даних у розподілених системах. Всі лабораторні виконуються на кафедральному кластері *Hadoop* і включають в себе розробку програм для обробки даних за парадигмою *MapReduce*, а також використання інструментів *Apache Pig Latin*, *Hive* та диспетчеру формування робіт *Oozie*. Роботи послідовно та логічно впорядковані за складністю та охоплюють всі теми, що вивчаються в курсі.

Матеріал для кожної лабораторної роботи містить мету, основні теоретичні відомості, загальне завдання, вказівки стосовно обрання варіанту для індивідуальних завдань, список питань для самоперевірки. Також дається список рекомендованих інформаційних джерел для підготовки та виконання лабораторних робіт. У додатку надаються інструкції по налаштуванню локального кластеру *Hadoop*.

ЗАГАЛЬНІ ПОЛОЖЕННЯ ПРО ЛАБОРАТОРНІ РОБОТИ

Лабораторні роботи виконуються на кафедральному кластері *Hadoop* для розподілених обчислень.

Перед кожною роботою наведено короткі теоретичні відомості, що містять достатній обсяг інформації для виконання завдання лабораторної роботи.

Оформлення звіту та порядок його подання

Для позитивної оцінки по кожній роботі студент надає викладачу оформлений звіт.

Звіт має містити:

- титульний аркуш (на ньому вказують назву міністерства, назву університету, назву кафедри, тему роботи, виконавця, особу, яка приймає звіт, рік);
- мету, варіант і завдання роботи;
- детальний опис виконаної роботи, лістинг програми (за необхідності);
- змістовний аналіз отриманих результатів та висновки.

Під час захисту роботи студенту потрібно: продемонструвати знання по змісту роботи, по теоретичному матеріалу, аналізувати кожен етап роботи, виконувати завдання сумісні з роботою, що викладач може попросити виконати на місці. Студент має вміти правильно аналізувати отримані результати.

ЛАБОРАТОРНА РОБОТА №1
ОСНОВНІ ПОНЯТТЯ *MAPREDUCE*

Мета та основне завдання роботи: ознайомитися із парадигмою програмування *MapReduce,* отримати навички написання та запуску простої програми, яка слідує парадигмі *MapReduce*. Спробувати реалізувати етапи, що знаходяться між *Map* та *Reduce* (наприклад, *Shuffle* та *Sorting*) без використання фреймворку.

ОСНОВНІ ТЕОРЕТИЧНІ ВІДОМОСТІ

MapReduce – парадигма програмування для обробки великих обсягів даних, розподілених між вузлами кластера. Ідея цієї парадигми полягає у розділенні вхідних даних на частини, обробки цих частин та об’єднання проміжного результату в остаточне рішення.

Розділені вхідні дані мають формат списку, елементами якого є пари «ключ, значення». На прикладі задачі розрахунку кількості різних слів у тексті, матимемо елементи формату «слово, 1». При подальших розрахунках, це значення - «одиниця» буде сумуватись для отримання загальної кількості появи певного слова у тексті. На рис. 1.1 показано приклад обробки текстової інформації, використовуючи парадигму *MapReduce.*

Канонічна програма *MapReduce* має містити реалізацію принаймні двох головних обчислювальних фаз *Map* та *Reduce.* В них описується як потрібно створити пари «ключ, значення» та яким чином зібрати кінцевий результат. Задання обчислювального процесу у даний спосіб дозволяє розподіляти дані на частини і виконувати їх обробку на різних обчислювальних вузлах. При цьому, сам процес розподілу виконується автоматично і не вимагаю явного керування користувачем.

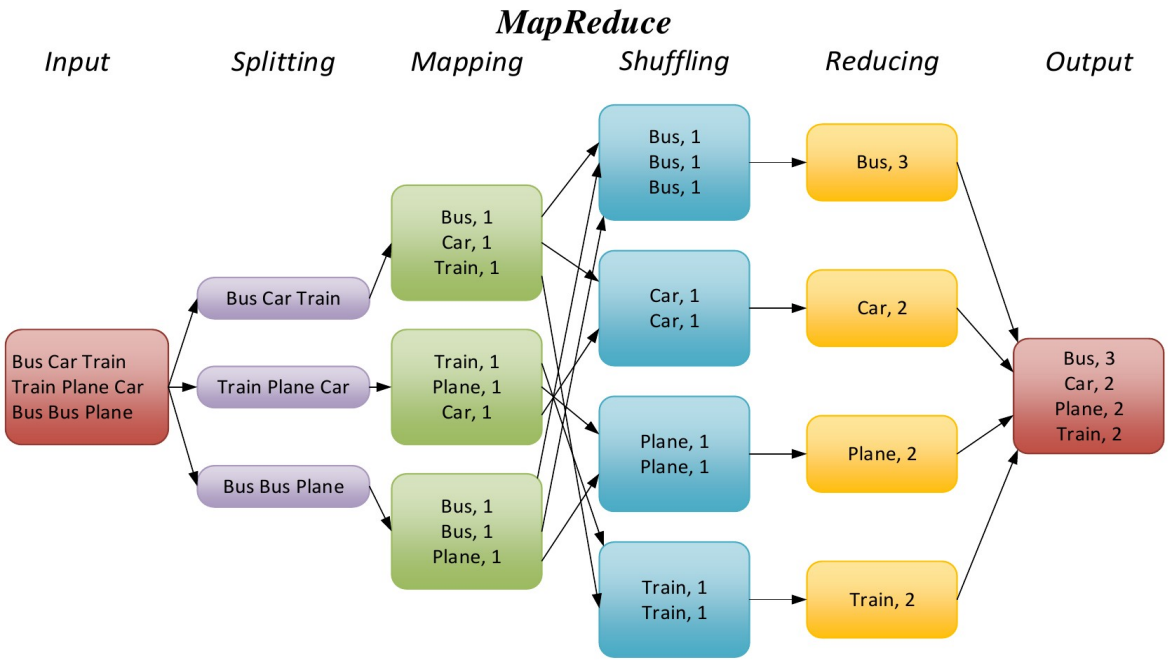


Рис. 1.1. Основні фази парадигми *MapReduce*.

В загальному вигляді проста програма *MapReduce*, що виконує підрахунок слів в наборі документів, матиме наступний вигляд.

```
function map(String name, String document):
  // name: document name
  // document: document contents
  for each word w in document:
    emit (w, 1)

function reduce(String word, Iterator partialCounts):
  // word: a word
  // partialCounts: a list of aggregated partial counts
  sum = 0
  for each pc in partialCounts:
    sum += pc
  emit (word, sum)
```

Далі розглянемо приклад роботи коду *mapper.py* та *reducer.py* для мови програмування *Python.*

1. Створіть директорію, перейдіть в неї, та завантажте приклад коду з репозиторію (https://github.com/iorch/test_hadoop).

mapper.py

```
#!/usr/bin/env python
"""A more advanced Mapper, using Python iterators and generators."""

import sys

def read_input(file):
    for line in file:
        # split the line into words
        yield line.split()

def main(separator='\t'):
    # input comes from STDIN (standard input)
    data = read_input(sys.stdin)
    for words in data:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        #
        # tab-delimited; the trivial word count is 1
        for word in words:
            print '%s%s%d' % (word, separator, 1)

if __name__ == "__main__":
    main()
```

reducer.py

```
#!/usr/bin/env python


from itertools import groupby
from operator import itemgetter
import sys


def read_mapper_output(file, separator='\t'):
    for line in file:
        yield line.rstrip().split(separator, 1)


def main(separator='\t'):
    # input comes from STDIN (standard input)
    data = read_mapper_output(sys.stdin, separator=separator)
    # groupby groups multiple word-count pairs by word,
    # and creates an iterator that returns consecutive keys and their
     group:
    #   current_word - string containing a word (the key)
    #   group - iterator yielding all ["<current_word>", "<count>"] items
    for current_word, group in groupby(data, itemgetter(0)):
        try:
            total_count = sum(int(count) for current_word, count in group)
            print "%s%s%d" % (current_word, separator, total_count)
        except ValueError:
            # count was not a number, so silently discard this item
            pass


if __name__ == "__main__":
    main()
```

2. Змініть прав доступу

```
# chmod a+x mapper.py
# chmod a+x reducer.py
```

3. Запустіть програму із командного рядку. На вхід подається текстовий рядок, який розбивається на пари «ключ, значення». Цей крок відповідає етапу *map*. Результат виконання скрипту *mapper.py* показано на рис. 1.2.

```
# echo "abc ddd hello abc yyy def hello" | ./mapper.py | sort
```

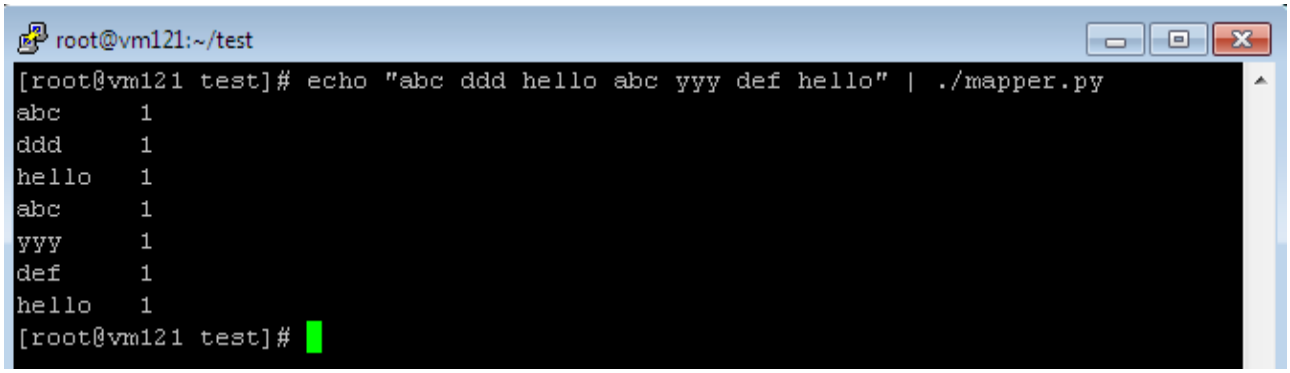


Рис. 1.2. Імітація процесу розбиття вхідних даних та створення пар.

Програма *MapReduce* зазвичай розбиває набір вхідних даних на незалежні частини, які обробляються задачами *map* паралельно. Фреймворк *Hadoop* виконує сортування результатів від задач *map*, які потім поступають на вхід задач *reduce* у сортованому за ключем вигляді. Це дозволяє фреймворку збирати пари з однаковими ключами на виході різних задач *map* та передавати їх в один *reduce*. Цей процес називається *shuffle*. Для спрощення прикладу, не використовуючи фреймворк, скористаємося командою *sort* в *Linux* (рис. 1.3).

```
# echo "abc ddd hello abc yyy def hello" | ./mapper.py | sort
```

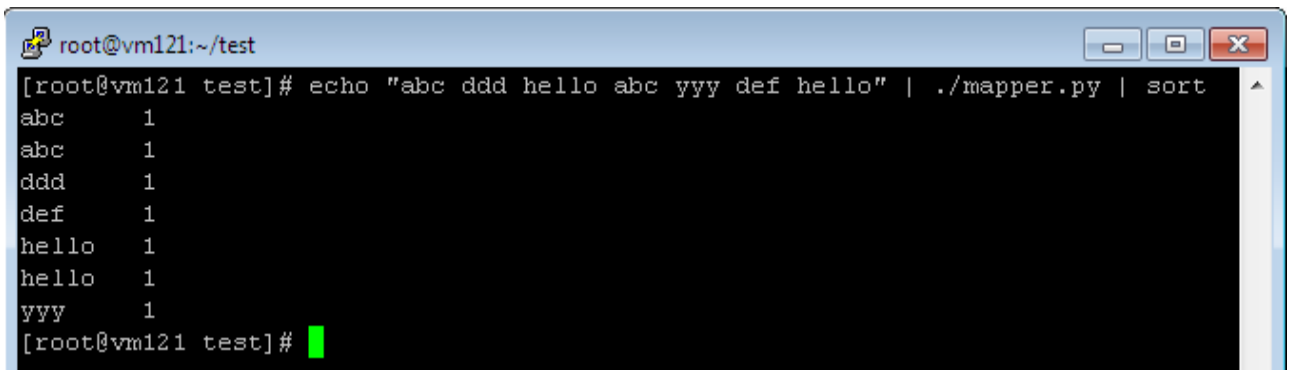


Рис. 1.3. Імітація процесу *shuffle* засобами утиліти *sort* в *Linux*.

4. Наступним кроком потрібно виконати обробку отриманих на етапі *map* даних і сформувати кінцевий результат згідно поставленої задачі. Для цього перенаправляємо попередній вивід до скрипту *reducer.py* (рис. 1.4).

```
# echo "abc ddd hello abc yyy def hello" | ./mapper.py | sort | ./reducer.py
```

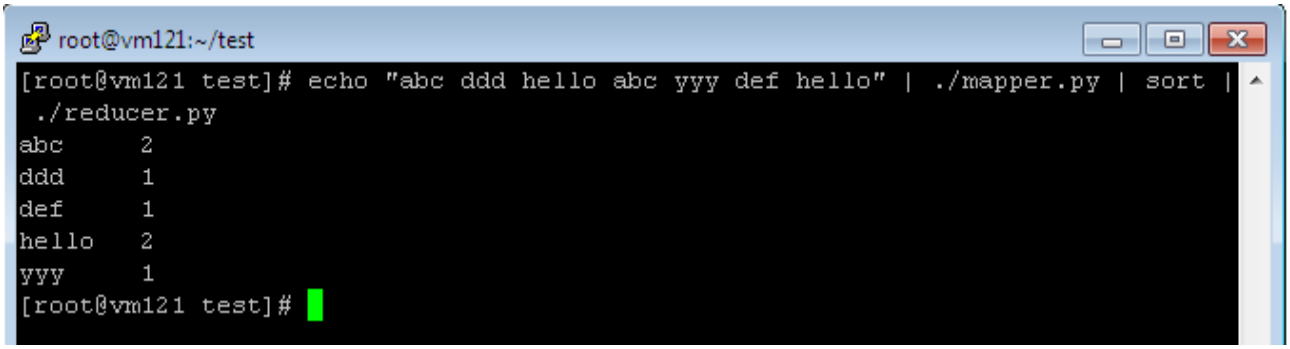


Рис. 1.4. Імітація процесу збору даних у остаточне рішення.

В результаті даного прикладу було змодельовано обробку текстових даних за парадигмою *MapReduce* із використанням двох скриптів, що реалізують два основні етапи *map* та *reduce*. Дані етапи було об’єднано в одну послідовність у командному рядку терміналу *Linux*.

ПРАКТИЧНЕ ЗАВДАННЯ

Припустимо, є дві задачі *map*, які виводять дані відповідно до *file1* та *file2*.

1. Реалізуйте крок *shuffle*. Для цього розробіть програму або скрипт, який читає вхідні файли, і готує дані для задач *reduce*. Ви можете використовувати мову програмування на ваш вибір;

2. Запустіть програму і переконайтесь, що все працює правильно.

СПИСОК КОНТРОЛЬНИХ ПИТАНЬ

Дайте визначення парадигми *MapReduce?*

Які основні етапи обробки даних визначені у парадигмі *MapReduce?*

ЛАБОРАТОРНА РОБОТА №2
ОСНОВНІ ПОНЯТТЯ *HADOOP*

Мета та основне завдання роботи: ознайомитися із програмним забезпеченням для розподілених обчислень *Hadoop*, що включає в себе розподілену файлову систему *Hadoop – HDFS*, менеджер ресурсів кластеру *Hadoop – YARN* та засоби *Hadoop* для моніторингу кластеру. Навчитися розробляти та запускати програму *MapReduce* на кластері *Hadoop*.

ОСНОВНІ ТЕОРЕТИЧНІ ВІДОМОСТІ

Компоненти *Hadoop*

Для розподіленого зберігання даних на вузлах кластера, *Hadoop* має розподілену файлову систему – *HDFS*.

HDFS має наступні властивості:

- відмовостійкість;
- може застосовуватись на стандартному апаратному забезпеченні;
- можливість обробки великих обсягів даних;
- принцип *master/slave*.

Основними елементами *HDFS* є:

- *NameNode* – «*master*» вузол, який виконує функції підтримки каталогів, файлів та управління блоками даних, розподіленими між вузлами кластера;
- *DataNode* – «*slave*» вузол, що забезпечує фізичний простір для зберігання даних та обробляє запити читання / запису з головного вузла.
- *Secondary-namenode* – резервний екземпляр для *NameNode*, що працює на іншому вузлі кластеру.

На рис. 2.1 показано архітектуру *HDFS*.

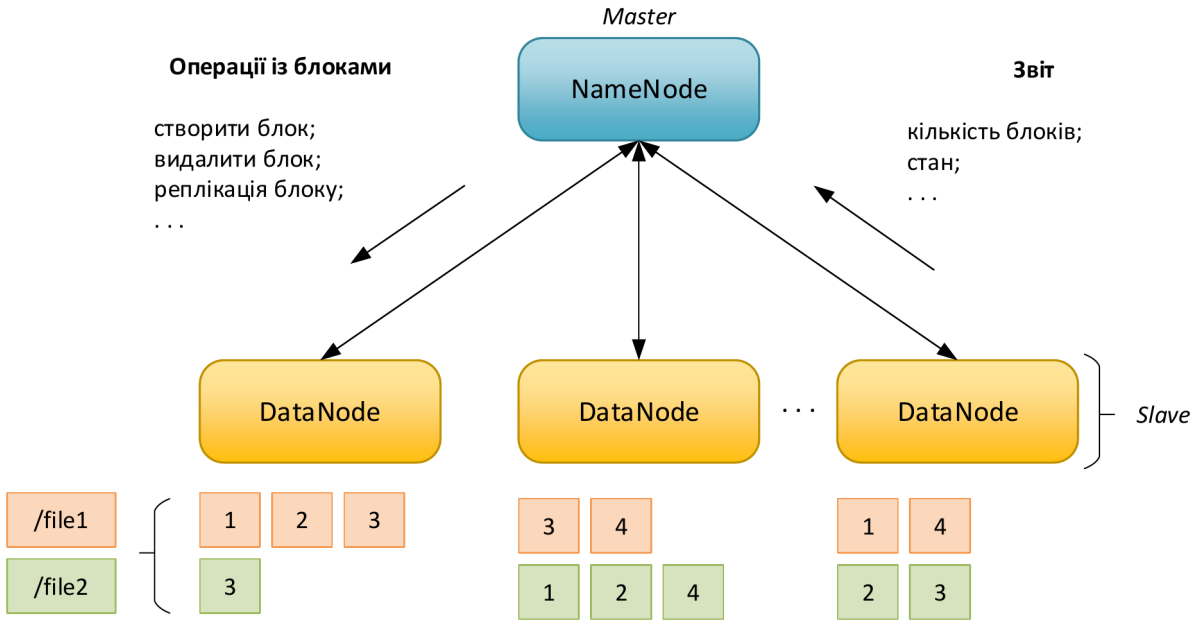


Рис. 2.1. Архітектура *HDFS*.

MapReduce – парадигма програмування для обробки великих обсягів даних, розподілених між вузлами кластера. Ідея цієї парадигми полягає у розділенні вхідних даних на частини, обробки цих частин на обчислювальних вузлах та об’єднання проміжного результату в остаточне рішення на головному вузлі.

За планування/виконання задач *MapReduce* та управлінням ресурсами на кластері *Hadoop* відповідає менеджер ресурсів *YARN – Yet Another Resource Negotiator*.

Основними елементами *YARN* є:

• *Resource Manager* – «*master*» вузол *YARN* для виконання задач *MapReduce*, відповідає за постановку задач на обробку;

• *Node Manager* – «*slave*» вузол *YARN* для виконання задач *MapReduce*, надає обчислювальні ресурси;

• *App Master* – назначається «*master*» вузлом на певному «*slave*» вузлі, контролює виконання задачі *MapReduce* із певним *ID*;

• *Container* – запускається на інших «*slave*» вузлах, виконує обробку певної задачі *MapReduce* і контролюється своїм *App Master*.

На рис. 2.2 показано архітектуру *YARN*.

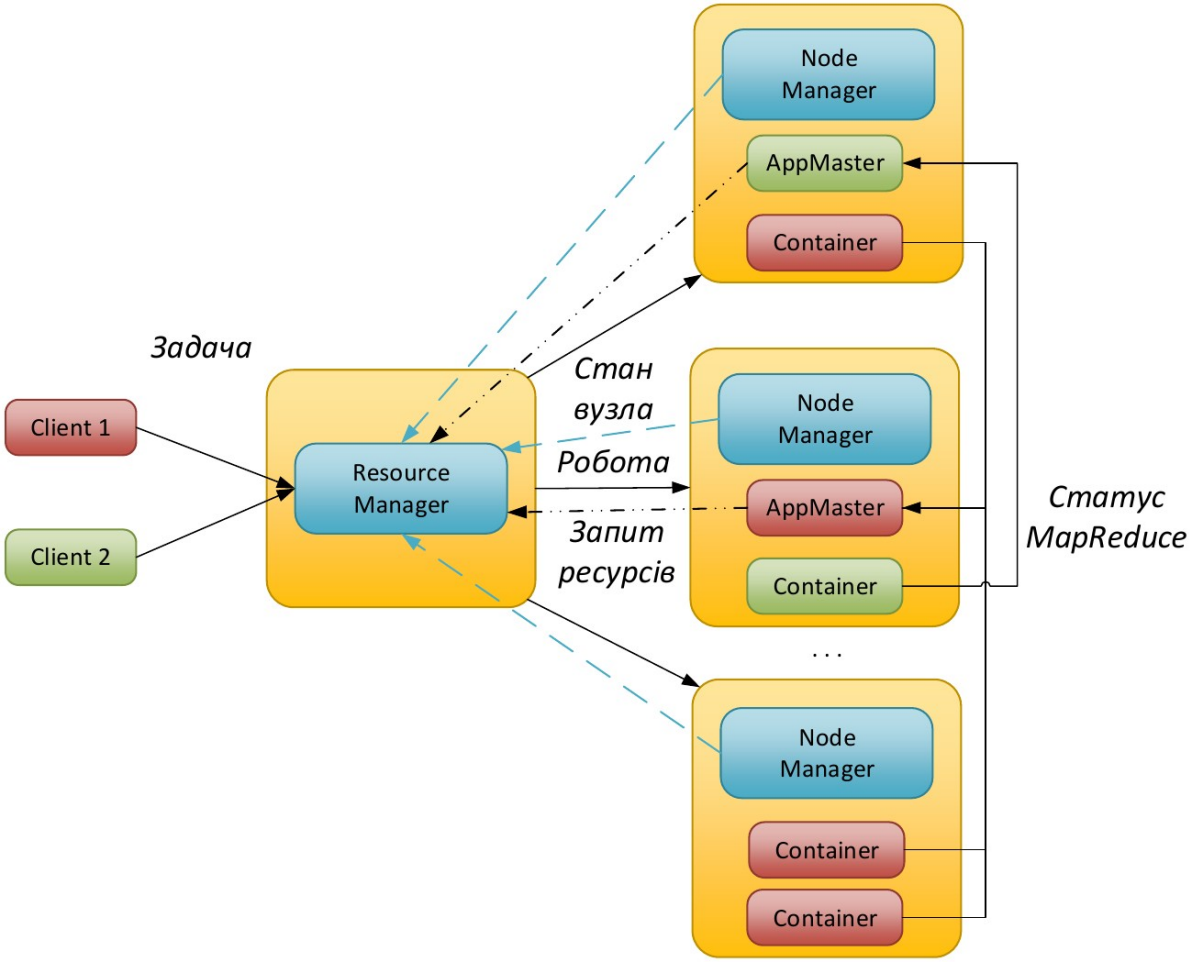


Рис. 2.2. Архітектура *YARN*.

Переваги програмного забезпечення *Hadoop* полягають у тому, що воно:

• не потребує спеціалізованого обладнання для побудови обчислювального кластеру;

• має меншу вартість зберігання та обробки даних на терабайт;

• має високу масштабованість, продуктивність, відмовостійкість;

• не потребує перетворення даних для їх зберігання в розподіленій файловій системі.

Засоби моніторингу кластеру *Hadoop*

Hadoop має декілька ресурсів за допомогою яких можна слідкувати за певними аспектами роботи кластеру:

- *http://<master-IP>:50070/dfshealth.html* – веб-ресурс стану *HDFS*;
- *http://<master-IP>:8088/cluster* – веб-ресурс менеджера ресурсів кластеру;
- *http://<master-IP>:19888/jobhistory* – веб-ресурс історії оброблених задач.

На рис. 2.3 показано головну сторінку із оглядом *HDFS*, де можна ознайомитися із основною інформацією, а саме кількість файлів, доступний простір сховища даних, кількість вузлів, тощо.

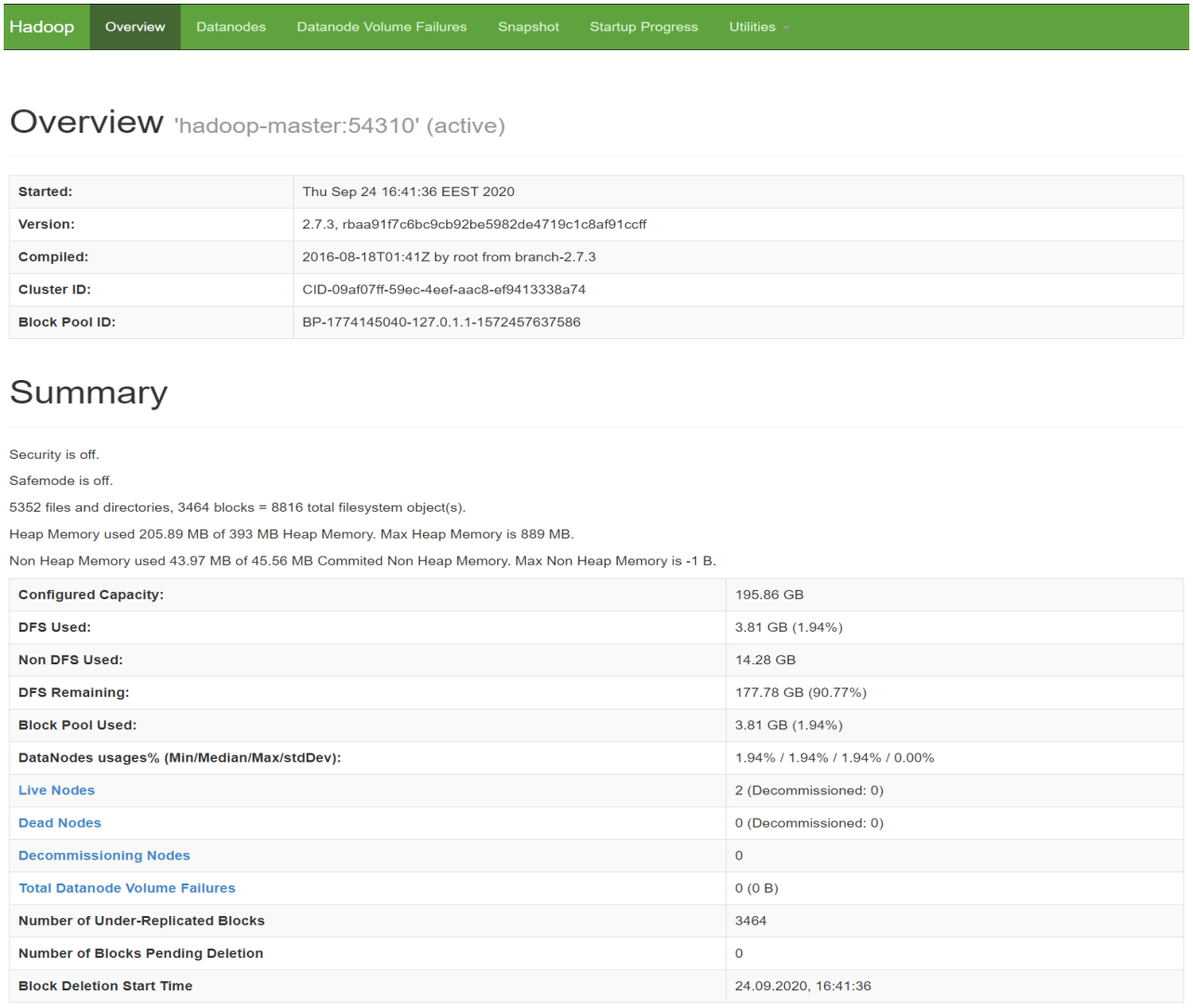


Рис. 2.3. Ресурс із основною інформацією про *HDFS*.

Також можна отримати детальну інформацію по кожному вузлу *HDFS* (рис. 2.4).

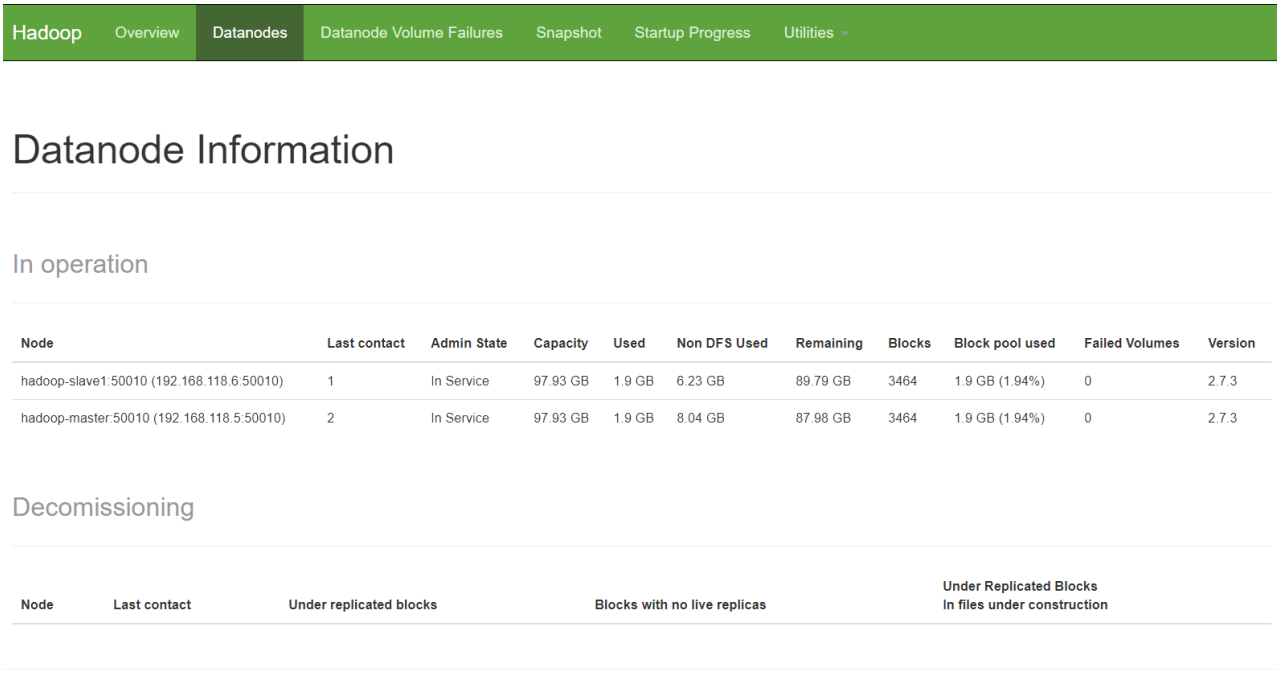


Рис. 2.4. Детальна інформація по кожному вузлу *HDFS*.

Веб-ресурс *HDFS* має декілька утиліт, одна з яких дозволяє переглядати файли, що зберігаються у розподіленій файловій системі (рис. 2.5).

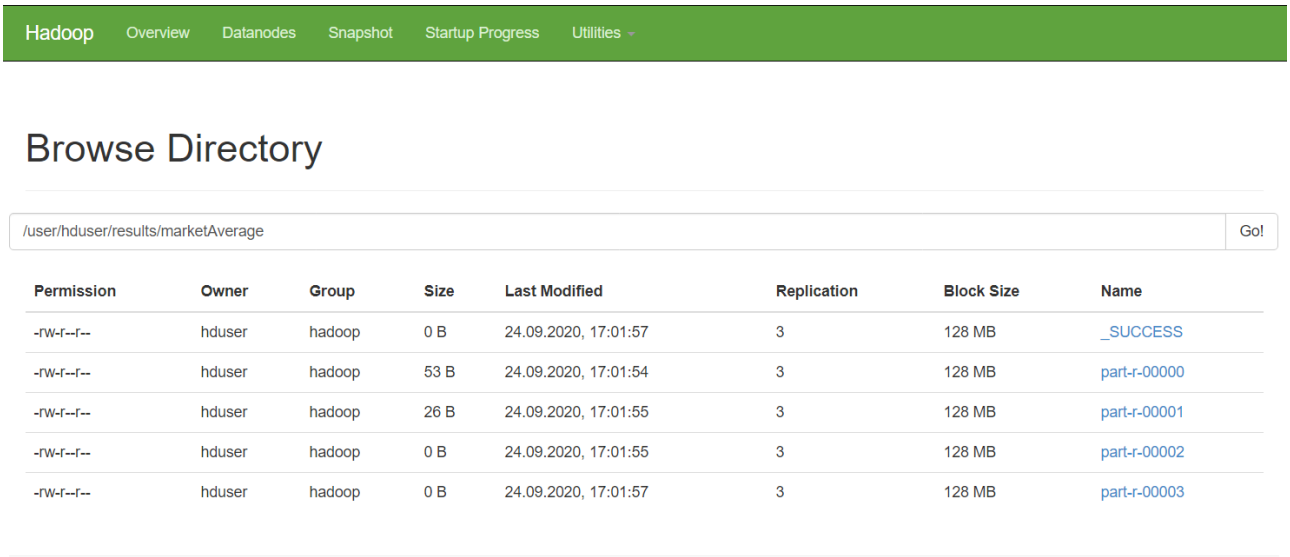


Рис. 2.5. Перегляд файлів *HDFS*.

Веб-сторінка менеджера ресурсів кластеру надає детальну інформацію про кількість обчислювальних вузлів кластеру та їх ресурси. Вона дозволяє переглядати перелік задач із різними статусами: виконуються, завершено, тощо (рис. 2.6-2.8).

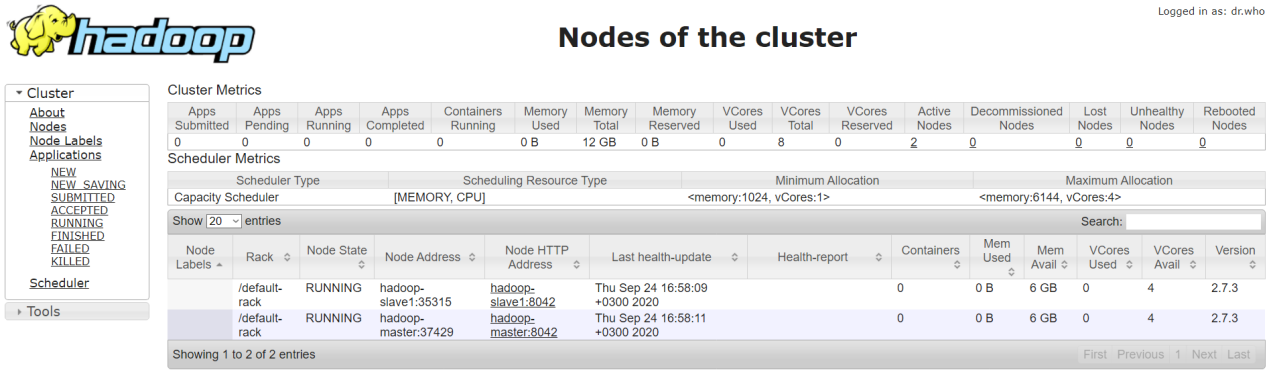


Рис. 2.6. Перелік обчислювальних вузлів кластеру.

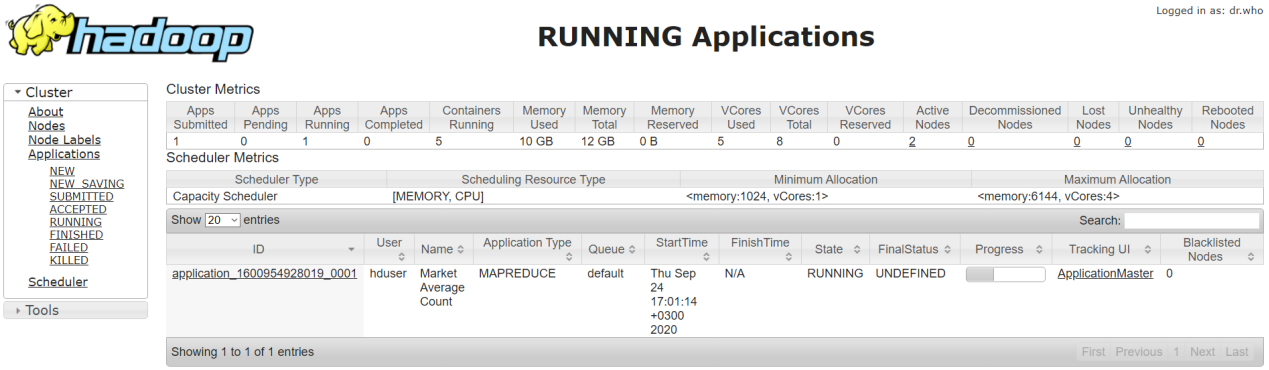


Рис. 2.7. Перелік задач, що виконуються.

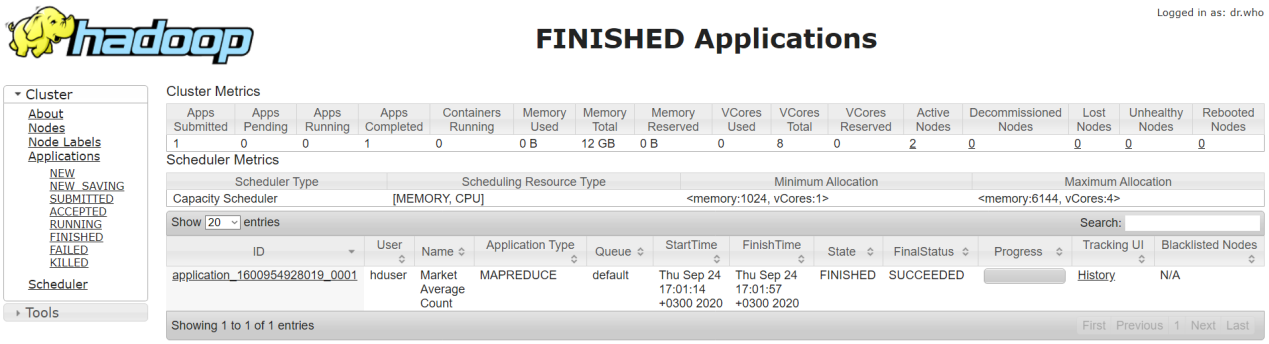


Рис. 2.8. Перелік завершених задач.

Менеджер ресурсів не зберігає перелік оброблених задач і після перезапуску кластеру дана інформація втрачається. Для зберігання історії у *Hadoop* передбачено спеціальний сервіс, що зберігає історію задач у *HDFS* (рис. 2.9).

Рис. 2.9. Веб-ресурс історії оброблених задач.

Розробка та запуск програми *MapReduce* на кластері *Hadoop*

Для написання програм *MapReduce* для *Hadoop* використовують мови програмування *Java*, *Python*, *Scala*, *R*, тощо. Деякі з цих мов потребують встановлення додаткових компонентів на кластері *Hadoop*.

В даній лабораторній роботі буде виконуватись розрахунок статистичної інформації для певного набору даних. В якості набору даних використовується історія котирування валют на *Forex* для 3-х валютних пар. Маємо статистику за 5 років по 2 млн. записів для кожної пари. Приклад набору даних наведено нижче.

```
1. 2019.09.30,17:43,1.0902,1.0903,1.0902,1.0903,2,2019.09.30,18:00,0.8862,0.8862,0.8861,0.8861,4,2019.09.30,18:01,1.0880,
   1.0881,1.0879,1.0880,5
2. 2019.09.30,17:46,1.0904,1.0905,1.0904,1.0904,3,2019.09.30,18:01,0.8862,0.8863,0.8861,0.8862,7,2019.09.30,18:02,1.0881,
   1.0881,1.0879,1.0880,4
3. 2019.09.30,17:47,1.0905,1.0905,1.0905,1.0905,2,2019.09.30,18:02,0.8863,0.8863,0.8863,0.8863,1,2019.09.30,18:03,1.0881,
   1.0881,1.0880,1.0881,5
```

Кожний рядок містить наступні поля для 3-х пар, а саме:

1. Дата;
2. Час;
3. Ціна відкриття;
4. Максимум ціни;
5. Мінімум ціни;
6. Ціна закриття;
7. Обсяг торгів.

Для прикладу розглянемо програму *MapReduce*, написану на мові програмування *Java*, що розраховує середнє значення обсягу торгів для 3-х валютних пар.

Програма *MapReduce* для *Hadoop* має містити щонайменше наступні компоненти:

- Реалізація етапу *Map*;
- Реалізація етапу *Reduce*;
- Реалізація коду *Driver*, що описує задачу *MapReduce* для *Hadoop*.

Додатковими компонентами можуть бути реалізації *Combiner*, *Partitioner*, тощо.

Нижче наведено реалізації *Map*, *Reduce* та *Driver* для програми *MapReduce*, розробленій на мові *Java* (рис. 2.10-2.12).

```
public static class AverageCountMapper extends Mapper<Object, Text, Text, MarketAverage>{

public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
String csvLine = value.toString();
String[] csvField = csvLine.split(",");
context.write(MARKET_KEY1, new MarketAverage(Double.parseDouble(csvField[6]), 1));
context.write(MARKET_KEY2, new MarketAverage(Double.parseDouble(csvField[13]), 1));
context.write(MARKET_KEY3, new MarketAverage(Double.parseDouble(csvField[20]), 1));
}
```

Рис. 2.10. Реалізація етапу *Map*.

```
public static class AverageCountReducer extends Reducer<Text, MarketAverage, Text,
MarketAverage>{

public void reduce(Text key, Iterable<MarketAverage> values, Context context) throws
IOException, InterruptedException {

double sum=0;
int count=0;

for(MarketAverage marketAverage: values) {
sum+=(marketAverage.getAverage()*marketAverage.getCount());
count+=marketAverage.getCount();
}

context.write(key, new MarketAverage(sum/count, count));
}

 }
```

Рис. 2.11. Реалізація етапу *Reduce*.

```
...
public static void main(String... args) throws Exception{

Configuration conf = new Configuration();
String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
if (otherArgs.length != 3) {
System.err.println("Usage: AverageCount <hdfs://> <in> <out>");
System.exit(2);
}

FileSystem hdfs=FileSystem.get(new URI(args[0]), conf);
Path resultFolder=new Path(args[2]);
if(hdfs.exists(resultFolder))
hdfs.delete(resultFolder, true);

Job job = Job.getInstance(conf, "Market Average Count");
job.setJarByClass(AverageCount.class);
job.setMapperClass(AverageCountMapper.class);
job.setCombinerClass(AverageCountReducer.class);
job.setReducerClass(AverageCountReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(MarketAverage.class);

for (int i = 1; i < otherArgs.length - 1; i++) {
FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
}
FileOutputFormat.setOutputPath(job, new Path(otherArgs[(otherArgs.length - 1)]));

boolean finishState = job.waitForCompletion(true);
System.out.println("Job Running Time: " + (job.getFinishTime() - job.getStartTime()));

System.exit(finishState ? 0 : 1);
 }

...
```

Рис. 2.12. Реалізація *Driver*, що описує задачу *MapReduce* для *Hadoop*.

Компіляцію програми можна виконувати локально або завантаживши код на кафедральний кластер *Hadoop*.

Розглянемо приклад компіляції на кафедральному кластері. Для цього потрібно підключитись до головного вузла кластеру через *SSH*. Адреса вузла наведена у таблиці 2.1.

```
1. hduser@hadoop-master:~/HadoopMarketAverage$ javac -cp
   /usr/local/hadoop/share/hadoop/common/hadoop-common-2.7.3.jar:/usr/local/hadoop/share/
   hadoop/mapreduce/hadoop-mapreduce-client-core-2.7.3.jar:/usr/local/hadoop/share/hadoop/
   common/lib/commons-cli-1.2.jar -d bin/ src/AverageCount.java
2. hduser@hadoop-master:~/HadoopMarketAverage$ jar cvf HadoopMarketAverage.jar -C bin/ .
   added manifest
   adding: AverageCount$AverageCountReducer.class(in = 1720) (out= 744)(deflated 56%)
   adding: AverageCount$MarketAverage.class(in = 1366) (out= 728)(deflated 46%)
   adding: AverageCount.class(in = 3115) (out= 1576)(deflated 49%)
   adding: AverageCount$AverageCountMapper.class(in = 1693) (out= 702)(deflated 58%)
```

Розроблена програма має наступні аргументи:

- <*hdfs://namenode-ip*> – адреса головного вузла *HDFS*;
- <*in*> – вхідний файл;
- <*out*> – вихідна директорія для результату.

Щоб запустити розроблену програму, необхідно на головному вузлі кластеру *Hadoop* виконати наступну команду:

```
1. hduser@hadoop-master:~$ hadoop jar ~/HadoopMarketAverage/HadoopMarketAverage.jar
   AverageCount hdfs://hadoop-master:54310 market/EURUSD_GBP_CHF.csv
   results/marketAverage
```

Вивід буде подібний наступному:

```
19/10/06 17:24:47 INFO client.RMProxy: Connecting to ResourceManager at hadoop-
master /192.168.11.200:8032
19/10/06 17:24:48 INFO input.FileInputFormat: Total input paths to process : 1
19/10/06 17:24:48 INFO mapreduce.JobSubmitter: number of splits:3
19/10/06 17:24:49 INFO mapreduce.JobSubmitter: Submitting tokens for job:
job_1570371609249_0003
19/10/06 17:24:49 INFO impl.YarnClientImpl: Submitted application
application_1570371609249_0003
19/10/06 17:24:49 INFO mapreduce.Job: The url to track the job: http:// hadoop-
master:8088/proxy/application_1570371609249_0003/
19/10/06 17:24:49 INFO mapreduce.Job: Running job: job_1570371609249_0003
19/10/06 17:24:55 INFO mapreduce.Job: Job job_1570371609249_0003 running in uber
mode : false
19/10/06 17:24:55 INFO mapreduce.Job:  map 0% reduce 0%
19/10/06 17:25:06 INFO mapreduce.Job:  map 13% reduce 0%
19/10/06 17:25:08 INFO mapreduce.Job:  map 70% reduce 0%
19/10/06 17:25:09 INFO mapreduce.Job:  map 95% reduce 0%
19/10/06 17:25:10 INFO mapreduce.Job:  map 100% reduce 0%
19/10/06 17:25:12 INFO mapreduce.Job:  map 100% reduce 8%
19/10/06 17:25:13 INFO mapreduce.Job:  map 100% reduce 21%
```

```
19/10/06 17:25:14 INFO mapreduce.Job:  map 100% reduce 33%
19/10/06 17:25:15 INFO mapreduce.Job:  map 100% reduce 67%
19/10/06 17:25:16 INFO mapreduce.Job:  map 100% reduce 88%
19/10/06 17:25:17 INFO mapreduce.Job:  map 100% reduce 100%
19/10/06 17:25:18 INFO mapreduce.Job: Job job_1570371609249_0003 completed
successfully
19/10/06 17:25:18 INFO mapreduce.Job: Counters: 50
        File System Counters
                FILE: Number of bytes read=915
                FILE: Number of bytes written=839061
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=285874588
                HDFS: Number of bytes written=79
                HDFS: Number of read operations=21
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=8
        Job Counters
                Killed map tasks=1
                Launched map tasks=3
                Launched reduce tasks=4
                Data-local map tasks=3
                Total time spent by all maps in occupied slots (ms)=94250
                Total time spent by all reduces in occupied slots (ms)=79394
                Total time spent by all map tasks (ms)=47125
                Total time spent by all reduce tasks (ms)=39697
                Total vcore-milliseconds taken by all map tasks=47125
                Total vcore-milliseconds taken by all reduce tasks=39697
                Total megabyte-milliseconds taken by all map tasks=96512000
                Total megabyte-milliseconds taken by all reduce tasks=81299456
        Map-Reduce Framework
                Map input records=2000000
                Map output records=6000000
                Map output bytes=114000000
                Map output materialized bytes=387
                Input split bytes=387
                Combine input records=6000000
                Combine output records=15
                Reduce input groups=3
                Reduce shuffle bytes=387
                Reduce input records=15
                Reduce output records=3
                Spilled Records=42
                Shuffled Maps =12
                Failed Shuffles=0
                Merged Map outputs=12
                GC time elapsed (ms)=737
                CPU time spent (ms)=35230
                Physical memory (bytes) snapshot=2240106496
                Virtual memory (bytes) snapshot=27811844096
                Total committed heap usage (bytes)=2287468544
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=285874201
        File Output Format Counters
                Bytes Written=79
Job Running Time: 29807
```

Щоб переглянути результат роботи програми, збережений у *HDFS*, виконайте наступну команду:

```
hduser@hadoop-master:~$ hdfs dfs -cat /user/hduser/results/marketAverage/*
EURCHF  8.873911 (2000000)
EURGBP  5.2978175 (2000000)
EURUSD  6.823027 (2000000)
```

Таблиця 2.1. Параметри кластеру.

	Параметр	**Значення**
1.	Адреса головного вузла	77.47.193.172
2.	Порт SSH	227
3.	Логін	hduser
4.	Пароль	#Tc4j81P

Після підключення до головного вузла ви потрапите до домашнього каталогу користувача.

• У директорії *Labs* створіть власну директорію (наприклад <LabX_Прізвище_Ім’я>) та завантажте до неї свою програму;

• За необхідності скомпілюйте програму;

• Результат виконання програми зберігайте у *HDFS* – */user/hduser/results/*<своя директорія>.

• Після завершення роботи ви можете видалити свої директорії у домашньому каталозі та *HDFS*.

Для роботи із SSH можете використовувати такі програми, як Putty, WinSCP, тощо.

ПРАКТИЧНЕ ЗАВДАННЯ

1. Розробити програму *MapReduce* для кластеру *Hadoop* на мові програмування *Java* або за допомогою *Hadoop Streaming API*. Програма має обробляти вхідний набір даних згідно варіанту (див. таблиця варіантів). Вхідний набір даних знаходиться у директорії *HDFS* – */user/hduser/market/EURUSD_GBP_CHF.csv*;

2. Завантажити та запустити розроблену програму на кластері *Hadoop* (адреса головного вузла наведена у табл. 2.1). Для підключення використовуйте *SSH*.

3. Перевірити коректність роботи розробленої програми. Прокоментуйте вивід *Hadoop* після завершення програми.

СПИСОК КОНТРОЛЬНИХ ПИТАНЬ

Які основні компоненти має кластер *Hadoop*, дайте їх визначення?

Для чого використовується *HDFS*?

Для чого використовується *YARN*?

Назвіть основні компоненти *HDFS*?

Назвіть основні компоненти *YARN*?

Назвіть головні переваги у використанні програмного забезпечення *Hadoop*?

Які засоби моніторингу є у кластері *Hadoop*?

Які основні компоненти має містити програма *MapReduce* для *Hadoop*?

ЛАБОРАТОРНА РОБОТА №3
ОСНОВНІ ПОНЯТТЯ *APACHE PIG*

Мета та основне завдання роботи: ознайомитися із програмним забезпеченням високого рівня для виконання розподілених обчислень на кластері *Hadoop*. Ознайомитися із мовою програмування високого рівня *Pig Latin*. Навчитися розробляти та запускати програму/скрипт *Pig Latin* на кластері *Hadoop*.

ОСНОВНІ ТЕОРЕТИЧНІ ВІДОМОСТІ

Apache Pig – це платформа високого рівня для створення програм, які працюють на *Apache Hadoop*. Мова для цієї платформи називається *Pig Latin*. Вона дозволяє описувати прикладну задачу у вигляді нотацій. Це дає можливість розробляти програми на більш високому рівні і абстрагуватись від програмної моделі *MapReduce*. Використання нотацій у мові *Pig Latin* робить її подібною до мови *SQL* для реляційних систем управління базами даних.

Apache Pig також містить компоненти, що дозволяють виконувати програми, написані на мові *Pig Latin*, шляхом їх конвертації у *MapReduce*. Це робить *Apache Pig* зручним інструментом для аналізу великих обсягів даних.

Мова *Pig Latin* має такі ключові властивості:

• Простота програмування – складні задачі описуються у вигляді послідовних операцій (нотацій), що полегшує розробку та розуміння.

• Можливості оптимізації – формат опису завдань дозволяє системі оптимізувати їх виконання автоматично, дозволяючи користувачеві зосередитись на семантиці, а не на ефективності.

• Розширюваність – розробники можуть створювати власні функції для обробки даних.

Спрощена архітектура платформи *Apache Pig* показана на рис. 3.1.

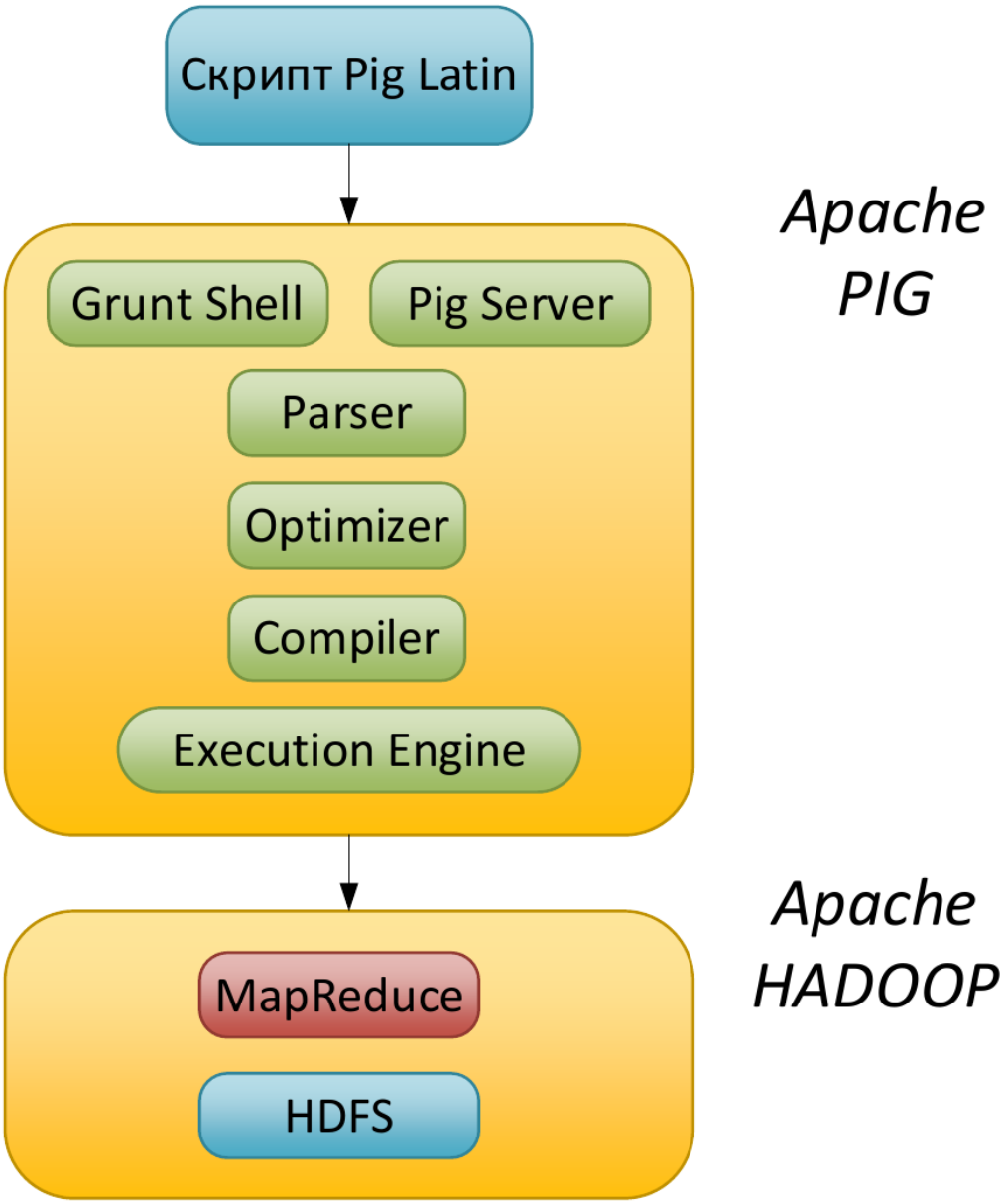


Рис. 3.1. Архітектура *Apache Pig*.

Для прикладу буде розглянута програма, написана на мові *Pig Latin*, що розраховує середнє значення обсягів торгів. Детальний опис задачі та набору даних приведений у лабораторній роботі №2.

```
1.  forex_history = LOAD
    'hdfs://hadoop-master:54310/user/hduser/market/EURUSD_GBP_CHF.csv' USING
    PigStorage(',') as (eurusd_date:chararray, eurusd_time:chararray,
    eurusd_open:float, eurusd_max:float, eurusd_min:float, eurusd_close:float,
    eurusd_volume:float, eurgbp_date:chararray, eurgbp_time:chararray,
    eurgbp_open:float, eurgbp_max:float, eurgbp_min:float, eurgbp_close:float,
    eurgbp_volume:float, eurchf_date:chararray, eurchf_time:chararray,
    eurchf_open:float, eurchf_max:float, eurchf_min:float, eurchf_close:float,
    eurchf_volume:float);

2.  eurusd_data = FOREACH forex_history GENERATE eurusd_date, eurusd_time,
    eurusd_open, eurusd_max, eurusd_min, eurusd_close, eurusd_volume;
3.  eurgbp_data = FOREACH forex_history GENERATE eurgbp_date, eurgbp_time,
    eurgbp_open, eurgbp_max, eurgbp_min, eurgbp_close, eurgbp_volume;
4.  eurchf_data = FOREACH forex_history GENERATE eurchf_date, eurchf_time,
    eurchf_open, eurchf_max, eurchf_min, eurchf_close, eurchf_volume;

5.  eurusd_group = GROUP eurusd_data All;
6.  eurgbp_group = GROUP eurgbp_data All;
7.  eurchf_group = GROUP eurchf_data All;

8.  eurusd_volume_avg = FOREACH eurusd_group GENERATE AVG(eurusd_data.eurusd_volume);
9.  eurgbp_volume_avg = FOREACH eurgbp_group GENERATE AVG(eurgbp_data.eurgbp_volume);
10. eurchf_volume_avg = FOREACH eurchf_group GENERATE AVG(eurchf_data.eurchf_volume);

11. result = UNION eurusd_volume_avg, eurgbp_volume_avg, eurchf_volume_avg;

12. STORE result INTO
    'hdfs://hadoop-master:54310/user/hduser/results/marketAverageLab3' USING
    PigStorage (',');
```

Програма зчитує вхідний набір даних із розподіленої файлової системи за допомогою функції *LOAD* та розділяє його на поля через *PigStorage(',')*

Далі за допомогою функції *FOREACH* для кожного елемента/рядка виділяються поля, що відповідають окремим валютним парам – *GENERATE*

Щоб обчислити середнє для обсягу торгів, спочатку згрупуємо всі дані в спільний ключ *All* за допомогою функції *GROUP*.

В кінці обчислюємо середнє за допомогою функції *AVG* та об’єднуємо результати для трьох валютний пар в остаточний результат – *UNION*.

Щоб зберегти результат до *HDFS* використовуємо функцію *STORE*.

Щоб запустити програму, необхідно на головному вузлі кластеру *Hadoop* виконати наступну команду:

```
1.  hduser@hadoop-master:~$ pig -x mapreduce script_lab3.pig
```

Вивід матиме подібний вигляд:

```
19/10/25 20:53:17 INFO pig.ExecTypeProvider: Picked MAPREDUCE as the ExecType
2019-10-25 20:53:17,453 [main] INFO  org.apache.pig.Main - Apache Pig version
0.17.0 (r1797386) compiled Jun 02 2017, 15:41:58
2019-10-25 20:53:17,453 [main] INFO  org.apache.pig.Main - Logging error messages
to: /usr/local/pig/pig_1572025997451.log
2019-10-25 20:53:18,162 [main] INFO  org.apache.pig.impl.util.Utils - Default
bootup file /home/hduser/.pigbootup not found
2019-10-25 20:53:18,257 [main] INFO
org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is
deprecated. Instead, use mapreduce.jobtracker.address
2019-10-25 20:53:18,257 [main] INFO
org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to
hadoop file system at: hdfs://US64-M1:54310
2019-10-25 20:53:18,724 [main] INFO  org.apache.pig.PigServer - Pig Script ID for
the session: PIG-script_lab3.pig-5209c1eb-3c4a-456f-a129-cc04a8ee64e4
2019-10-25 20:53:18,724 [main] WARN  org.apache.pig.PigServer - ATS is disabled
since yarn.timeline-service.enabled set to false
2019-10-25 20:53:19,610 [main] INFO
org.apache.hadoop.conf.Configuration.deprecation -
mapred.textoutputformat.separator is deprecated. Instead, use
mapreduce.output.textoutputformat.separator
2019-10-25 20:53:19,636 [main] INFO  org.apache.pig.tools.pigstats.ScriptState -
Pig features used in the script: GROUP_BY,UNION
2019-10-25 20:53:19,670 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key
[pig.schematuple] was not set... will not generate code.
2019-10-25 20:53:19,697 [main] INFO
org.apache.pig.newplan.logical.optimizer.LogicalPlanOptimizer -
{RULES_ENABLED=[AddForEach, ColumnMapKeyPrune, ConstantCalculator,
GroupByConstParallelSetter, LimitOptimizer, LoadTypeCastInserter, MergeFilter,
MergeForEach, NestedLimitOptimizer, PartitionFilterOptimizer,
PredicatePushdownOptimizer, PushDownForEachFlatten, PushUpFilter, SplitFilter,
StreamTypeCastInserter]}
2019-10-25 20:53:19,768 [main] INFO
org.apache.pig.impl.util.SpillableMemoryManager - Selected heap (PS Old Gen) of
size 699400192 to monitor. collectionUsageThreshold = 489580128, usageThreshold =
489580128
2019-10-25 20:53:19,906 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MRCompiler - File
concatenation threshold: 100 optimistic? false
2019-10-25 20:53:19,929 [main] INFO
org.apache.pig.backend.hadoop.executionengine.util.CombinerOptimizerUtil - Choosing
to move algebraic foreach to combiner
2019-10-25 20:53:19,970 [main] INFO
org.apache.pig.backend.hadoop.executionengine.util.CombinerOptimizerUtil - Choosing
to move algebraic foreach to combiner
2019-10-25 20:53:19,972 [main] INFO
org.apache.pig.backend.hadoop.executionengine.util.CombinerOptimizerUtil - Choosing
to move algebraic foreach to combiner
2019-10-25 20:53:19,993 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MultiQueryOptimizer -
MR plan size before optimization: 5
2019-10-25 20:53:19,995 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MultiQueryOptimizer -
Merged MR job 149 into MR job 146
2019-10-25 20:53:19,995 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MultiQueryOptimizer -
Merged MR job 151 into MR job 146

. . .
```

```
. . .
2019-10-25 20:54:36,639 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher -
100% complete
2019-10-25 20:54:36,710 [main] INFO
org.apache.pig.tools.pigstats.mapreduce.SimplePigStats - Script Statistics:

HadoopVersion   PigVersion      UserId  StartedAt       FinishedAt      Features
2.7.3   0.17.0  hduser  2019-10-25 20:53:20     2019-10-25 20:54:36
GROUP_BY,UNION

Success!

Job Stats (time in seconds):
JobId   Maps    Reduces MaxMapTime      MinMapTime      AvgMapTime
MedianMapTime   MaxReduceTime   MinReduceTime   AvgReduceTime   MedianReducetime
   Alias    Feature Outputs
job_1572025593211_0001  8       1       40      23      36      38      16      16
    16      16
eurchf_data,eurchf_group,eurchf_volume_avg,eurgbp_data,eurgbp_group,eurgbp_volume_a
vg,eurusd_data,eurusd_group,eurusd_volume_avg,forex_history  MULTI_QUERY,COMBINER
job_1572025593211_0002  3       0       3       3       3       3       0       00
    0       result  MAP_ONLY
hdfs://hadoop-master:54310/user/hduser/results/marketAverageLab3,

Input(s):
Successfully read 2000000 records (285875374 bytes) from: "hdfs://hadoop-
master:54310/user/hduser/market/EURUSD_GBP_CHF.csv"

Output(s):
Successfully stored 3 records (28 bytes) in:
"hdfs://hadoop-master:54310/user/hduser/results/marketAverageLab3"

Counters:
Total records written : 3
Total bytes written : 28
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_1572025593211_0001  ->      job_1572025593211_0002,
job_1572025593211_0002


2019-10-25 20:54:36,713 [main] INFO  org.apache.hadoop.yarn.client.RMProxy -
Connecting to ResourceManager at US64-M1/192.168.11.200:8032
2019-10-25 20:54:36,717 [main] INFO  org.apache.hadoop.mapred.ClientServiceDelegate
- Application state is completed. FinalApplicationStatus=SUCCEEDED. Redirecting to
job history server
. . .
2019-10-25 20:54:36,891 [main] INFO  org.apache.hadoop.mapred.ClientServiceDelegate
- Application state is completed. FinalApplicationStatus=SUCCEEDED. Redirecting to
job history server
2019-10-25 20:54:36,917 [main] INFO  org.apache.hadoop.yarn.client.RMProxy -
Connecting to ResourceManager at US64-M1/192.168.11.200:8032
2019-10-25 20:54:36,921 [main] INFO  org.apache.hadoop.mapred.ClientServiceDelegate
- Application state is completed. FinalApplicationStatus=SUCCEEDED. Redirecting to
job history server
2019-10-25 20:54:36,948 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher -
Success!
2019-10-25 20:54:36,972 [main] INFO  org.apache.pig.Main - Pig script completed in
1 minute, 19 seconds and 780 milliseconds (79780 ms)
```

Щоб переглянути результат роботи програми, збережений у *HDFS*, виконайте наступну команду:

```
hduser@hadoop-master:~$ hdfs dfs -cat /user/hduser/results/marketAverageLab3/*
6.823027
5.2978175
8.873911
```

ПРАКТИЧНЕ ЗАВДАННЯ

1. Розробити програму *Apache Pig* для кластеру *Hadoop* на мові програмування *Pig Latin.* Програма має обробляти вхідний набір даних згідно варіанту (див. таблиця варіантів). Вхідний набір даних знаходиться у директорії *HDFS – /user/hduser/market/EURUSD_GBP_CHF.csv*;

2. Завантажити та запустити розроблену програму на кластері *Hadoop* (адреса головного вузла наведена у табл. 2.1). Для підключення використовуйте *SSH*;

3. Перевірити коректність роботи розробленої програми.

СПИСОК КОНТРОЛЬНИХ ПИТАНЬ

Дайте визначення *Apache Pig*?

Для чого використовується *Apache Pig*?

Які переваги дає використання *Apache Pig*?

Назвіть ключові властивості мови *Pig Latin*?

ЛАБОРАТОРНА РОБОТА №4
ОСНОВНІ ПОНЯТТЯ *APACHE HIVE*

Мета та основне завдання роботи: ознайомитися із програмним забезпеченням високого рівня для виконання розподілених обчислень на кластері *Hadoop*. Ознайомитися із засобами *Apache Hive* та мовою *HiveQL*. Навчитися розробляти та виконувати запити *HiveQL* на кластері *Hadoop*.

ОСНОВНІ ТЕОРЕТИЧНІ ВІДОМОСТІ

Apache Hive – це інфраструктура сховища даних, яка полегшує роботу та управління великими обсягами даних, що знаходиться в розподіленій системі зберігання даних. *Hive* базується на *Hadoop* та розроблений компанією *Facebook*. *Hive* дозволяє виконувати запити даних за допомогою *SQL*-подібної мови під назвою *HiveQL*.

Внутрішній компілятор перетворює оператори *HiveQL* в завдання *MapReduce,* які потім передаються до кластеру *Hadoop* на виконання.

Hive дуже схожий на традиційну базу даних з доступом за допомогою *SQL*. Однак, оскільки *Hive* базується на операціях *Hadoop* та *MapReduce*, є кілька ключових особливостей:

- Оскільки *Hadoop* призначений для тривалого послідовного зчитування даних, запити *Hive* мають великий час затримки. Це означає, що *Hive* не підходить для програм, які потребують дуже швидкого часу відгуку, який можна очікувати при використанні традиційних баз даних *RDBMS*;
- *Hive* призначений для операцій читання і тому він не оптимізований для обробки запитів, які мають високий відсоток операцій запису.

Відмінності *Apache Hive* від *Apache Pig:*

- *Apache Hive* — розподілене сховище даних. *Apache Pig* — середовище для дослідження великих обсягів даних;
- *HiveQL* – це декларативна мова подібна до *SQL*. *PigLatin* – мова потоку даних.

Спрощена архітектура *Apache Hive* показана на рис. 4.1.

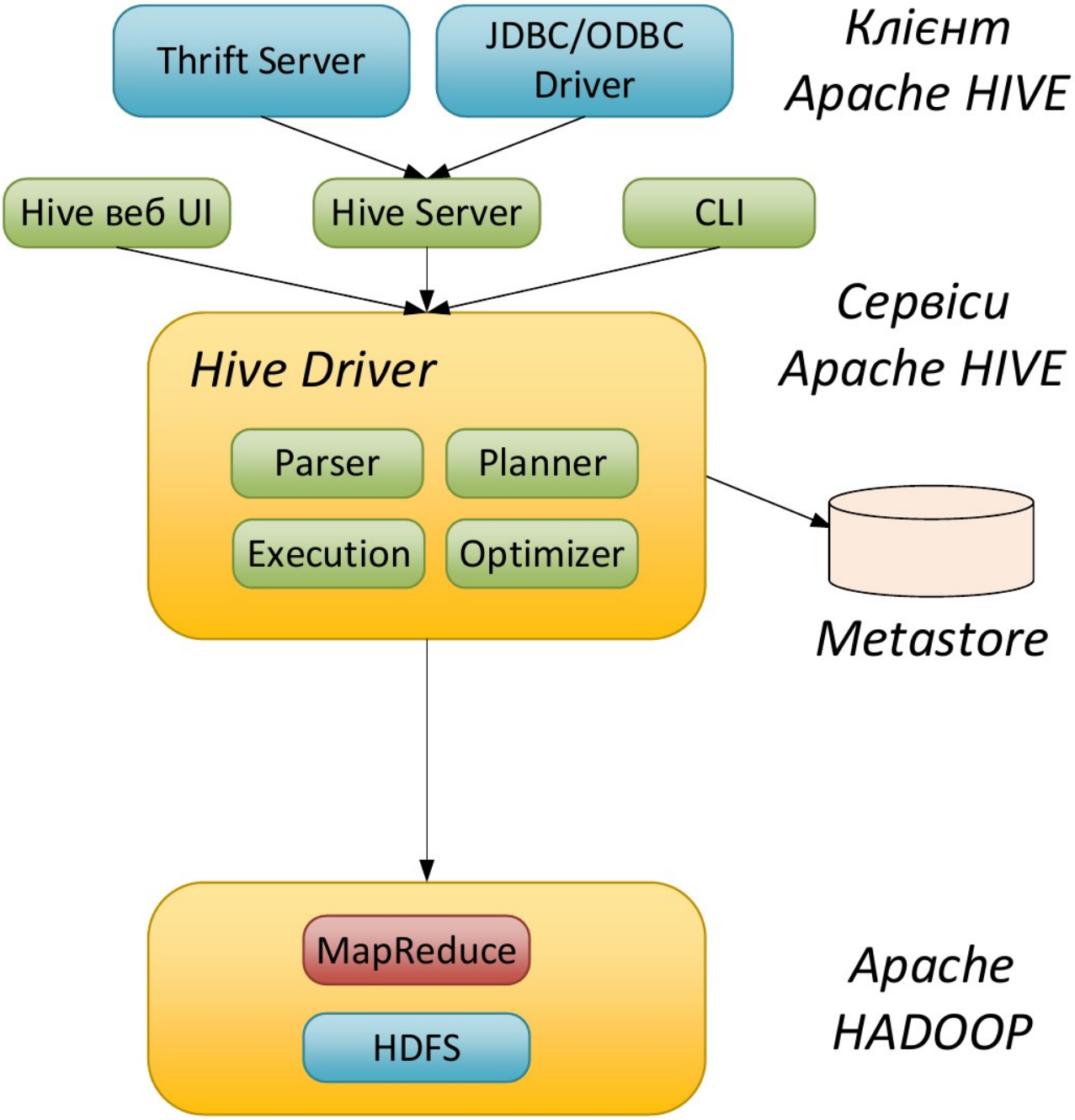


Рис. 4.1. Архітектура *Apache Hive*.

Для прикладу буде розглянутий запит написаний на мові *HiveQL*, що дозволяє отримати середнє значення обсягу торгів для 2-х пар.

Для початку роботи із *Hive* необхідно із домашнього каталогу зайти у *Hive command line interface – CLI*.

```
hduser@hadoop-master:~$ hive
SLF4J: Class path contains multiple SLF4J bindings.
. . .
hive>
```

Далі у *Hive CLI* пишемо запит, щоб отримати середнє значення обсягу торгів.

```
hive> SELECT avg(eurusd_volume), avg(eurgbp_volume)
> FROM market.forex;
```

Після опрацювання запиту у *Hive*, отримаємо наступний вивід із значенням середнього обсягу торгів.

```
Query ID = hduser_20191109161600_ec9403d2-95d1-4763-94c0-8f23f9d233f1
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1572808850918_0323, Tracking URL =
http://hadoop-master:8088/proxy/application_1572808850918_0323/
Kill Command = /usr/local/hadoop/bin/hadoop job  -kill job_1572808850918_0323
Hadoop job information for Stage-1: number of mappers: 2; number of reducers: 1
2019-11-09 16:16:27,830 Stage-1 map = 0%,  reduce = 0%
2019-11-09 16:16:42,001 Stage-1 map = 50%,  reduce = 0%, Cumulative CPU 5.58 sec
2019-11-09 16:16:45,203 Stage-1 map = 67%,  reduce = 0%, Cumulative CPU 14.44 sec
2019-11-09 16:16:46,262 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 16.01 sec
2019-11-09 16:16:56,757 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 20.86
sec
MapReduce Total cumulative CPU time: 20 seconds 860 msec
Ended Job = job_1572808850918_0323
MapReduce Jobs Launched:
Stage-Stage-1: Map: 2  Reduce: 1   Cumulative CPU: 20.86 sec   HDFS Read: 285892624
HDFS Write: 118 SUCCESS
Total MapReduce CPU Time Spent: 20 seconds 860 msec
OK
6.823027        5.2978175
Time taken: 57.438 seconds, Fetched: 1 row(s)
```

Формат таблиці *market.forex* приведений нижче.

```
hive> describe market.forex;
OK
eurusd_date             string
eurusd_time             string
eurusd_open             float
eurusd_max              float
eurusd_min              float
eurusd_close            float
eurusd_volume           float
eurgbp_date             string
eurgbp_time             string
eurgbp_open             float
eurgbp_max              float
eurgbp_min              float
eurgbp_close            float
eurgbp_volume           float
eurchf_date             string
eurchf_time             string
eurchf_open             float
eurchf_max              float
eurchf_min              float
eurchf_close            float
eurchf_volume           float
Time taken: 7.372 seconds, Fetched: 21 row(s)
```

ПРАКТИЧНЕ ЗАВДАННЯ

1. Створити запит на мові *HiveQL*, що буде обробляти дані (у таблиці *market.forex* або *market.forex_shift*) згідно варіанту (див. таблиця варіантів);

2. Перевірити коректність виконання створеного запиту *HiveQL* на кластері *Hadoop* (адреса головного вузла наведена у табл. 2.1). Для підключення використовуйте *SSH*.

СПИСОК КОНТРОЛЬНИХ ПИТАНЬ

Дайте визначення *Apache Hive*?

Яким чином виконується обробка даних у *Apache Hive*?

В чому різниця між *Apache Pig* та *Apache Hive*?

Які переваги та недоліки має *Apache Hive*?

ЛАБОРАТОРНА РОБОТА №5
ДИСПЕТЧЕР *APACHE OOZIE*

Мета та основне завдання роботи: ознайомитися із диспетчером робіт для кластеру *Hadoop*. Ознайомитися із принципами роботи диспетчера *Apache Oozie*. Навчитися створювати простий потік завдань для кластеру *Hadoop*, використовуючи диспетчер *Apache Oozie*.

ОСНОВНІ ТЕОРЕТИЧНІ ВІДОМОСТІ

Apache Oozie – диспетчер, що спрощує процес створення потоків завдань і їх координацію на кластері *Hadoop*. Він дозволяє об’єднувати послідовності декількох завдань в одну логічну одиницю роботи. За допомогою *Apache Oozie* можна виконувати складні перетворення даних, об'єднувати обробку різноманітних індивідуальних завдань і навіть підпотоків робіт. Це дозволяє краще контролювати виконання складних завдань і полегшує повторення цих завдань із заздалегідь заданими інтервалами. Оскільки *Oozie* інтегрований зі стеком *Apache Hadoop*, він підтримує *MapReduce*, *Pig*, *Hive*, тощо. Також підтримуються операції файлової системи *HDFS* та *Java*-додатки.

Виділяють декілька типів робіт *Oozie*:

- *Oozie Workflow* – має вигляд орієнтованого ациклічного графу, що задає послідовність виконуваних завдань (дій);
- *Oozie Coordinator* – задає потік завдань *Oozie*, ініційованих за часом або за наявності даних;
- *Oozie Bundle* – дозволяє організовувати групи завдань *Oozie*.

Типовий потік завдань *Oozie* може складатись із вузлів управління (*Start*, *Decision*, *Fork*, *Join* та *End*) і вузлів дії (*MapReduce*, *Pig*, *Hive*, *Java*, *Shell*, тощо). Приклад потоку роботи *Oozie* наведений на рис. 5.1.

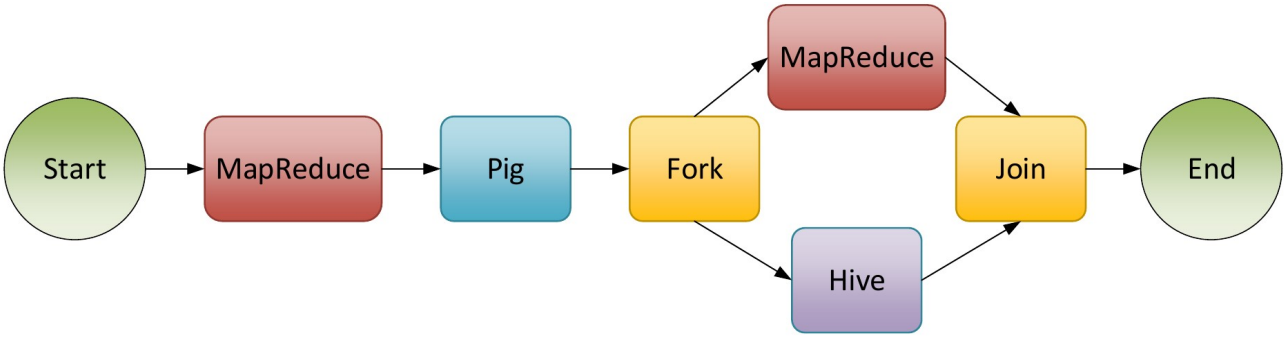


Рис. 5.1. Приклад потоку завдань *Apache Oozie*.

Нижче приведено спрощену архітектуру *Apache Oozie* (рис. 5.2).

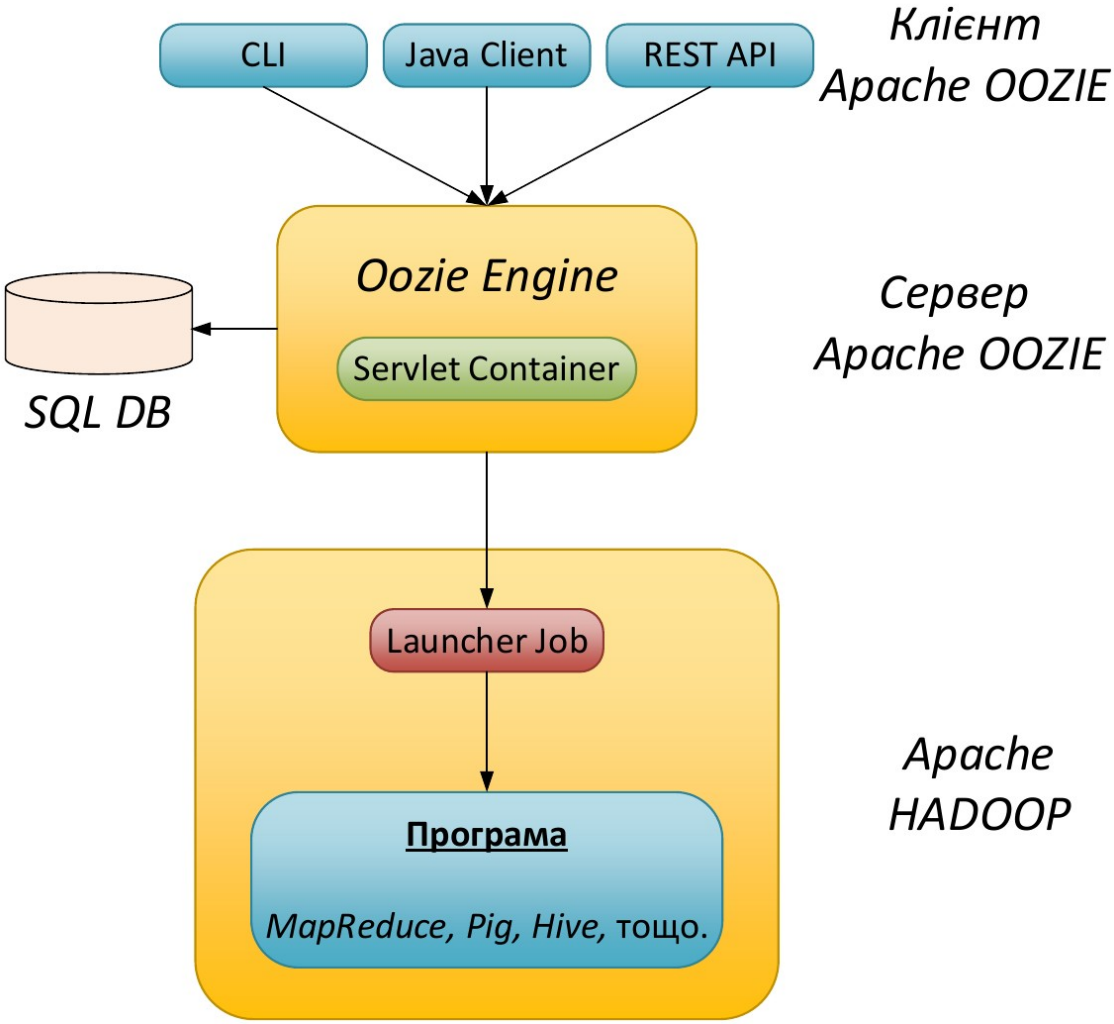


Рис. 5.2. Архітектура *Apache Oozie*.

Для прикладу розглянемо роботу *Oozie*, що складається з одного завдання *MapReduce* із лабораторної роботи №2 – розрахунок середнього значення обсягу торгів. Програма скомпільована та запакована у *JAR*-архів.

Щоб описати роботу *Oozie* необхідно створити два файли:

- *job.properties* – містить системні параметри та змінні користувача;
- *workflow.xml* – містить послідовності завдань (дій) в роботі *Oozie*.

Також необхідно створити директорію у *HDFS*, в якій будуть міститись файл *workflow.xml* та інші файли, що необхідні для виконання роботи.

В даному прикладі додатково треба створити директорію *lib*, в яку потрібно завантажити *JAR*-архів програми *MapReduce*.

Вміст файлу *job.properties*, для даного прикладу, приведений нижче.

```
nameNode=hdfs://hadoop-master:54310
jobTracker=hadoop-master:8032
workflowRoot=oozieLab5Example
oozie.wf.application.path=${nameNode}/user/hduser/results/${workflowRoot}
mapredInput=${nameNode}/user/hduser/market/EURUSD_GBP_CHF.csv
mapredOutput=${nameNode}/user/hduser/results/${workflowRoot}/mapred
```

де *nameNode* та *jobTracker* – адреса головного вузла *HDFS* і *YARN* відповідно.

oozie.wf.application.path – задає директорію *HDFS*, в якій зберігаються файли, що необхідні для виконання роботи *Oozie*.

Вміст файлу *workflow.xml*:

```
<workflow-app xmlns="uri:oozie:workflow:0.1" name="OozieLab5">

  <start to="jar-mapreduce"/>

  <action name="jar-mapreduce">
    <java>
      <job-tracker>${jobTracker}</job-tracker>
      <name-node>${nameNode}</name-node>
      <configuration>
        <property>
          <name>oozie.launcher.mapreduce.map.memory.mb</name>
          <value>1024</value>
        </property>
        <property>
          <name>oozie.launcher.mapreduce.reduce.memory.mb</name>
          <value>1024</value>
        </property>
        <property>
          <name>oozie.launcher.yarn.app.mapreduce.am.resource.mb</name>
          <value>1024</value>
        </property>
      </configuration>
      <main-class>AverageCount</main-class>
      <arg>${nameNode}</arg>
      <arg>${mapredInput}</arg>
      <arg>${mapredOutput}</arg>

    </java>

    <ok to="end"/>
    <error to="fail"/>
  </action>

  <kill name="fail">
    <message>Map/Reduce failed, error message</message>
  </kill>
  <end name="end"/>

</workflow-app>
```

В даному прикладі використовуємо завдання — дію <*java*>, оскільки розроблена програма *MapReduce* вже містить клас *Driver*, в якому задано реалізації фаз *Map/Reduce* та вказано типи вхідних/вихідних даних.

У тегу <*configuration*> задаються параметри для завдання *MapReduce*, наприклад, обсяг пам’яті контейнерів *Map/Reduce*, *ApplicationMaster*, тощо.

Тег <*main-class*> задає основний клас нашої програми.

Теги <*arg*> задають параметри запуску програми.

В результаті отримаємо два файли, що описують роботу *Oozie*. Далі необхідно завантажити їх та *JAR*-архів програми *MapReduce* на кластер *Hadoop*.

Для цього, у домашньому каталозі у директорії *Labs* створюємо свою директорію, наприклад, *oozieLab5Example* і завантажуємо до неї файли.

```
hduser@hadoop-master:~/Labs/oozieLab5Example$ ls -l
total 16
-rw-r--r-- 1 hduser hadoop 4710 Dec 17 20:20 HadoopMarketAverage.jar
-rw-r--r-- 1 hduser hadoop  306 Dec 17 20:21 job.properties
-rw-rw-r-- 1 hduser hadoop 1056 Dec 17 20:20 workflow.xml
```

Далі створюємо директорію у *HDFS* та завантажуємо до неї *workflow.xml*. Додатково створюємо директорію *lib* і завантажуємо до неї *JAR*-архів програми.

```
hduser@hadoop-master:~/Labs/oozieLab5Example$ hdfs dfs -mkdir
/user/hduser/results/oozieLab5Example
hduser@hadoop-master:~/Labs/oozieLab5Example$ hdfs dfs -put workflow.xml
/user/hduser/results/oozieLab5Example
hduser@hadoop-master:~/Labs/oozieLab5Example$ hdfs dfs -mkdir
/user/hduser/results/oozieLab5Example/lib
hduser@hadoop-master:~/Labs/oozieLab5Example$ hdfs dfs -put
HadoopMarketAverage.jar /user/hduser/results/oozieLab5Example/lib
```

Щоб запустити роботу *Oozie*, виконуємо наступну команду.

```
hduser@hadoop-master:~/Labs/oozieLab5Example$ oozie job -oozie
http://localhost:11000/oozie -config job.properties –run
```

Переглядаємо результат після завершення роботи *Oozie* (див. наступну сторінку).

```
hduser@hadoop-master:~/Labs/oozieLab5Example$ hdfs dfs -cat
/user/hduser/results/oozieLab5Example/mapred/*
EURCHF  8.873911 (2000000)
EURGBP  5.2978175 (2000000)
EURUSD  6.823027 (2000000)
```

Слідкувати за ходом виконання роботи *Oozie* можна на спеціальному ресурсі. Для цього зробіть *SSH port forwarding* порта 11000 – веб-консоль *Oozie* (рис. 5.3) та 8088 – веб сторінка менеджеру ресурсів *YARN* (рис. 5.4).

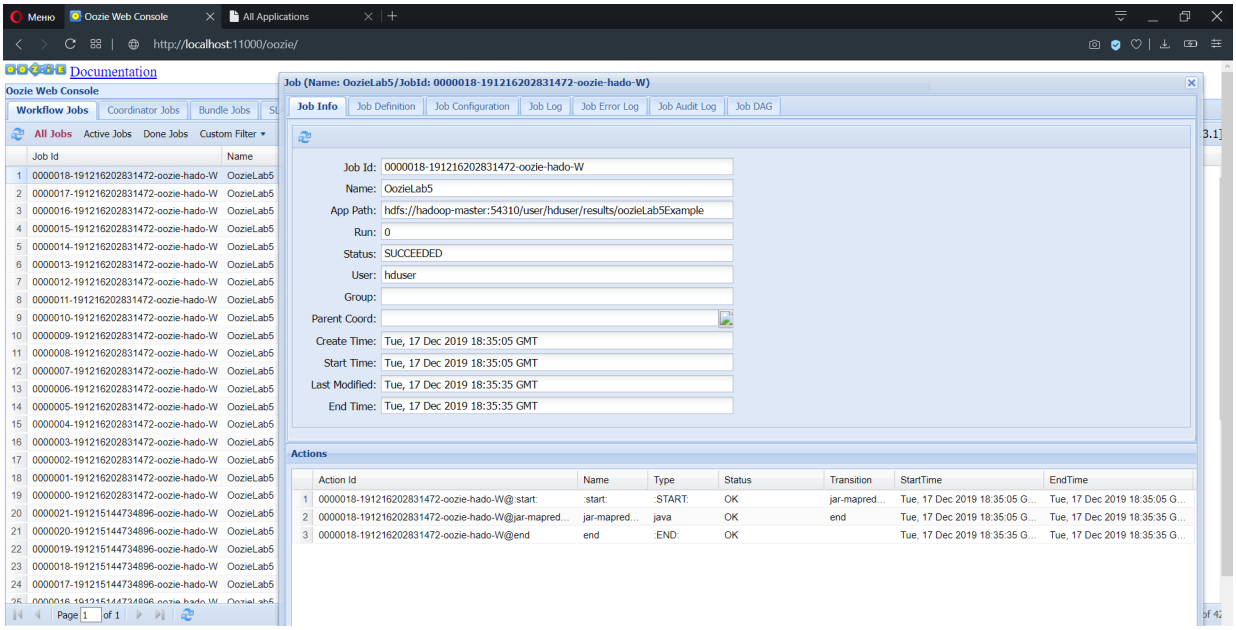


Рис. 5.3. Веб-консоль *Oozie*.

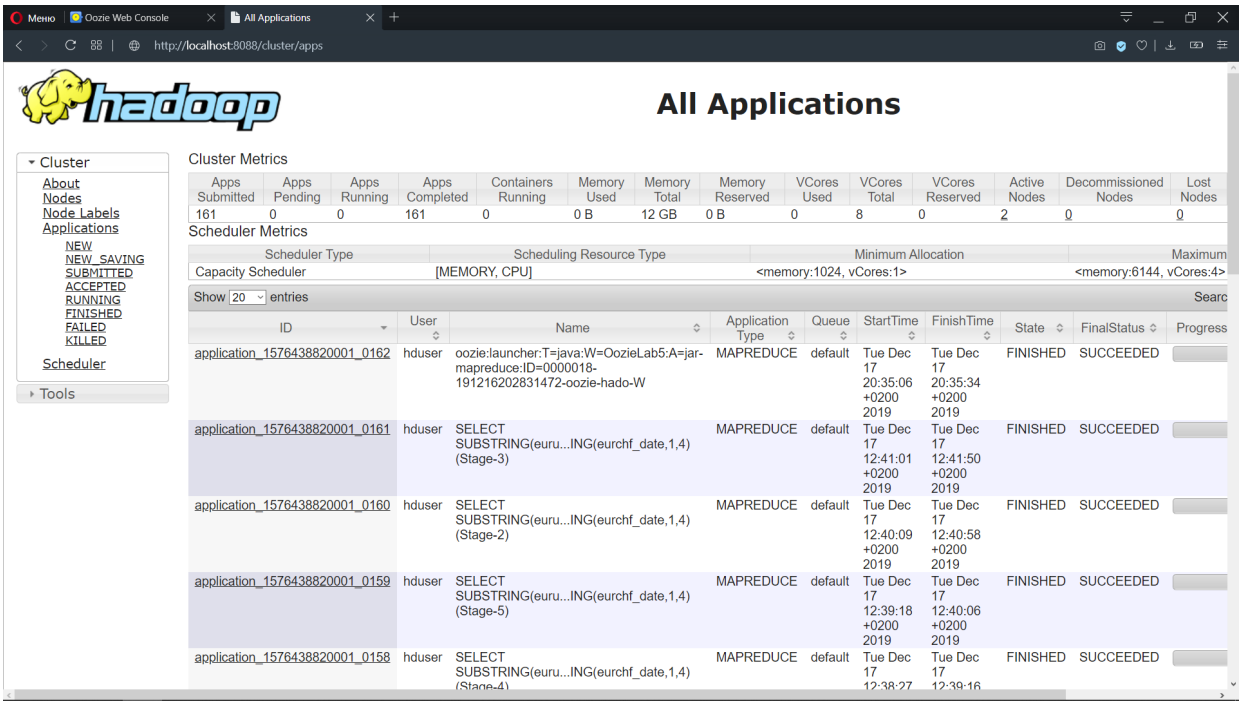
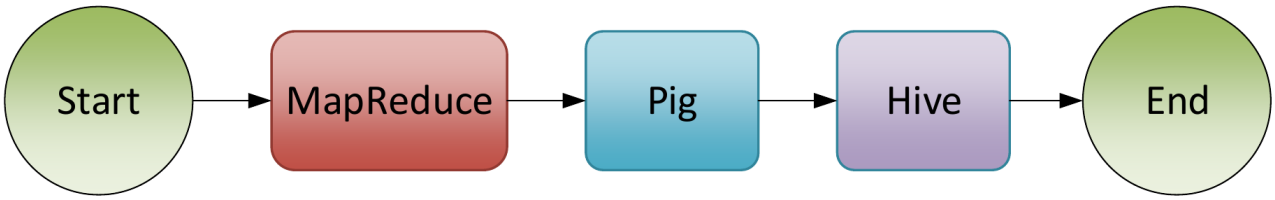


Рис. 5.4. Веб-сторінка менеджеру ресурсів *YARN*.

ПРАКТИЧНЕ ЗАВДАННЯ

1. Створити роботу для диспетчеру *Apache Oozie*, що складається із послідовності трьох завдань: *MapReduce* (лаб. 2), *Pig* (лаб. 3) та *Hive* (лаб. 4);



2. Запустити створену роботу *Oozie* на кластері *Hadoop*.

СПИСОК КОНТРОЛЬНИХ ПИТАНЬ

Дайте визначення *Apache Oozie*?

Для чого використовується *Apache Oozie*?

Які типи робіт дає можливість створювати *Apache Oozie*?

Дайте визначення потоку завдань *Apache Oozie*?

ДОДАТОК 1
НАЛАШТУВАННЯ ЛОКАЛЬНОГО КЛАСТЕРА *HADOOP*

В даному додатку будуть представлені основні кроки, які необхідно виконати для успішного встановлення та налаштування кластеру *Hadoop.*

В якості тестової інфраструктури було створено три віртуальні машини *KVM:*

1. *VT-USHM1*;
2. *VT-USHS1*;
3. *VT-USHS2*.

Для кожної віртуальної машини виділені наступні ресурси:

- Процесор – 4 ядра;
- Пам’ять – 8 ГБ;
- Диск – 64 ГБ.

На кожній віртуальній машині встановлено *Ubuntu Server 18.04.6 LTS*

Налаштування мережевих інтерфейсів віртуальних машин:

1. *VT-USHM1* – 192.168.1.230;
2. *VT-USHS1* – 192.168.1.231;
3. *VT-USHS2* – 192.168.1.232.

Попереднє налаштування системи

Наступні кроки необхідно виконати на кожному вузлі, який буде працювати у кластері Hadoop.

Встановлення Java

Для роботи *Hadoop* необхідно встановити *Java*. В даному прикладі використовувалась версія *Java – 1.8.0_111*.

У терміналі вводимо команди:

```
1. root@VT-USHM1:~# sudo add-apt-repository ppa:webupd8team/java
2. root@VT-USHM1:~# sudo apt-get update
3. root@VT-USHM1:~# sudo apt-get install oracle-java8-installer
```

Створення користувача *Hadoop*

Для зручності роботи із *Hadoop*, у системі необхідно створити окремого користувача. Назвемо його “*hduser*”.

```
1. root@VT-USHM1:~# sudo addgroup hadoop
2. root@VT-USHM1:~# sudo adduser --ingroup hadoop hduser
```

Налаштування *hosts* файлу

Для взаємодії між вузлами кластера, *Hadoop* використовує доменні імена. Тому для можливості доступу до вузла по імені, задамо їх у файлі “*/etc/hosts*”.

```
   # Редагуємо файл (у терміналі відкриється редактор)
1. root@VT-USHM1:~# nano /etc/hosts
   # Після редагування файл /etc/hosts повинен мати наступний зміст.
2. root@VT-USHM1:~# cat /etc/hosts
    127.0.0.1       localhost
    192.168.1.230  VT-USHM1
    192.168.1.231  VT-USHS1
    192.168.1.232  VT-USHS2

    # The following lines are desirable for IPv6 capable hosts
    ::1     localhost ip6-localhost ip6-loopback
    ff02::1 ip6-allnodes
    ff02::2 ip6-allrouters
```

Налаштування *SSH*

Для коректної взаємодії вузлів кластеру необхідно виконати налаштування *SSH* доступу на основі ключів.

```
   # Генеруємо ключ
1. root@VT-USHM1:~# su hduser
2. hduser@VT-USHM1:/root$ ssh-keygen -t rsa -P ""
   Generating public/private rsa key pair.
   Enter file in which to save the key (/home/hduser/.ssh/id_rsa):
   Created directory '/home/hduser/.ssh'.
   Your identification has been saved in /home/hduser/.ssh/id_rsa.
   Your public key has been saved in /home/hduser/.ssh/id_rsa.pub.
   The key fingerprint is:
   9b:82:ea:58:b4:e0:35:d7:ff:19:66:a6:ef:ae:0e:d2 hduser@ubuntu
   The key's randomart image is:
   [...snipp...]

   # Додаємо ключ у список авторизованих ключів (необхідно, щоб
   отримати доступ до машини за згенерованим ключем)

3. hduser@VT-USHM1:/root$ cat ~/.ssh/id_rsa.pub >>
   ~/.ssh/authorized_keys

   # Копіюємо ключ на інші вузли кластеру (необхідно для отримання
   доступу до вузлів використовуючи ключ, згенерований на кроці 2)

4. hduser@VT-USHM1:/root$ ssh-copy-id -i ~/.ssh/id_rsa.pub hduser@VT-
   USHS1
5. hduser@VT-USHM1:/root$ ssh-copy-id -i ~/.ssh/id_rsa.pub hduser@VT-
   USHS2
```

Перевірка *SSH* доступу

Перевірку можливості підключення потрібно виконати “з кожного вузла до кожного вузла”. Нижче показано перевірку підключення з вузла *VT-USHM1* до вузла *VT-USHS1*.

```
1. hduser@VT-USHM1:~$ ssh hduser@VT-USHS1
   The authenticity of host 'vt-ushs1 (192.168.1.231)' can't be
   established.
   ECDSA key fingerprint is
   b5:cf:ba:53:4c:ce:10:20:75:c4:af:c1:5f:e9:de:6f.
   Are you sure you want to continue connecting (yes/no)? yes

   # Відповідаємо ‘yes’, після чого повинно виконатись підключення до
   вузла VT-USHS1 без запиту паролю

   Warning: Permanently added 'vt-ushs1,192.168.1.231' (ECDSA) to the
   list of known hosts.

   Welcome to Ubuntu 18.04.6 LTS

    * Documentation:  https://help.ubuntu.com/

     . . .


   Last login: Thu Dec  1 19:07:51 2020 from vt-ushm1
   hduser@VT-USHS1:~$
```

Після виконання показаних вище кроків, вузли майбутнього тестового кластеру готові для встановлення на них програмного забезпечення *Hadoop.*

Встановлення *Hadoop*

Показані нижче кроки необхідно виконати на кожному вузлі.

В даному прикладі використовується версія *Hadoop – 2.7.3.*

Завантажити дистрибутив *Hadoop* можна з офіційного сайту https://hadoop.apache.org/releases.html

Розпаковуємо завантажений архів та копіюємо дані у директорію де буде встановлений *Hadoop* (у прикладі це "*/usr/local/hadoop*").

```
# Розпаковка архіву
root@VT-USHM1:/home/hduser# tar xzf hadoop-2.7.3.tar.gz

# Архів буде розпаковано у директорію /home/hduser/hadoop-2.7.3
# Копіюємо Hadoop у директорію встановлення (/usr/local/hadoop)
root@VT-USHM1:/home/hduser# cp –r hadoop-2.7.3 /usr/local/Hadoop

# Змінюємо права доступу та власника директорії на користувача hduser
root@VT-USHM1:/home/hduser# sudo chown -R hduser:hadoop /usr/local/hadoop
```

Редагування файлу */home/hduser/.bashrc*

```
# Відкриваємо файл .bashrc на редагування і додаємо наступні параметри у кінець файлу
root@VT-USHM1:/home/hduser# nano .bashrc

# Setting the evironment variables for running Java and Hadoop commands
export HADOOP_HOME=/usr/local/hadoop
export JAVA_HOME=/usr/lib/jvm/java-8-oracle

# Alias for Hadoop commands
unalias fs &> /dev/null
alias fs="hadoop fs"
unalias hls &> /dev/null
alias hls="fs -ls"

# Defining the function for compressing the MapReduce job output
lzohead () {
  hadoopfs -cat $1 | lzop -dc | head -1000 | less
}

#Adding HADOOP_HOME variable to PATH
export PATH=$PATH:$HADOOP_HOME/bin
```

Файли конфігурації *Hadoop*

Список файлів, які необхідно редагувати:

• */usr/local/Hadoop/etc/hadoop/core-site.xml* – основні налаштування;

• */usr/local/Hadoop/etc/hadoop/hdfs-site.xml* – налаштування файлової системи *Hadoop*;

• */usr/local/Hadoop/etc/hadoop/mapred-site.xml* – налаштування виконання задач *MapReduce*;

• */usr/local/Hadoop/etc/hadoop/yarn-site.xml* – налаштування менеджера ресурсів *YARN*.

Детальний опис значення параметрів можна знайти на офіційному сайті:

• https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/core-default.xml

• https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/hdfs-default.xml

• https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/mapred-default.xml

• https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-common/yarn-default.xml

Файли конфігурації повинні мати наступні основні параметри:

1. *core-site.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing,
software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
  See the License for the specific language governing permissions
and
  limitations under the License. See accompanying LICENSE file.
-->

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

<configuration>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/usr/local/hadoop/app/hadoop/tmp</value>
  </property>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://VT-USHM1:54310</value>
  </property>
  <property>
    <name>io.file.buffer.size</name>
    <value>131072</value>
  </property>
</configuration>
```

2. *hdfs-site.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing,
software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
  See the License for the specific language governing permissions
and
  limitations under the License. See accompanying LICENSE file.
-->

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

<configuration>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:/usr/local/hadoop/store/hdfs/namenode</value>
  </property>
  <property>
    <name>dfs.datanode.name.dir</name>
    <value>file:/usr/local/hadoop/store/hdfs/datanode</value>
  </property>
  <property>
    <name>dfs.namenode.checkpoint.dir</name>
    <value>file:/usr/local/hadoop/store/hdfs/namesecondary</value>
  </property>
  <property>
    <name>dfs.block.size</name>
    <value>134217728</value>
  </property>
</configuration>
```

3. *mapred-site.xml*

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
  <property>
    <name>mapreduce.jobhistory.address</name>
    <value>VT-USHM1:10020</value>
  </property>
  <property>
    <name>mapreduce.jobhistory.webapp.address</name>
    <value>VT-USHM1:19888</value>
  </property>
  <property>
    <name>yarn.app.mapreduce.am.staging-dir</name>
    <value>/user/hduser</value>
  </property>
  <property>
    <name>mapred.child.java.opts</name>
    <value>-Djava.security.egd=file:/dev/../dev/urandom</value>
  </property>
  <property>
    <name>mapreduce.map.memory.mb</name>
    <value>2048</value>
  </property>
  <property>
    <name>mapreduce.reduce.memory.mb</name>
    <value>4096</value>
  </property>
  <property>
    <name>mapreduce.map.java.opts</name>
    <value>-Xmx1024m</value>
  </property>
  <property>
    <name>mapreduce.reduce.java.opts</name>
    <value>-Xmx3072m</value>
  </property>

</configuration>
```

4. *yarn-site.xml*

```
<?xml version="1.0"?>
<configuration>

  <property>
    <name>yarn.resourcemanager.hostname</name>
    <value>VT-USHM1</value>
  </property>

  <property>
    <name>yarn.resourcemanager.bind-host</name>
    <value>0.0.0.0</value>
  </property>

  <property>
    <name>yarn.nodemanager.bind-host</name>
    <value>0.0.0.0</value>
  </property>

  <property>
    <name>yarn.nodemanager.resource.memory-mb</name>
    <value>6144</value>
  </property>

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

  <property>

<name>yarn.nodemanager.aux-services.mapreduce_shuffle.class</name>
    <value>org.apache.hadoop.mapred.ShuffleHandler</value>
  </property>

  <property>
    <name>yarn.log-aggregation-enable</name>
    <value>true</value>
  </property>

  <property>
    <name>yarn.nodemanager.local-dirs</name>
    <value>file:/usr/local/hadoop/app/yarn/local</value>
  </property>

  <property>
    <name>yarn.nodemanager.log-dirs</name>
    <value>file:/usr/local/hadoop/app/yarn/log</value>
  </property>

  <property>
    <name>yarn.nodemanager.remote-app-log-dir</name>
    <value>hdfs://VT-USHM1:54310/var/log/yarn/apps</value>
  </property>

</configuration>
```

Файли налаштування вузлів

Необхідно створити наступні файли:

• */usr/local/Hadoop/etc/hadoop/masters* – список головних вузлів (на них будуть запускатись служби *NameNode* та *ResourceManager*);

• */usr/local/Hadoop/etc/hadoop/slaves* – список підлеглих вузлів (на них будуть запускатись служби *DataNode* та *NodeManager*).

Додаємо у файли наступні рядки:

1. *masters*

```
VT-USHM1
```

2. *slaves*

```
VT-USHS1
VT-USHS2
```

Створення директорій необхідних для роботи *Hadoop*

Необхідно створити директорії, які були вказані у конфігураційних файлах вище.

```
1.  root@VT-USHM1:~# su hduser
2.  hduser@VT-USHM1:/root$ cd /usr/local/hadoop
3.  hduser@VT-USHM1:/usr/local/hadoop$ mkdir app
4.  hduser@VT-USHM1:/usr/local/hadoop$ mkdir app/hadoop
5.  hduser@VT-USHM1:/usr/local/hadoop$ mkdir app/hadoop/tmp
6.  hduser@VT-USHM1:/usr/local/hadoop$ mkdir app/yarn
7.  hduser@VT-USHM1:/usr/local/hadoop$ mkdir app/yarn/local
8.  hduser@VT-USHM1:/usr/local/hadoop$ mkdir app/yarn/log
9.  hduser@VT-USHM1:/usr/local/hadoop$ mkdir store/hdfs/namenode
10. hduser@VT-USHM1:/usr/local/hadoop$ mkdir store/hdfs/datanode
11. hduser@VT-USHM1:/usr/local/hadoop$ mkdir store/hdfs/ namesecondary
```

Запуск *Hadoop*

Перед першим запуском *Hadoop* необхідно відформатувати розподілену файлову систему *HDFS*. Наведені нижче кроки потрібно виконувати на головному вузлі кластеру (в даному прикладі це вузол *VT-USHM1*).

Форматування файлової системи

Даний крок виконується лише під час першого запуску кластеру Hadoop.

```
1. hduser@VT-USHM1:/usr/local/hadoop$ bin/hdfs namenode -format
```

Запуск файлової системи *Hadoop*

```
1. hduser@VT-USHM1:/usr/local/hadoop$ sbin/start-dfs.sh

   Starting namenodes on [VT-USHM1]
   VT-USHM1: starting namenode, logging to
   /usr/local/hadoop/logs/hadoop-hduser-namenode-VT-USHM1.out
   VT-USHS2: starting datanode, logging to
   /usr/local/hadoop/logs/hadoop-hduser-datanode-VT-USHS2.out
   VT-USHS1: starting datanode, logging to
   /usr/local/hadoop/logs/hadoop-hduser-datanode-VT-USHS1.out
   Starting secondary namenodes [0.0.0.0]
   0.0.0.0: starting secondarynamenode, logging to
   /usr/local/hadoop/logs/hadoop-hduser-secondarynamenode-VT-USHM1.out
```

Після першого запуску необхідно створити службові директорії.

```
1. hduser@VT-USHM1:/usr/local/hadoop$ bin/hdfs dfs –mkdir /user
2. hduser@VT-USHM1:/usr/local/hadoop$ bin/hdfs dfs –mkdir /user/hduser
3. hduser@VT-USHM1:/usr/local/hadoop$ bin/hdfs dfs –mkdir /var
4. hduser@VT-USHM1:/usr/local/hadoop$ bin/hdfs dfs –mkdir /var/log
5. hduser@VT-USHM1:/usr/local/hadoop$ bin/hdfs dfs –mkdir
   /var/log/yarn
6. hduser@VT-USHM1:/usr/local/hadoop$ bin/hdfs dfs –mkdir
   /var/log/yarn/apps
```

Запуск менеджеру ресурсів *Hadoop*

```
1. hduser@VT-USHM1:/usr/local/hadoop$ sbin/start-yarn.sh

   starting yarn daemons
   starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-
   hduser-resourcemanager-VT-USHM1.out
   VT-USHS1: starting nodemanager, logging to
   /usr/local/hadoop/logs/yarn-hduser-nodemanager-VT-USHS1.out
   VT-USHS2: starting nodemanager, logging to
   /usr/local/hadoop/logs/yarn-hduser-nodemanager-VT-USHS2.out
```

Запуск серверу історії *Hadoop*

```
hduser@VT-USHM1:/usr/local/hadoop$ sbin/mr-jobhistory-daemon.sh
start historyserver

starting historyserver, logging to /usr/local/hadoop/logs/mapred-
hduser-historyserver-VT-USHM1.out
```

Запуск задачі на кластері *Hadoop*

Дистрибутив *Hadoop* включає в себе приклади задач, які можна запустити, щоб перевірити працездатність та коректність налаштувань кластеру. В даному кластері, що був налаштований, приклади знаходяться у директорії */usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.3.jar*

В подальшому на кластері можна додатково налаштувати високорівневі засоби роботи та керування такі, як *Apache Pig*, *Apache Hive*, *Apache Oozie*, тощо.

Зупинка кластеру *Hadoop*

```
hduser@VT-USHM1:/usr/local/hadoop$ sbin/mr-jobhistory-daemon.sh
stop historyserver

stopping historyserver

hduser@VT-USHM1:/usr/local/hadoop$ sbin/stop-yarn.sh
stopping yarn daemons
stopping resourcemanager
VT-USHS1: stopping nodemanager
VT-USHS2: stopping nodemanager
no proxyserver to stop

hduser@VT-USHM1:/usr/local/hadoop$ sbin/stop-dfs.sh
Stopping namenodes on [VT-USHM1]
VT-USHM1: stopping namenode
VT-USHS2: stopping datanode
VT-USHS1: stopping datanode
Stopping secondary namenodes [0.0.0.0]
0.0.0.0: stopping secondarynamenode
```

ДОДАТОК 2
СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. Alex Holmes. Hadoop in Practice. – Manning Publications, 2012;

2. Tom White. Hadoop: The Definitive Guide: Storage and Analysis at Internet Scale. – O'Reilly Media, 2010;

3. Donald Miner, Adam Shook. MapReduce Design Patterns. – O'Reilly Media, 2012.

4. Apache Hadoop (https://hadoop.apache.org/docs/stable/)

5. Apache Pig (https://pig.apache.org/)

6. Apache Hive (https://cwiki.apache.org/confluence/display/Hive/LanguageManual)

7. Apache Oozie (https://oozie.apache.org/)

ДОДАТОК 3
ТАБЛИЦЯ ВАРІАНТІВ

Наведені нижче варіанти надаються для ознайомлення і в подальшому можуть бути змінені.

Варіант розраховується за формулою:

$$x = <\text{порядковий номер у групі}> \ mod \ 10$$

Таблиця Д3.1. Варіанти роботи

x	Варіант – обчислити
0	кількість значень “обсягу торгів” певної величини *(оберіть самостійно)* для 3-х пар за кожен місяць
1	мінімальне значення “ціни відкриття” для 2-х пар за кожен рік
2	середнє суми “обсягу торгів” 3-х пар за кожні півроку
3	суму “максимум цін” 3-х пар за певні три місяці
4	максимальне різниці між “обсягами торгів” 2-х валютних пар за певний місяць
5	кількість записів, коли значення “обсягу торгів” не перевищувало певну величину *(оберіть самостійно)*
6	знайти дати, коли “обсяг торгів” 2-х пар були однаковими за певний рік
7	суму “ціни закриття” 3-х пар за кожен рік
8	середнє значення “ціни закриття” для 3-х пар за кожні два роки
9	кількість записів, коли “ціна відкриття” та “ціна закриття” були однаковими, за кожні три місяці