ISIT312 Big Data Management

# MapReduce Data Processing Model

Dr Fenghui Ren

School of Computing and Information Technology -
University of Wollongong

# MapReduce Data Processing Model

## Outline

# Key-value pairs

Key-Value pairs: MapReduce basic data model

```sql
Key                     Value
City                    Sydney
Employer                Cloudera
```

Input, output, and intermediate records in MapReduce are represented as key-value pairs (aka name-value/attribute-value pairs)

A key is an identifier, for example, a name of attribute

- In MapReduce, a key is not required to be unique.

A value is a data associated with a key

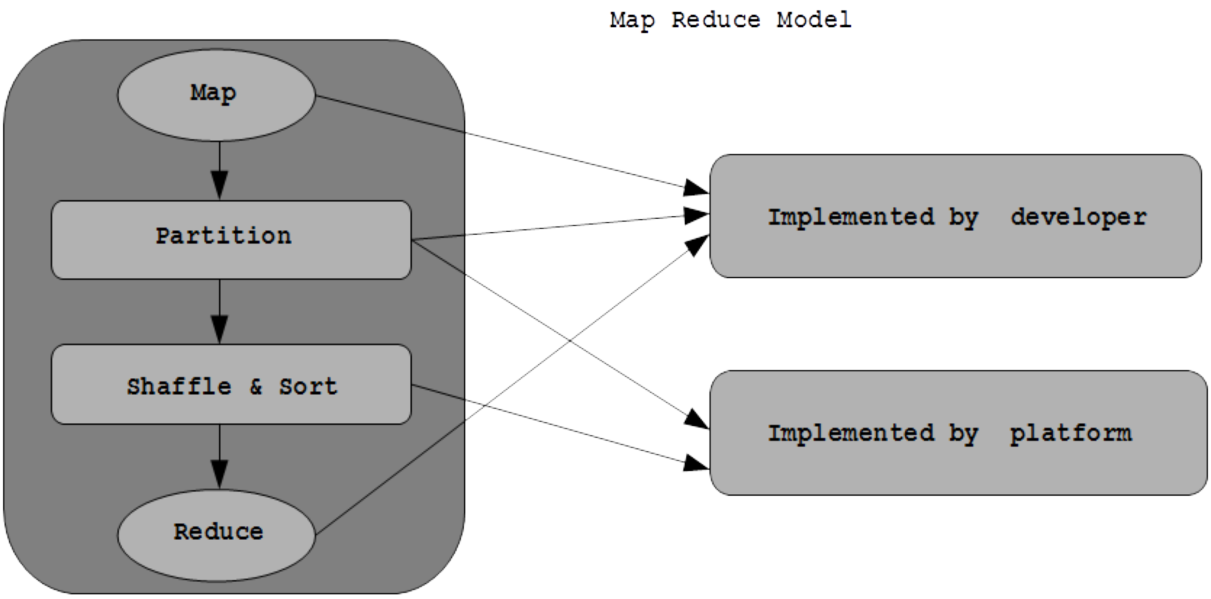
- It may be simple value or a complex object

# MapReduce Data Processing Model

## Outline

# MapReduce Model

MapReduce data processing model is a sequence of Map, Partition, Shuffle and Sort, and Reduce stages



# MapReduce Model

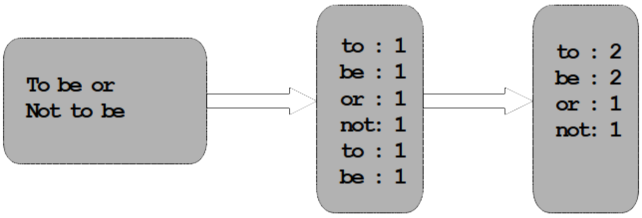
## An abstract MapReduce program: `WordCount`

Function Map

```
function Map(Long lineNo, String line):
  lineNo: the position no. of a line in the text
  line: a line of text
    for each word w in line:
      emit (w, 1)
```

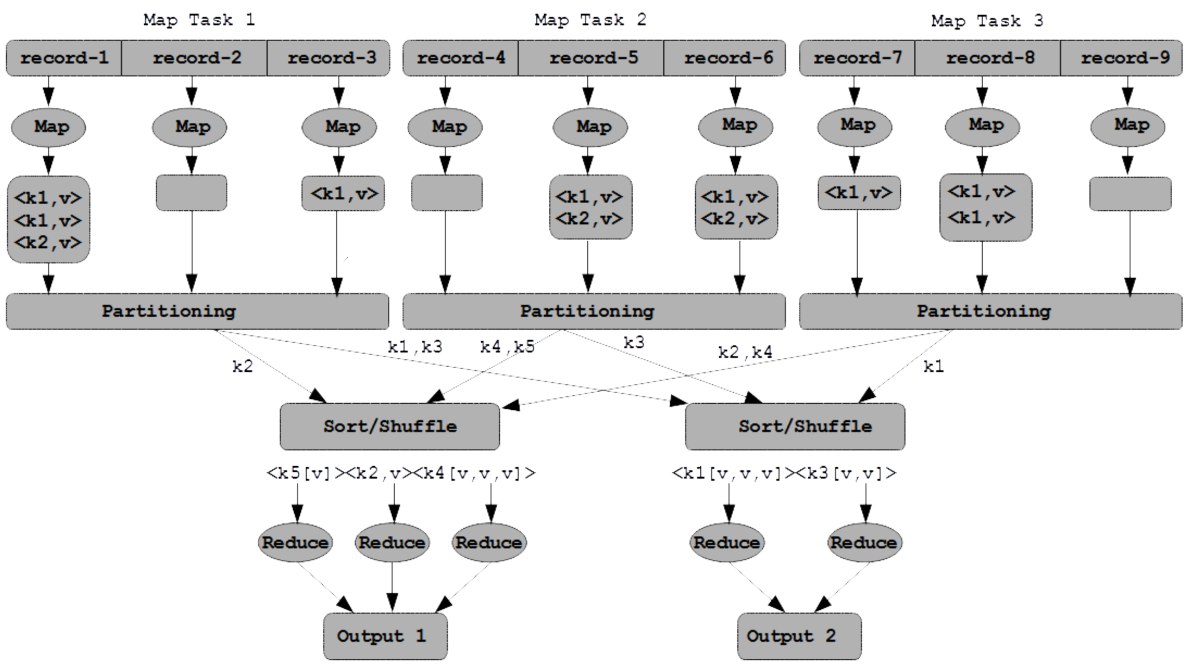
Function Reduce

```
function Reduce(String w, List loc):
  w: a word
  loc: a list of counts outputted from map instances
    sum = 0
    for each c in loc:
      sum += c
    emit (word, sum)
```



# MapReduce Model

A diagram of data processing in MapReduce model



# MapReduce Data Processing Model

## Outline

# Map phase

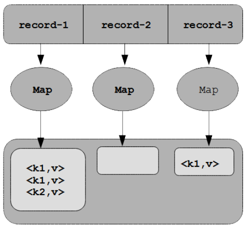
Map phase uses input format and record reader functions to derive records in the form of key-value pairs for the input data

Map phase applies a function or functions to each key-value pair over a portion of the dataset

- In the case of a dataset hosted in HDFS, this portion is usually called as a block
- If there are n blocks of data in the input dataset, there will be at least n Map tasks (also referred to as Mappers)

Each Map task operates against one filesystem (HDFS) block

In the diagram fragment, a Map task will call its `map()` function, represented by M in the diagram, once for each record, or key-value pair; for example, rec1, rec2, and so on.



# Map phase

Each call of the `map()` function accepts one key-value pair and emits zero or more key-value pairs

A call of Map() function

```
map (in_key, in_value) -> list (intermediate_key, intermediate_value)
```

The emitted data from Mapper, also in the form of lists of key-value pairs, will be subsequently processed in the Reduce phase

Different Mappers do not communicate or share data with each other

Common `Map()` functions include filtering of specific keys, such as filtering log messages if you only wanted to count or analyse ERROR log messages

Sample Map() function

```
Map (k, v) = if (ERROR in v) then emit (k, v)
```

Another example of `Map()` function would be to manipulate values, such as a function that converts a text value to lowercase

Sample Map() function

```
Map (k, v) = emit (k, v.toLowercase ( ))
```

# Map phase

Partition function, or Partitioner, ensures each key and its list of values is passed to one and only one Reduce task or Reducer

The number of partitions is determined by the (default or user-defined) number of Reducers

Custom Partitioners are developed for various practical purposes

# MapReduce Data Processing Model

## Outline

# Reduce Phase

Input of the Reduce phase is output of the Map phase (via shuffle-and sort)

Each Reduce task (or Reducer) executes a `reduce()` function for each intermediate key and its list of associated intermediate values

The output from each `reduce()` function is zero or more key-values

A call of Reduce() function

```
reduce (intermediate_key, list (intermediate_value)) -> (out_key, out_value)
```

Note that, in the reality, an output from Reducer may be an input to another Map phase in a complex multistage computational workflow

# Example of Reduce Functions

The simplest and most common `reduce()` function is the Sum Reducer, which simply sums a list of values for each key

```
reduce (k, list ) =
  {
    sum = 0
    for int i in list  :
      sum + = i
    emit (k, sum)
  }
```

Sum reducer

A count operation is as simple as summing a set of numbers representing instances of the values you wish to count

Other examples of `reduce()` function are `max()` and `average()`

# MapReduce Data Processing Model

## Outline

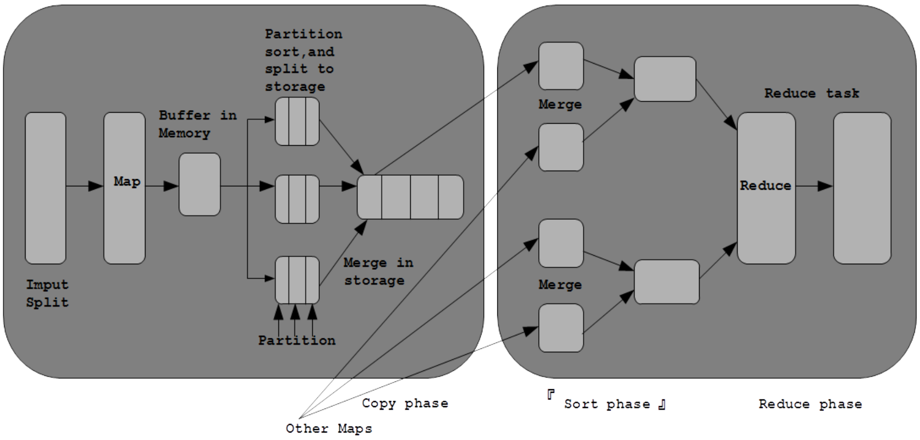
# Shuffle and Sort

Shuffle-and-sort is the process where data are transferred from Mapper to Reducer

- It is the heart of MapReduce where the "magic" happens

The most important purpose of Shuffle-and-sort is to minimise data transmission through a network

In general, in Shuffle-and-Sort, the Mapper output is sent to the target Reduce task according to the partitioning function

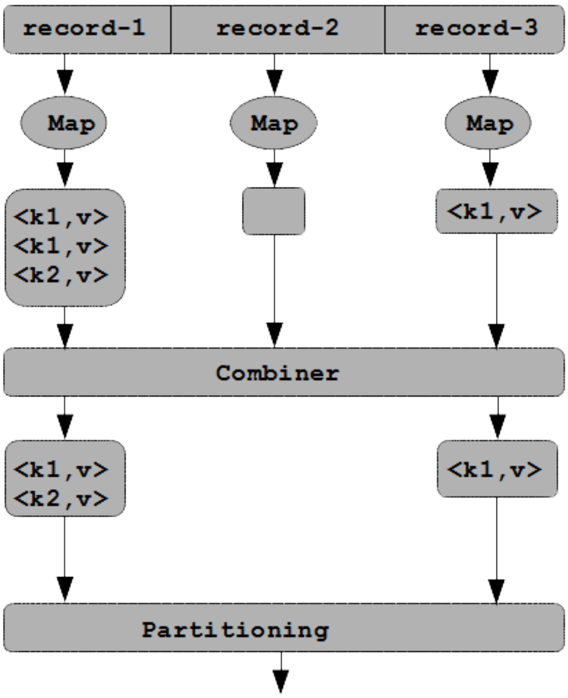


# MapReduce Data Processing Model

## Outline

# Combine phase

A structure of Combine phase



# Combine phase

If the Reduce function is commutative and associative then it can be performed before the Shuffle-and-Sort phase

In this case, the Reduce function is called a Combiner function

For example, `sum` and `count` is commutative and associative, but `average` is not

The use of a Combiner can minimise the amount of data transferred to Reduce phase and in such a way reduce the network transmit overhead

A MapReduce application may contain zero Reduce tasks

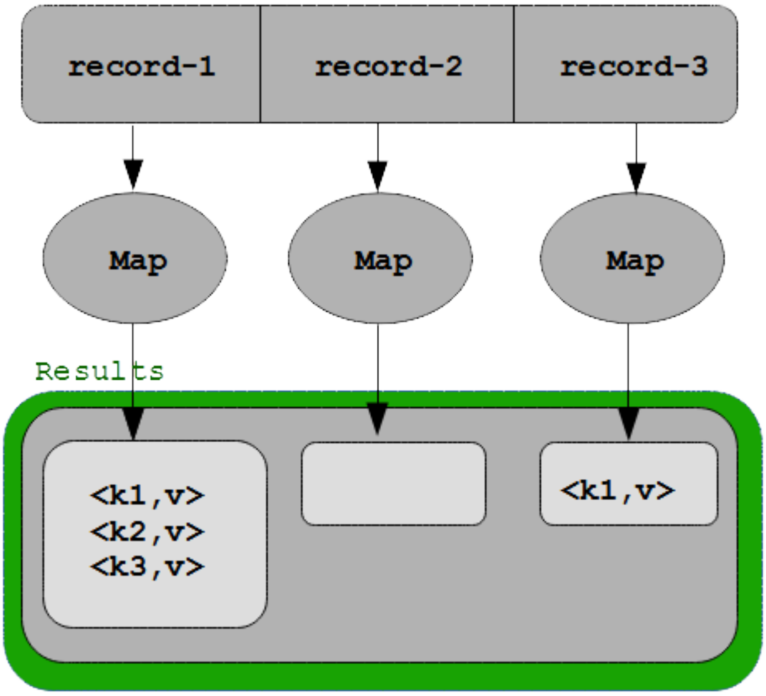
In this case, it is a Map-Only application

Examples of Map-only MapReduce jobs

- ETL routines without data summarization, aggregation and reduction
- File format conversion jobs
- Image processing jobs

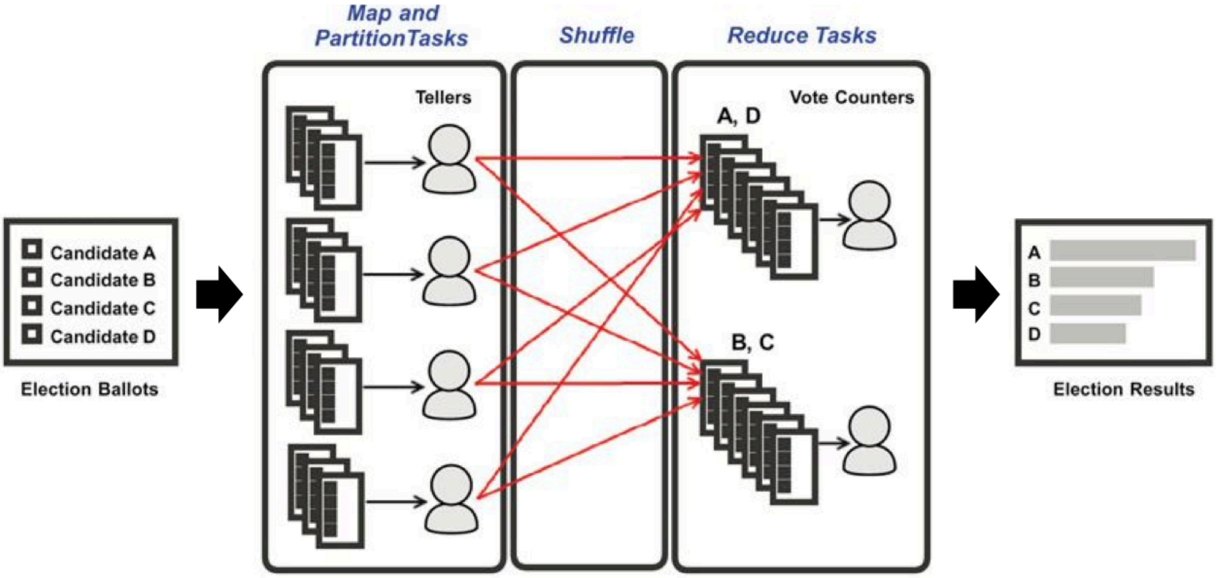
# Combine phase

Map-Only MapReduce



# Combine phase

An election Analogy for MapReduce



# MapReduce Data Processing Model

## Outline

# Example

For a database of 1 billion people, compute the average number of social contacts a person has according to age

In SQL like language

SELECT statement

```
SELECT age, AVG(contacts)
FROM social.person
GROUP BY age
```

If the records are stored in different datanodes then in Map function is the following

Map function

```
function Map is
      input: integer K between 1 and 1000, representing a batch of 1
      million social.person records
      for each social.person record in the K-th batch do
        let Y be the person age
        let N be the number of contacts the person has
          produce one output record (Y,(N,1))
      repeat
end function
```

# Example

Then Reduce function is the following

```
function Reduce is                                                    Reduce function
        input: age (in years) Y
        for each input record (Y,(N,C)) do
          Accumulate in S the sum of N*C
          Accumulate in C_new the sum of C
        repeat
        let A be S/C_new
        produce one output record (Y,(A,C_new ))
end function
```

MapReduce sends the codes to the location of each data batch (not the other way around)

Question: the output from Map is multiple copies of `(Y, (N, 1))`, but the input to Reduce is `(Y, (N, C))`, so what fills the gap?

# Example

A MapReduce application in Hadoop is a Java implementation of the MapReduce model for a specific problem, for example, word count

# Example

## Sample processing on a screen

```
[root@hadoop2 sbin]# hadoop jar /opt/yarn/hadoop-2.6.0/share/hadoop/mapreduce/
    hadoop-mapreduce-examples-2.6.0.jar wordcount /shakes shake_output
16/05/22 10:44:00 INFO input.FileInputFormat: Total input paths to process : 1
16/05/22 10:44:01 INFO mapreduce.JobSubmitter: number of splits:1
16/05/22 10:44:01 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1440603749764_0001
16/05/22 10:44:03 INFO impl.YarnClientImpl: Submitted application  application_1440603749764_0001
16/05/22 10:44:03 INFO mapreduce.Job: The url to track the job: http://hadoop1.localdomain:8081/
    proxy/application_1440603749764_0001/
16/05/22 10:44:44 INFO mapreduce.Job: Job job_1440603749764_0001 running in uber mode : false
16/05/22 10:44:44 INFO mapreduce.Job:  map 0% reduce 0%
16/05/22 10:45:07 INFO mapreduce.Job:  map 67% reduce 0%
16/05/22 10:45:17 INFO mapreduce.Job:  map 100% reduce 0%
16/05/22 10:45:33 INFO mapreduce.Job:  map 100% reduce 100%
16/05/22 10:45:34 INFO mapreduce.Job: Job job_1440603749764_0001 completed successfully
16/05/22 10:45:36 INFO mapreduce.Job: Counters: 49
       File System Counters
              FILE: Number of bytes read=983187
              FILE: Number of bytes written=2178871
              HDFS: Number of bytes read=5590008
              HDFS: Number of bytes written=720972
              HDFS: Number of read operations=6

              HDFS: Number of write operations=2
```

**The application**

# Example

## Sample processing on a screen

```
Job Counters
        Launched map tasks=1
        Launched reduce tasks=1
        Data-local map tasks=1
        Total time spent by all maps in occupied slots (ms)=30479
        Total time spent by all reduces in occupied slots (ms)=13064
        Total time spent by all map tasks (ms)=30479
        Total time spent by all reduce tasks (ms)=13064
        Total vcore-seconds taken by all map tasks=30479
        Total vcore-seconds taken by all reduce tasks=13064
        Total megabyte-seconds taken by all map tasks=31210496
        Total megabyte-seconds taken by all reduce tasks=13377536
Map-Reduce Framework
        Map input records=124787
        Map output records=904061
        Map output bytes=8574733
        Map output materialized bytes=983187
        Input split bytes=119
        Combine input records=904061
        Combine output records=67779
        Reduce input groups=67779
        Reduce shuffle bytes=983187
        Reduce input records=67779
        Reduce output records=67779
        Spilled Records=135558
        Shuffled Maps =1
                      Merged Map outputs=1
        GC time elapsed (ms)=454
        CPU time spent (ms)=10520
        Physical memory (bytes) snapshot=302411776
        Virtual memory (bytes) snapshot=1870229504
        Total committed heap usage (bytes)=168497152

File Input Format Counters
        Bytes Read=5589889
File Output Format Counters
        Bytes Written=720972
```

# MapReduce Data Processing Model

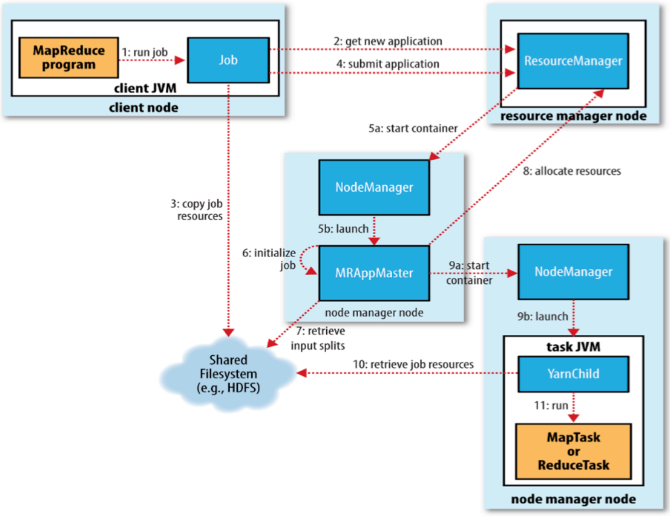
## Outline

# Running MapReduce Jobs

Client submits Mapreduce job

YARN resource manager coordinates the allocation of computing resources in the cluster

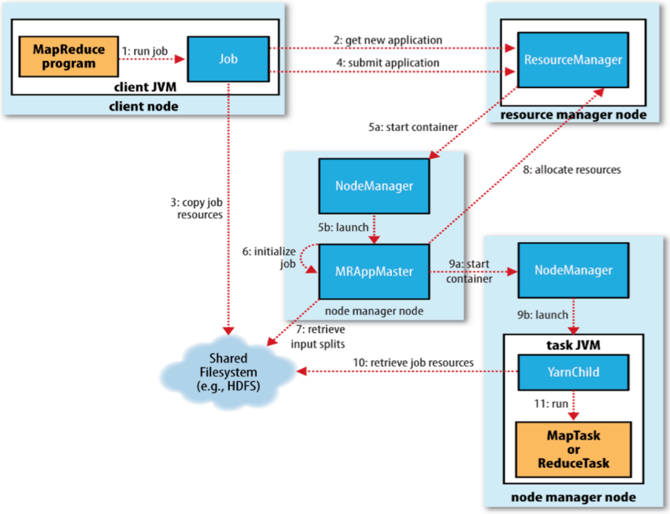
YARN node manager(s): launch & monitor containers on machines in the cluster



# Running MapReduce Jobs

MapReduce application master runs in a container, and coordinates the tasks in a MapReduce job

HDFS is used for sharing job files between the other files



# References

White T., Hadoop The Definitive Guide: Storage and Analysis at Internet Scale, O'Reilly, 2015