Objective

- What is MapReduce
- MapReduce Benefits
- WordCount Program Flow
- MapReduce Flow Chart
- MapReduce Phases
- MapReduce Input/Output Format
- MapReduce DataTypes
- MapReduce Advance Features
- Questions?

What is MapReduce

- ✓ MapReduce is a framework for writing applications that process large amounts of structured and unstructured data stored in the Hadoop Distributed File System (HDFS).
- ✓ It is used to process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.
- ✓ The MapReduce framework and the Hadoop Distributed File System run on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster.
- ✓ A MapReduce program has two components: one that implements the mapper, and another that implements the reducer.
- ✓ A MapReduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner.
- ✓ The framework shuffle and sorts the outputs of the maps, which are then input to the reduce tasks.
- ✓ Both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.
- ✓ In MapReduce, users write a client application that submits one or more jobs that contain user-supplied map and reduce code and a job configuration file to a cluster of machines.
- ✓ The job contains a map function and a reduce function, along with job configuration information that controls various aspects of its execution.
- ✓ The map and reduce functions in Hadoop MapReduce have the following general form:

  Map: (K1, V1) → list (K2, V2)

  Reduce: (K2, list (V2)) → list (K3, V3)

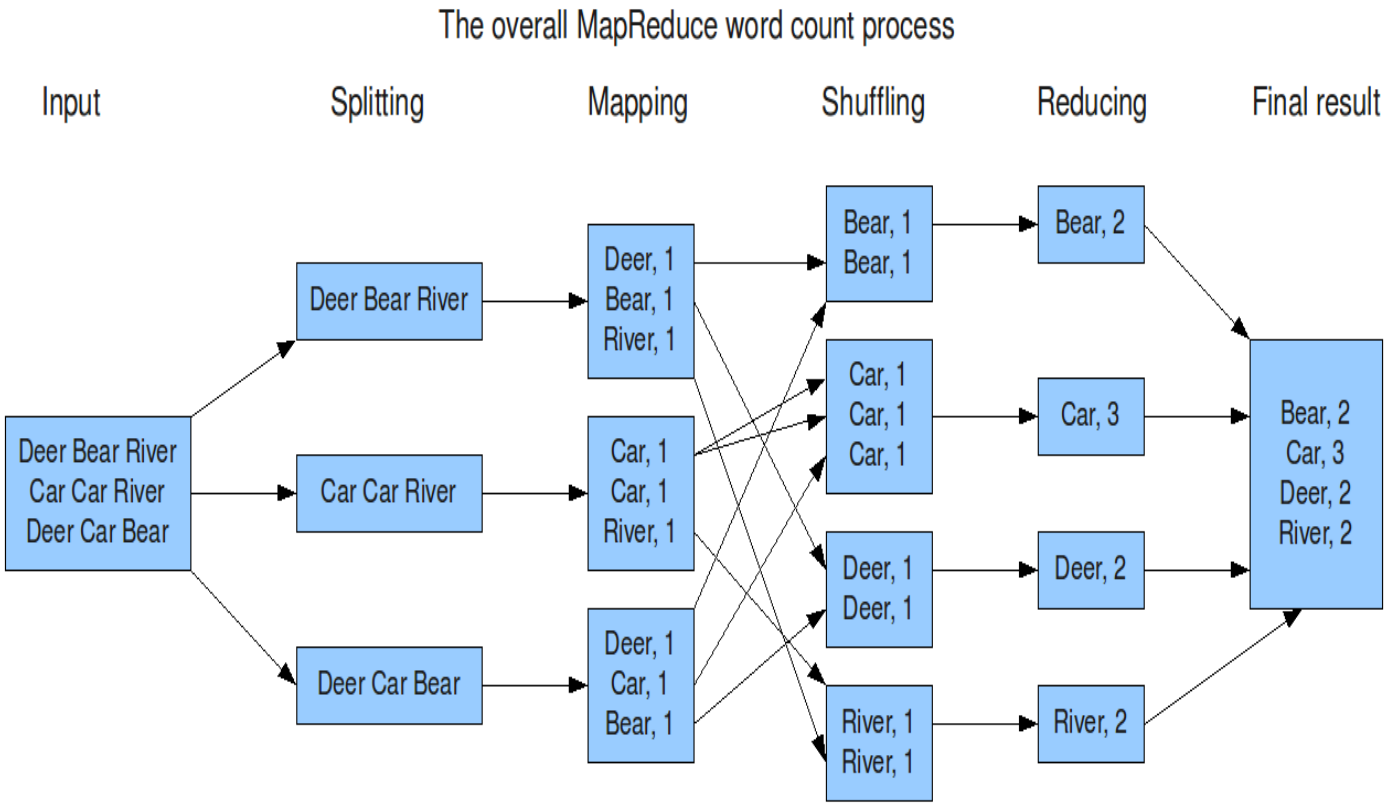
MapReduce Benefits

MapReduce is useful for batch processing on terabytes or petabytes of data stored in Apache Hadoop.

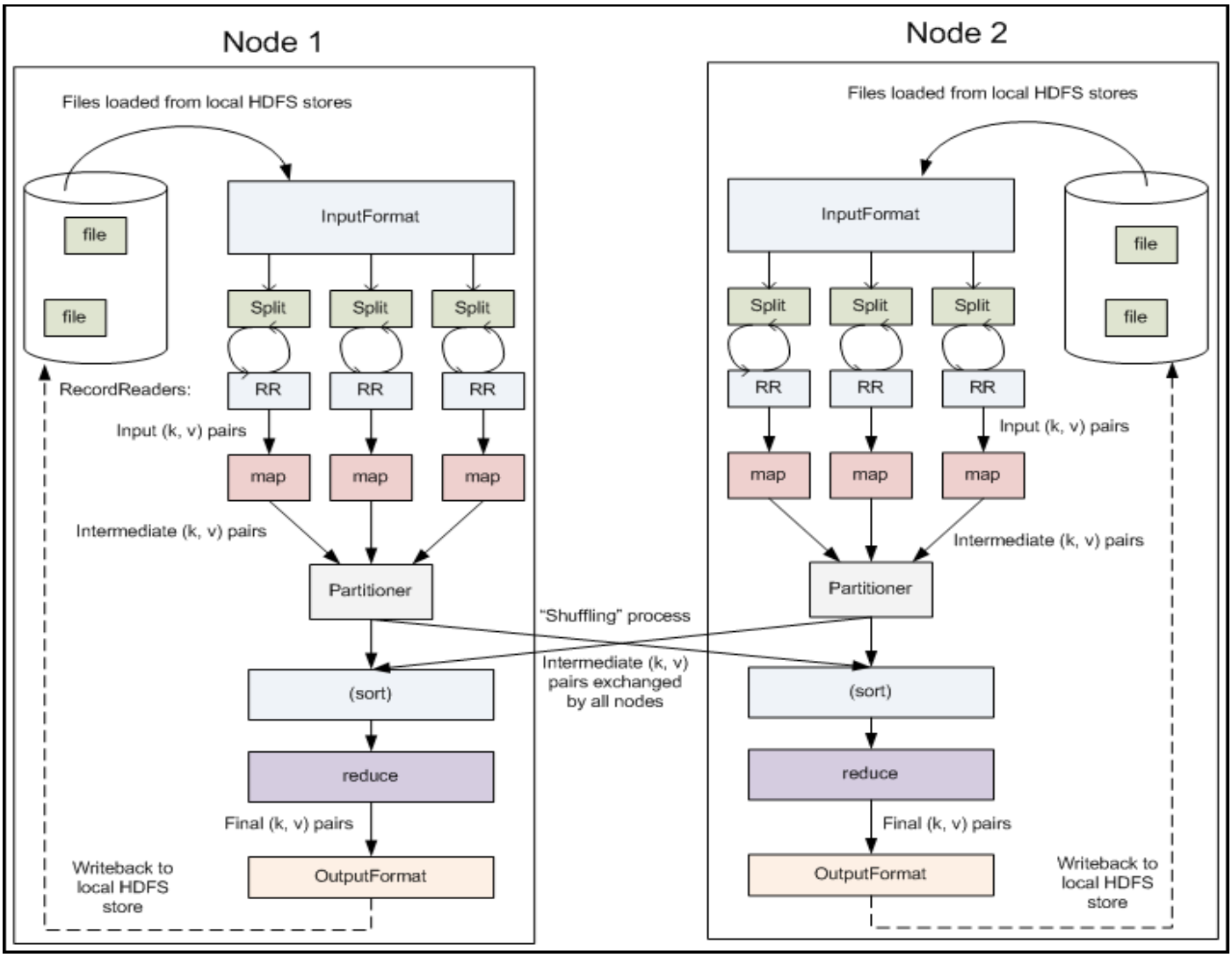
The following tables describes some of MapReduce's key benefits:

Benefit	Description
Simplicity	Developers can write applications in their language of choice, such as Java, C++ or Python, and MapReduce jobs are easy to run
Scalability	MapReduce can process petabytes of data, stored in HDFS on one cluster
Speed	Parallel processing means that MapReduce can take problems that used to take days to solve and solve them in hours or minutes
Recovery	MapReduce takes care of failures. If a machine with one copy of the data is unavailable, another machine has a copy of the same key/value pair, which can be used to solve the same sub-task. The JobTracker keeps track of it all.
Minimal data motion	MapReduce moves compute processes to the data on HDFS and not the other way around. Processing tasks can occur on the physical node where the data resides. This significantly reduces the network I/O patterns and contributes to Hadoop's processing speed.

MapReduce WordCount Program Flow



MapReduce Flow Chart



MapReduce Phases

InputSplits:

- InputSplit represents the data to be processed by an individual Mapper.
- Input splits are represented by the Java class InputSplit.
- A split doesn’t contain the input data; it is just a reference to the data.
- Typically InputSplit presents a byte-oriented view of the input, and it is the responsibility of RecordReader to process and present a record-oriented view.
- FileSplit is the default InputSplit.

RecordReader:

- RecordReader reads <key, value> pairs from an InputSplit.
- Typically the RecordReader converts the byte-oriented view of the input, provided by the InputSplit, and presents a record-oriented to the Mapper implementations for processing.
- The RecordReader class actually loads the data from its source and converts it into (key, value) pairs suitable for reading by the Mapper.

Mapper:

- Mapper maps input key/value pairs to a set of intermediate key/value pairs.
- A new instance of Mapper is instantiated in a separate Java process for each map task (InputSplit).
- map method of Mapper class called for each InputSplit.
- The individual mappers can’t communicate with one another in any way.

Reducer:

Reducer has **3** primary phases: shuffle, sort and reduce.

1. **Shuffle:** Shuffling is a phase on intermediate data to combine all values into a collection associated to same key. After this there will be no duplicate key in intermediate data.

2. **Sort:** The set of intermediate keys on a single node is automatically sorted by Hadoop before they are presented to the Reducer.

Sorting is done because of Box Classes.

   The shuffle and sort phases occur simultaneously; while map-outputs are being fetched they are merged.

3. **Reduce:** Shuffled and sorted output data of mapper is provided to Reducer. In this phase the reduce(WritableComparable, Iterator, OutputCollector, Reporter) method is called for each <key, (list of values)> pair in the grouped inputs.

RecordWriter:

It is used to get one (Key,Value) data from Reducer and write it into output file Ex: part-00000

MapReduce Input/Output Format

InputFormat:

How these input files are split up and read is defined by the InputFormat. An InputFormat is a class that provides the following functionality:

- Selects the files or other objects that should be used for input.
- Defines the InputSplits that break a file into tasks.
- Provides a factory for RecordReader objects that read the file.

An InputFormat is responsible for creating the input splits and dividing them into records

We can choose InputFormat to apply to our input files for a job by calling the **setInputFormat()** method of the JobConf object that defines the job.

A table of standard InputFormats is given below.

InputFormat:	Description:	Key:	Value:
TextInputFormat	Default format; reads lines of text files	The byte offset of the line	The line contents
KeyValueInputFormat	Parses lines into key, value pairs	Everything up to the first tab character	The remainder of the line
SequenceFileInputFormat	A Hadoop-specific high-performance binary format	user-defined	user-defined

OutputFormat:

A table of standard OutputFormats is given below.

OutputFormat:	Description
TextOutputFormat	Default; writes lines in "key \t value" form
SequenceFileOutputFormat	Writes binary files suitable for reading into subsequent MapReduce jobs
NullOutputFormat	Disregards its inputs

MapReduce DataTypes

Objective types used for (Key,Value) pairs.

Wrapper Classes (In Java)	Primitive Types	Box Classes (In Hadoop)
Integer	int	IntWritable
Long	long	LongWritable
Float	float	FloatWritable
Double	double	DobuleWritable
String	String	Text

All these Box classes implemented Writable and WritableComparable interfaces so that they can compare with each other.

Sorting of intermediate data is done because of these Box classes.

Conversion from Primitive type to BoxType

Primitive Types	Box Classes (In Hadoop)	Method
int	IntWritable	(new IntWritable (int))
long	LongWritable	(new LongWritable (long))
float	FloatWritable	(new FloatWritable (float))
double	DobuleWritable	(new DoubleWritable (double))
String	Text	(new Text (String))

Conversion from BoxType to Primitive type

Box Classes (In Hadoop)	Primitive Types	Method
IntWritable	int	(get ( ))
LongWritable	long	(get ( ))
FloatWritable	float	(get ( ))
DobuleWritable	double	(get ( ))
Text	String	(String (toString ( ))

MapReduce Advance Features

1. Combiner

- It is called mini reducer.
- It perform same task (shuffle and sort) as reducer does, on each mapper output data i.e (Key,value)
- Number of combiner will be equal to number of Mapper.
- Each and every combiner will work on its individual mapper.
- When all combiner will finish the work, then reducer take input from all combiners.
- It increase application performance by reducing network traffic.

2. Partitioner

- It increase application performance.
- HashPartitioner is default class in MapReduce. This class split all Key,value pair to different reducer depending on hashcode of our object.
- If we want specific (Key,Value) pair should move to specific reducer then we must write our own partitioner code.
- It will push the data into specific reducer based on over partitioner code logic.

3. Speculative Execution

- The MapReduce model is to break jobs into tasks and run the tasks in parallel to make the overall job execution time smaller than it would be if the tasks ran sequentially.
- When a job consists of hundreds or thousands of tasks, the possibility of a few struggling tasks is very real.
- Speculative execution is an optimization, and not a feature to make jobs run more reliably.
- Speculative execution is turned on by default. It can be enabled or disabled independently for map tasks and reduce tasks, on a cluster-wide basis, or on a per-job basis.
- We can disable speculative execution for the mappers and reducers by setting the mapred.map.tasks.speculative.execution and mapred.reduce.tasks.speculative.execution JobConf options to false, respectively.

4. Distributed Cache

- DistributedCache is a facility provided by the MapReduce framework to cache files (text, archives, jars and so on) needed by applications.
- DistributedCache distributes application-specific, large, read-only files efficiently.
- Applications specify the files to be cached via urls (hdfs://) in the JobConf. The DistributedCache assumes that the files specified via hdfs:// urls are already present on the FileSystem.
- The framework will copy the necessary files to the slave node before any tasks for the job are executed on that node.

5. Data Compression

- Hadoop MapReduce provides facilities for the application-writer to specify compression for both intermediate map-outputs and the job-outputs.
- It also comes bundled with CompressionCodec implementation for the zlib compression algorithm.
- The gzip file format is also supported.

6. Skipping bad records

- Hadoop provides an option where a certain set of bad input records can be skipped when processing map inputs.
- Applications can control this feature through the **SkipBadRecords** class.
- By default this feature is disabled.

7. Hadoop Streaming

Streaming is a generic API that allows programs written in any language to be used as Hadoop Mapper and Reducer implementations.