

XỬ LÝ DỮ LIỆU LỚN VỚI MAPREDUCE

Giảng viên: TS. Nguyễn Văn Quyết



- Giới thiệu chung về MapReduce
- Nguyên lý hoạt động của MapReduce
- Chức năng của các thành phần trong MapReduce
- Thực hành lập trình với MapReduce
- Hỏi & đáp

Tại sao cần MapReduce ?

- Xử lý lượng lớn dữ liệu
 - Google xử lý khoảng 24PB dữ liệu/ngày vào 2009
- Một máy chủ không thể đảm nhiệm việc xử lý toàn bộ dữ liệu lớn
 - Cần hệ thống phân tán để lưu trữ và xử lý song song
- Lập trình song song?
 - Đa luồng – khó
 - Làm sao để thuận tiện truyền dữ liệu giữa các nodes?
 - Làm thế nào để mở rộng thêm nhiều máy?
 - Làm thế nào để việc xử lý thuật toán khi máy lỗi?

MapReduce là gì?

- MapReduce là một mô hình lập trình được thiết kế cho việc xử lý dữ liệu lớn một cách song song bằng việc chia nhỏ công việc (work) thành tập các nhiệm vụ (tasks) độc lập.
- Hình thành từ 2004
 - MapReduce: Simplified Data Processing on Large Clusters
 - **Jeffrey Dean** and **Sanjay Ghemawat**

Vai trò của MapReduce

- Hỗ trợ tự động song song hóa và phân tán
- Hỗ trợ lập lịch vào/ra dữ liệu
 - Cân bằng tải
 - Tối ưu hóa việc truyền dữ liệu trên mạng
- Hỗ trợ việc mở rộng hệ thống, tăng hiệu suất
 - **Scale out**, **not up**

Các vấn đề cơ bản có thể giải quyết dùng MapReduce

- Đọc để xử lý một lượng lớn dữ liệu
- **Map**: trích lọc thông tin từ một lượng lớn dữ liệu
- Trộn và sắp xếp dữ liệu lớn
- **Reduce**: tổng hợp, lọc kết quả, chuyển đổi dữ liệu
- Ghi lại các kết quả lên hệ thống phân tán

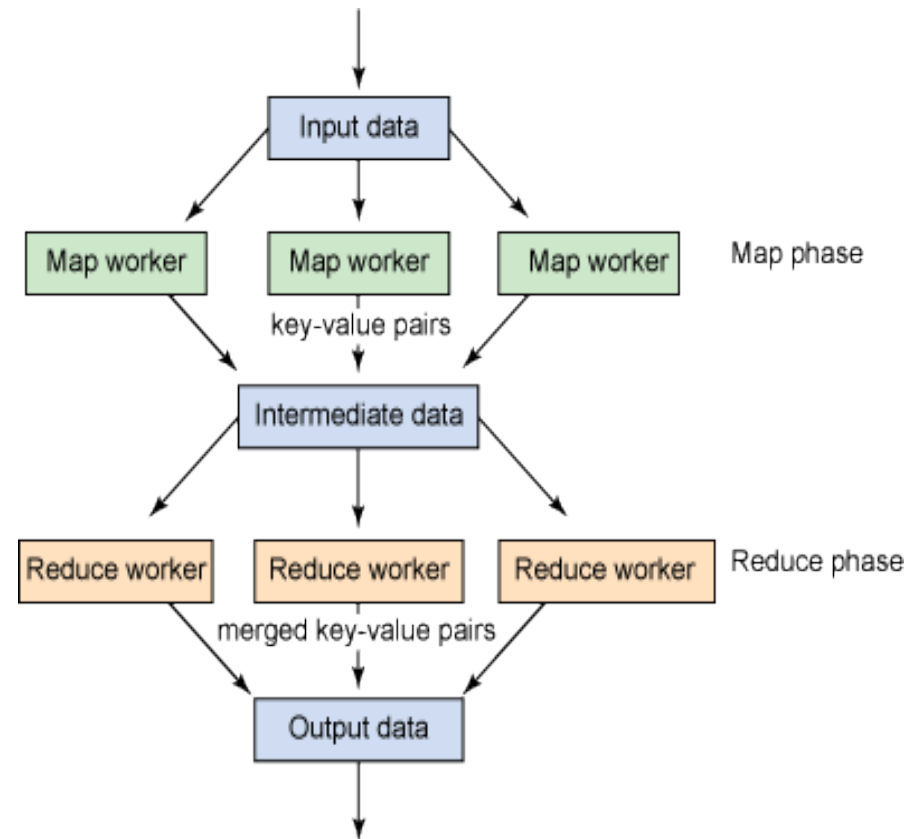
Các thành phần chính của MapReduce

● Các thành phần người dùng

- Mapper
- Reducer
- Combiner (optional)
- Partitioner (optional) – Shuffle

● Các thành phần hệ thống

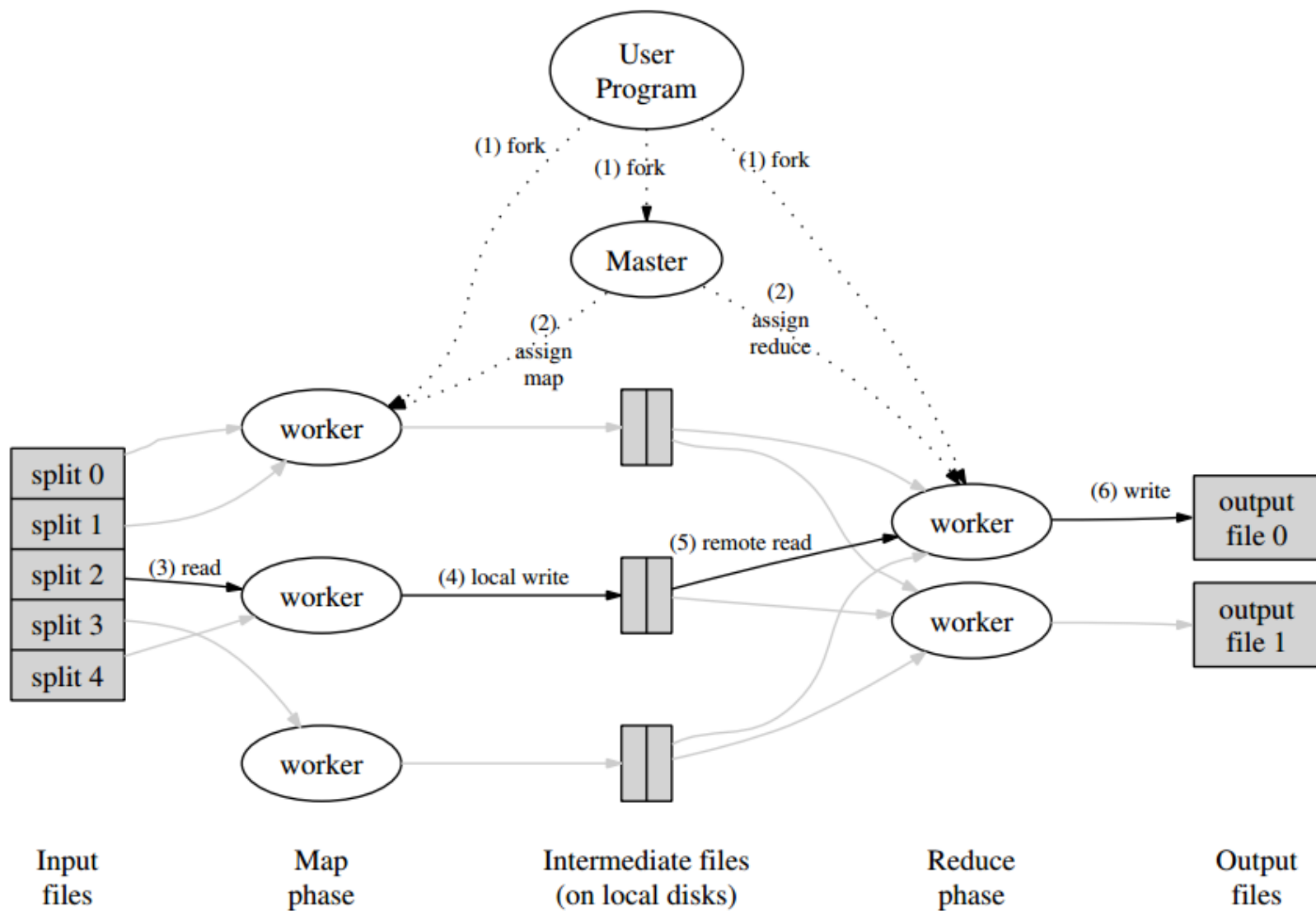
- Master
- InputSplitter*
- OutputCommitter*



Nguyên lý hoạt động chung của MapReduce

- MapReduce hoạt động dựa vào nguyên tắc chính là **“Chia để trị”**
 - Phân chia dữ liệu thành nhiều phần nhỏ trước khi thực hiện xử lý
 - Xử lý các vấn đề nhỏ theo phương thức song song và độc lập trên các máy tính phân tán.
 - Tiến hành tổng hợp những kết quả thu được để đưa ra kết quả sau cùng.

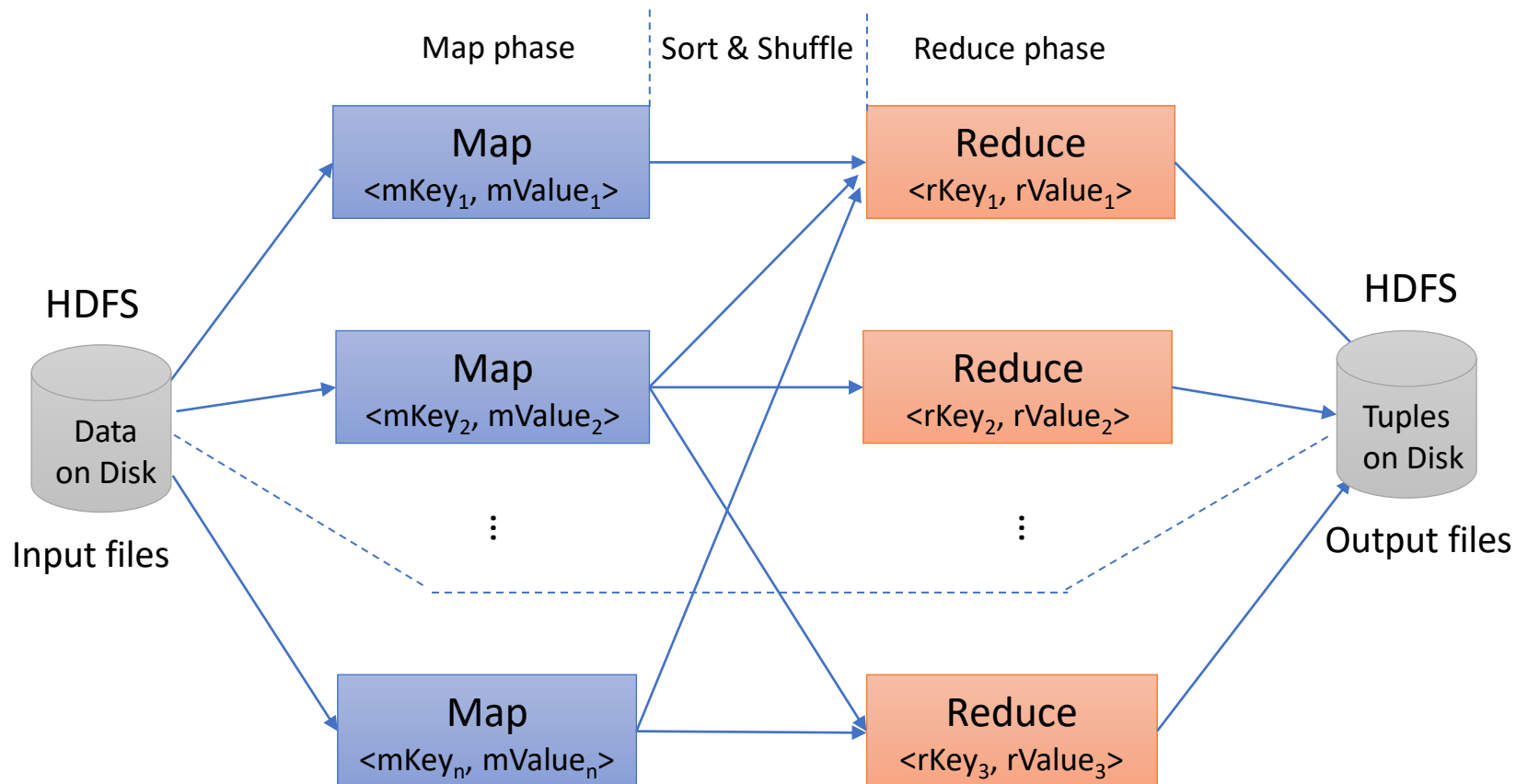
MapReduce Data Flow



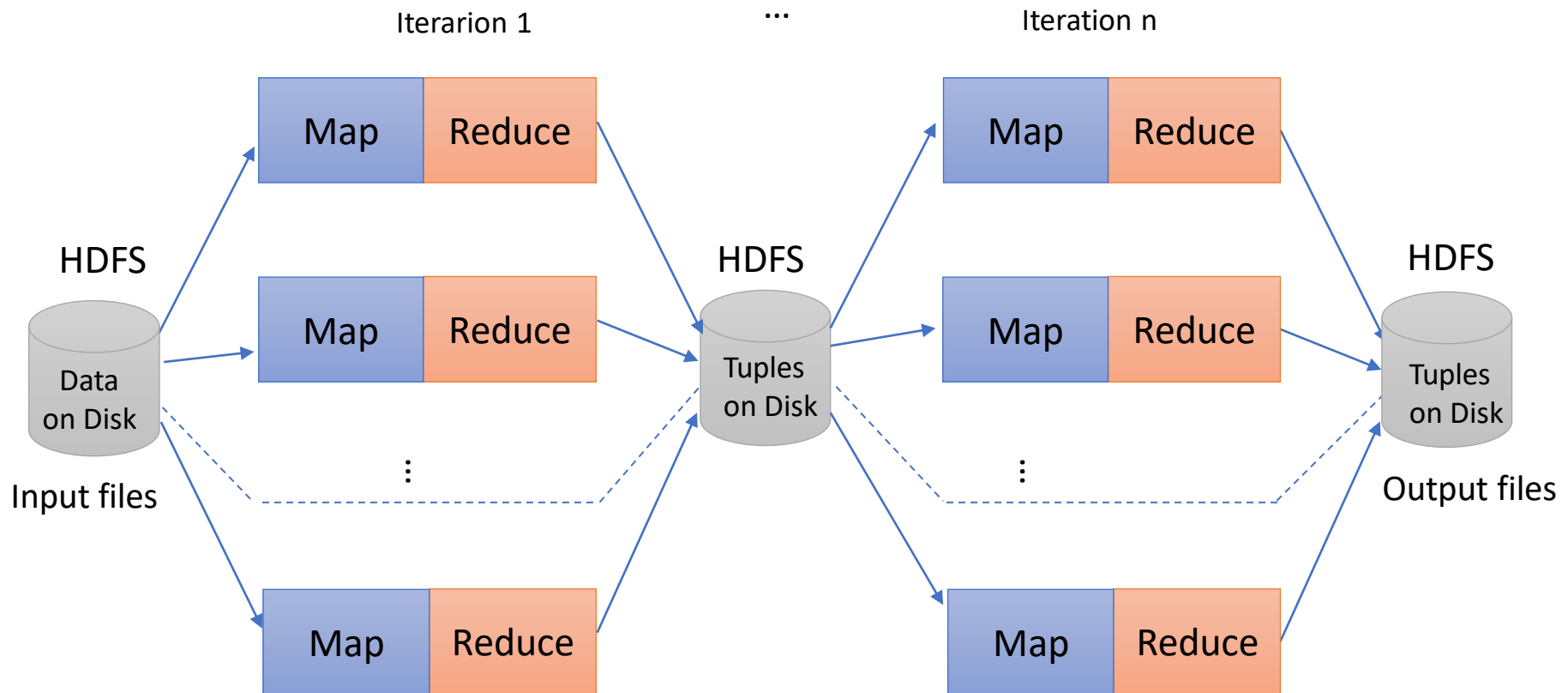
Các bước xử lý của MapReduce

- **Bước 1:** Đọc dữ liệu đầu vào với **Input Splitter**
- **Bước 2:** Xử lý dữ liệu đầu vào với **Mapper**
- **Bước 3:** Sắp xếp và trộn các kết quả thu được từ các hàm Map() trên các máy tính phân tán với **Combiner & Partitioner**.
- **Bước 4:** Tổng hợp và xử lý các kết quả trung gian thu được với **Reducer**
- **Bước 5:** Ghi ra kết quả cuối cùng với **Output Committer**

Sơ đồ hoạt động của MapReduce



Sơ đồ hoạt động của MapReduce



Multiple MapReduce Jobs

Input Splitter

- Chia nhỏ tệp dữ liệu thành nhiều mảnh (chunks)
- Các chunks là đầu vào cho các Mapper
- Chia dữ liệu theo ràng buộc
 - Kích thước mặc định là 128MB
- Thông thường sử dụng các kiểu chia đã được cài đặt sẵn, có thể tùy biến

Input Splitter

● InputFormat

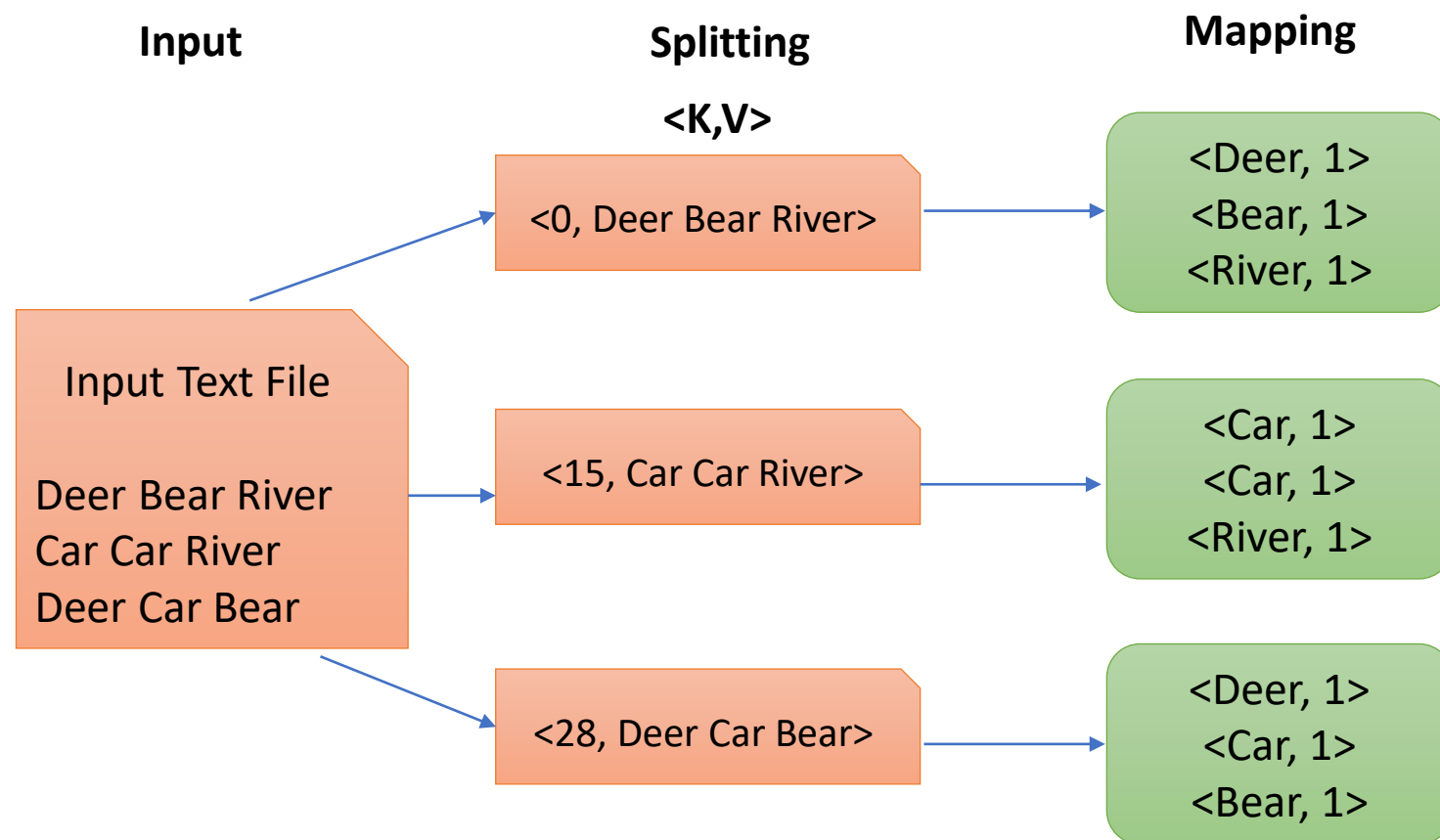
- **FileInputFormat**: cơ sở của các InputFileFormat khác
- **TextFileFormat**
- KeyValueTextInputFormat
- **SequenceFileInputFormat**
- SequenceFileAsTextInputFormat
- SequenceFileAsBinaryInputFormat
- NLineInputFormat
- DBInputFormat

Mapper (1/3)

- Đọc dữ liệu vào dưới dạng cặp key-value $\langle K, V \rangle$
- Ghi dữ liệu ra dưới dạng key-value $\langle K', V' \rangle$
- **$\langle \text{Key}, \text{Value} \rangle$** được sinh ra như thế nào trong Hadoop?
 - **InputSplit**: biểu diễn dữ liệu mức logic (*kích thước và vị trí*)
 - **RecordReader**: lấy thông tin từ InputSplit, đọc dữ liệu và chuyển nó về dạng $\langle \text{key}, \text{value} \rangle$ cho các Mapper

Mapper (2/3)

● Ví dụ: Bài toán Word Count



Mapper (3/3)

● Bao nhiêu Map Tasks trong 1 chương trình Hadoop MapReduce?

● **Số Map Tasks = Số block of input files**

- Mỗi file có kích thước < block size vẫn được coi là 1 block

● **Số Map Tasks = Tổng kích thước file dữ liệu / input split size**

- Ví dụ: 1 tệp 1GB = 1024MB, input split size = 128MB

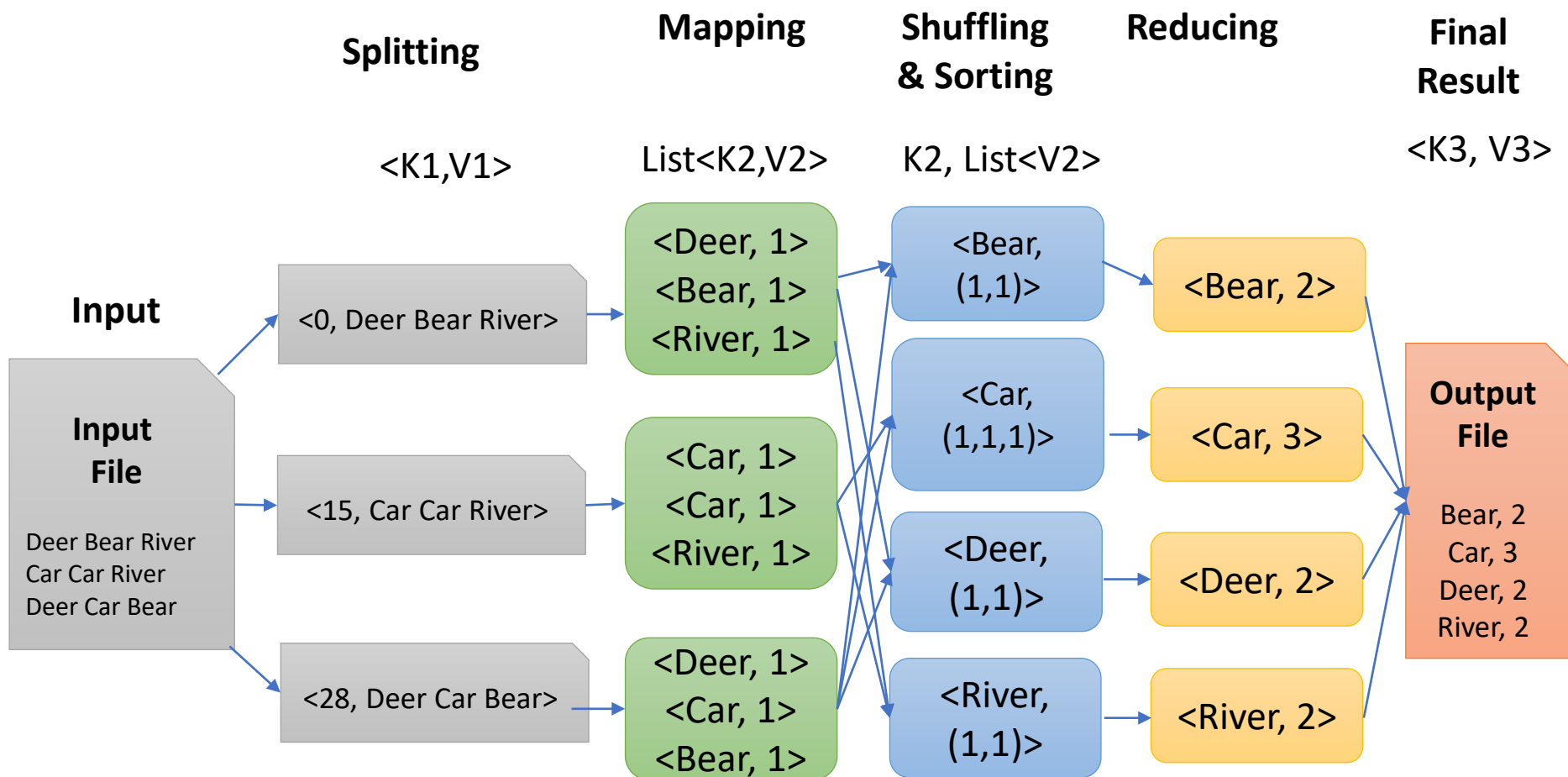
→ #MapTask = $1024/128 = 8$ map tasks

Reducer (1/3)

- Nhận Input là output của các Mapper
 - Tất cả các input có **cùng Key** phải vào cùng 1 Reducer
- Input thường đã được trộn và sắp xếp
 - Shuffle & Sorting phase
- Output là kết quả mong đợi, lưu trên HDFS

Reducer (2/3)

● Ví dụ: Bài toán Word Count



Reducer (3/3)

● Số Reducers cần thiết

- Người dùng có thể thiết lập số Reducer cho Job
 - Sử dụng lệnh: ***Job.setNumreduceTasks(int)***
 - Thông thường, số Reducers = **(0.95 hoặc 1.75) x #số nodes x #max số containers/node**
 - Với 0.95: tất cả các reducer sẽ chạy khi các map hoàn thành
 - Với 1.75: vòng đầu của các reducer sẽ kết thúc bởi các node tính toán nhanh, và tiếp tục các vòng tiếp theo

Combiner (1/2)

- Gom các **output của Mapper có cùng key** với nhau
- Chạy trước Reducer
- Tùy chọn (có hoặc không)
- Thuận lợi
 - Giảm lượng dữ liệu truyền qua mạng
 - Giảm lượng dữ liệu cần xử lý của Reducer
 - Tăng hiệu suất tổng thể của chương trình
- Bất lợi
 - Thao tác nhiều trên ổ đĩa khi làm việc với các local filesystem

Combiner (2/2)

- So sánh **có** và **không** sử dụng Combiner

- Ví dụ: Bài toán Word Count

Mapper Output

<Deer, 1>
<Bear, 1>
<River, 1>

<Car, 1>
<Car, 1>
<River, 1>

<Deer, 1>
<Car, 1>
<Bear, 1>

**9 cặp
key-value
gửi đến Reducer**

Combiner Output

<Bear,
(1,1)>

<Car,
(1,1,1)>

<Deer,
(1,1)>

<River,
(1,1)>

**4 cặp
key-value
gửi đến Reducer**

Partitioner (Shuffler)

- Quyết định **Key nào** của Map Output thực hiện bởi **Reducer nào**
- Sử dụng Hash Function
 - Mặc định: **Key.hashCode() % numOfReducers**
- Người dùng có thể ghi đè cách thức hoạt động của Partitioner
 - Dùng các hàm phân bố nhằm cân bằng tải
 - Một vài Value cần gửi vào cùng 1 reducer
 - Ví dụ: để tính toán mối tương quan trong giữa các cặp từ nên gửi 2 từ đó vào cùng 1 reducer.
- **Partitioner thực hiện sau Combiner, trước Reducer**

Output Committer (1/2)

- Nhận kết quả của Reducer và ghi kết quả ra HDFS với RecordWriter
- Kết quả được ghi theo định dạng của OutputFormat
 - Các Job khác có thể đọc được
- Kiểu Split có thể Customize
 - Tương tự Input Splitter

Output Committer (2/2)

● **OutputFormats**

- **FileOutputFormat**: cơ sở của các OutputFormat khác
- **TextOutputFormat**
- **SequenceFileOutputFormat**
- **SequenceAsBinaryOutputFormat**
- **MultipleOutput**
- **LazyOutputFormat**
- **DBOutputFormat**

Master

● Lập lịch và quản lý các Job

- Lập lịch tính toán sao cho các máy xử lý dữ liệu gần nơi lưu trữ → Giảm băng thông

● Nếu 1 task lỗi: → Kill → Re-launch với cùng Input

- Số lần mặc định contact với 1 task khi phát hiện crash = 4, có thể thay đổi
- Master restart các node failed, các node khác không ảnh hưởng

Chuẩn bị môi trường – phần mềm

● Cài đặt Eclipse (*có thể trên máy master*) (1/3)

- **Bước 1:** Tìm và Download Eclipse phiên bản 2019-12 R, 348MB:

https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2019-12/R/eclipse-jee-2019-12-R-linux-gtk-x86_64.tar.gz&mirror_id=105

- **Bước 2:** Chuyển (Move To) tệp tin vào thư mục /usr/local

```
$ sudo mv /home/hduser/Downloads/eclipse-jee-2019-12-R-linux-gtk-x86_64.tar.gz /usr/local/
```

Hoặc bạn có thể copy trên giao diện

- **Bước 3:** Giải nén tệp

```
$ cd /usr/local
```

```
$ tar xvf eclipse-jee-2019-12-R-linux-gtk-x86_64.tar.gz
```

Chuẩn bị môi trường – phần mềm

● Cài đặt Eclipse (*có thể trên máy master*) – (2/3)

- **Bước 4:** Tạo Desktop Icon để chạy chương trình:

\$ **sudo gedit** ~/.local/share/applications/eclipse.desktop

Copy nội dung sau vào file:

```
[Desktop Entry]
Name=Eclipse Eclipse
Type=Application
Exec=/usr/local/eclipse/eclipse
Terminal=false
Icon=/usr/local/eclipse/icon.xpm
Comment=Eclipse Integrated
Development Environment
NoDisplay=false
Categories=Development;IDE;
Name[en]=Eclipse
```

Chạy lệnh sau để make it executable

\$ **sudo chmod +x** ~/.local/share/applications/eclipse.desktop

Chuẩn bị môi trường – phần mềm

● Cài đặt Eclipse (*có thể trên máy master*) – (3/3)

- **Bước 5:** chạy file thực thi của Eclipse từ đường dẫn
`/usr/local/eclipse/`

Giao diện khởi động tương tự sau



Chuẩn bị môi trường – phần mềm

● Cài đặt Hadoop-Eclipse-Plugin (1/2)

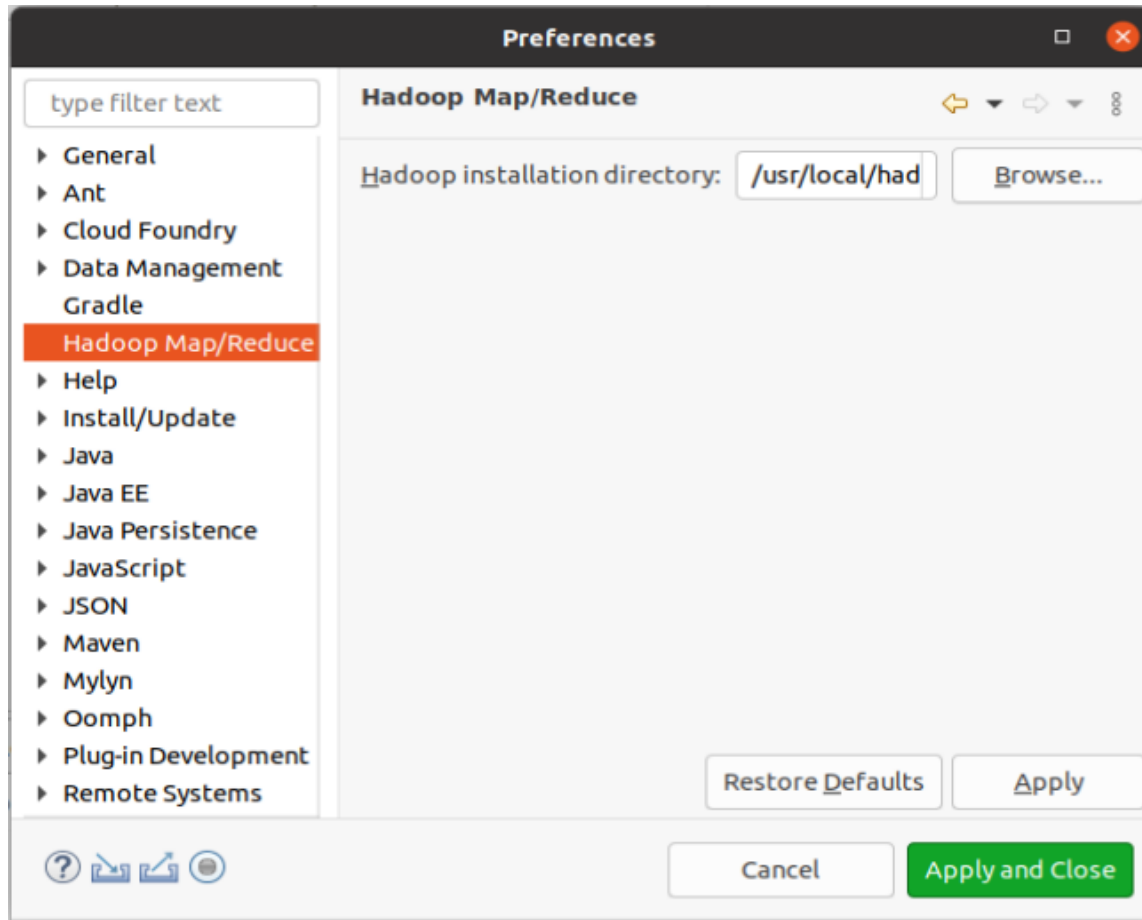
- **Bước 1:** Download **hadoop-eclipse-plugin-3.3.0.jar** từ link sau:

<https://drive.google.com/file/d/1Tfhz1k1imgp7FBOxEEjwxwOdn-8KYW/view?usp=sharing>

- **Bước 2:** Copy tệp tin **hadoop-eclipse-plugin-3.3.0-jar** vào thư mục **/usr/local/eclipse/plugins**
- **Bước 3:** Khởi động lại Eclipse để tải **hadoop-eclipse-plugin**
- **Bước 4:** Cấu hình đường dẫn tới thư mục cài đặt Hadoop trên Eclipse
(*hình bên*)

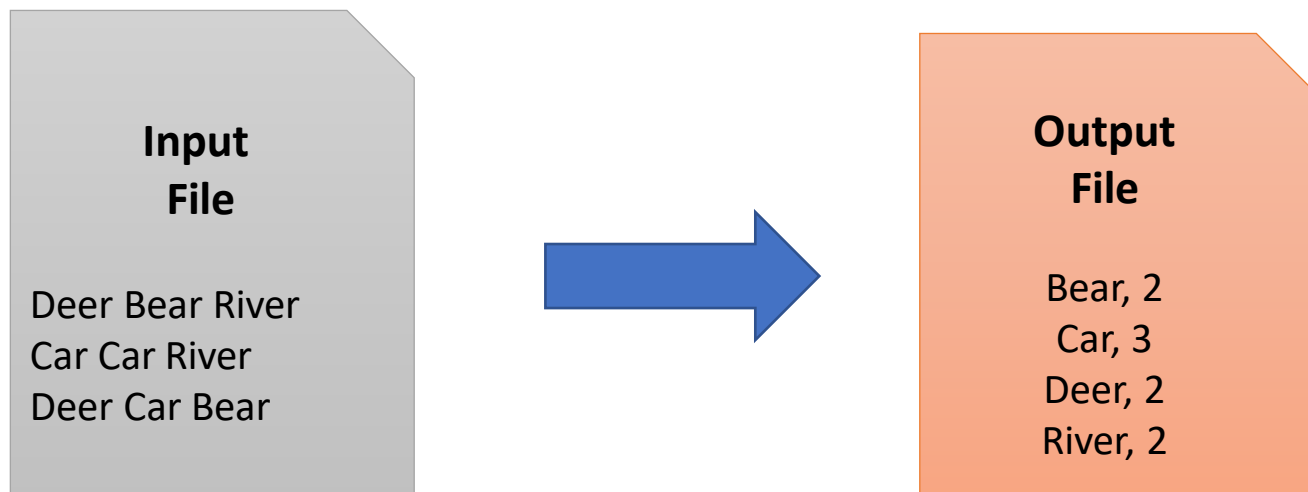
Chuẩn bị môi trường – phần mềm

● Cài đặt Hadoop-Eclipse-Plugin (2/2)



Bài tập thực hành 1

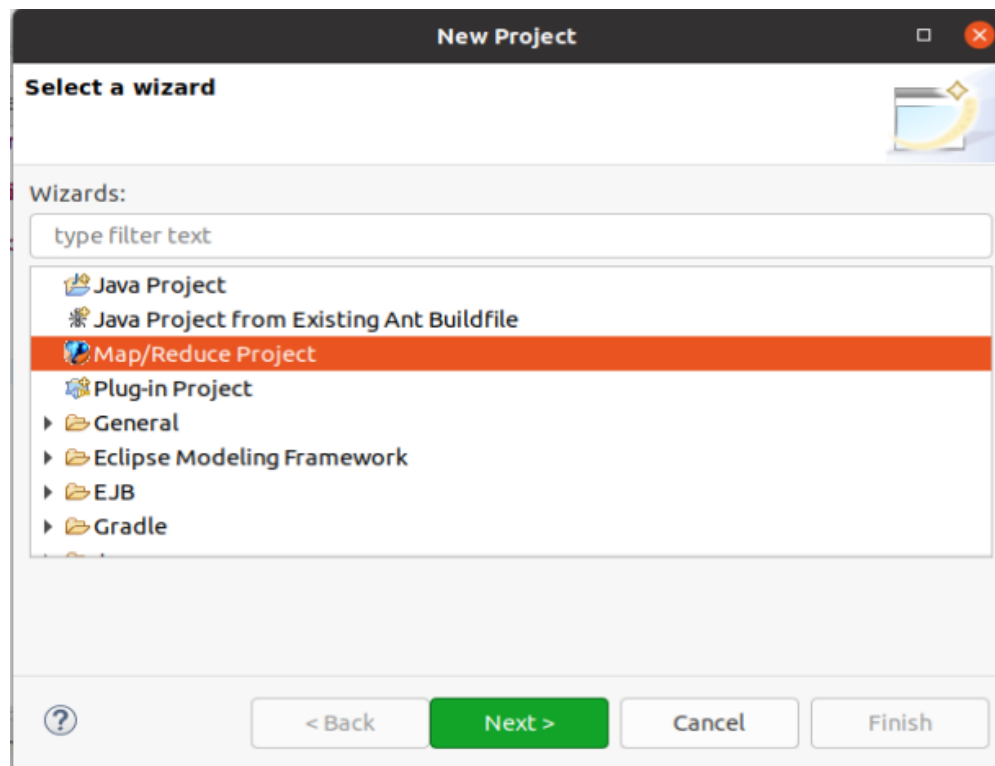
- Viết chương trình MapReduce cho bài toán đếm từ (Word Count)
 - **Input:** Tập các tệp tin văn bản
 - **Output:** Số lần xuất hiện của mỗi từ trong tập văn bản đó



Bài tập thực hành 1

● Bước 1: Tạo dự án MapReduce

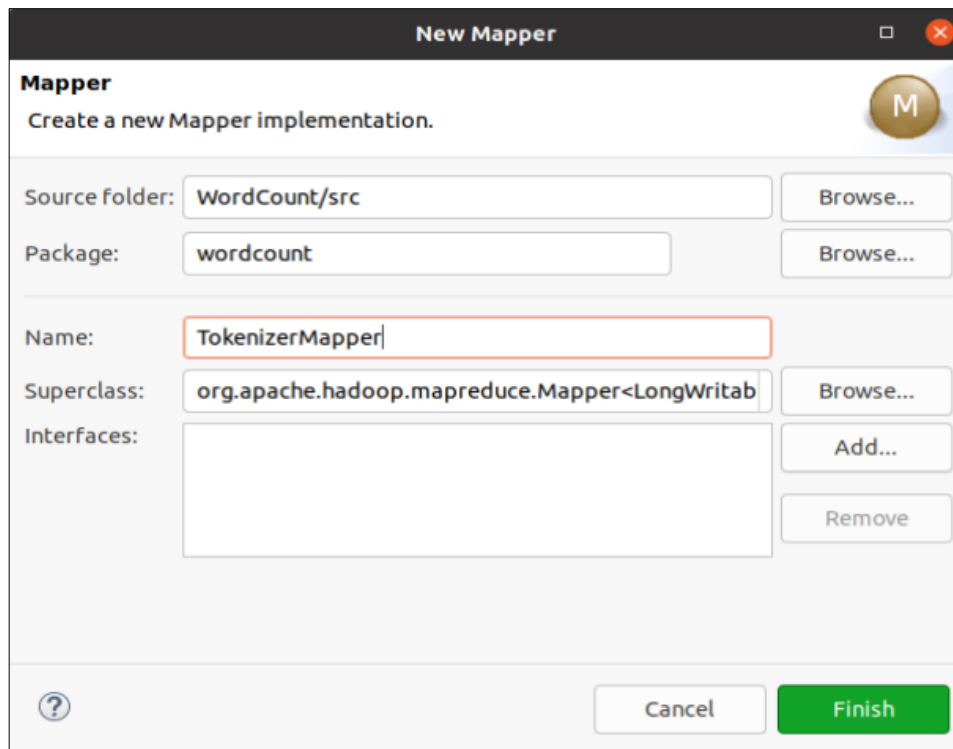
- File → New → Project → Map/Reduce Project → Next
- Đặt tên project: **WordCount** → Nhấn **Finish**



Bài tập thực hành 1

● Bước 2: Tạo hàm Map (1/2)

- File → New → Mapper
- Đặt tên Package: **wordcount**
- Đặt tên tệp: **TokenizerMapper**
- Nhấn **Finish**



New Mapper

Mapper
Create a new Mapper implementation.

Source folder: WordCount/src Browse...

Package: wordcount Browse...

Name: TokenizerMapper

Superclass: org.apache.hadoop.mapreduce.Mapper<LongWritable> Browse...

Interfaces: Add... Remove

? Cancel Finish

Bài tập thực hành 1

● Bước 2: Tạo hàm Map (1/2)

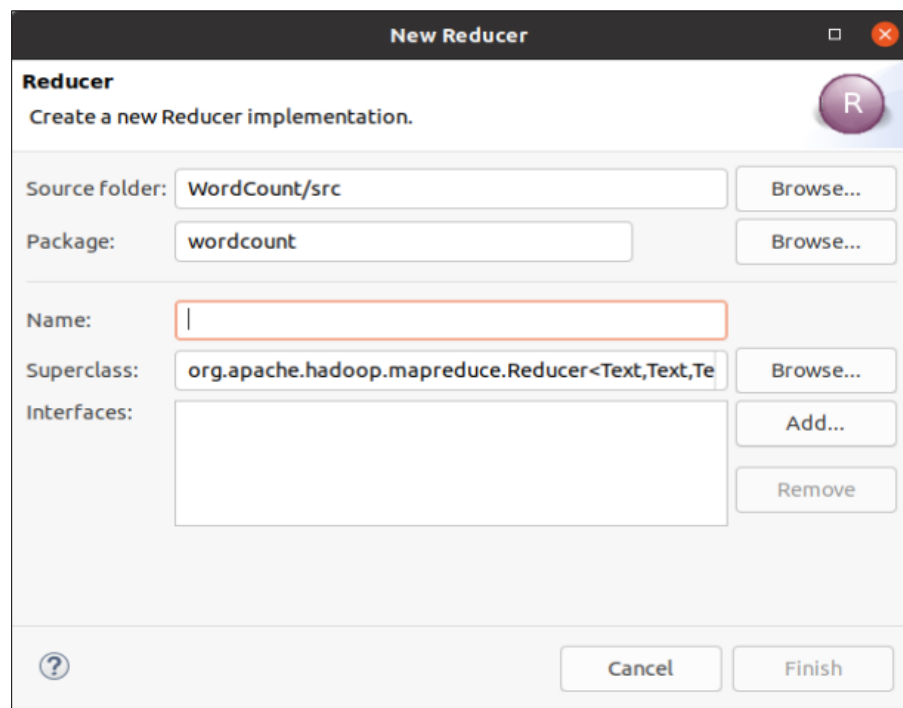
- Viết code cho hàm Map như sau:

```
TokenizerMapper.java IntSumReducer.java WordCountDriver.java
1 package wordcount;
2
3 import java.io.IOException;
4 import java.util.StringTokenizer;
5
6 import org.apache.hadoop.io.IntWritable;
7 import org.apache.hadoop.io.LongWritable;
8 import org.apache.hadoop.io.Text;
9 import org.apache.hadoop.mapreduce.Mapper;
10
11 public class TokenizerMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
12
13     private final static IntWritable one = new IntWritable(1);
14     private Text word = new Text();
15
16     public void map(LongWritable key, Text value, Context context)
17         throws IOException, InterruptedException {
18         StringTokenizer itr = new StringTokenizer(value.toString());
19         while (itr.hasMoreTokens()) {
20             word.set(itr.nextToken());
21             context.write(word, one);
22         }
23     }
24 }
```

Bài tập thực hành 1

● **Bước 3: Tạo hàm Reduce (1/2)**

- **File → New → Reducer**
- Đặt tên Package: **wordcount**
- Điền tên tệp: **IntSumReducer**
- Nhấn **Finish**



Bài tập thực hành 1

● Bước 3: Tạo hàm Reduce (2/2)

- Viết code cho hàm Reduce như sau:

```
1 package wordcount;
2
3 import java.io.IOException;
4
5 import org.apache.hadoop.io.IntWritable;
6 import org.apache.hadoop.io.Text;
7 import org.apache.hadoop.mapreduce.Reducer;
8
9 public class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
10
11     private IntWritable result = new IntWritable();
12
13     public void reduce(Text key, Iterable<IntWritable> values, Context context)
14         throws IOException, InterruptedException {
15         int sum = 0;
16         for (IntWritable val : values) {
17             sum += val.get();
18         }
19         result.set(sum);
20         context.write(key, result);
21     }
22 }
```

Bài tập thực hành 1

● **Bước 4:** Tạo hàm chương trình chính (1/2)

- **File → New → MapReduce Driver**

- Đặt tên Package: **wordcount**

- Điền tên tệp: **WordCountDriver**

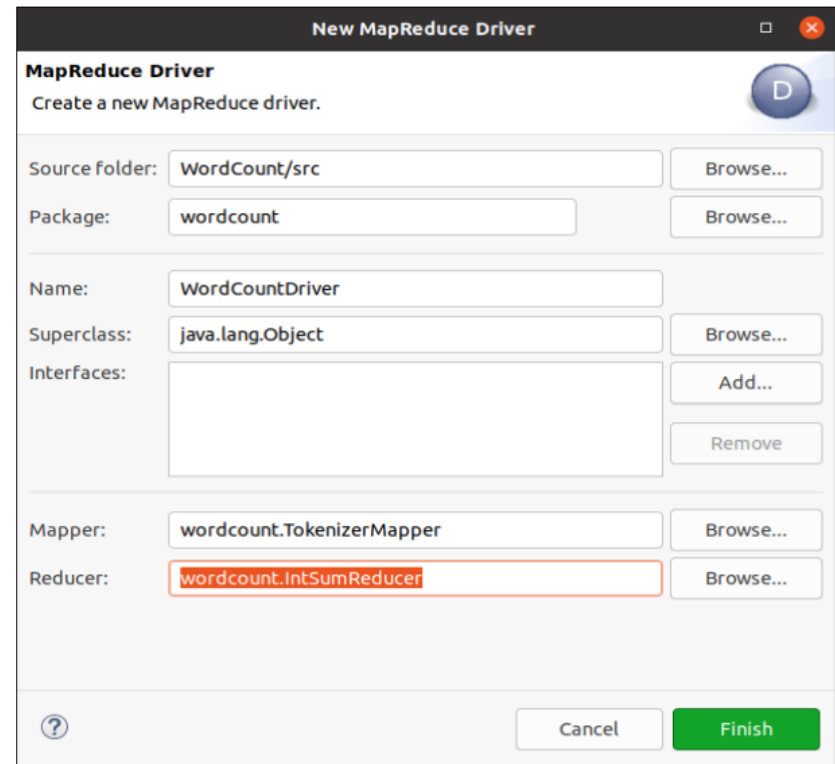
- Điền tên Mapper:

 - **wordcount.TokenizerMapper**

- Điền tên Reducer:

 - **wordcount.IntSumReducer**

- Nhấn **Finish**



New MapReduce Driver

Create a new MapReduce driver.

Source folder: WordCount/src Browse...

Package: wordcount Browse...

Name: WordCountDriver

Superclass: java.lang.Object Browse...

Interfaces: Add... Remove

Mapper: wordcount.TokenizerMapper Browse...


Reducer: wordcount.IntSumReducer Browse...

Cancel Finish

Bài tập thực hành 1

● Bước 4: Tạo hàm chương trình chính (2/2)

- Viết code cho hàm Main (Driver) như sau:



```
1 package wordcount;
2
3 import org.apache.hadoop.conf.Configuration;
4
5
6
7
8
9
10
11 public class WordCountDriver {
12
13     public static void main(String[] args) throws Exception {
14         Configuration conf = new Configuration();
15         Job job = Job.getInstance(conf, "WordCountApp");
16         job.setJarByClass(wordcount.WordCountDriver.class);
17         job.setMapperClass(wordcount.TokenizerMapper.class);
18         job.setReducerClass(wordcount.IntSumReducer.class);
19
20         // TODO: specify output types
21         job.setOutputKeyClass(Text.class);
22         job.setOutputValueClass(IntWritable.class);
23
24         // TODO: specify input and output DIRECTORIES (not files)
25         FileInputFormat.setInputPaths(job, new Path(args[0]));
26         FileOutputFormat.setOutputPath(job, new Path(args[1]));
27
28         if (!job.waitForCompletion(true))
29             return;
30     }
31 }
```

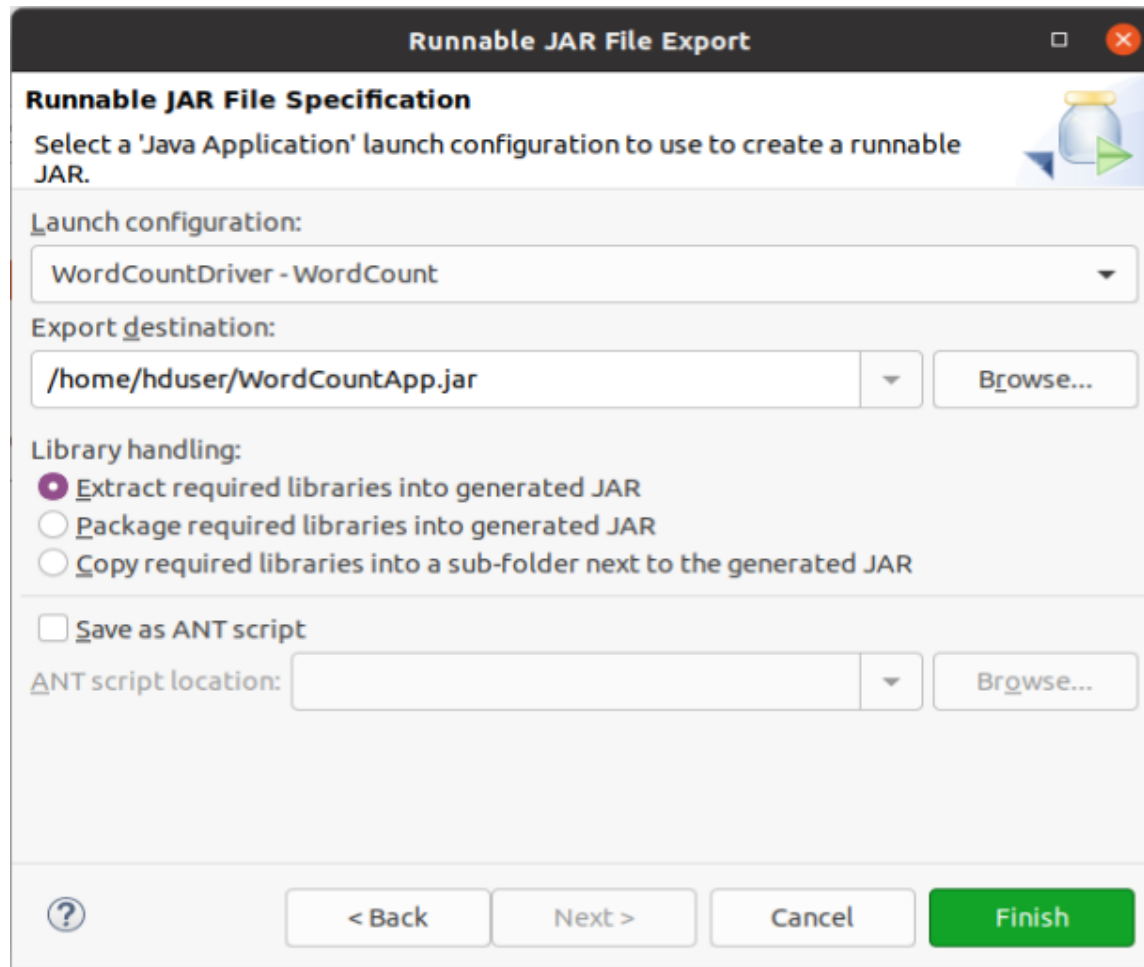
Bài tập thực hành 1

● **Bước 5:** Đóng gói chương trình → Tạo file *.jar (1/2)

- Chọn vào Project cần đóng gói
- File → Export → Java → Runnable JAR file
- Giao diện Runnable JAR file hiển thị
 - Launch Configuration: **Chọn tên tệp chương trình chính**
 - Export Destination: **Chọn nơi chứa Jar file tên máy tính**

Bài tập thực hành 1

● Bước 5: Đóng gói chương trình → Tạo file *.jar (2/2)



Bài tập thực hành 1

● **Bước 6:** Chạy chương trình trên Hadoop Cluster

- Khởi động Hadoop System

```
$ start-all.sh
```

- Chạy chương trình:

```
$ cd /usr/local/hadoop/bin
```

```
$ hadoop jar /home/hduser/WordCountApp.jar /input /output
```

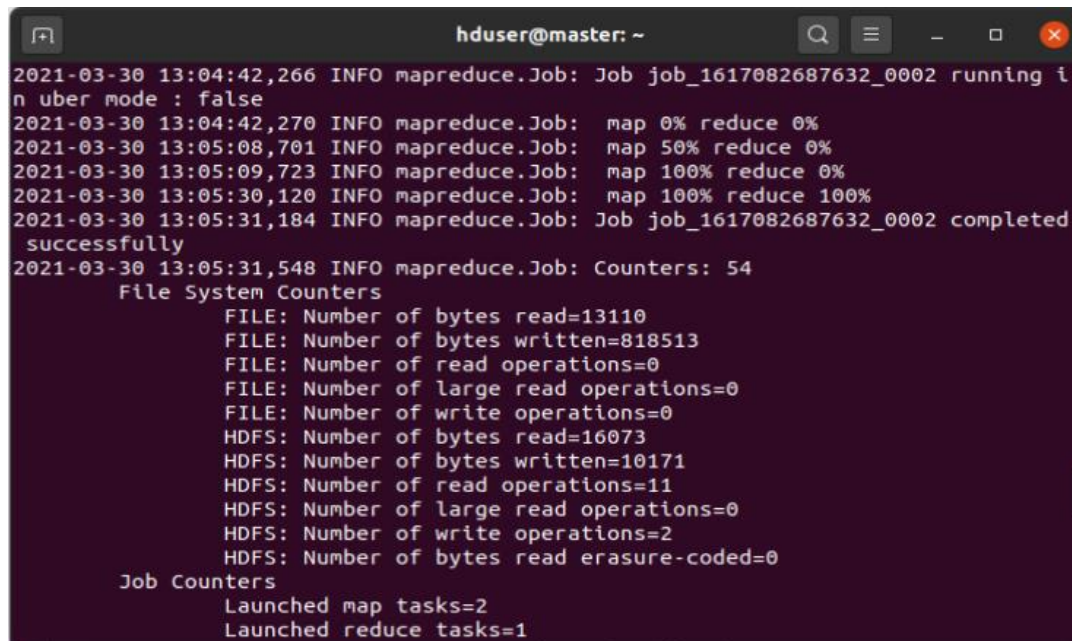
Trong đó:

/input là thư mục chứa các tệp text trên HDFS

/output là thư mục trên HDFS (*ko nên tồn tại trước*) chứa kết quả đầu ra của của chương trình

Bài tập thực hành 1

- **Bước 7:** Theo dõi tiến trình xử lý của MapReduce và check kết quả
 - Chương trình hoàn thành khi 100% map tasks và 100% reduce tasks kết thúc, khi đó **Job** ở trạng thái **Completed!**



```
hduser@master: ~  
2021-03-30 13:04:42,266 INFO mapreduce.Job: Job job_1617082687632_0002 running i  
n uber mode : false  
2021-03-30 13:04:42,270 INFO mapreduce.Job: map 0% reduce 0%  
2021-03-30 13:05:08,701 INFO mapreduce.Job: map 50% reduce 0%  
2021-03-30 13:05:09,723 INFO mapreduce.Job: map 100% reduce 0%  
2021-03-30 13:05:30,120 INFO mapreduce.Job: map 100% reduce 100%  
2021-03-30 13:05:31,184 INFO mapreduce.Job: Job job_1617082687632_0002 completed  
successfully  
2021-03-30 13:05:31,548 INFO mapreduce.Job: Counters: 54  
  File System Counters  
    FILE: Number of bytes read=13110  
    FILE: Number of bytes written=818513  
    FILE: Number of read operations=0  
    FILE: Number of large read operations=0  
    FILE: Number of write operations=0  
    HDFS: Number of bytes read=16073  
    HDFS: Number of bytes written=10171  
    HDFS: Number of read operations=11  
    HDFS: Number of large read operations=0  
    HDFS: Number of write operations=2  
    HDFS: Number of bytes read erasure-coded=0  
  Job Counters  
    Launched map tasks=2  
    Launched reduce tasks=1
```

- Kiểm tra kết quả ở thư mục **/output** trên HDFS

Bài tập thực hành 2

- **Bài toán:** Một hệ thống bán hàng gồm N cửa hàng bán lẻ (shop). Dữ liệu bán hàng của các shop được gửi về hệ thống máy chủ trung tâm để phân tích theo định kỳ. Dữ liệu của mỗi shop được lưu trữ trong 1 tệp *.csv (*Shop-k.csv*, $k \leq N$). Trong đó, mỗi dòng thể hiện tên các sản phẩm được mua trong mỗi hóa đơn của khách hàng, được ngăn cách bởi dấu phẩy (,). Giả sử lượng dữ liệu này là rất lớn. Bạn hãy viết chương trình MapReduce thống kê giúp chủ hệ thống số lượng mỗi sản phẩm bán được trong kỳ

```
MILK,BREAD,BISCUIT  
BREAD,MILK,BISCUIT,CORNFLAKES  
BREAD,TEA,BOURNVITA  
JAM,MAGGI,BREAD,MILK  
MAGGI,TEA,BISCUIT  
BREAD,TEA,BOURNVITA
```

Shop-01.csv

- Giới thiệu tổng quan về MapReduce
- Nguyên lý hoạt động của MapReduce
- Chi tiết đặc điểm và vai trò của các thành phần trong MapReduce
- Hướng dẫn thực hành lập trình MapReduce trên Eclipse

Trân trọng cảm ơn!
Q&A