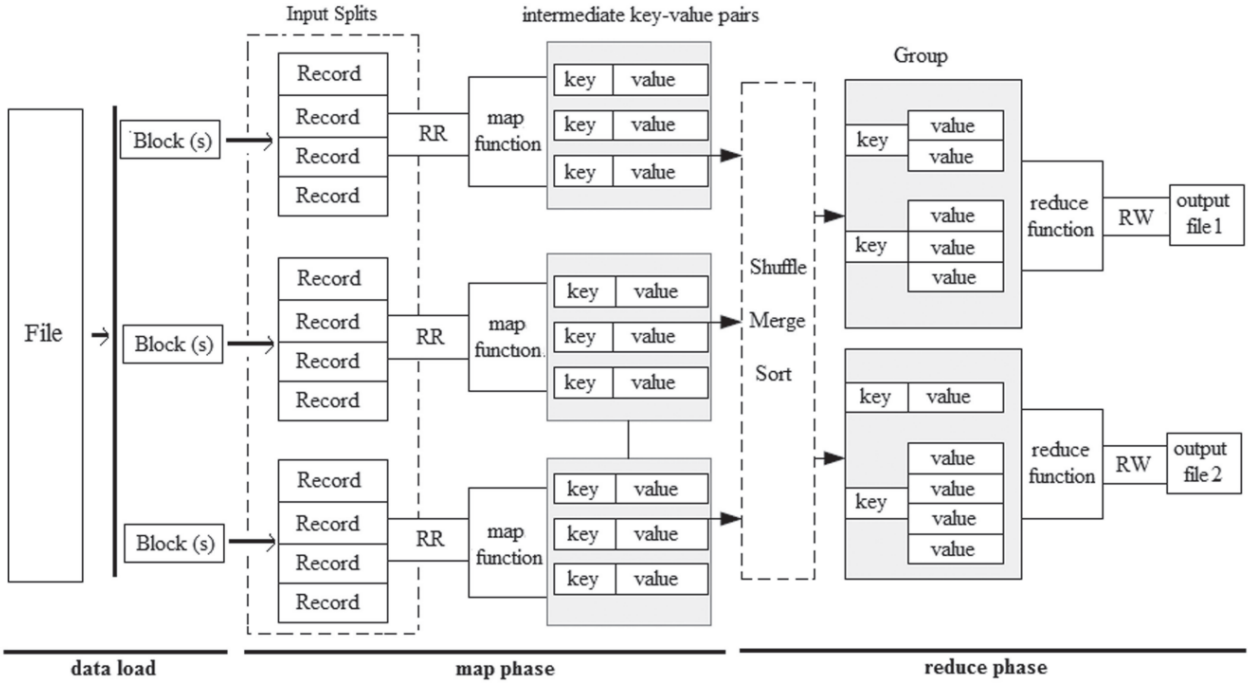
Bài tập 1
MapReduce

1. Tóm tắt MapReduce

Chúng ta đã thực thi wordcount được tạo sẵn trong bản phân phối Hadoop. Bây giờ, chúng ta sẽ tập làm quen việc viết một chương trình wordcount bằng cách sử dụng các Java IDE (NetBeans, Eclipse, …) và chạy trong một môi trường Hadoop.

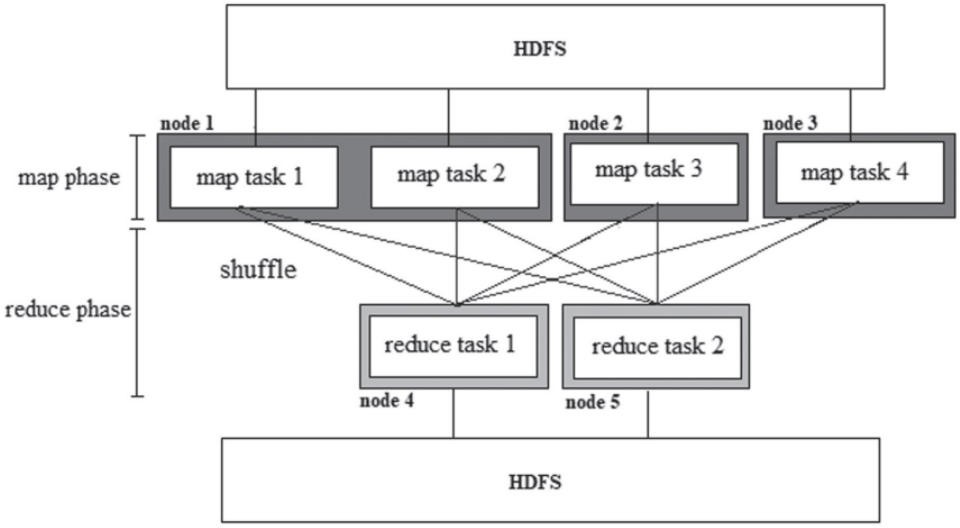
Kiến trúc tổng quan của một chương trình MapReduce như sau:



Để phát triển một công việc MapReduce cho một thuật toán, bạn cần viết các hàm map và reduce. Một số phần của thuật toán được bao gồm trong hàm map, và một số phần khác được bao gồm trong hàm reduce. Cách phân chia công việc giữa hàm map và hàm reduce trong một công việc MapReduce thường phụ thuộc vào đặc thuật toán mà bạn đang triển khai. Tuy nhiên, chúng ta sẽ làm các bài luyện tập cơ bản trong lab này.

- Hàm Map thực hiện đọc dữ liệu từ đĩa dưới dạng cặp khóa-giá trị và tạo ra một số lượng tùy ý các cặp khóa-giá trị trung gian.
- Hàm Reduce thực hiện thu thập đầu ra từ tất cả các tác vụ map, kết hợp, sắp xếp dựa trên khóa, nhóm các danh sách giá trị thuộc cùng một khóa và tạo ra kết quả cuối cùng.

Trong một công việc MapReduce phức tạp, có thể có nhiều hàm map và reduce, và chúng có thể được kết hợp và xử lý tuần tự hoặc song song để thực hiện thuật toán của bạn.



1.1. Pha Map

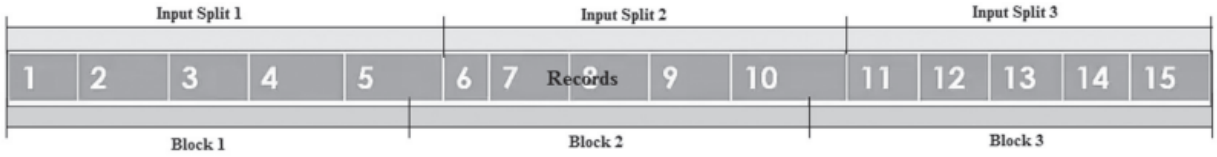
HDFS chia tập tin đầu vào thành các khối (block) có kích thước bằng nhau và lưu trữ trên các DataNode khác nhau. Các khối dữ liệu cần thiết được đưa vào bộ nhớ để cung cấp cho tác vụ map thực thi và bắt đầu giai đoạn map.

Tuần tự của các hàm được thực thi cùng với hàm map gồm:

Input Split → Record Reader → Mapper → Partitioner → combiner

1.1.1. Bước 1: Input Split

IS (InputSplit) là một tập các block được xử lý bởi *một* tác vụ map. Số lượng khối cho một IS tùy vào cấu hình được thiết lập. Mỗi dữ liệu (data) được liên kết với một khóa (key). Một cặp key-value được gọi là một bản ghi (**record**).



Định dạng đầu vào của tập tin (FileInputFormat) có trách nhiệm tạo ra IS và chuẩn bị các record từ nội dung của các block. Mỗi định dạng đầu vào xác định kiểu dữ liệu cho khóa (key) và giá trị (value).

File Input Format	Key	Default value
TextInputFormat	byte offset	all texts until new line (/n)
KeyValueTextInputFormat	text until first tab (customizable)	all texts after the first tab until new line (/n)
NlineInputFormat	byte offset	texts of "N" lines
TableInputFormat (HBase)	row key	entire row
WholeFileInputFormat	Null	entire file
MultiFileInputFormat	per path basis	per path basis
SequenceFileInputFormat	User-defined	User-defined

Do HDFS không biết record nào được chứa trong block hoặc việc một record có nằm vắt giữa hai block không, nên IS có trách nhiệm xác định vị trí bắt đầu của record đầu tiên trong một block và vị trí kết thúc của record cuối trong block. Nếu record cuối trong block không hoàn chỉnh, IS sẽ chứa thông tin về vị trí của block tiếp theo và byte offset của dữ liệu cần để hoàn thành record.

1.1.2. Bước 2: Record Reader (RR)

IS đã chuẩn bị các cặp key:value (record) từ nội dung của một block dựa trên định dạng đầu vào của tập tin và cung cấp một cách nhìn dựa trên byte. Tuy nhiên, tác vụ map không biết cách đọc các record từ IS. Do đó, RR (RecordReader) được sử dụng để đọc các cặp key:value từ IS và chuyển đổi cách nhìn dựa trên byte thành các kiểu dữ liệu Hadoop, sau đó cung cấp cho hàm map.

Ví dụ, đối với định dạng TextInputFormat, IS hình thành byte offset như là khóa và toàn bộ dòng (cho đến khi gặp dòng mới) là giá trị. Sau đó, RR sẽ chuyển đổi khóa (byte offset) thành LongWritable và giá trị (toàn bộ dòng) thành Text.

Java primitive type	**MR data type**
Boolean	BooleanWritable
Byte	ByteWritable
Byte[]	BytesWritable
Short	ShortWritable
Integer	IntWritable VIntWritable
Float	FloatWritable
Double	DourbleWritable
Java objective type	**MR data type**
String	Text
Object	ObjectWritable
	NullWritable
Java collection	**MR data type**
Array	ArrayWritable ArrayPrimitiveWritable TwoDArrayWritable
Map	MapWritable
SortedMap	SortedMapWritable
Enum	EnumSetWritable

Mỗi InputFormat cung cấp triển khai RR riêng của nó để đọc các cặp khóa/giá trị.

Khi viết chương trình bằng Java, chúng ta cần chuyển đổi kiểu dữ liệu giữa kiểu trong Java và kiểu dữ liệu MR để thực hiện nhập xuất và tính toán. Hàm constructor có dạng ***Writable() giúp chuyển đổi kiểu dữ liệu trong Java thành các kiểu dữ liệu MR.

Ví dụ:

```
int a=20;
IntWritable num=new IntWritable(a);

String str="Hello, World";
Text t=new Text(str);
```

Để chuyển đổi kiểu dữ liệu MR thành kiểu dữ liệu Java, ta sử dụng hàm get() cho số và toString() cho chuỗi.

Ví dụ:

```
IntWritable num=new IntWritable (10);
int a=num.get();

Text t=new Text("Hello, World");
String str=t.toString();
```

1.1.3. Bước 3: Mapper

Mapper là một hàm map do người dùng xác định, thường được sử dụng để tiền xử lý các bản ghi. Do đó, đầu ra của tác vụ map được gọi là các cặp khóa-giá trị trung gian (intermediate key-value pairs), không phải là kết quả của công việc MapReduce.

RR cung cấp một record cho hàm map. Kiểu dữ liệu cho khóa và giá trị đầu vào của map dựa trên bản ghi của RR.

Hàm map cho phép người dùng viết logic của riêng họ để quyết định xử lý như thế nào với các bản ghi. Bạn có thể phân tích cú pháp (parse) và trích xuất chỉ các trường liên quan, hoặc lọc các bản ghi không mong muốn/hỏng, hoặc biến đổi các bản ghi đầu vào. Đầu ra của tác vụ map được lưu trữ trong một bộ đệm trong bộ nhớ (in-memory buffer).

```
class MapTask extends Mapper{          // mapper/map task
        public void  map (){           // map function
                // user defined logic
        }
}
```

1.1.4. Bước 4: Partitioner

Partitioner (bộ phân vùng) chia đầu ra của một tác vụ map thành nhiều phân vùng. Mỗi phân vùng là một phần của đầu ra map và được gửi đến một reducer cụ thể. Số lượng phân vùng bằng số lượng tác vụ reducer. Mục tiêu của việc phân vùng là đưa các khóa giống nhau từ các tác vụ map khác nhau vào một reducer duy nhất.

1.1.5. Bước 5: Combiner

Nếu tác vụ map tạo ra đầu ra lớn, điều này dẫn đến lưu lượng mạng nội bộ lớn hơn để chuyển chúng đến tác vụ reduce. Khi đó, việc truyền dữ liệu từ tác vụ map đến tác vụ reduce trên mạng cục bộ sẽ gây ra áp lực lớn và tăng giao thông mạng. Bên cạnh đó, khi kích thước đầu ra của tác vụ map vượt qua ngưỡng bộ đệm trong bộ nhớ, nó sẽ được ghi vào đĩa. Điều này gây ra việc chuyển dữ liệu qua I/O đĩa nhiều hơn. Ảnh hưởng đến thời gian của hệ thống.

Để giải quyết những vấn đề này, hàm **combiner** được sử dụng để giảm kích thước đầu ra của tác vụ map tại cục bộ sau khi phân vùng đã được chuẩn bị. Do đó, combiner còn được gọi là mini/local reducer. Nếu không có cách nào để giảm kích thước đầu ra của map, việc sử dụng combiner sẽ không có ý nghĩa. Do đó, combiner hoàn toàn phụ thuộc vào ứng dụng cụ thể.

Hàm combiner có thể được định nghĩa bởi người dùng hoặc có thể giống như tác vụ reduce. Chữ ký của tác vụ combiner do người dùng xác định hoàn toàn giống với tác vụ reduce. Tuy nhiên, không phải ứng dụng nào cũng có thể sử dụng tác vụ reduce làm combiner. Các ứng dụng thỏa mãn tính chất hoán vị và tính chất kết hợp có thể sử dụng tác vụ reduce như tác vụ combiner. Ví dụ: sum, max, min.

Tính chất hoán vị: a + b = b + a (đổi chỗ các toán hạng sẽ cho cùng kết quả)

Tính chất kết hợp: a + (b + c) = (a + b) + c (nhóm các toán hạng khác nhau cho cùng kết quả)

Ví dụ: max(max(a,b), max(c,d,e)) = = max (a, b, c, d, e)

Tuy nhiên, hàm trung bình mean thì không được vì mean(mean(a,b), mean(c,d,e)) != mean(a, b, c, d, e).

1.2. Pha Reduce

Tuần tự của các hàm được thực thi cùng với hàm reducer gồm:

shuffle → merge → sort → group → reducer → file output format → record writer

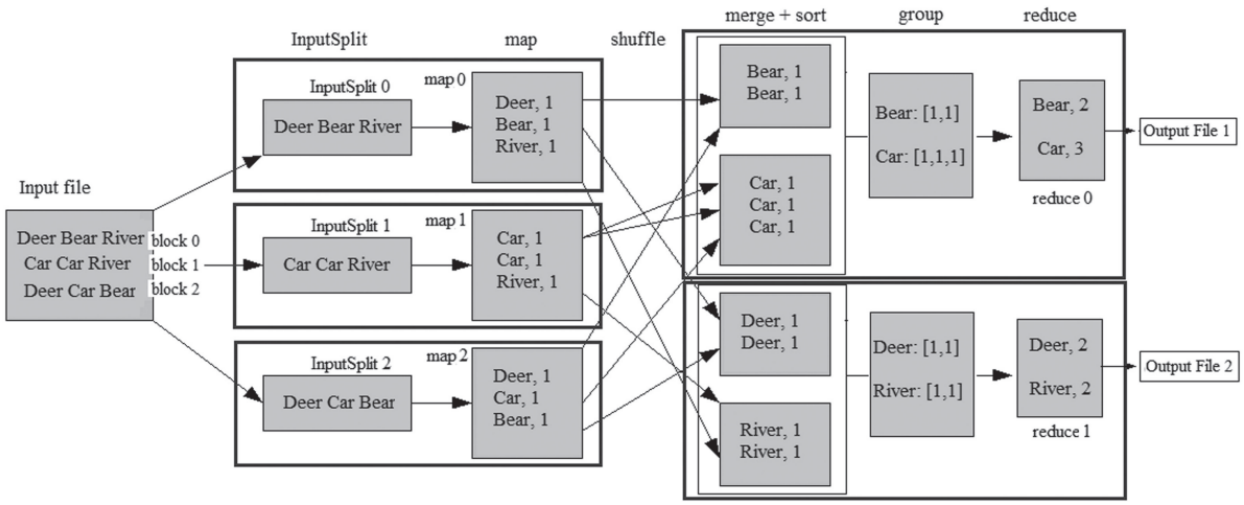
Các phân vùng (partitions) được di chuyển từ các nút map đến các nút reduce (shuffle), sau đó được hợp nhất (merge) thành một và được sắp xếp dựa trên khóa (sort). Các giá trị thuộc cùng một khóa được nhóm lại (group) để loại bỏ sự trùng lặp của khóa. Cuối cùng, đầu ra (key: list(values)) được đưa vào tác vụ reduce.

Hàm reduce xử lý một danh sách các giá trị cho mỗi khóa và tạo ra một hoặc nhiều bản ghi đầu ra. Tại đây, các phép tổng hợp (aggregation) và nối kết (join) được thực hiện.

Sau đó, hàm reduce đưa khóa và giá trị đầu ra cho RW (RecordWriter), tiếp theo RW ghi vào HDFS với sự phân tách bằng tab dựa trên TextOutputFormat. Quy ước đặt tên của các tập đầu ra là part-r-00000, trong đó r là reduce task, 5 ký số mô tả chỉ số tập tin đầu ra.

2. Ví dụ tiến trình MapReduce

2.1. Bài word count



2.2. Bài tính tổng tiền

Tính tổng số tiền mà mỗi khách hàng đã trả dựa trên bảng thông tin bán hàng có dạng sau:

Name	Price	Item	Date
Sham	7000	mobile	12-2-2014 ….
Ravi	1000	earphone	10-2-2015 ….
Raja	10000	laptop	2-1-2015 …..
Anil	2000	charger	3-2-2014 ….
Bala	1000	books	5-3-2011 ….
Ravi	500	pizza	23-11-2015 ….
Sham	100	soap	12-12-2015 ….
Anil	2000	charger	3-2-2014 ….
Ravi	1000	snacks	5-3-2011 ….

- Pha map:
  - Input: phụ thuộc vào định dạng lưu trữ

- Output: <key là tên khách hàng, value là số tiền>

- Pha reduce:
  - Input: <key là tên khách hàng, list(value) là danh sách số tiền của khách hàng>
  - Ouput: <key là tên khách hàng, tổng số tiền>

3. Viết chương trình Java

Chúng ta sử dụng Eclipse để lập trình Java (bạn có thể chọn Java IDE khác) và tạo ra tập tin jar để submit lên Hadoop. Đảm bảo phiên bản Java giống với môi trường Hadoop (Java 8.x.x).

3.1. Cài đặt Eclipse trong Ubuntu

Lên trang chủ Eclipse để tải file *.tar.gz (x86_64). Giả sử file được lưu về thư mục Download, ta thực hiện giải nén:

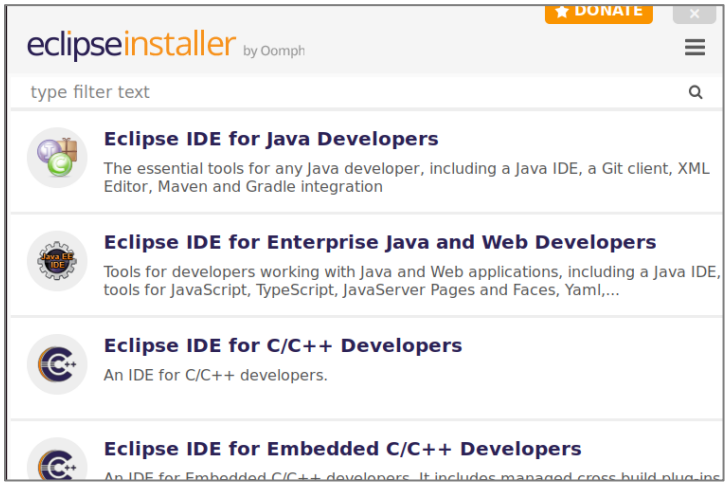
```
$ tar xfz ~/Downloads/eclipse-inst-jre-linux64.tar.gz
```

(Lưu ý, cập nhật tên file nếu khác)

Chạy tập tin cài đặt:

```
$ ~/Downloads/eclipse-installer/eclipse-inst
```

Chọn Eclips IDE for Java Developers và thực hiện theo các hướng dẫn cài đặt.



3.2. Tạo dự án WordCount

Quá trình được thực hiện như sau trên Eclipse:

Bước 1: Tạo dự án

- File >> New >> Java Project
- Đặt tên dự án
- Chọn môi trường JRE là JavaSE-1.8
- Finish

Bước 2: Tạo package để chứa các thư viện Hadoop

- Click phải trên Project Name >> New >> Package
- Đặt tên cho package (ví dụ: com.myproject.wc)
- Finish

Bước 3: Thêm các thư viện Hadoop

- Click phải trên Project Name >> Build Path >> Configure Build Path
- Chọn thẻ Libraries
- Nhấn nút "Add External JARs"
- Thêm tất cả các tập tin jar trong thư mục hadoop-3.3.6 >> share >> hadoop >> client
- Thêm tất cả các tập tin jar trong thư mục hadoop-3.3.6 >> share >> hadoop >> common
- Thêm tất cả các tập tin jar trong thư mục hadoop-3.3.6 >> share >> hadoop >> common >> lib
- Thêm tất cả các tập tin jar trong thư mục hadoop-3.3.6 >> share >> hadoop >> yarn
- Thêm tất cả các tập tin jar trong thư mục hadoop-3.3.6 >> share >> hadoop >> mapreduce
- Thêm tất cả các tập tin jar trong thư mục hadoop-3.3.6 >> share >> hadoop >> hdfs
- Nhấn Apply and Close sau khi thêm xong

Bước 4: Tạo class mới cho Mapper

- Click phải trên **Package Name** >> New >> Class
- Đặt tên cho lớp (Ví dụ: WordCountMapper)
- Finish
- Sao chép đoạn code sau vào lớp vừa tạo:

```
package com.myproject.wc; //cap nhat lai theo ten package của bạn

import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.io.LongWritable;

public class WordCountMapper extends Mapper<LongWritable, Text, Text,
IntWritable> {
      private Text wordToken = new Text();

      public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
            StringTokenizer tokens = new
StringTokenizer(value.toString()); // Dividing String into tokens
            while (tokens.hasMoreTokens()) {
                  wordToken.set(tokens.nextToken());
                  context.write(wordToken, new IntWritable(1));
            }
      }
}
```

- Để format lại code cho đẹp có thể dùng tổ hợp phím: Ctrl + Shift + F
- Save tập tin lại.

Bước 5: Tạo class mới cho Reducer

- Click phải trên **Package Name** >> New >> Class
- Đặt tên cho lớp (Ví dụ: WordCountReducer)
- Finish
- Sao chép đoạn code sau vào lớp vừa tạo:

```
package com.myproject.wc; //cap nhat lai theo ten package của bạn

import java.io.IOException;
```

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class WordCountReducer extends Reducer<Text, IntWritable, Text,
IntWritable> {
      private IntWritable count = new IntWritable();

      public void reduce(Text key, Iterable<IntWritable> values,
Context context)
                  throws IOException, InterruptedException {
// gurukul [1 1 1 1 1 1….]
            int valueSum = 0;
            for (IntWritable val : values) {
                  valueSum += val.get();
            }
            count.set(valueSum);
            context.write(key, count);
      }
}
```

- Save tập tin lại.

Bước 6: Tạo driver class (main)

- Click phải trên **Package Name** >> New >> Class
- Đặt tên cho lớp (Ví dụ: WordCount)
- Finish
- Sao chép đoạn code sau vào lớp vừa tạo:

```
package com.myproject.wc; //cap nhat lai theo ten package của bạn

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class WordCount {
      public static void main(String[] args) throws Exception {
            Configuration conf = new Configuration();
            String[] pathArgs = new GenericOptionsParser(conf,
args).getRemainingArgs();
            if (pathArgs.length < 2) {
                  System.err.println("MR Project Usage: wordcount
<input-path> […] <output-path>");
                  System.exit(2);
            }
            Job wcJob = Job.getInstance(conf, "MapReduce WordCount");
            wcJob.setJarByClass(WordCount.class);
            wcJob.setMapperClass(WordCountMapper.class);
            wcJob.setCombinerClass(WordCountReducer.class);
            wcJob.setReducerClass(WordCountReducer.class);
            wcJob.setOutputKeyClass(Text.class);
            wcJob.setOutputValueClass(IntWritable.class);
            for (int i = 0; i < pathArgs.length - 1; ++i) {
                  FileInputFormat.addInputPath(wcJob, new
Path(pathArgs[i]));
```

```
            }
            FileOutputFormat.setOutputPath(wcJob, new
Path(pathArgs[pathArgs.length - 1]));
            System.exit(wcJob.waitForCompletion(true) ? 0 : 1);
      }
}
```

- Save tập tin lại.

Bước 7: Tạo tập tin Jar

- Click phải trên tên project >> Export
- Chọn JAR File
- Next
- Chọn project, nơi lưu và tên tập tin jar (Browse…ví dụ: WordCount.jar)
- Next
- Chọn Main class bằng cách nhấn nút Browse và chọn lớp chính (driver) là WordCount (giống tên tập tin ở bước 6)
- Finish

Bước 8: Thử nghiệm

Chạy thử tập tin jar vừa tạo trên hệ thống Hadoop đã thiết lập.

```
yarn jar WordCount.jar /data/input.txt /output
```

Kiểm tra kết quả chạy ra.

Lưu ý: nếu có lỗi thì kiểm tra lại Hadoop đã chạy bình thường chưa như kiểm tra các tiến trình bằng jps, chạy thử code tích hợp sẵn trong Hadoop.

3.3. Bài tập

Tìm kiếm các java code Hadoop khác để thực hiện trên Eclipse.

4. Viết chương trình bằng Python

4.1. Cài đặt PIP, MRJOB, NANO

Việc viết bằng ngôn ngữ Java (ngôn ngữ chính được hỗ trợ bởi MapReduce) thường phức tạp đối với một số lập trình viên. Việc streaming qua ngôn ngữ Python giúp cho việc lập trình trở nên dễ dàng hơn. Có nhiều công cụ để hỗ trợ streaming như Hadoop Streaming, MRJob, Dumbo, Hadoopy. Trong khuôn khổ bài lab này, ta sử dụng thư viện MRJob được phát triển bởi Yelp (Amazon Web Services).

Cài đặt thư viện streaming python để có thể sử dụng Python để lập trình Map Reduce.

- Cài đặt python:

```
$ sudo apt install wget build-essential libncursesw5-dev libssl-
dev libsqlite3-dev tk-dev libgdbm-dev libc6-dev libbz2-dev
libffi-dev zlib1g-dev

$ sudo add-apt-repository ppa:deadsnakes/ppa

$ sudo apt install python3.11
```

- Cài đặt pip:

```
$ yum install hadoop-pip
```

- Cập nhật pip:

```
$ python -m pip install --upgrade pip
```

- Cài đặt MRJob:

```
$ pip install mrjob
```

(Lưu ý tùy phiên bản MRJob có thể phù hợp với phiên bản python có trên hệ thống)

- Khởi động lại hệ thống

4.2. Viết chương trình WordCount

- Tạo một tập tin python (ví dụ: word_count.py)
- Chép mã nguồn sau:

```
from mrjob.job import MRJob

class MRWordFrequencyCount(MRJob):

    def mapper(self, _, line):
        yield "chars", len(line)
        yield "words", len(line.split())
        yield "lines", 1

    def reducer(self, key, values):
        yield key, sum(values)


if __name__ == '__main__':
    MRWordFrequencyCount.run()
```

4.3. Giải thích chương trình MRJOB

Một chương trình được xác định bằng một lớp kế thừa từ MRJob. Lớp này chứa các phương thức xác định các bước (step) của công việc của bạn.

Một "step" bao gồm mapper, combiner và reducer. Tất cả những bước này đều là tùy chọn, mặc dù bạn phải có ít nhất một bước. Vì vậy, bạn có thể có một bước chỉ là mapper, hoặc chỉ có combiner và reducer. Khi bạn chỉ có một bước, bạn chỉ cần viết các phương thức được gọi là mapper(), combiner() và reducer().

Phương thức mapper() nhận một khóa và một giá trị như các đối số (trong trường hợp này, khóa bị bỏ qua (_) và một dòng văn bản đầu vào là giá trị) và tạo ra nhiều cặp khóa-giá trị tuỳ ý. Phương thức reduce() nhận một khóa và một bộ lặp giá trị và cũng tạo ra nhiều cặp khóa-giá trị tùy ý. (Trong trường hợp này, nó tổng hợp các giá trị cho mỗi khóa, mà thể hiện số ký tự, từ và dòng trong đầu vào.)

Câu lệnh bắt buộc trong một chương trình là hai dòng ở cuối. Những dòng này chuyển quyền điều khiển qua các đối số dòng lệnh và thực thi cho mrjob. Nếu thiếu chúng, công việc sẽ không hoạt động.

4.4. Chạy MRJOB trên local

Trước khi chạy code thực sự trên HDFS của Hadoop, ta có thể test trên hệ thống local với kích thước dữ liệu nhỏ trước để đảm bảo không lỗi:

Cú pháp:

```
python word_count.py input.txt
```

Nếu muốn xuất kết quả ra file thay vì ra màn hình ta dùng dấu ">". Ví dụ:

```
python word_count.py input.txt > output.txt
```

4.5. Cách chạy MRJOB trên HDFS

Cú pháp:

```
python word_count.py -r hadoop hdfs:///data/input.txt
```

4.6. Ví dụ với nhiều step

Để xác định nhiều bước, ta cần ghi đè phương thức steps() để trả về một danh sách MRSteps. Dưới đây là một ví dụ về một chương trình tìm từ phổ biến nhất:

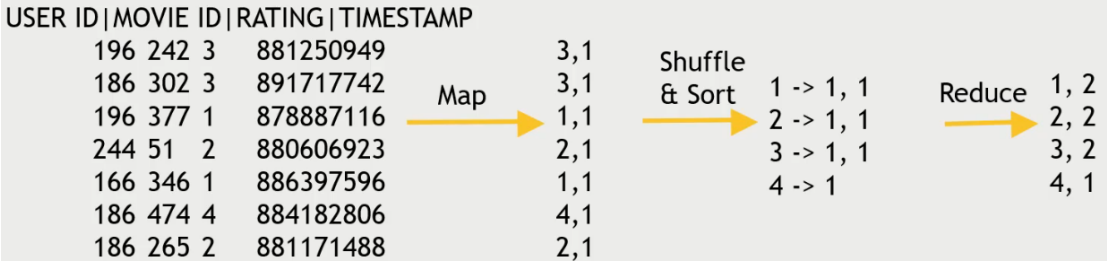
```
from mrjob.job import MRJob
from mrjob.step import MRStep
import re

WORD_RE = re.compile(r"[\w']+")

class MRMostUsedWord(MRJob):

    def steps(self):
        return [
            MRStep(mapper=self.mapper_get_words,
                   combiner=self.combiner_count_words,
                   reducer=self.reducer_count_words),
            MRStep(reducer=self.reducer_find_max_word)
        ]

    def mapper_get_words(self, _, line):
        # yield each word in the line
        for word in WORD_RE.findall(line):
            yield (word.lower(), 1)

    def combiner_count_words(self, word, counts):
        # optimization: sum the words we've seen so far
        yield (word, sum(counts))

    def reducer_count_words(self, word, counts):
        # send all (num_occurrences, word) pairs to the same reducer.
        # num_occurrences is so we can easily use Python's max() function.
        yield None, (sum(counts), word)

    # discard the key; it is just None
    def reducer_find_max_word(self, _, word_count_pairs):
        # each item of word_count_pairs is (count, word),
        # so yielding one results in key=counts, value=word
        yield max(word_count_pairs)

if __name__ == '__main__':
    MRMostUsedWord.run()
```

4.7. Bài tập

4.7.1. Bài 1

Xét tập dữ liệu ratings (u.data) trong MovieLens, thực hiện chương trình thống kê số lượng người bình chọn ở mỗi mức.

Quá trình Map-Reduce như sau:



Ta lần lượt định nghĩa code cho mapper và reducer. Việc định nghĩa được thực hiện thông qua lớp kế thừa của MRJob. Trong lớp này ta định nghĩa các hàm để thực hiện cho mapper và hàm thực thi cho reducer. Cuối cùng là khai báo các bước thực hiện.

```
from mrjob.job import MRJob
from mrjob.step import MRStep

class RatingsBreakdown(MRJob):
    def steps(self):
        return [
            MRStep(mapper=self.mapper_get_ratings,
                   reducer=self.reducer_count_ratings)
        ]

    def mapper_get_ratings(self, _, line):
        (userID, movieID, rating, timestamp) = line.split('\t')
        yield rating, 1

    def reducer_count_ratings(self, key, values):
        yield key, sum(values)

if __name__ == '__main__':
    RatingsBreakdown.run()
```

4.7.2. Bài 2

Sắp xếp các phim theo số lượt bình chọn.

- Bước map sẽ đọc từng dòng và mỗi bộ phim được bình chọn sẽ được đếm là 1.
- Bước shuffle và sort sẽ gom tác bộ phim có cùng mã để đưa cho reducer
- Bước reduce sẽ tính tổng các bình chọn. Để có thể sort theo số lượt bình chọn, ta lấy tổng số bình chọn làm khóa cho bước shuffle và sort tiếp theo.
- Bước reduce thứ 2 chỉ cần đảo ngược thông tin phim và số lượt bình chọn để có được danh sách sắp xếp.

```
from mrjob.job import MRJob
from mrjob.step import MRStep

class RatingsBreakdown(MRJob):
    def steps(self):
        return [
            MRStep(mapper=self.mapper_get_ratings,
                   reducer=self.reducer_count_ratings),
            MRStep(reducer=self.reducer_sorted_output)
        ]

    def mapper_get_ratings(self, _, line):
        (userID, movieID, rating, timestamp) = line.split('\t')
        yield movieID, 1

    def reducer_count_ratings(self, key, values):
        yield str(sum(values)).zfill(5), key

    def reducer_sorted_output(self, count, movies):
        for movie in movies:
            yield movie, count


if __name__ == '__main__':
    RatingsBreakdown.run()
```

4.7.3. Bài 3

Thống kê mỗi từ xuất hiện trong tài liệu cho trước:

a. Trường hợp phân biệt hoa thường

b. Trường hợp không phân biệt hoa thường

Tự tạo dữ liệu là một tập tin văn bản ngắn để test trên local.

Khi thành công, thực hiện tải dữ liệu sau lên HDFS và thực hiện đếm. Dữ liệu gồm 3 cuốn sách:

http://www.gutenberg.org/ebooks/20417

http://www.gutenberg.org/ebooks/5000

http://www.gutenberg.org/ebooks/4300

(Tải Plain Text)

4.7.4. Bài 4

Tìm từ xuất hiện nhiều nhất trong tài liệu (không phân biệt hoa thường). Dữ liệu có thể lấy từ bài tập trên.

Gợi ý: hàm max() trong python để trả về giá trị lớn nhất trong tập hợp.