

Cloud Computing, ITNT404

Lecturer: Dr. Omar Abusada

E-mail: omar.abusaeeda@uot.edu.ly

MapReduce as Hadoop Tool

What is Map-Reduce?

Map-Reduce could be defined as **two parts**:

- **A model** for writing programs that can easily be made to process data in parallel.
- **A framework** that runs these programs in parallel, automatically handling the details of division of labor, distribution, synchronization, and fault-tolerance.

The **model** and the **framework** work together to make programs that are scalable, distributed, and fault-tolerant.

Apache MapReduce

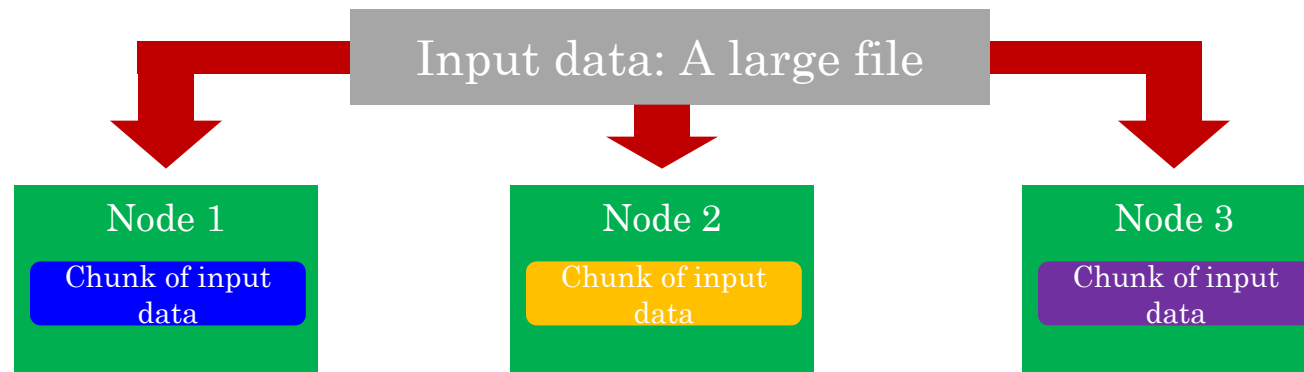
- All the world turned digital
- A software framework for distributed processing of large data sets
- The framework takes care of **scheduling tasks, monitoring** them and **re-executing any failed tasks**.
- It splits the input data set into independent chunks that are processed in a completely parallel manner.
- MapReduce framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically, both the input and the output of the job are stored in a file system.

MapReduce programming model

- **MapReduce consists of two phases and its key innovation is:**
- The ability to take a query over a data set, divide it, and run it in parallel over many nodes.
- Solves the issue of data too large (Big Data) to fit onto a single machine
 - Distributed computing over many servers
 - Batch processing model
- **Map phase**, input data is processed, item by item, and transformed into an intermediate data set.
- **Reduce phase**, these intermediate results are reduced to a summarized data set, which is the desired end result.

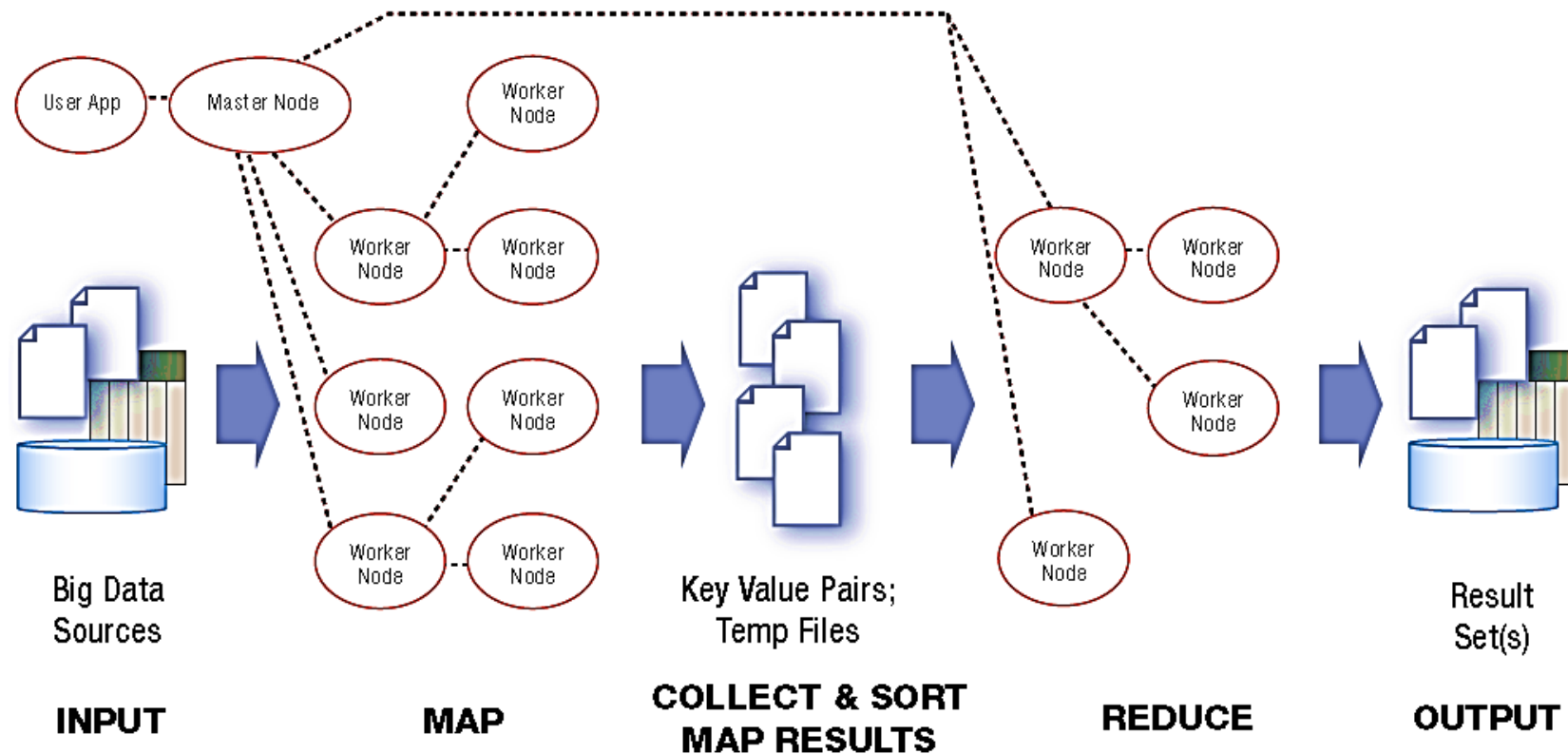
Data Distribution

- In a MapReduce cluster, data is distributed to all the nodes of the cluster as it is being loaded in.
- An underlying distributed file systems (e.g., GFS) splits large data files into chunks which are managed by different nodes in the cluster
- Even though the file chunks are distributed across several machines, they form *a single namespace*

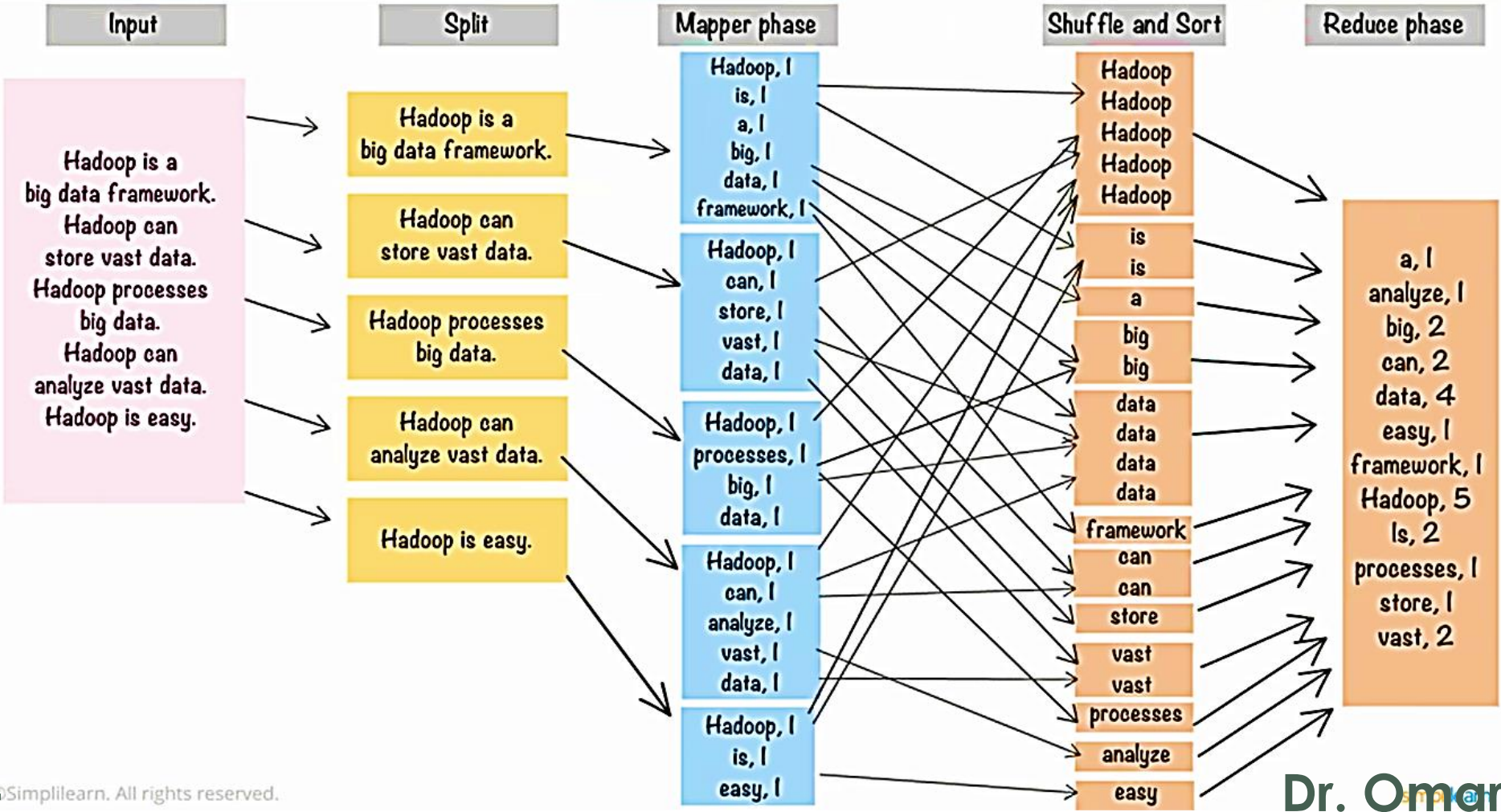


MapReduce: Very high level overview

- process data in a batch-oriented fashion and may take minutes or hours to process (normally).



MapReduce: Very high level overview



MapReduce: Very high level overview

- **Loading the data**

- This operation is properly called **Extract, Transform, Load (ETL)** in data warehousing terminology.
- Data must be extracted from its source, structured to make it ready for processing, and loaded into the storage layer for MapReduce to operate on it.

- **MapReduce**

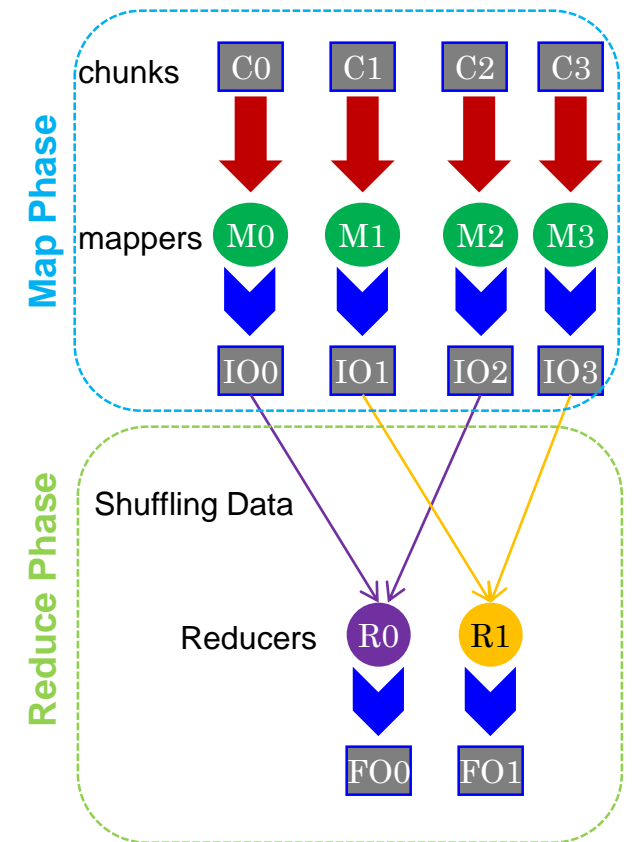
- This phase will recover data from storage,
- Process it (map, collect and sort map results, reduce)
- And return the results to the storage.

- **Extracting the result**

- Once processing is complete, for the result to be useful, it must be retrieved from the storage and presented.

MapReduce: A Bird's-Eye View

- In MapReduce, chunks are processed in isolation by tasks called Mappers
- The outputs from the mappers are denoted as intermediate outputs (IOs) and are brought into a second set of tasks called Reducers
- The process of bringing together IOs into a set of Reducers is known as shuffling process
- The Reducers produce the final outputs (FOs)
- Overall, MapReduce breaks the data flow into two phases, map phase and reduce phase



MapReduce: Very high level overview

- **MapReduce Programming Model**

- **Data type: key-value *records***

- **Map function:**

$$(K_{in}, V_{in}) \rightarrow \text{list}(K_{inter}, V_{inter})$$

- **Reduce function:**

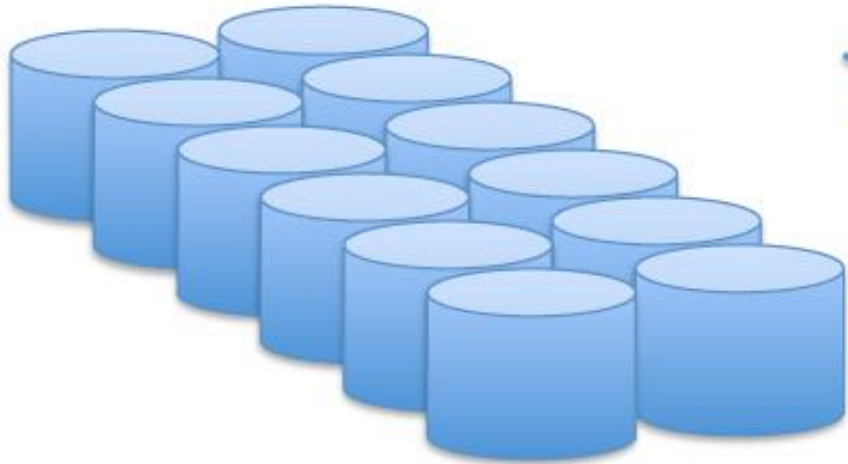
$$(K_{inter}, \text{list}(V_{inter})) \rightarrow \text{list}(K_{out}, V_{out})$$

Benefits of MapReduce Model

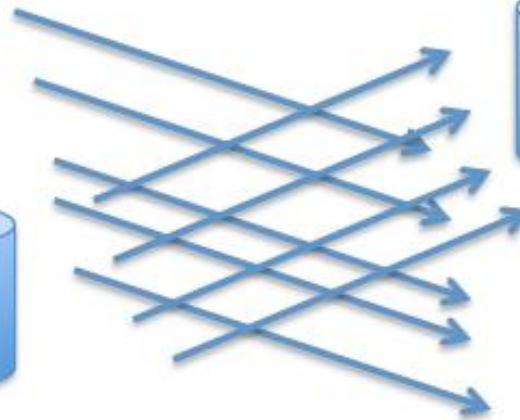
- By providing a data-parallel programming model, MapReduce can control job execution in useful ways:
 - Automatic division of job into tasks
 - Automatic placement of computation near data
 - Automatic load balancing
 - Recovery from failures & stragglers
- User focuses on application, not on complexities of distributed computing (Implicit Parallelism)

Example: Word Count

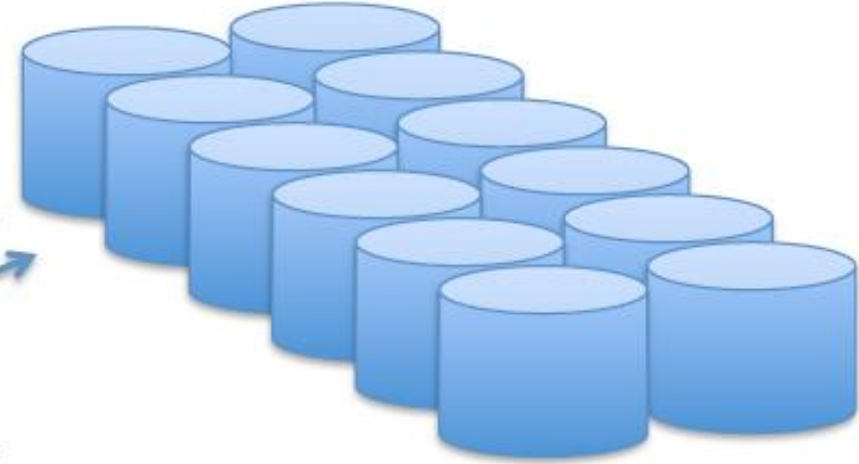
- Basic Pattern: Strings



1. Extract words from web pages in parallel.



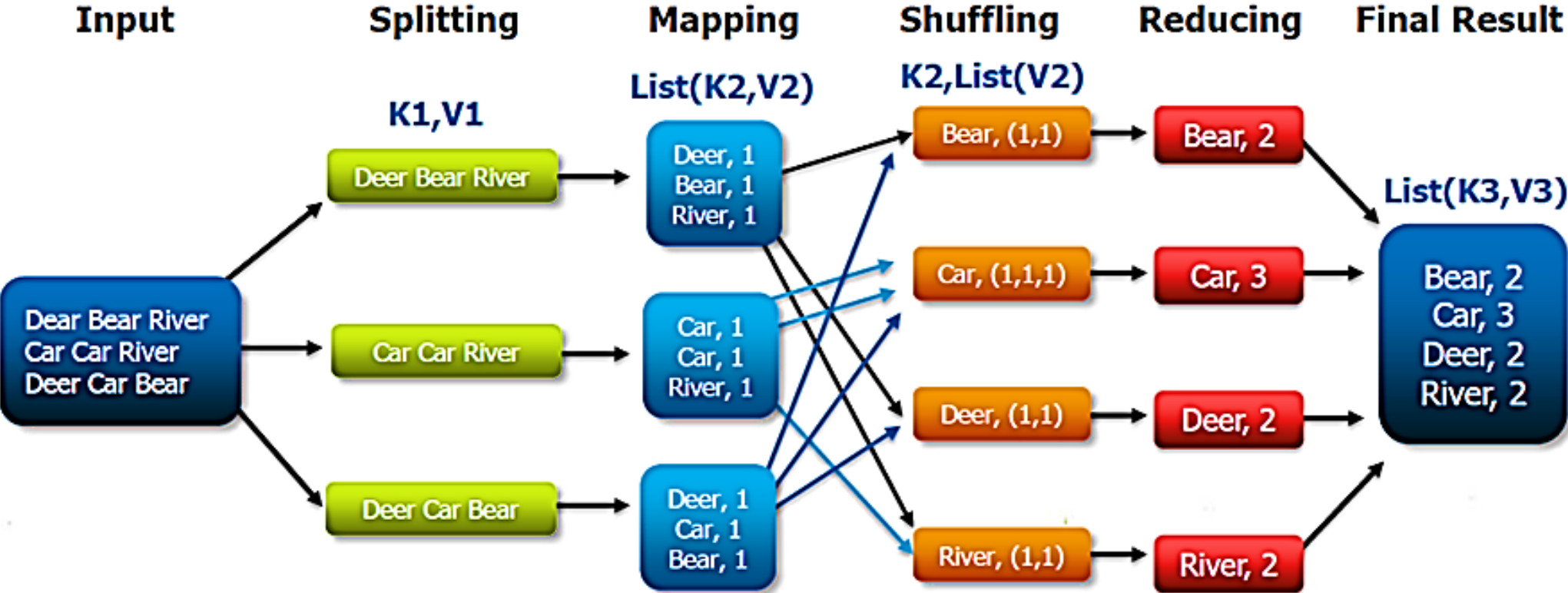
2. Hash and sort words.



3. Count in parallel.

Common word_Count

The Overall MapReduce Word Count Process



Common word_Count

```
void map(string i, string line):  
    for word in line:  
        print word, 1
```

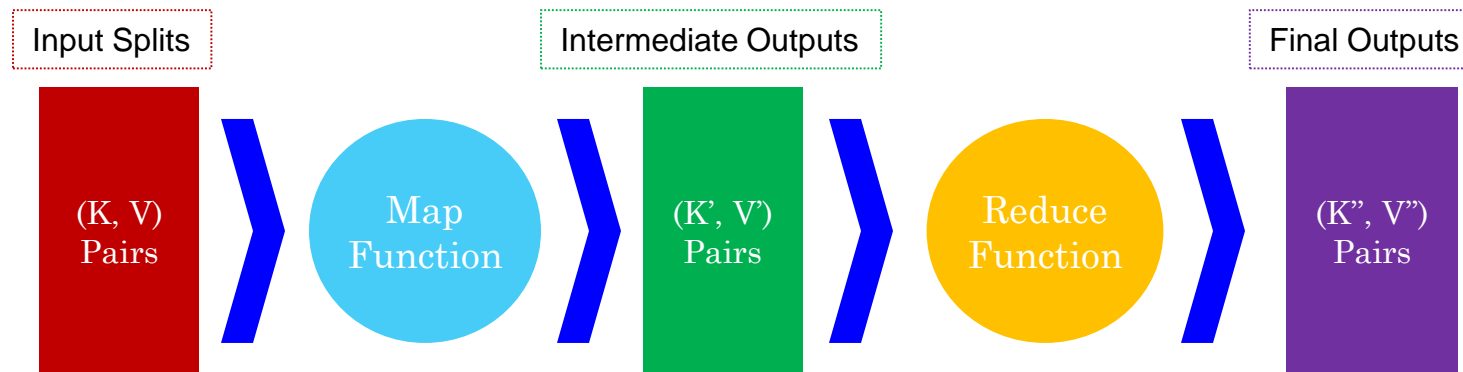
Word_count – map function

```
void reduce(string word, list partial_counts):  
    total = 0  
    for c in partial_counts:  
        total += c  
    print word, total
```

Word_count – reduce function

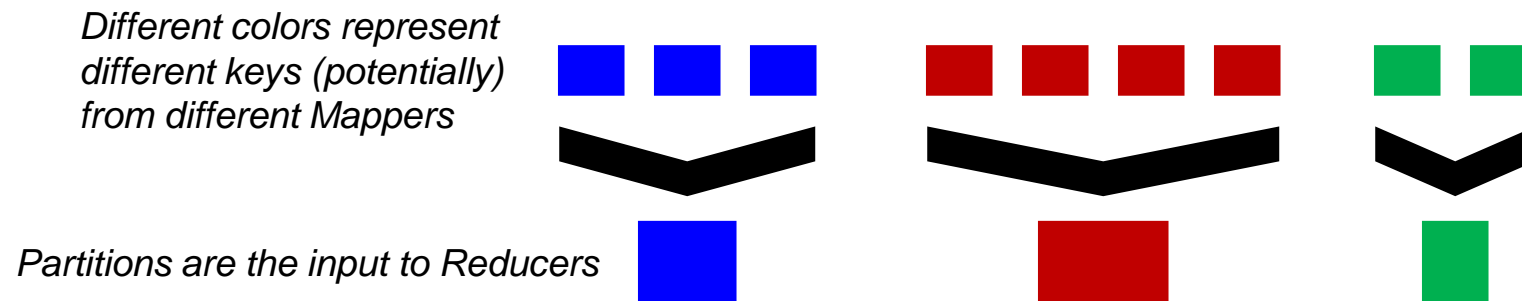
Keys and Values

- The programmer in MapReduce has to specify two functions, the map function and the reduce function that implement the Mapper and the Reducer in a MapReduce program.
- In MapReduce data elements are always structured as key-value (i.e., (K, V)) pairs
- The map and reduce functions receive and emit (K, V) pairs

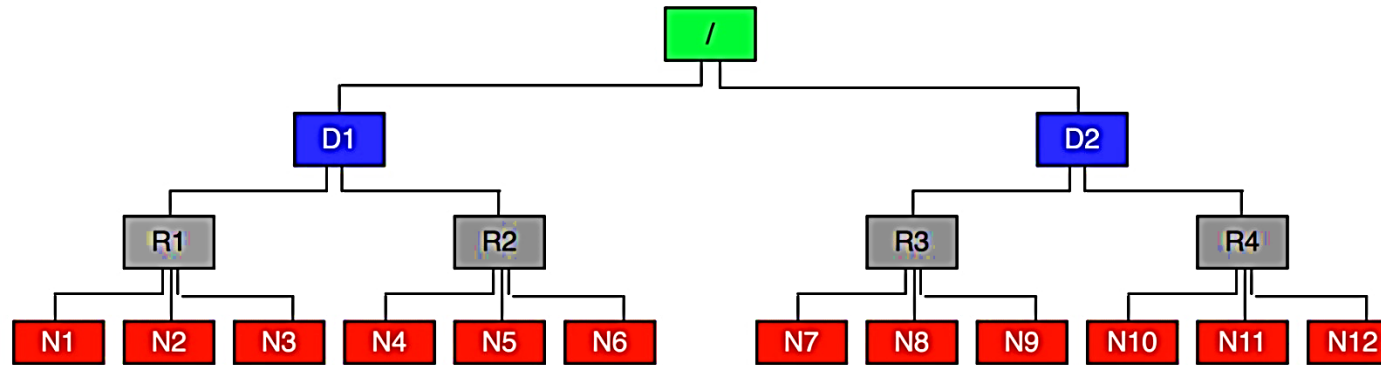


Partitions

- In MapReduce, intermediate output values are not usually reduced together
- All values with the same key are presented to a single Reducer together
- More specifically, a different subset of intermediate key space is assigned to each Reducer
- These subsets are known as partitions



Network Topology In MapReduce

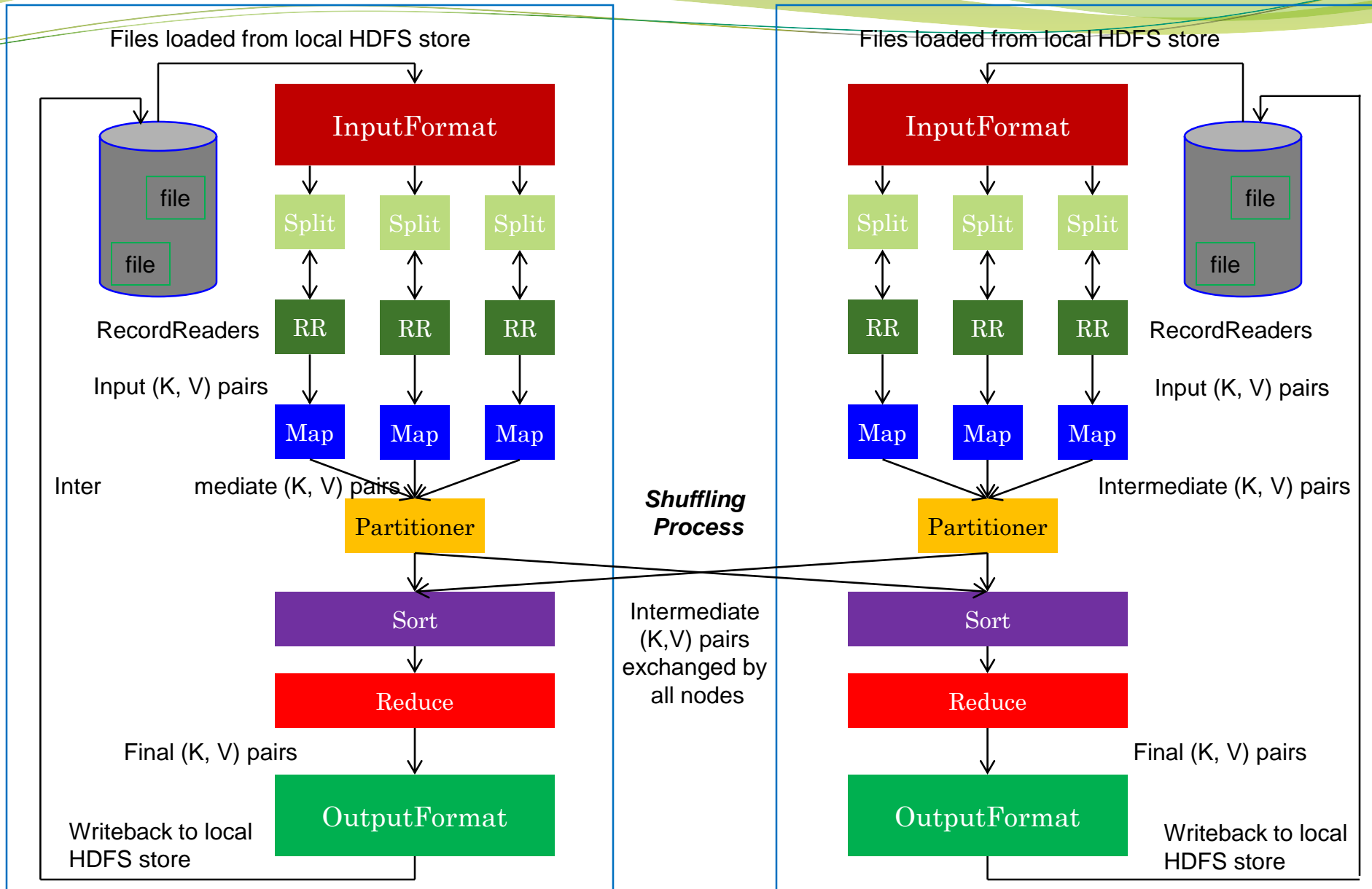


- MapReduce assumes a tree style network topology.
- Nodes are spread over different racks embraced in one or many data centers.
- A salient point is that the bandwidth between two nodes is dependent on their relative locations in the network topology.
- For example, nodes that are on the same rack will have higher bandwidth between them as opposed to nodes that are off-rack.

Hadoop MapReduce: A Closer Look

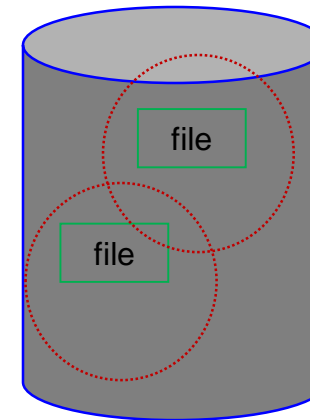
Node 1

Node 2



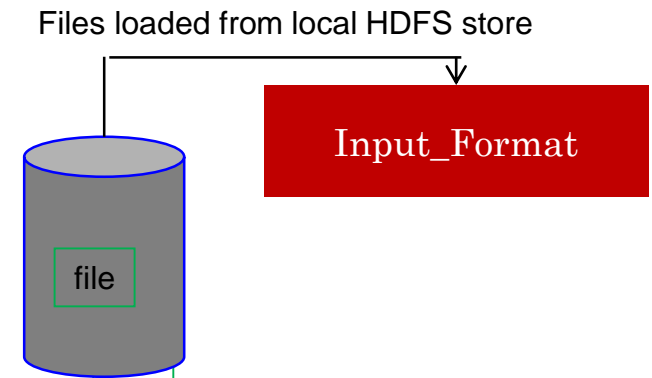
Input Files

- Input files are where the data for a **MapReduce** task is initially stored
- The input files typically reside in a distributed file system (e.g. HDFS)
- The format of input files is arbitrary
 - Line-based log files
 - Binary files
 - Multi-line input records
 - Or something else entirely



Input Format

- The format of input files is arbitrary
- How the input files are split up and read is defined by the Input_Format
- Input_Format is a class that does the following:
 - Selects the files that should be used for input
 - Defines the InputSplits that break a file
 - Provides a factory for Record_Reader objects that read the file



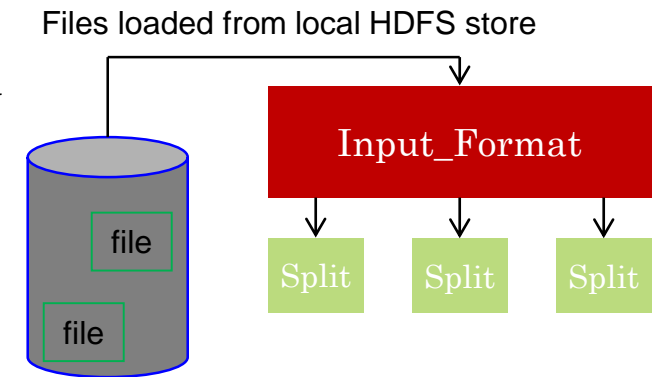
Input Format Type

- Several Input_Formats are provided with Hadoop:

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 (K, V) 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

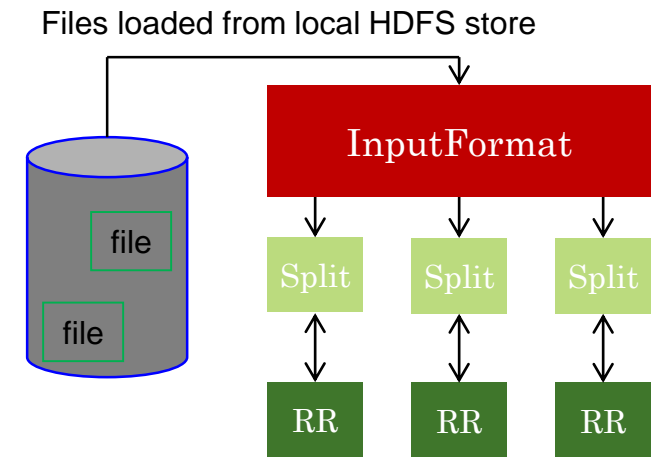
Input Splits

- An input split describes a unit of work that comprises a single map task in a MapReduce program.
- By default, the Input_Format breaks a file up into 64MB splits.
- By dividing the file into splits, we allow several map tasks to operate on a single file in parallel.
- If the file is very large, this can improve performance significantly through parallelism.
- Each map task corresponds to a single input split.



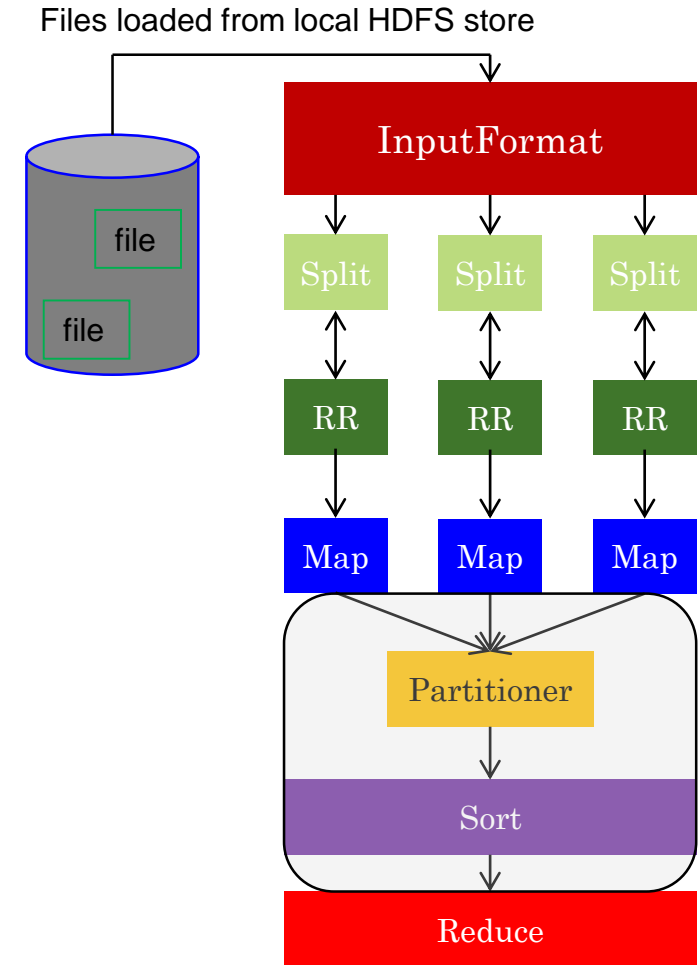
Record_Reader

- The input split defines a slice of work but does not describe how to access it.
- The Record_Reader class actually loads data from its source and converts it into (K, V) pairs suitable for reading by Mappers.
- The Record_Reader is invoked repeatedly on the input until the entire split is consumed.
- Each invocation of the Record_Reader leads to another call of the map function defined by the programmer.



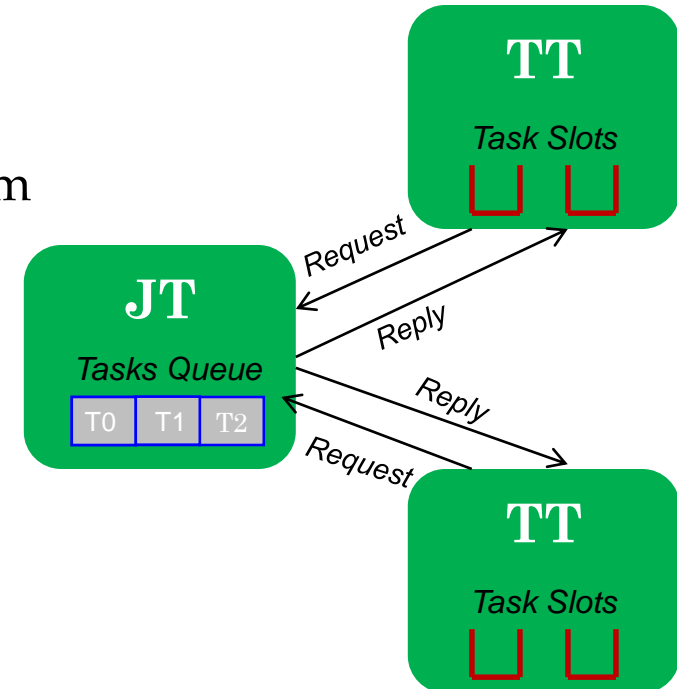
Mapper and Reducer

- The Mapper performs the user-defined work of the first phase of the MapReduce program
- A new instance of Mapper is created for each split
- The Reducer performs the user-defined work of the second phase of the MapReduce program
- A new instance of Reducer is created for each partition
- For each key in the partition assigned to a Reducer, the Reducer is called once.



Task Scheduling in MapReduce

- MapReduce adopts a master-slave architecture
- The master node in MapReduce is referred to as Job Tracker (JT)
- Each slave node in MapReduce is referred to as Task Tracker (TT)
- MapReduce adopts a pull scheduling strategy rather than a push one
 - I.e., JT does not push map and reduce tasks to TTs but rather TTs pull them by making pertaining requests



Job_Tracker

- Job_Tracker is the soul service for submitting and tracking MapReduce jobs in Hadoop.
- Job_Tracker performs following actions in Hadoop :
 - It accepts the MapReduce Jobs from client applications
 - Talks to Name_Node to determine data location
 - Locates available Task_Tracker Node
 - Submits the work to the chosen Task_Tracker Node

Task_Tracker

- A Task_Tracker node accepts map, reduce or shuffle operations from a Job_Tracker.
- Its configured with a set of slots, these indicate the number of tasks that it can accept.
- Job_Tracker seeks for the free slot to assign a job.
- Task_Tracker notifies the Job_Tracker about job success status.
- Task_Tracker also sends the heartbeat signals to the job tracker to ensure its availability, it also reports the no. of available free slots with it.

Map and Reduce Task Scheduling

- Every TT sends a heartbeat message periodically to JT encompassing a request for a map or a reduce task to run.
- I. Map Task Scheduling:
 - JT satisfies requests for map tasks via attempting to schedule mappers in the area of their input splits (i.e., it considers locality).
 - II. Reduce Task Scheduling:
 - However, JT simply assigns the next yet-to-run reduce task to a requesting TT regardless of TT's network location and its implied effect on the reducer's shuffle time (i.e., it does not consider locality).

Job Scheduling in MapReduce

- In MapReduce, an application is represented as *a job*.
- A job encompasses multiple map and reduce tasks.
- MapReduce in Hadoop comes with a choice of schedulers:
 - The default is the *FIFO scheduler* which schedules jobs in order of submission
 - There is also a multi-user scheduler called the *Fair scheduler* which aims to give every user a fair share of the cluster capacity over time.

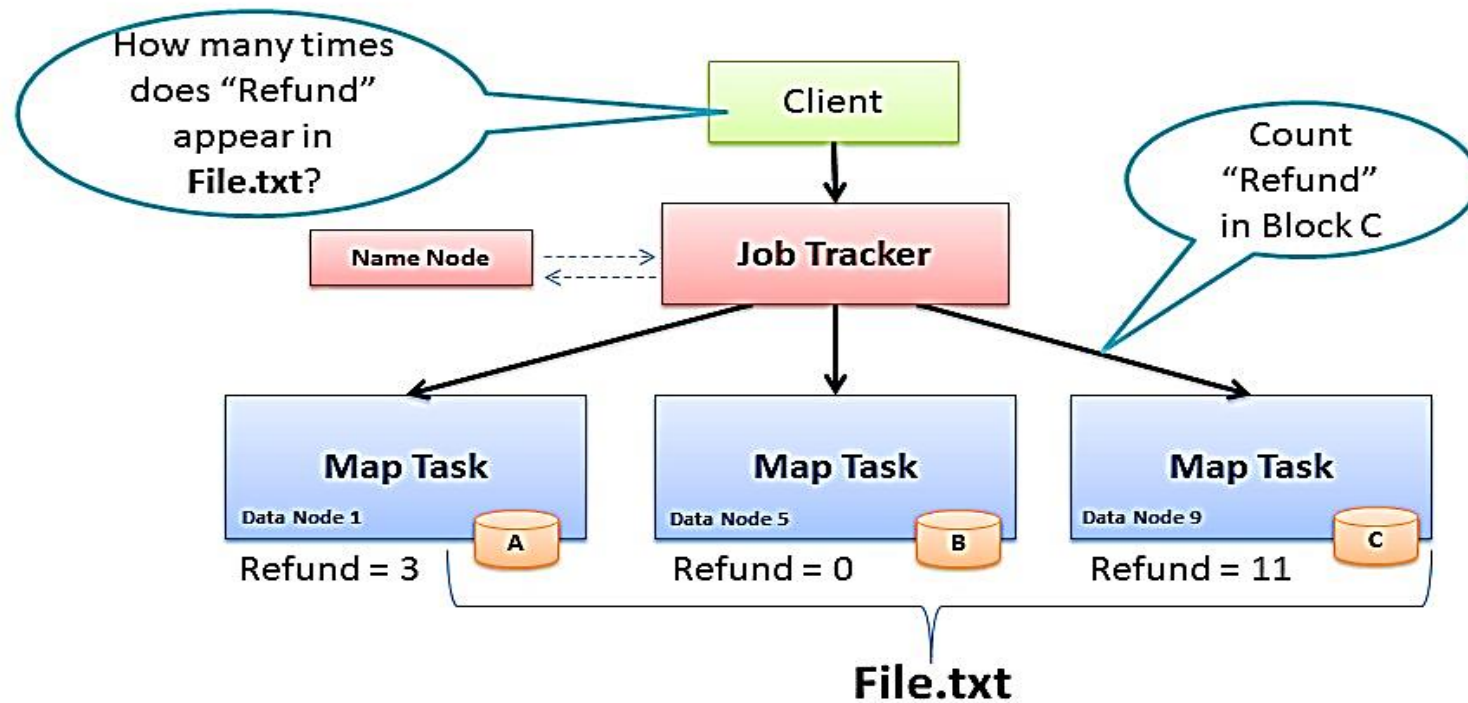
Fault Tolerance in Hadoop

- MapReduce can guide jobs toward a successful completion even when jobs are run on a large cluster where probability of failures increases
- The primary way that MapReduce achieves fault tolerance is through restarting tasks
- If a TT fails to communicate with JT for a period of time (by default, 1 minute in Hadoop), JT will assume that TT in question has crashed
 - If the job is still in the map phase, JT asks another TT to re-execute all Mappers that previously ran at the failed TT
 - If the job is in the reduce phase, JT asks another TT to re-execute all Reducers that were in progress on the failed TT

What Makes MapReduce Unique?

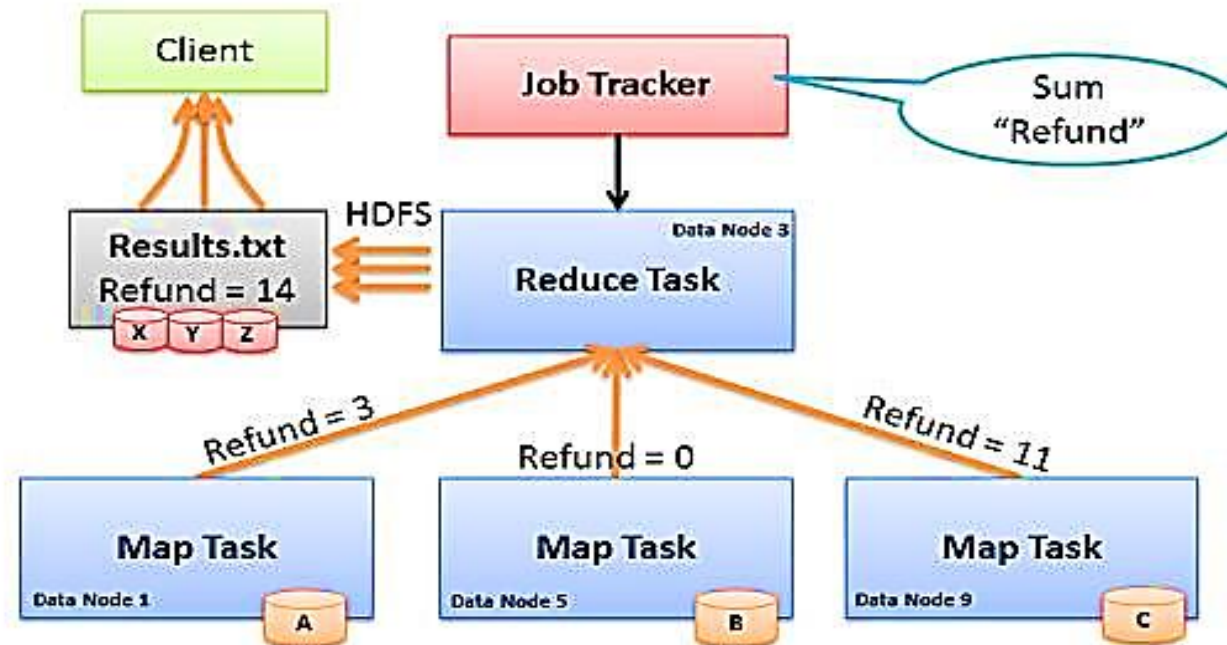
- MapReduce is characterized by:
 1. Its simplified programming model which allows the user to quickly write and test distributed systems
 2. Its efficient and automatic distribution of data and workload across machines
 3. Its flat scalability curve. Specifically, after a MapReduce program is written and functioning on 10 nodes, very little-if any- work is required for making that same program run on 1000 nodes

Map Task



- **Map:** "Run this computation on your local data"
- Job Tracker delivers Java code to Nodes with local data

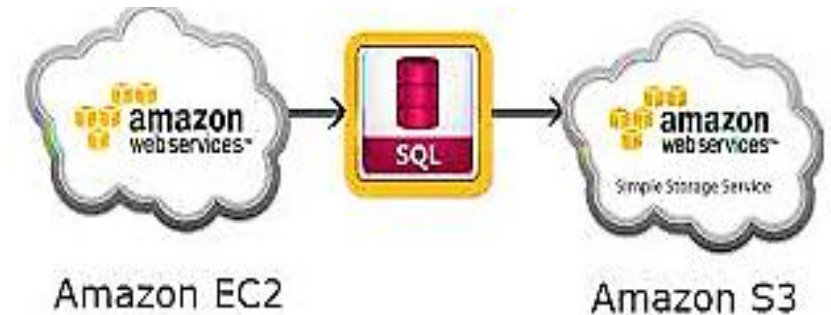
Reduce Task



- **Reduce:** “Run this computation across Map results”
- Map Tasks send output data to Reducer over the network
- Reduce Task data output written to and read from HDFS

MapReduce in the Cloud: AWS

- Provides a web-based interface and command-line tools for running Hadoop jobs on Amazon EC2 (Amazon Elastic Compute Cloud)
- Data stored in Amazon S3 (Amazon Simple Storage Service)
- Monitors job and shuts down machines after use
- Small extra charge on top of EC2 pricing
- An EC2 instance is like a remote computer running Windows or Linux and on which you can install whatever software you want, including a Web server running PHP code and a database server.
- Amazon S3 is just a storage service, typically used to store large binary files.



Amazon Elastic MapReduce



Sign Up

My Account / Console

English

AWS Products & Solutions

AWS Product Information



Developers

Support

Amazon EMR

- Amazon EMR Overview
- FAQs
- Pricing

Developer Resources

- EMR Training
- AWS Management Console
- Documentation
- Release Notes
- Sample Code & Libraries
- Developer Tools
- Articles & Tutorials
- Community Forum

Amazon Elastic MapReduce (Amazon EMR)

Amazon Elastic MapReduce (Amazon EMR) is a web service that enables businesses, researchers, data analysts, and developers to easily and cost-effectively process vast amounts of data. It utilizes a hosted Hadoop framework running on the web-scale infrastructure of Amazon Elastic Compute Cloud (Amazon EC2) and Amazon Simple Storage Service (Amazon S3).

Using Amazon Elastic MapReduce, you can instantly provision as much or as little capacity as you like to perform data-intensive tasks for applications such as web indexing, data mining, log file analysis, data warehousing, machine learning, financial analysis, scientific simulation, and bioinformatics research. Amazon Elastic MapReduce lets you focus on crunching or analyzing your data without having to worry about time-consuming set-up, management or tuning of Hadoop clusters or the compute capacity upon which they sit.

New to EMR? Check out these resources:

- Training
- Documentation
- FAQs
- EMR Forum

Easy to sign up,
pay only for what you
use

Sign Up Now


Featured Quote

razorfish.

"With Amazon Elastic MapReduce, there was no upfront investment in hardware, no hardware procurement delay, and no need to hire additional operations staff. Because of the flexibility of the platform, our first new online advertising campaign experienced a 500% increase in return on ad spend from a similar campaign a year before." Read the full case

Elastic MapReduce Workflow

Create a New Job Flow

Cancel 



Creating a job flow to process your data using Amazon Elastic MapReduce is simple and quick. Let's begin by giving your job flow a name and selecting its type. If you don't already have an application you'd like to run on Amazon Elastic MapReduce, samples are available to help you get started.

Job Flow Name*:

The name can be anything you like and doesn't need to be unique. It's a good idea to name the job flow something descriptive.

Type*: Streaming

A Streaming job flow allows you to write single-step mapper and reducer functions in a language other than java.

Custom Jar (advanced)

A custom jar on the other hand gives you more complete control over the function of Hadoop but must be a compiled java program. Amazon Elastic MapReduce supports custom jars developed for Hadoop 0.18.3.


Pig Program

Pig is a SQL-like language built on top of Hadoop. This option allows you to define a job flow that runs a Pig script, or set up a job flow that can be used interactively via SSH to run Pig commands.

Sample Applications

Select a sample application and click Continue. Subsequent forms will be filled with the necessary data to create a sample Job Flow.

Word count is a Python application that counts occurrences of each word in provided documents. [Learn more and view license](#)

Continue 

* Required field

Elastic MapReduce Workflow

Create a New Job Flow Cancel

DEFINE JOB FLOW **SPECIFY PARAMETERS** CONFIGURE EC2 INSTANCES REVIEW

Specify Mapper and Reducer functions to run within the Job Flow. The mapper and reducers may be either (i) class names referring to a mapper or reducer class in Hadoop or (ii) locations in Amazon S3. ([Click Here](#) for a list of available tools to help you upload and download files from Amazon S3.) The format for specifying a location in Amazon S3 is bucket_name/path_name. The location should point to an executable program, for example a python program. Extra arguments are passed to the Hadoop streaming program and can specify things such as additional files to be loaded into the distributed cache.

Input Location*:
The URL of the Amazon S3 Bucket that contains the input files.

Output Location*:
The URL of the Amazon S3 Bucket to store output files. Should be unique.

Mapper*:
The mapper Amazon s3 location or streaming command to execute.

Reducer*:
The reducer Amazon s3 location or streaming command to execute.

Extra Args:

[< Back](#) Continue * Required field

Elastic MapReduce Workflow

Create a New Job Flow Cancel X

DEFINE JOB FLOW SPECIFY PARAMETERS **CONFIGURE EC2 INSTANCES** REVIEW

Enter the number and type of EC2 instances you'd like to run your job flow on.


Number of Instances*:

The number of EC2 instances to run in your Hadoop cluster.
If you wish to run more than 20 instances, please complete the [limit request form](#).

Type of Instance*:

The type of EC2 instances to run in your Hadoop cluster ([learn more about instance types](#)).

[Show advanced options](#)

[Back](#) **Continue**  * Required field

Elastic MapReduce Workflow

The screenshot displays the Amazon Elastic MapReduce console interface. At the top, the Amazon Web Services logo is visible on the left, and navigation links for 'About AWS', 'Products', 'Solutions', 'Resources', 'Support', and 'Your Account' are in the center. On the right, there are links for 'Contact Us' and 'Create an AWS Account'. Below this, a breadcrumb trail shows 'Home > Resources > AWS Management Console BETA > Amazon Elastic MapReduce'. A user greeting 'Welcome, Rad Lab' and 'Settings | Sign Out' are also present.

The main content area is titled 'Your Elastic MapReduce Job Flows'. It includes a region selector set to 'US-East', buttons for 'Create New Job Flow' and 'Terminate', and utility buttons for 'Show/Hide', 'Refresh', and 'Help'. A viewing filter is set to 'All'. A table below shows one job flow:

Name	State	Creation Date	Elapsed Time	Normalized Instance Hours
My Job Flow	STARTING	2009-08-19 14:50 PDT	0 hours 0 minutes	0

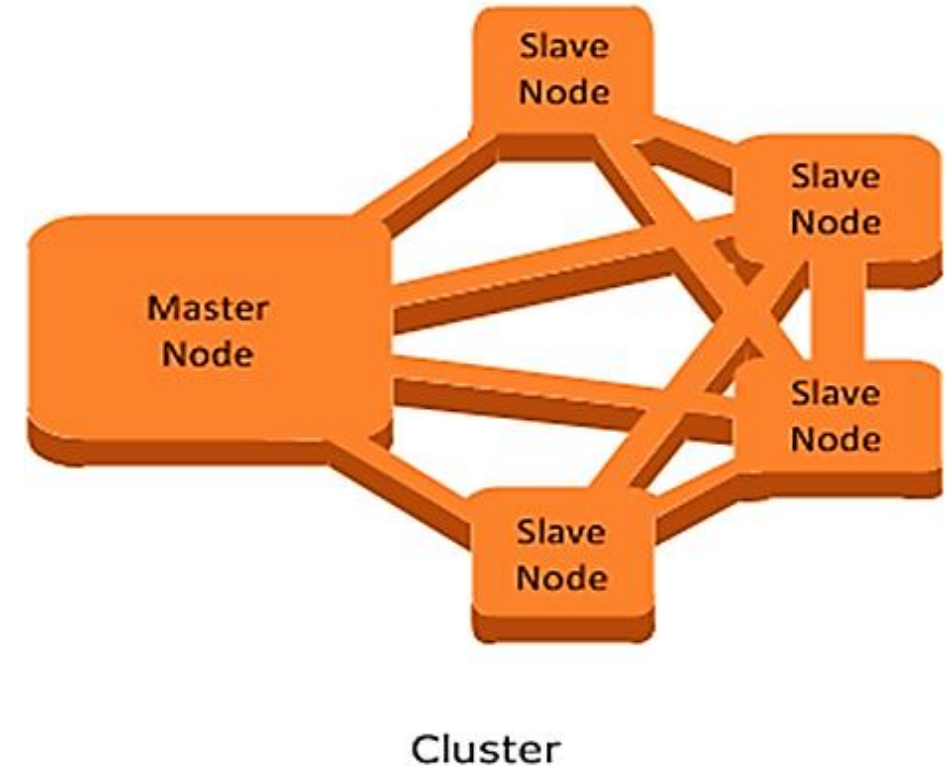
Below the table, a summary for the selected job flow is shown:

- 1 Job Flow selected**
- Id:** j-46JL0YQ7ZPH1
- Creation Date:** 2009-08-19 14:50 PDT
- Name:** My Job Flow
- Start Date:** -
- State:** STARTING
- End Date:** -
- Last State Change Reason:** Starting instances
- Availability Zone:** us-east-1b
- Instance Count:** 4

At the bottom of the console, there is a footer with copyright information: '© 2008 - 2009, Amazon Web Services LLC or its affiliates. All right reserved.' and links for 'Feedback', 'Support', 'Privacy Policy', and 'Terms of Use'.

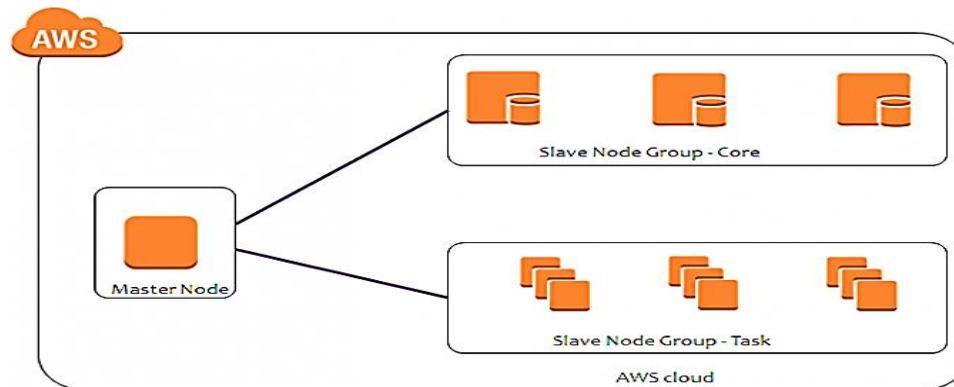
The central component of Amazon EMR

- The central component of Amazon **EMR** is the **cluster**.
- A **cluster** is a collection of Amazon Elastic Compute Cloud (Amazon **EC2**) instances.
- **Each instance** in the cluster is called a **node**.
- **Each node** has a role within the cluster, referred to as the node type. Amazon EMR also installs different software components on each node type, giving each node a role in a distributed application like Apache Hadoop.



The node types in Amazon EMR are as follows:

- **Master node:** A node that manages the cluster by running software components to coordinate the distribution of data and tasks among other nodes—collectively referred to as slave nodes—for processing. The master node tracks the status of tasks and monitors the health of the cluster.
- **Core node:** A slave node with software components that run tasks and store data in the Hadoop Distributed File System (HDFS) on your cluster.
- **Task node:** A slave node with software components that only run tasks. Task nodes are optional.



... Thank you ...

