

# Big Data & Hadoop

Module : 2 HDFS Architecture , Hadoop Configurations & Data  
Loading



# Topics

## MODULE 1

BIG DATA AND HADOOP  
INTRODUCTION

## MODULE 2

HDFS ARCHITECTURE,  
HADOOP CONFIGURATIONS  
& DATA LOADING

## MODULE 3

INTRODUCTION TO  
MAP REDUCE

## MODULE 4

ADVANCED MAP  
REDUCE CONCEPTS

## MODULE 5

INTRODUCTION TO PIG  
AND ADVANCE PIG

## MODULE 6

INTRODUCTION TO HIVE  
AND ADVANCE HIVE

## MODULE 7

INTRODUCTION TO  
HBASE AND ADVANCED  
HBASE

## MODULE 8

SQOOP AND FLUME

## MODULE 9

BASIC OOZIE AND  
OOZIE CONFIGURATION

## MODULE 10

PROJECT DISCUSSIONS



# Session Objectives

---

**This Session helps you to understand :**

- Hadoop Daemons
- Block Replication and Replication Factor
- Rack Awareness in HDFS
- Read and Write Operation in HDFS
- HDFS Architecture
- Hands –on HDFS
- YARN concept
- Basic MapReduce Execution Flow



# Hadoop Daemons

## **NameNode: (Single instance)**

- Runs in Master Node
- Manages the file metadata
- Client connects to NameNode for initiating all reads and writes in HDFS.

## **Data Node: (Multiple instances/ one in each slave node)**

- Runs in all slave nodes
- Data files are stored in the DataNodes in a distributed manner
- Reports to NameNode, periodically with lists of blocks they store

## **Secondary NameNode:**

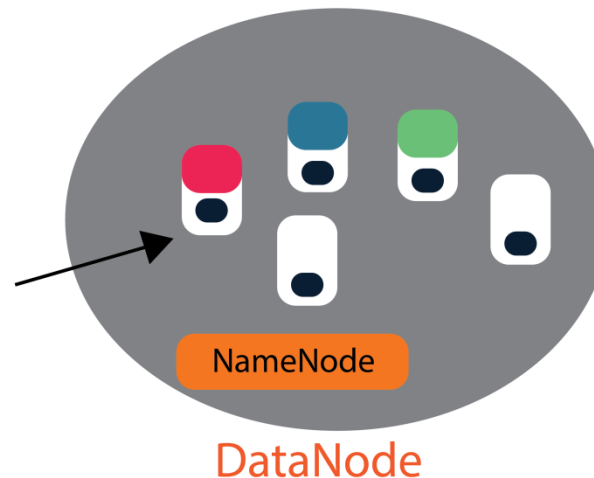
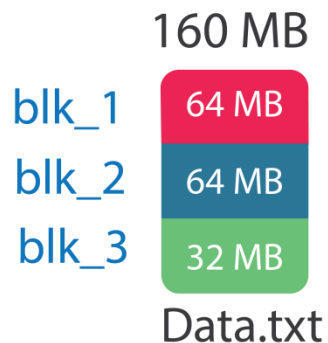
- Acts as back up node to name node
- It periodically copies metadata into its own storage



# Blocks

- HDFS stores the data in terms of blocks, the Data/File is broken down into multiple blocks when stored
- The default size of HDFS block is 64MB
- The files are split into 64MB blocks and then stored into the hadoop file system
- The hadoop application is responsible for distributing the data blocks across multiple nodes
- Excellent for storage of large files

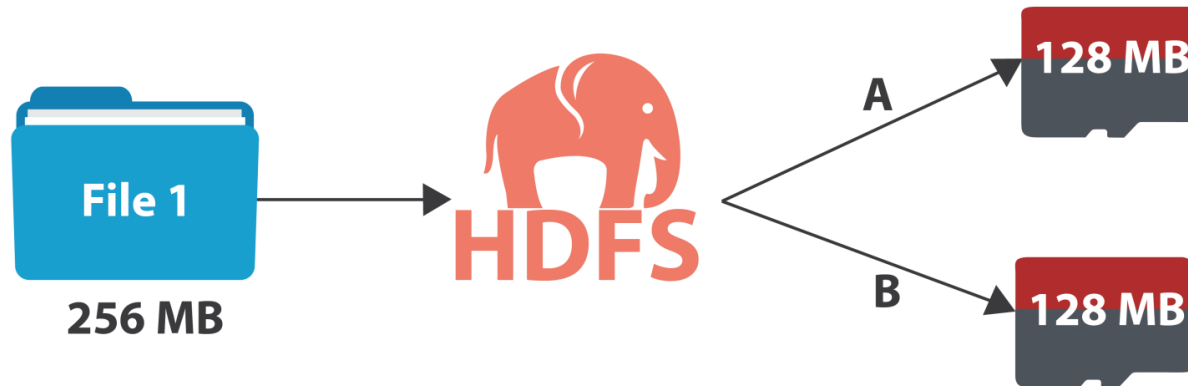
## HDFS



# Blocks in Hadoop 2.0

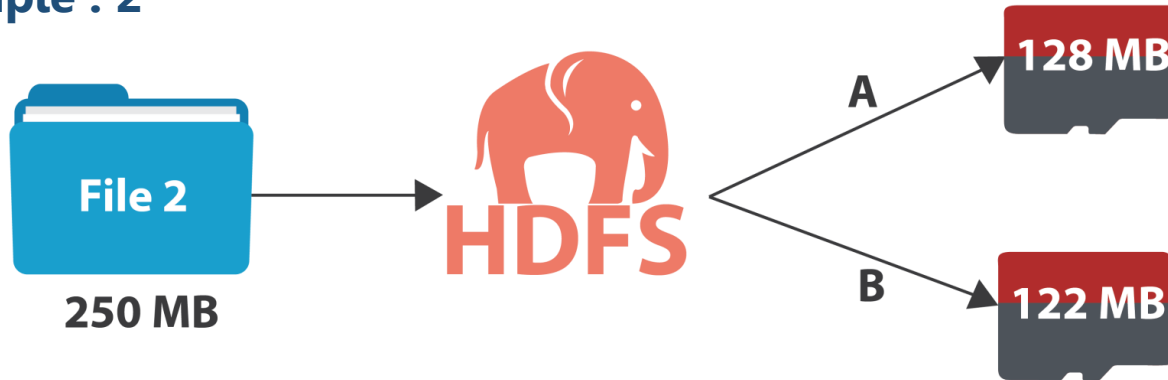
- Default size of the blocks in Hadoop 2.0 is 128 MB
- It is excellent for the storage of large files as the minimum size of file read & write is 128MB.

## Example : 1

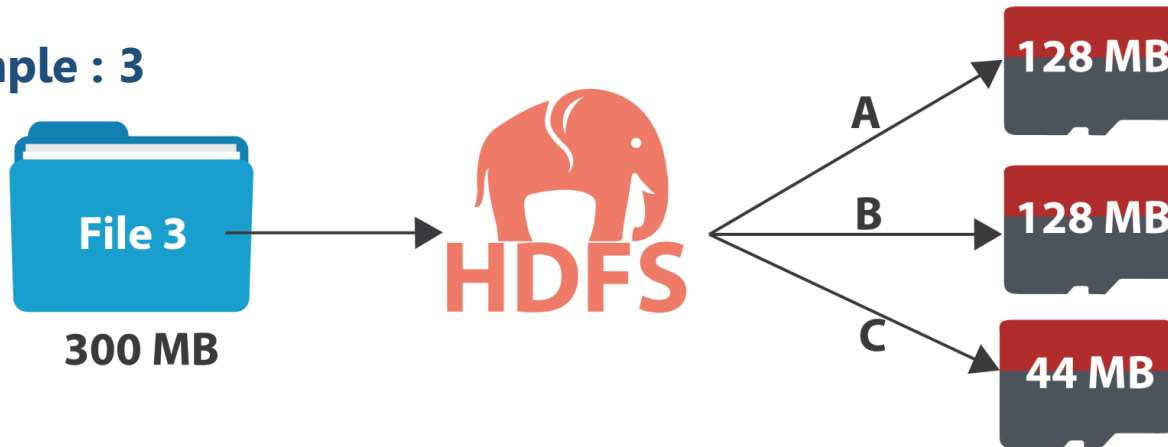


# Blocks in Hadoop 2.0(Cont'd)

## Example : 2



## Example : 3



# Block Replication and Replication Factor

## Block Replication

- Hadoop has a mechanism of Block replication among the Data Nodes.
- This means that the data that is available in one machine would get replicated and available on other machine as well.
- This will assure data availability all the time.
- Block replication provides redundancy and fault tolerance

## Replication Factor:

- The default replication factor is 3.





# Computer Racks

## Computer Racks :

- A computer racks is a metal frame used to hold various hardware devices such as servers, hard disk drives, modems and other electronic equipment.
- While racks come in many different shapes and sizes, the standard (traditional) size rack is 19-inches wide.



# Rack Awareness in HDFS

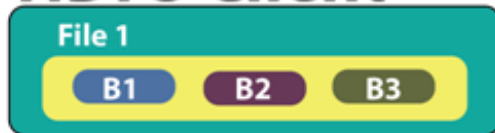
---

- Rack consists of sequence of Data Nodes.
- Single Hadoop cluster may have one or more Racks depending on the size of the cluster.
- HDFS stores blocks on the cluster in a rack aware fashion i.e. one block on one rack and the other two blocks on other rack.
- The HDFS is aware of the location of each server in the rack; this is referred to a rack awareness.

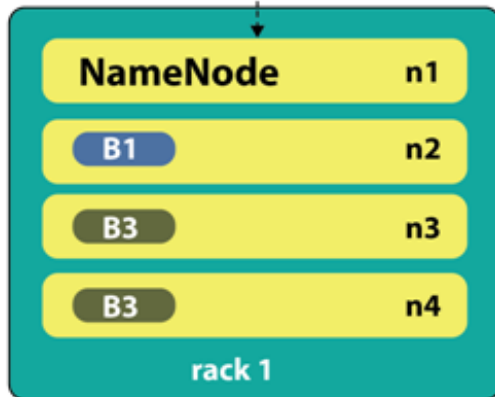


# Rack Awareness in HDFS(Cont'd)

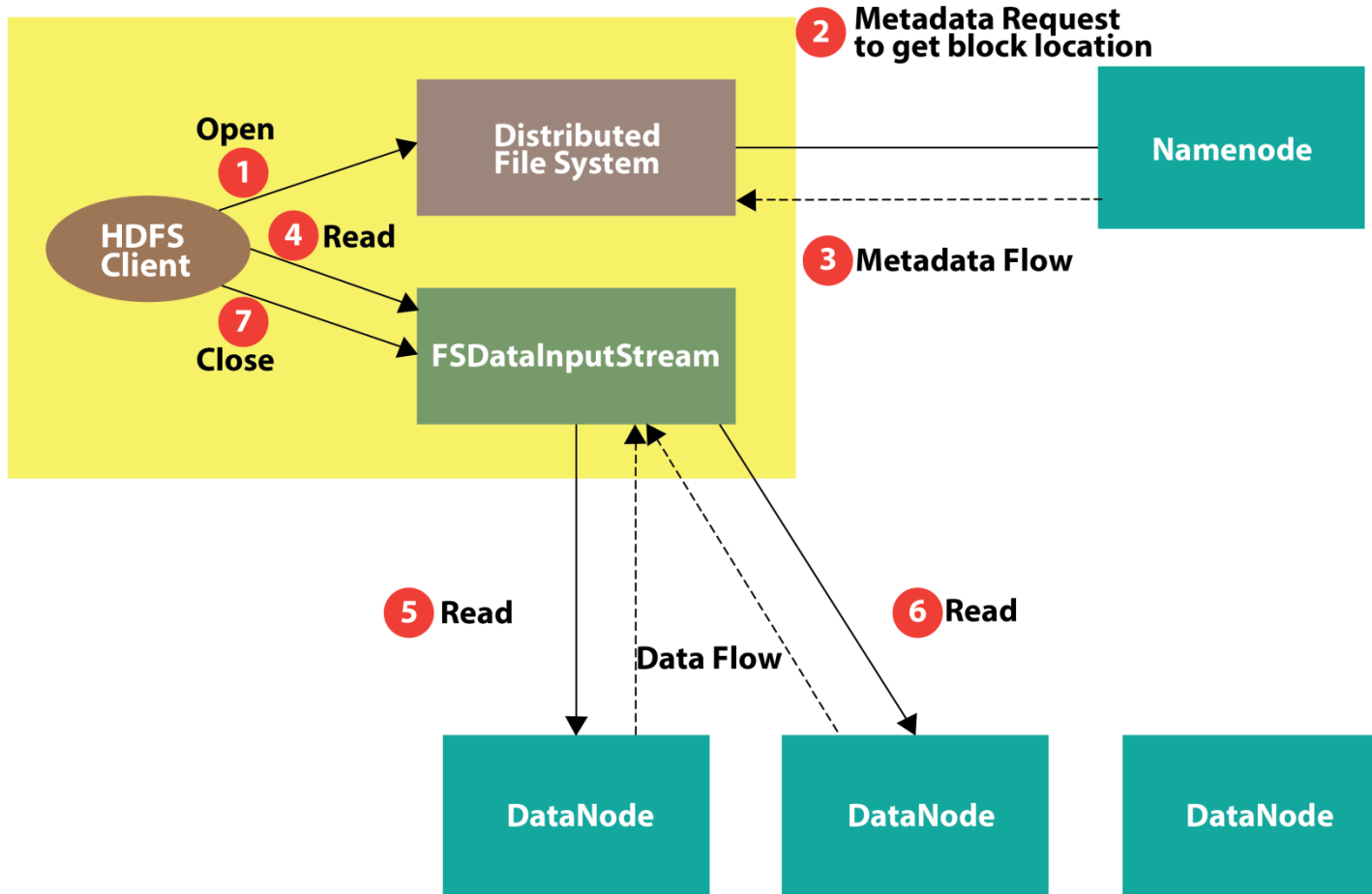
## HDFS Client



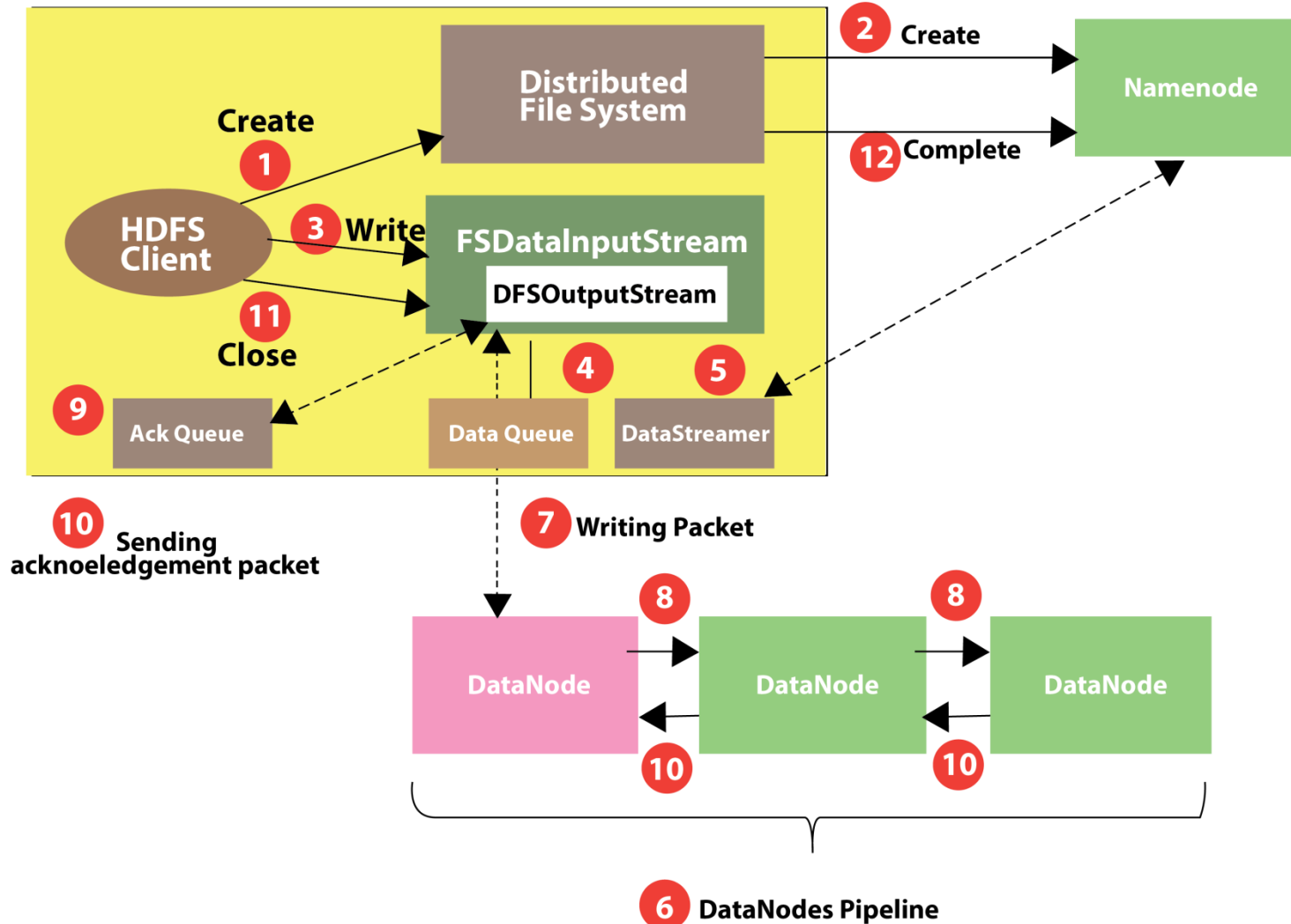
complete



# File Read Operation in HDFS

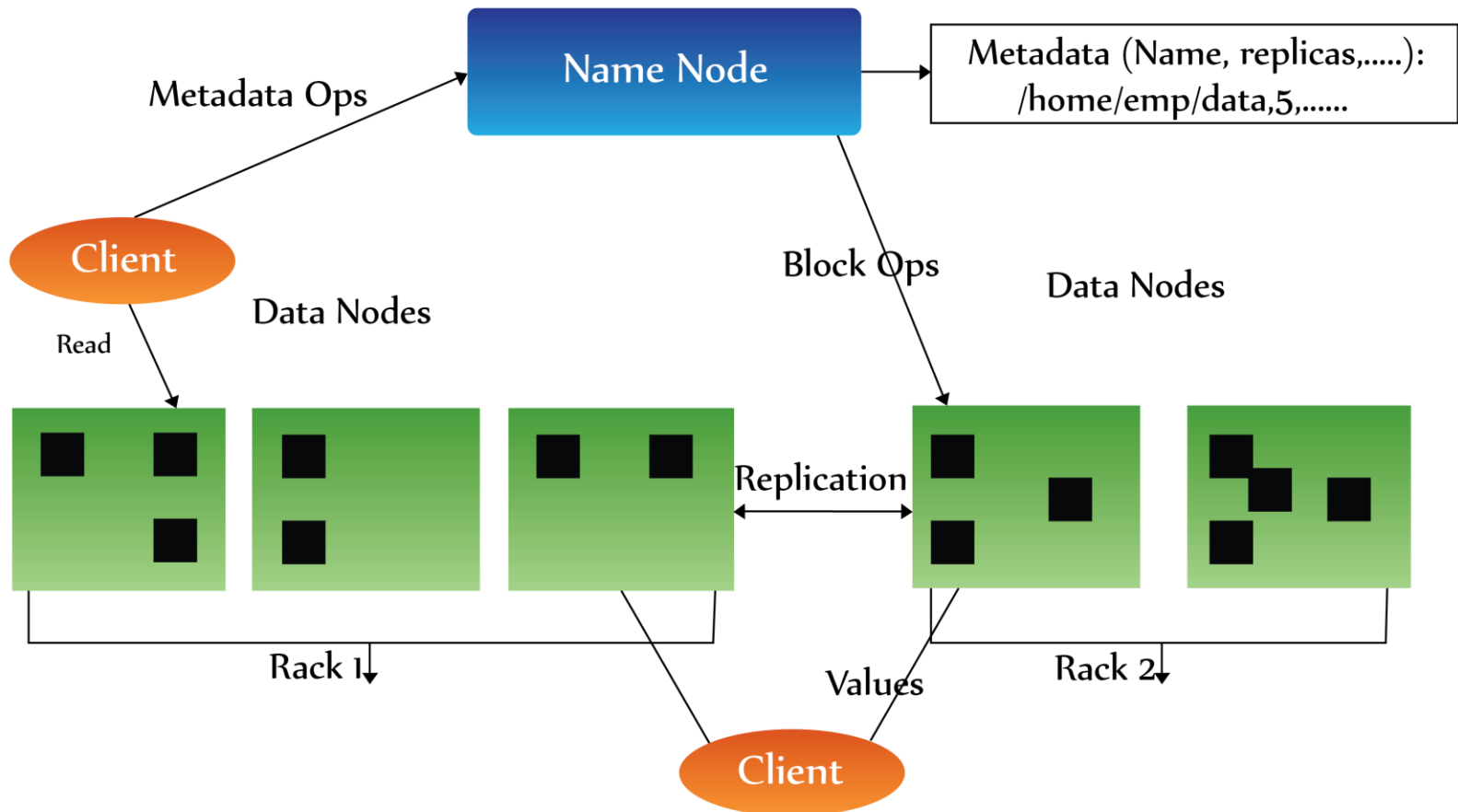


# File Write Operation in HDFS



# HDFS Architecture

## HDFS Architecture



# Hadoop Configuration Files

Configuration Filenames	Description of Log Files
hadoop-env.sh	Environment variable that are used in the scripts to run Hadoop
core-site	Configuration setting for Hadoop Core such as I/O settings that are common to HDFS and MapReuce
hdfs-site.xml	HDFS Configuration settings for HDFS daemons, the NameNode, the secondary NameNode and the data nodes
mapred-site.xml	MapReduce specific Configuration settings like Job History Server
yarn-site.xml	Configuration settings for Shuffle Mechanism with respect to YARN implementation.
masters	A list of machines (one per line) that each run a secondary NameNode
slaves	A list of machines (one per line) that each run a slave machine running DataNode and a NodeManager daemons



# Hadoop Configuration Files(Cont'd)

## conf/core-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--core-site.xml -->
<configuration>
<property>
<name>fs.default.name</name>
<value>hdfs://localhost:9000</value>
</property>
</configuration>
```





# Hadoop Configuration Files(Cont'd)

## conf/hdfs-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--hdfs-site.xml -->
<configuration> <property>
<name>dfs.replication</name>
<value>1</value> </property>
<property> <name>dfs.permissions</name>
<value>>false</value> </property>
<property> <name>dfs.namenode.name.dir</name>
<value>/home/user/hadoop-2.2.0/hadoop2_data/hdfs/namenode </value>
</property>
<property>
<name>dfs.datanode.data.dir</name>
<value>/home/user/hadoop-2.2.0/hadoop2_data/hdfs/datanode</value>
</property> </configuration>
```



# Hadoop Configuration Files(Cont'd)

## conf/mapred-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--mapred-site.xml -->
<configuration>
<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
</property>
</configuration>
```



# Hadoop Configuration Files(Cont'd)

## conf/yarn-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--yarn-site.xml -->
<configuration>
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>
<property>
<name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
<value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
</configuration>
```



# Hadoop Copy Options

- **put** : Copy single source, or multiple sources from local file system to the destination file system. Also reads input from stdin and writes to destination file system.

```
hadoop dfs -put weather.txt hdfs://<target Namenode>
```

```
hadoop dfs -copyFromLocal weather.txt hdfs://<target Namenode>
```

- **distcp** : Distributed copy to move data between clusters, used for backup and recovery.

```
hadoop distcp hdfs://<source NN> hdfs://<target NN>
```



# Demo - HDFS Commands

## Checking HDFS Commands

```
user@ubuntuvm: ~
user@ubuntuvm:~$ hdfs
Usage: hdfs [--config confdir] COMMAND
    where COMMAND is one of:
    dfs                run a filesystem command on the file systems supported in Hadoop.
    namenode -format   format the DFS filesystem
    secondarynamenode run the DFS secondary namenode
    namenode           run the DFS namenode
    journalnode        run the DFS journalnode
    zkfc               run the ZK Failover Controller daemon
    datanode           run a DFS datanode
    dfsadmin           run a DFS admin client
    haadmin            run a DFS HA admin client
    fsck              run a DFS filesystem checking utility
    balancer          run a cluster balancing utility
    jmxget            get JMX exported values from NameNode or DataNode.
    oiv              apply the offline fsimage viewer to an fsimage
    oev              apply the offline edits viewer to an edits file
    fetchdt          fetch a delegation token from the NameNode
    getconf           get config values from configuration
    groups            get the groups which users belong to
    snapshotDiff      diff two snapshots of a directory or diff the
                        current directory contents with a snapshot
                        Use -help to see options
    lsSnapshottableDir list all snapshottable dirs owned by the current user

    portmap          run a portmap service
    nfs3             run an NFS version 3 gateway

Most commands print help when invoked w/o parameters.
user@ubuntuvm:~$
```



# Demo - HDFS Commands(Cont'd)

## Checking HDFS Commands

```
user@ubuntuvm: ~
user@ubuntuvm:~$ hadoop
Usage: hadoop [--config confdir] COMMAND
      where COMMAND is one of:
      fs                run a generic filesystem user client
      version           print the version
      jar <jar>         run a jar file
      checknative [-a|-h] check native hadoop and compression libraries availability
      distcp <srcurl> <desturl> copy file or directories recursively
      archive -archiveName NAME -p <parent path> <src>* <dest> create a hadoop archive
      classpath         prints the class path needed to get the
                        Hadoop jar and the required libraries
      daemonlog         get/set the log level for each daemon
or
      CLASSNAME         run the class named CLASSNAME

Most commands print help when invoked w/o parameters.
user@ubuntuvm:~$
```



# Demo - HDFS Commands(Cont'd)

## Listing the HDFS Directories

```

user@ubuntuvm: ~
user@ubuntuvm:~$ hadoop fs -ls /
Found 6 items
drwxr-xr-x  - user supergroup    0 2014-10-15 19:34 /inputs
drwxr-xr-x  - user supergroup    0 2014-09-04 14:36 /mr_op
drwxr-xr-x  - user supergroup    0 2014-09-18 07:59 /op_dir
drwx----- - user supergroup    0 2014-10-15 19:49 /tmp
drwxr-xr-x  - user supergroup    0 2014-09-04 08:40 /user
drwxr-xr-x  - user supergroup    0 2014-09-01 22:57 /vishal
user@ubuntuvm:~$

```

## sbin Directory of Hadoop Instalation

```

user@ubuntuvm: ~/hadoop
user@ubuntuvm:~/hadoop$ ls sbin/
distribute-exclude.sh  mr-jobhistory-daemon.sh  start-dfs.cmd  stop-all.sh  stop-yarn.sh
hadoop-daemon.sh      refresh-namenodes.sh    start-dfs.sh   stop-balancer.sh  yarn-daemon.sh
hadoop-daemons.sh    slaves.sh                start-secure-dns.sh  stop-dfs.cmd     yarn-daemons.sh
hdfs-config.cmd       start-all.cmd           start-yarn.cmd  stop-dfs.sh
hdfs-config.sh        start-all.sh            start-yarn.sh   stop-secure-dns.sh
httpfs.sh             start-balancer.sh        stop-all.cmd   stop-yarn.cmd
user@ubuntuvm:~/hadoop$

```



# HDFS High Availability

## Problem Statement:

- If the NameNode process or machine fails, then the entire cluster will not be available in Hadoop 1.x until either the NameNode is rebooted.

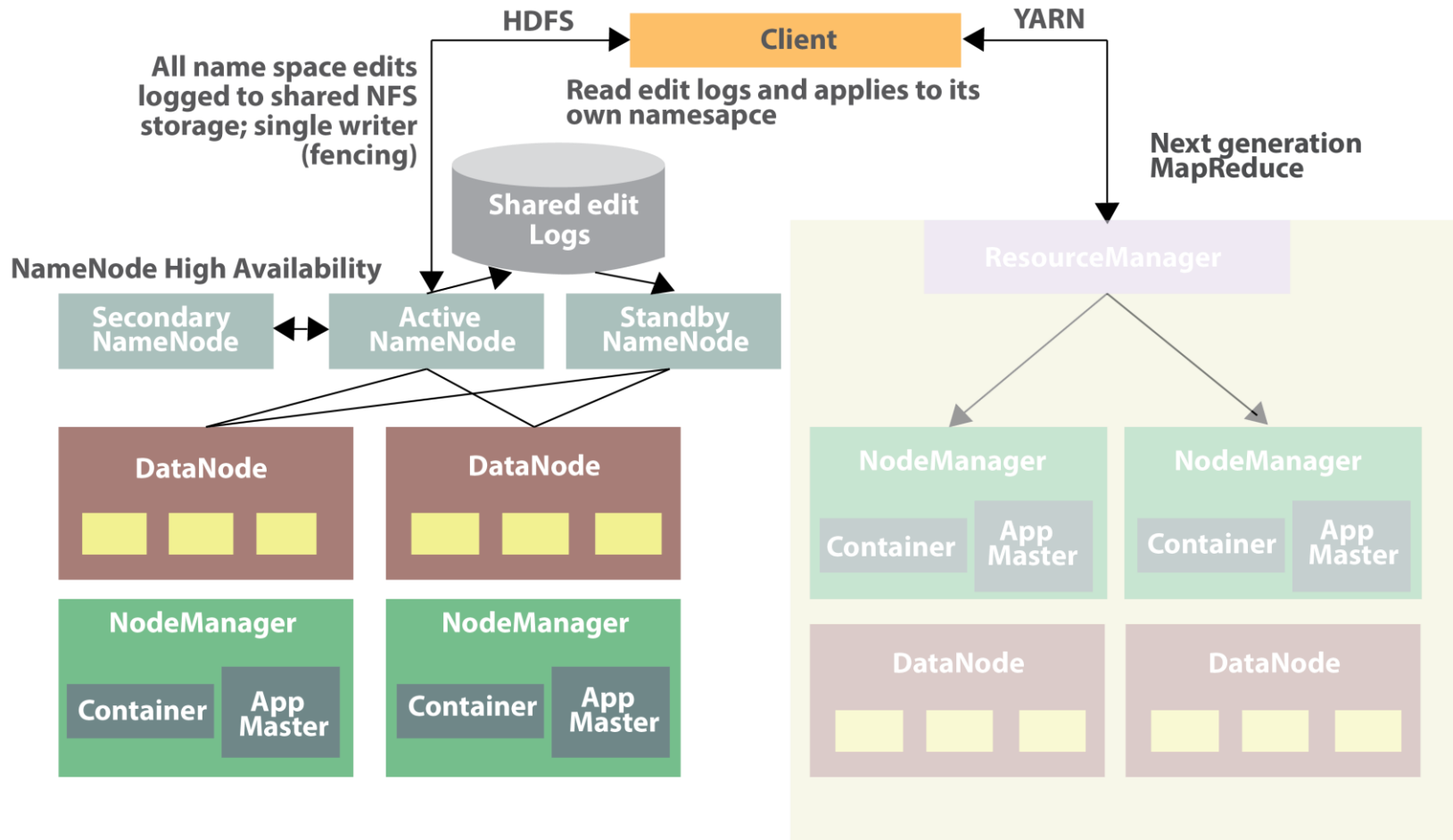
## Solution :

- This was solved in Hadoop 2.x by **HDFS High Availability (HDFS HA)**





# HDFS High Availability(Cont'd)



# HDFS Federation

## Problem Statement:

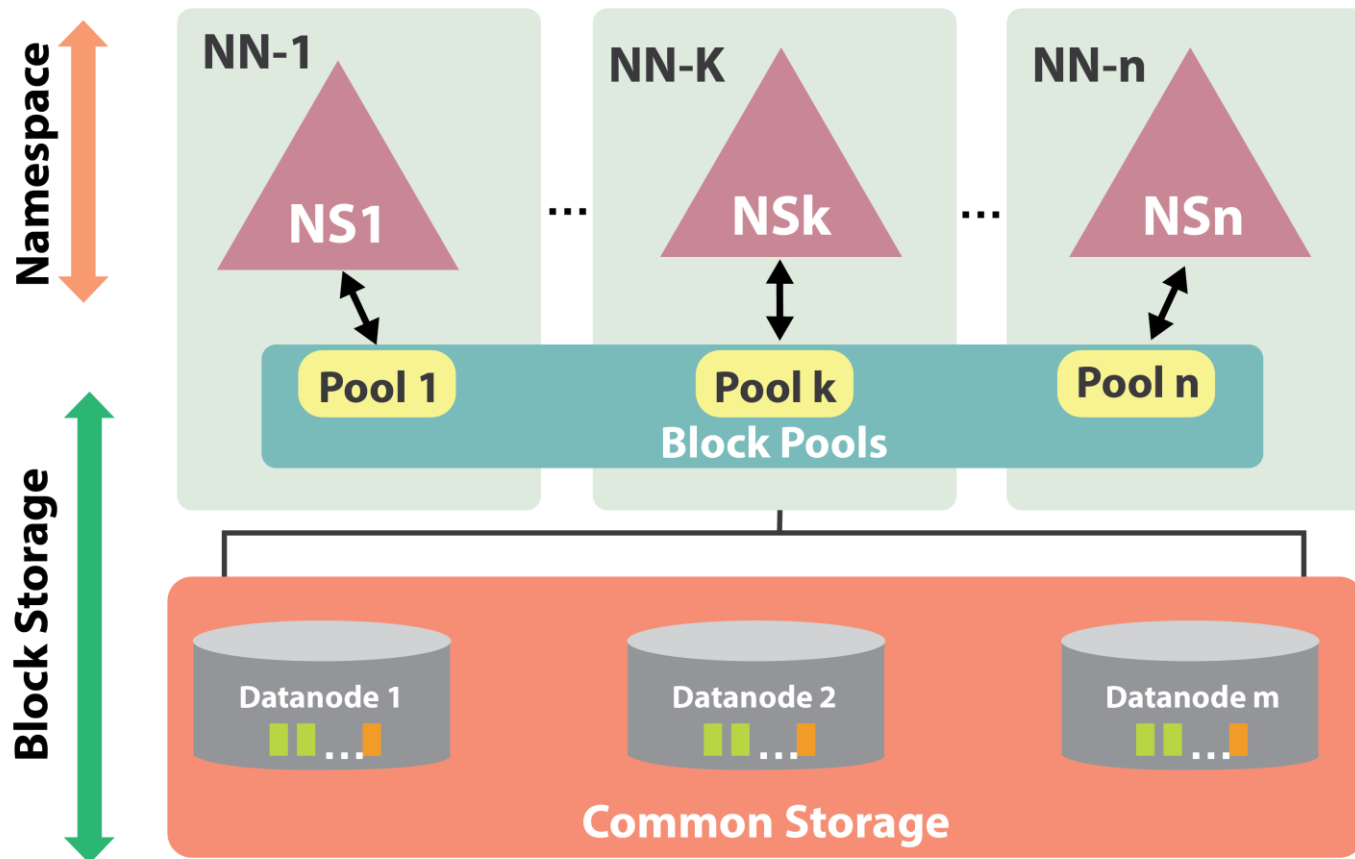
- The name node can store a limited amount of meta data(which is usually 64 GB in Hadoop 1.x)
- In Hadoop 1.x, the above limitation meant that the maximum amount of meta data that the Namenode can store is limited to Admin node's RAM
- Also, number of nodes that a Hadoop 1.x cluster could manage was 4000 node / slaves / commodity machines

## Solution:

- This was solved in Hadoop 2.x by **HDFS Federation**



# HDFS Federation(Cont'd)



# Key Benefits of Federation

---

- NameSpace Scalability – Due to the ScaleOut nature of HDFS Federation; instead of having only one name-node in a cluster; we can have multiple name-nodes. This results in horizontal scalability; wherein it is easy to carry out intensive operations.
- Greater Output – Due to greater scalability; via multiple name-nodes. The file system operations have a greater rate of output.
- Isolation – In large scale systems; the complete data is segregated by either function or vertical.
- New Implementations – As Federation opens up the architecture it is very friendly to customized applications & use-cases on a HDFS cluster.



# Basic Mapper - Reducer Concepts

## Mappers:

- Mappers are java programs conforming to Google's Map Reduce algorithm framework. These programs run on each of the blocks of big data file saved on the cluster
- The mapper's job is to process the input data. Generally the input data is in the form of file or directory and is stored in HDFS.
- The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.



# Basic Mapper - Reducer Concepts(Cont'd)

## Reducers:

- Similar to Mappers, Reducers are also java programs conforming to Google's Map Reduce algorithm framework. They are aggregate functions which are supposed to run on the outputs coming out of mappers
- The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.



## Problem statement :

- JobTracker runs on single machine doing several task like Resource management, Job scheduling, task scheduling and Monitoring etc although there are so many machines (DataNode) available, they are not getting used. This limits scalability.

## Solution :

- This was addressed in Hadoop 2.x with a new framework called YARN(This is an acronym for Yet Another Resource Negotiator)



## Problem statement :

- JobTracker runs on single machine doing several task like Resource management, Job scheduling, task scheduling and Monitoring etc although there are so many machines (DataNode) available, they are not getting used. This limits scalability.

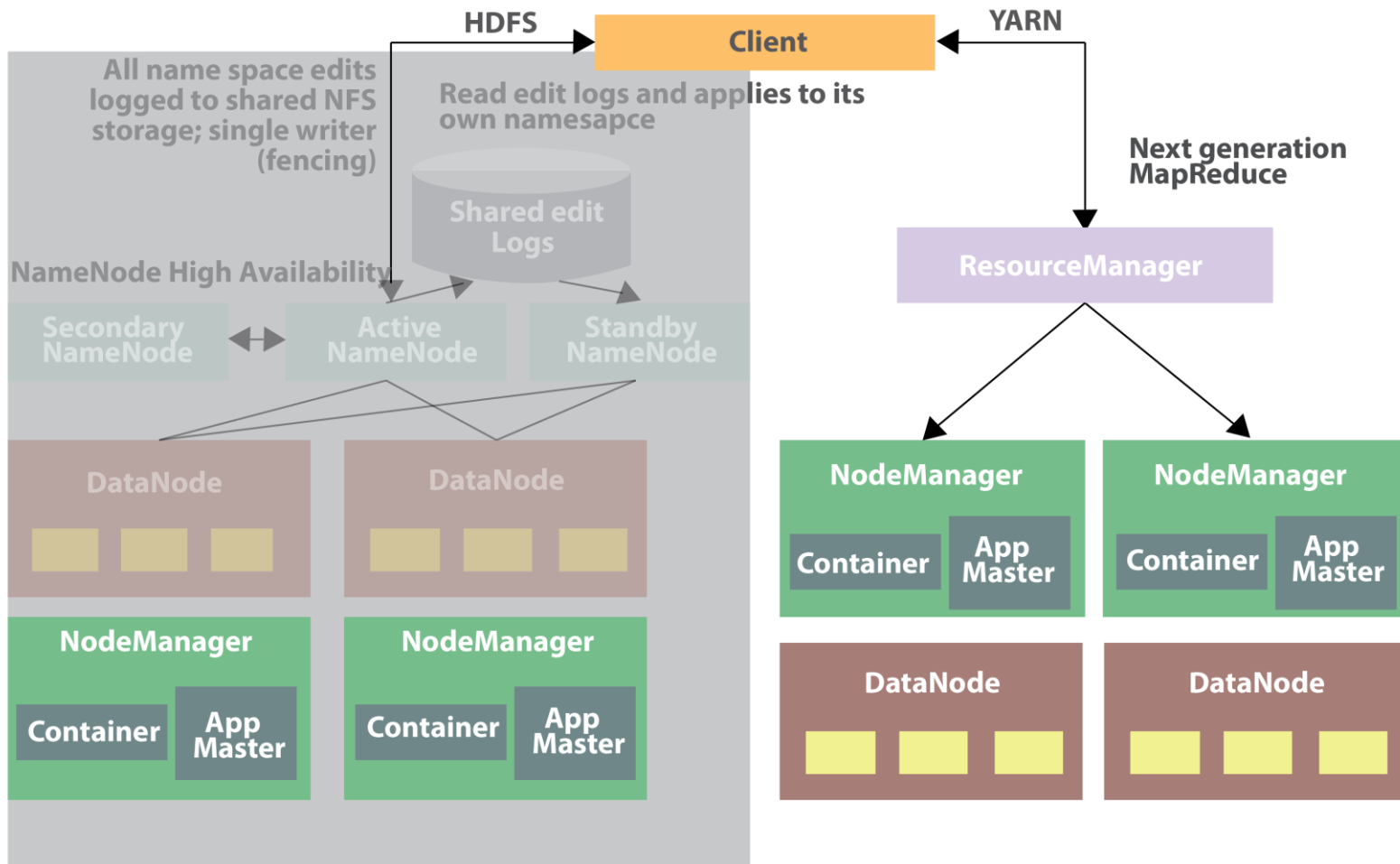
## Solution :

- This was addressed in Hadoop 2.x with a new framework called YARN(This is an acronym for Yet Another Resource Negotiator)

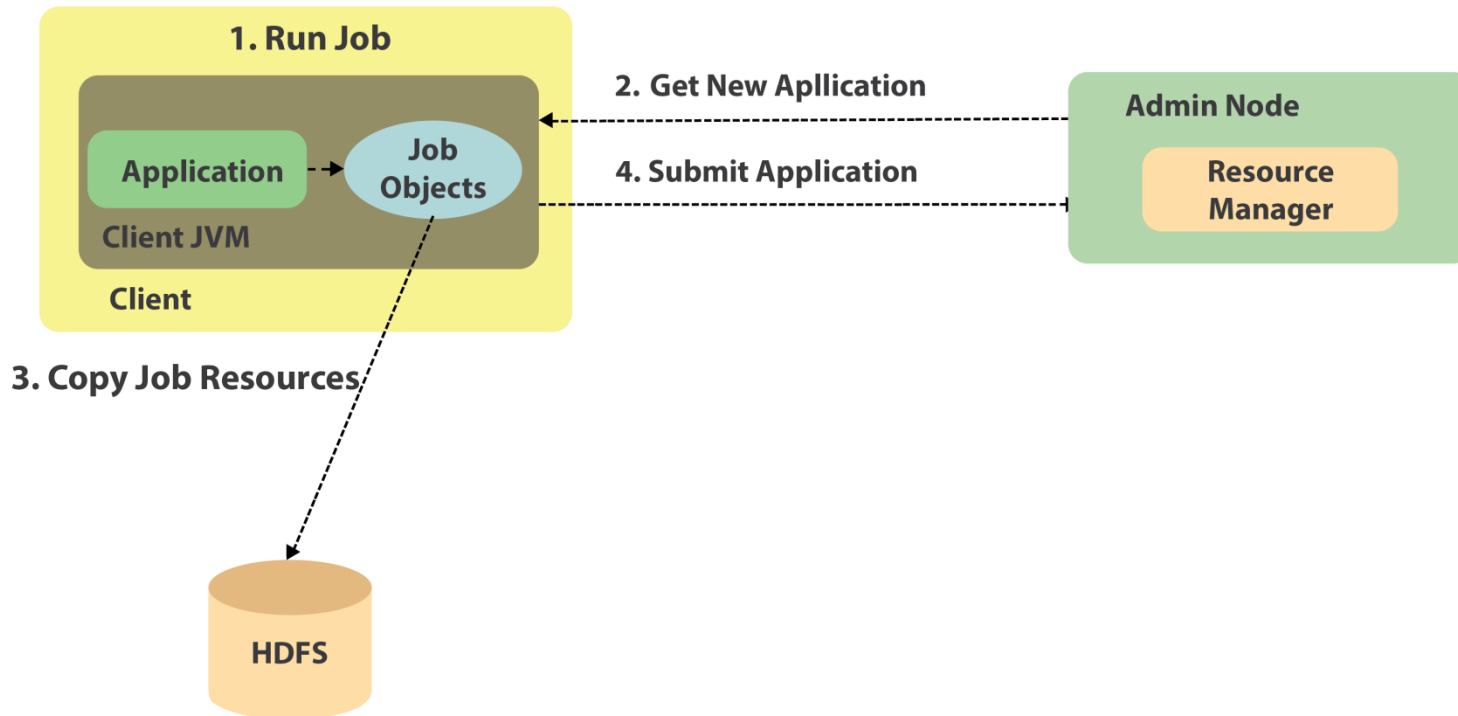




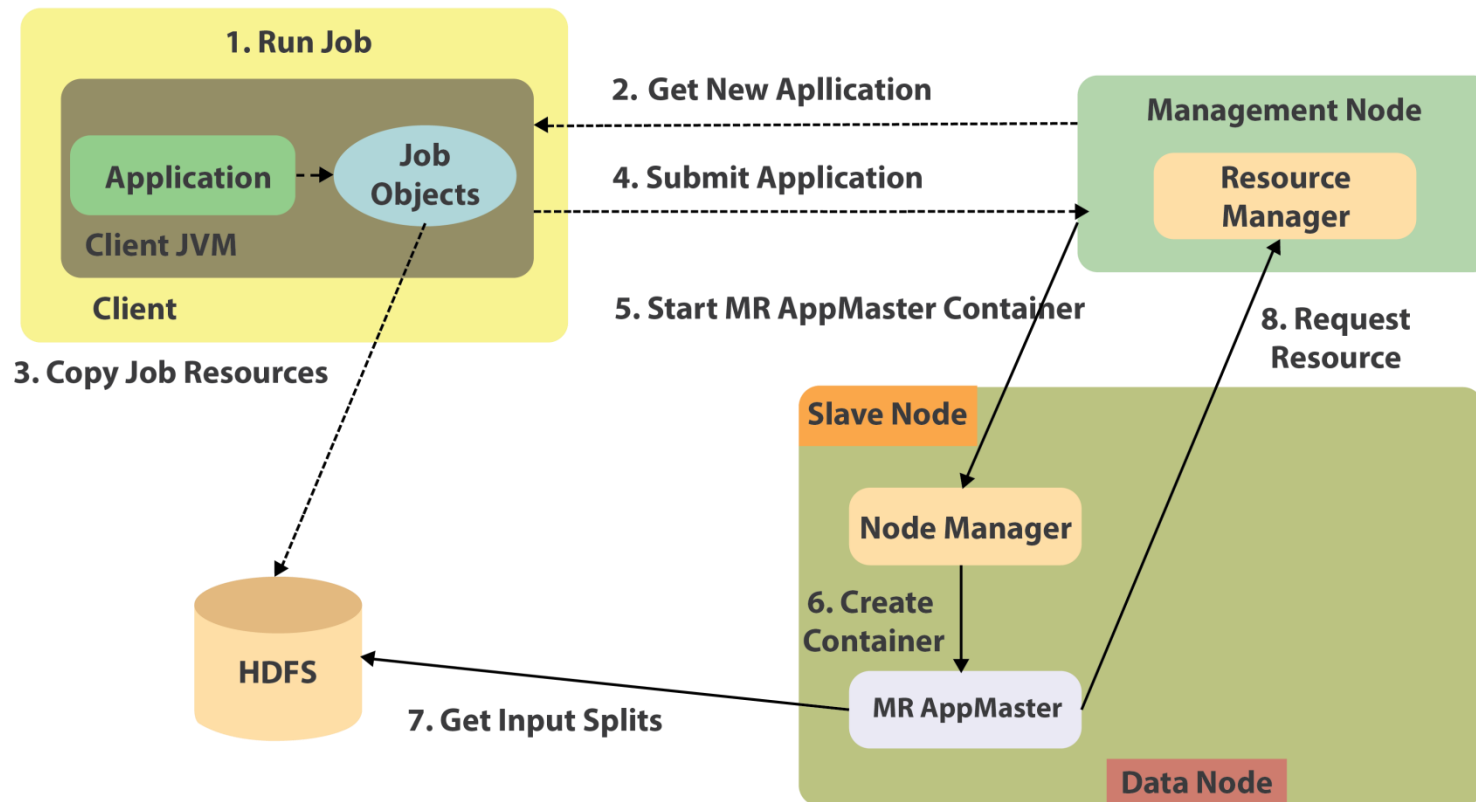
## YARN(Cont'd)



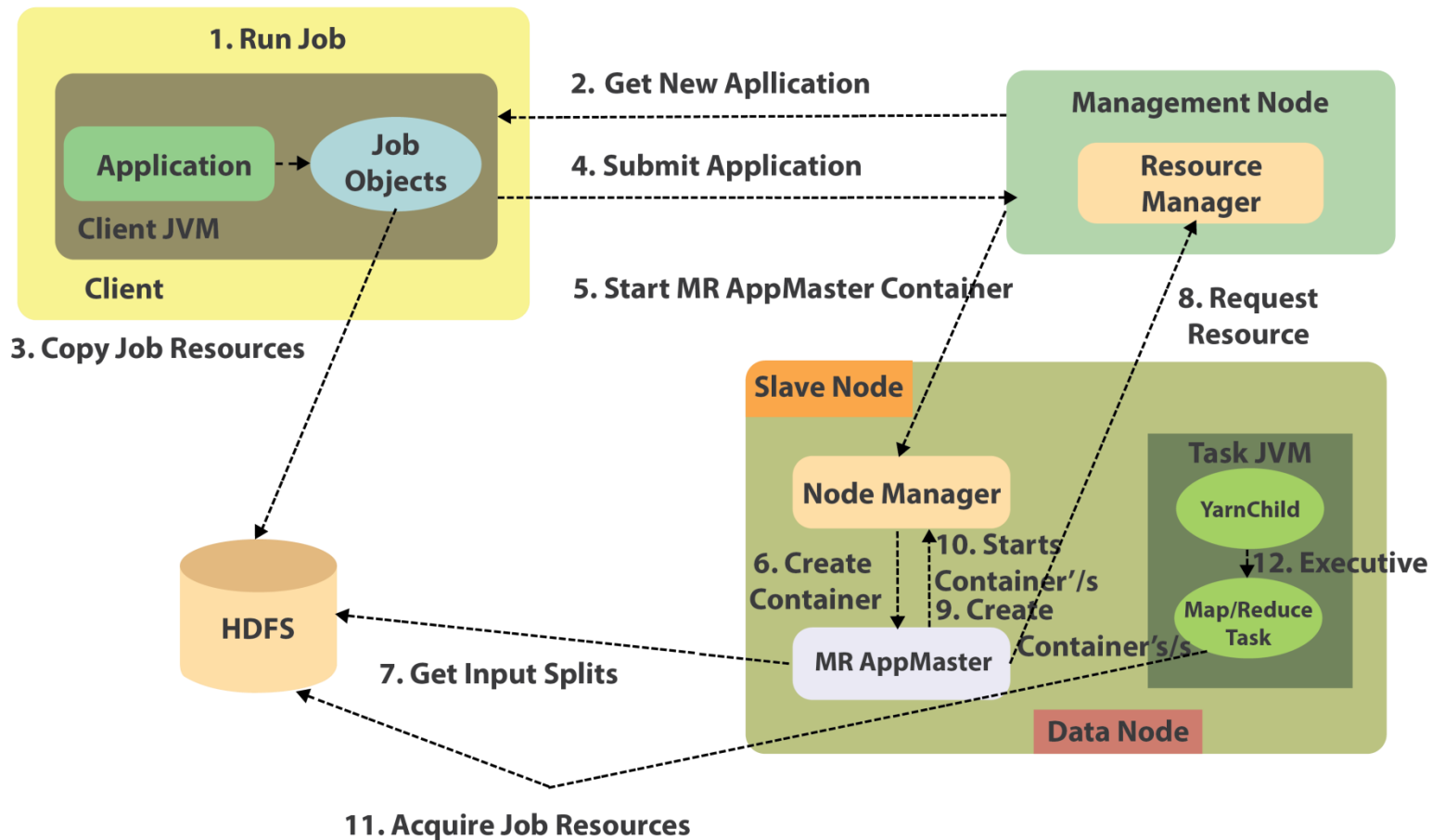
# YARN MR Application Execution Flow



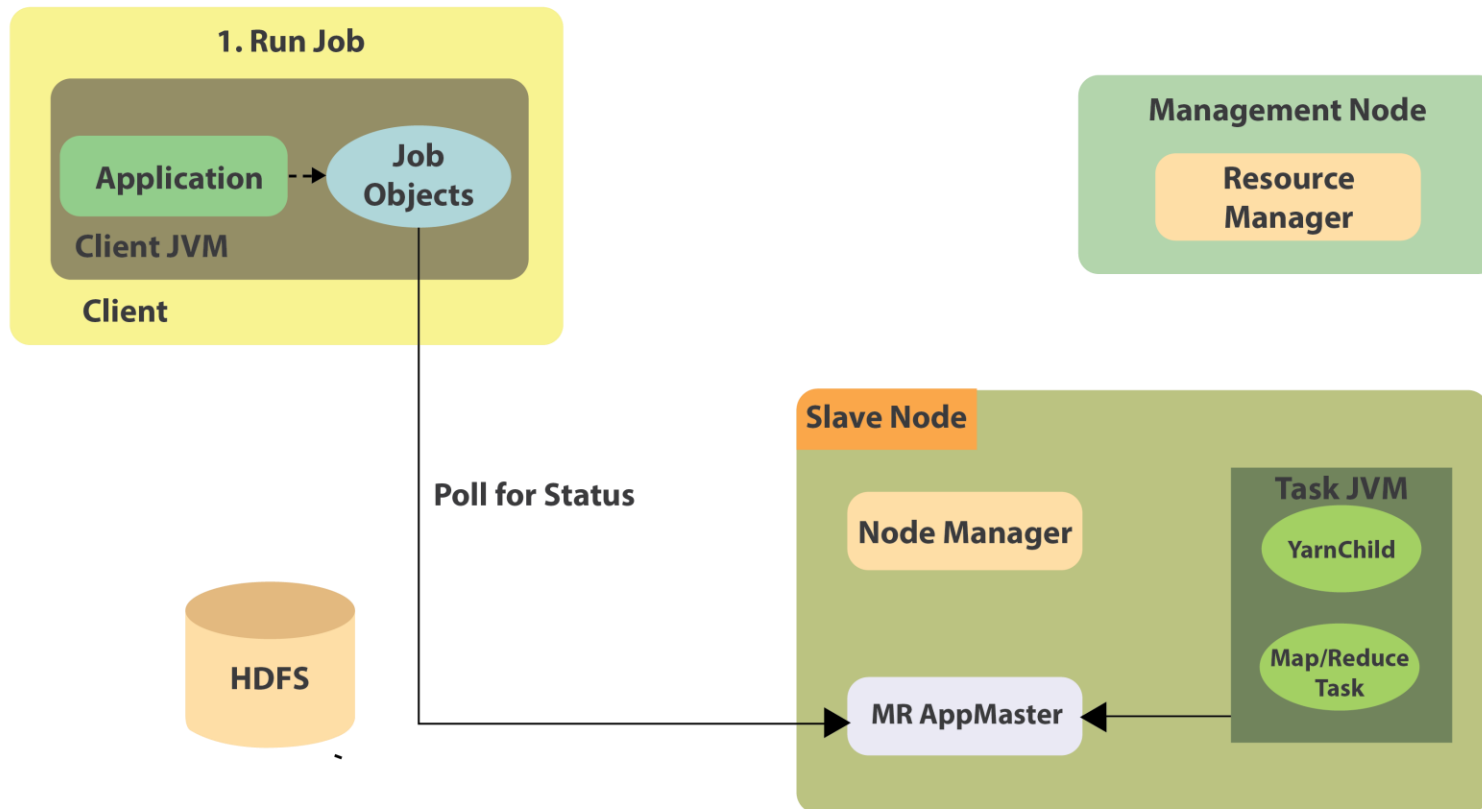
# YARN MR Application Execution Flow(Cont'd)



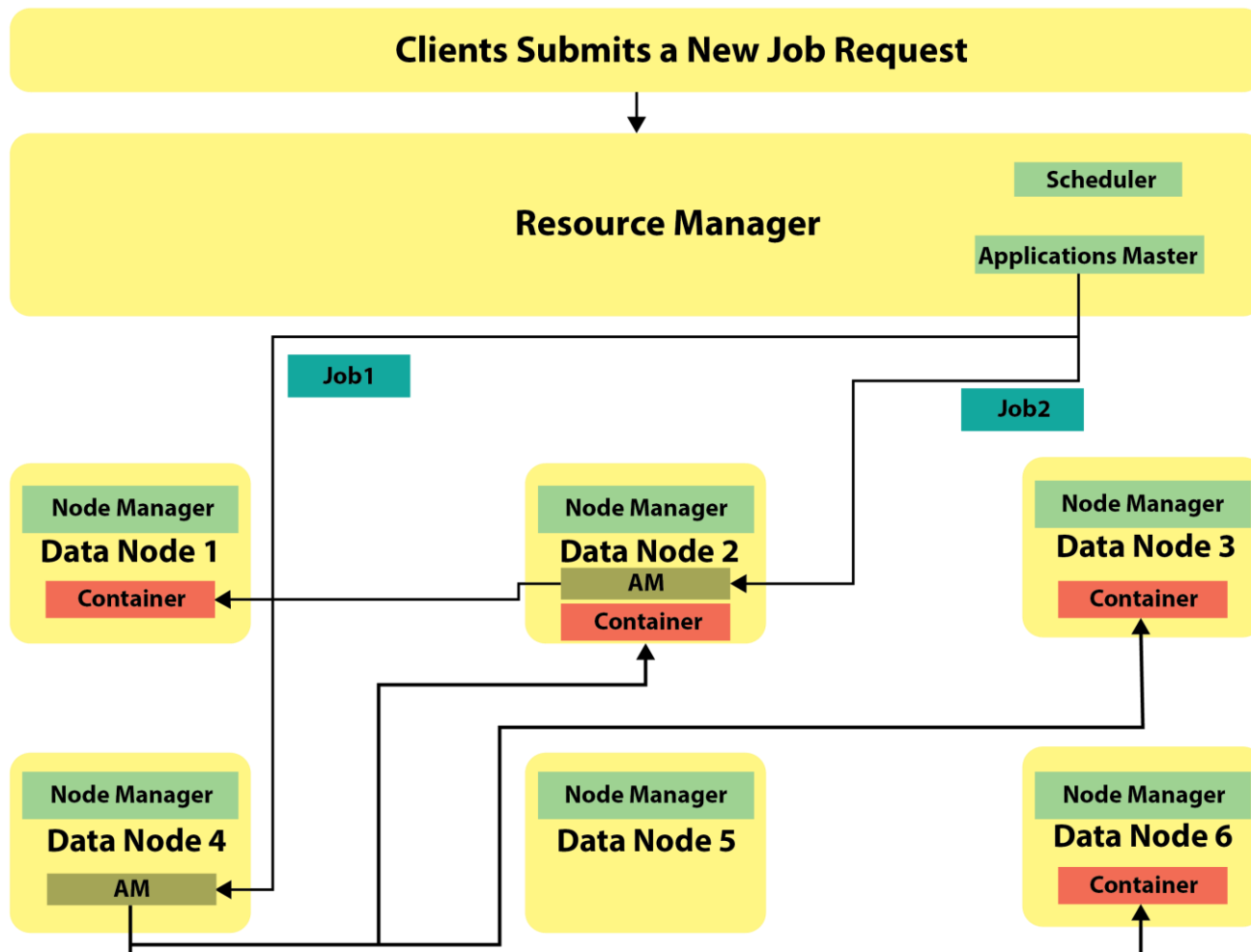
# YARN MR Application Execution Flow(Cont'd)



# YARN MR Application Execution Flow(Cont'd)



# YARN MR Application Execution Flow(Cont'd)



# Drawbacks of Hadoop 1.0

---

- Namenode is not at all backed up by any hot standby systems
- In case of primary Namenode failure, the entire cluster becomes inaccessible
- Also, the namespaces are limited to one, with no horizontal scalability
- The Job Tracker becomes a bottleneck with all the initiation, monitoring and re-creation of jobs
- It support only map task and reduce task ,other EcoSystem Tools have limited support
- The huge amount of cluster data cannot be utilized by other for different outputs such as graph Processing



# Solution is Hadoop 2.0

---

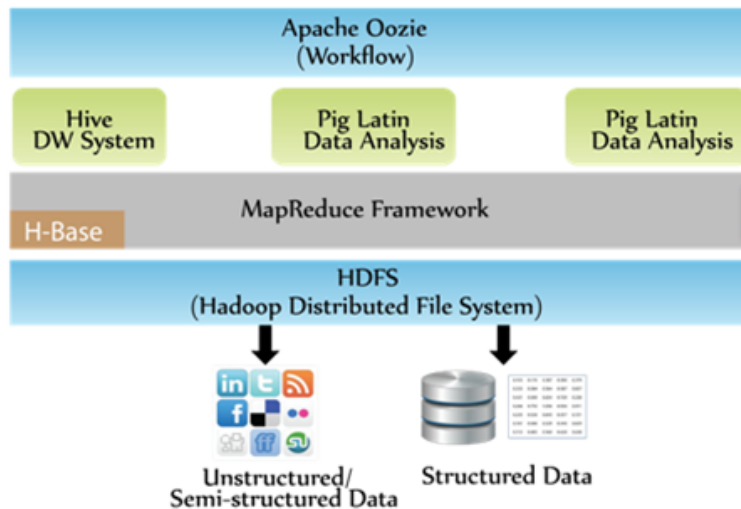
- Hadoop 2.x provides the following features to overcome the drawbacks of Hadoop 1.x.
- NameNode Federation for horizontal scalability
- NameNode High Availability
- YARN architecture for better job execution
- Supports almost all the latest EcoSystem processes



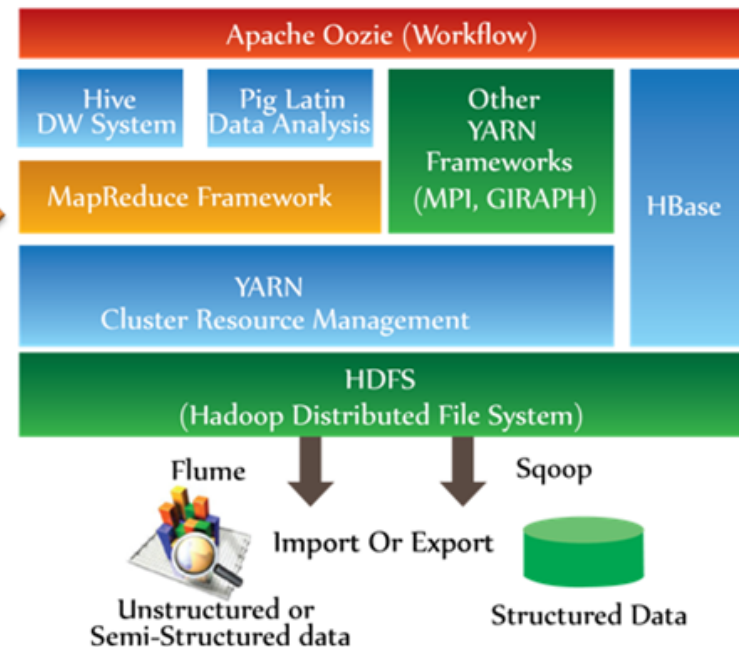


# Hadoop 1.x Vs. Hadoop 2.x

## Hadoop 1.x



## Hadoop 2.x



# Listing Hadoop Sample Programs

```
user@ubuntuvm: ~
user@ubuntuvm:~$ hadoop jar hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.2.0.jar
An example program must be given as the first argument.
Valid program names are:
aggregatewordcount: An Aggregate based map/reduce program that counts the words in the input files.
aggregatewordhist: An Aggregate based map/reduce program that computes the histogram of the words in the input files.
bbp: A map/reduce program that uses Bailey-Borwein-Plouffe to compute exact digits of Pi.
dbcount: An example job that count the pageview counts from a database.
distbbp: A map/reduce program that uses a BBP-type formula to compute exact bits of Pi.
grep: A map/reduce program that counts the matches of a regex in the input.
join: A job that effects a join over sorted, equally partitioned datasets
multifilewc: A job that counts words from several files.
pentomino: A map/reduce tile laying program to find solutions to pentomino problems.
pi: A map/reduce program that estimates Pi using a quasi-Monte Carlo method.
randomtextwriter: A map/reduce program that writes 10GB of random textual data per node.
randomwriter: A map/reduce program that writes 10GB of random data per node.
secondarysort: An example defining a secondary sort to the reduce.
sort: A map/reduce program that sorts the data written by the random writer.
sudoku: A sudoku solver.
teragen: Generate data for the terasort
terasort: Run the terasort
teravalidate: Checking results of terasort
wordcount: A map/reduce program that counts the words in the input files.
wordmean: A map/reduce program that counts the average length of the words in the input files.
wordmedian: A map/reduce program that counts the median length of the words in the input files.
wordstandarddeviation: A map/reduce program that counts the standard deviation of the length of the words in the input files.
user@ubuntuvm:~$
```



# Firing a MapReduce Job

```
user@ubuntuvm: ~
14/11/18 12:46:45 INFO mapreduce.Job: map 100% reduce 0%
14/11/18 12:46:49 INFO mapreduce.Job: map 100% reduce 100%
14/11/18 12:46:49 INFO mapreduce.Job: Job job_1416290290658_0003 completed successfully
14/11/18 12:46:49 INFO mapreduce.Job: Counters: 43
  File System Counters
    FILE: Number of bytes read=39
    FILE: Number of bytes written=158359
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=128
    HDFS: Number of bytes written=18
    HDFS: Number of read operations=6
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=2
  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)=2194
    Total time spent by all reduces in occupied slots (ms)=2012
  Map-Reduce Framework
    Map input records=4
    Map output records=8
    Map output bytes=116
    Map output materialized bytes=39
    Input split bytes=106
    Combine input records=8
    Combine output records=2
    Reduce input groups=2
    Reduce shuffle bytes=39
    Reduce input records=2
    Reduce output records=2
    Spilled Records=4
    Shuffled Maps =1
    Failed Shuffles=0
    Merged Map outputs=1
    GC time elapsed (ms)=93
    CPU time spent (ms)=610
    Physical memory (bytes) snapshot=219254784
    Virtual memory (bytes) snapshot=722239488
    Total committed heap usage (bytes)=137433088
  Shuffle Errors
```



# Understanding Dump of MapReduce Job

```
user@ubuntuvm: ~
14/11/18 12:46:45 INFO mapreduce.Job: map 100% reduce 0%
14/11/18 12:46:49 INFO mapreduce.Job: map 100% reduce 100%
14/11/18 12:46:49 INFO mapreduce.Job: Job job_1416298290658_0003 completed successfully
14/11/18 12:46:49 INFO mapreduce.Job: Counters: 43

File System Counters
  FILE: Number of bytes read=39
  FILE: Number of bytes written=158359
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=128
  HDFS: Number of bytes written=18
  HDFS: Number of read operations=6
  HDFS: Number of large read operations=0
  HDFS: Number of write operations=2

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)=2194
  Total time spent by all reduces in occupied slots (ms)=2012

Map-Reduce Framework
  Map input records=4
  Map output records=8
  Map output bytes=116
  Map output materialized bytes=30
  Input split bytes=106
  Combine input records=8
  Combine output records=2
  Reduce input groups=2
  Reduce shuffle bytes=39
  Reduce input records=2
  Reduce output records=2
  Spilled Records=4
  Shuffled Maps =1
  Failed Shuffles=0
  Merged Map outputs=1
  GC time elapsed (ms)=93
  CPU time spent (ms)=610
  Physical memory (bytes) snapshot=219254784
  Virtual memory (bytes) snapshot=722239488
  Total committed heap usage (bytes)=137433088

Shuffle Errors
```

1

2



# Understanding Dump of MapReduce Job

```
user@ubuntuvm: ~
14/11/18 12:46:45 INFO mapreduce.Job: map 100% reduce 0%
14/11/18 12:46:49 INFO mapreduce.Job: map 100% reduce 100%
14/11/18 12:46:49 INFO mapreduce.Job: Job job_1416298290658_0003 completed successfully
14/11/18 12:46:49 INFO mapreduce.Job: Counters: 43

File System Counters
  FILE: Number of bytes read=39
  FILE: Number of bytes written=158359
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=128
  HDFS: Number of bytes written=18
  HDFS: Number of read operations=6
  HDFS: Number of large read operations=0
  HDFS: Number of write operations=2

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)=2194
  Total time spent by all reduces in occupied slots (ms)=2012

Map-Reduce Framework
  Map input records=4
  Map output records=8
  Map output bytes=116
  Map output materialized bytes=30
  Input split bytes=106
  Combine input records=8
  Combine output records=2
  Reduce input groups=2
  Reduce shuffle bytes=39
  Reduce input records=2
  Reduce output records=2
  Spilled Records=4
  Shuffled Maps =1
  Failed Shuffles=0
  Merged Map outputs=1
  GC time elapsed (ms)=93
  CPU time spent (ms)=610
  Physical memory (bytes) snapshot=219254784
  Virtual memory (bytes) snapshot=722239488
  Total committed heap usage (bytes)=137433088

Shuffle Errors
```

1

2



# Quiz - 1

---

**Which one of the following nodes manages other nodes?**

- A** - Name node
- B** - Data node
- C** - slave node
- D** - None of these



# Quiz - Solution

---

**Which one of the following nodes manages other nodes?**

✓ **A** - Name node

**B** - Data node

**C** - slave node

**D** - None of these



## Quiz - 2

---

**Which of the following components retrieves the input splits directly from HDFS to determine the number of map tasks.**

- A - The NameNode.
- B - The TaskTrackers.
- C - The JobClient.
- D - The JobTracker.
- E - None of the options is correct.





# Quiz - Solution

---

**Which of the following components retrieves the input splits directly from HDFS to determine the number of map tasks.**

**A** - The NameNode.

**B** - The TaskTrackers.

**C** - The JobClient.

✓ **D** - The JobTracker.

**E** - None of the options is correct.



# Quiz - 3

## Under HDFS federation

- A** - Each namenode manages metadata of the entire filesystem.
- B** - Each namenode manages metadata of a portion of the filesystem.
- C** - Failure of one namenode causes loss of some metadata availability from the entire filesystem.
- D** - Each datanode registers with each namenode.

.



# Quiz - Solution

---

## Under HDFS federation

- A** - Each namenode manages metadata of the entire filesystem.
- ✓ **B** - Each namenode manages metadata of a portion of the filesystem.
- C** - Failure of one namenode causes loss of some metadata availability from the entire filesystem.
- D** - Each datanode registers with each namenode.

.



# Quiz - 4

---

## Under Hadoop High Availability, Fencing means

- A** - Preventing a previously active namenode from start running again.
- B** - Preventing the start of a failover in the event of network failure with the active namenode.
- C** - Preventing the power down to the previously active namenode.
- D** - Preventing a previously active namenode from writing to the edit log.

.



# Quiz - 4

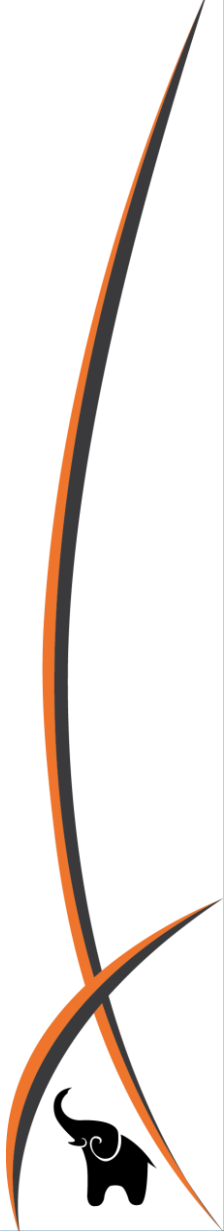
## Under Hadoop High Availability, Fencing means

- A** - Preventing a previously active namenode from start running again.
- B** - Preventing the start of a failover in the event of network failure with the active namenode.
- C** - Preventing the power down to the previously active namenode.
- ✓ **D** - Preventing a previously active namenode from writing to the edit log.



# Any Doubts?

---



- HDFS components are name node, secondary name node, data nodes , job tracker and task tracker.
- NameNode maintains, manages, and administers the data blocks saved on slave machines and managed by Data Nodes.
- An application adds data to HDFS by creating a new file and writing the data to it.
- Concept of rack awareness is important to prevent HDFS from placing all the copies of the block in same rack which might result in loss of data if rack fails.
- The mapper processes the data and creates several small chunks of data.
- The Reducer's job is to process the data that comes from the mapper.
- YARN – a resource-management platform responsible for managing computing resources in clusters and using them for scheduling of users' applications.



# Thank You!

