Introduction to Bigdata and Hadoop

Lecture4-6

Dr. S. Srivastava

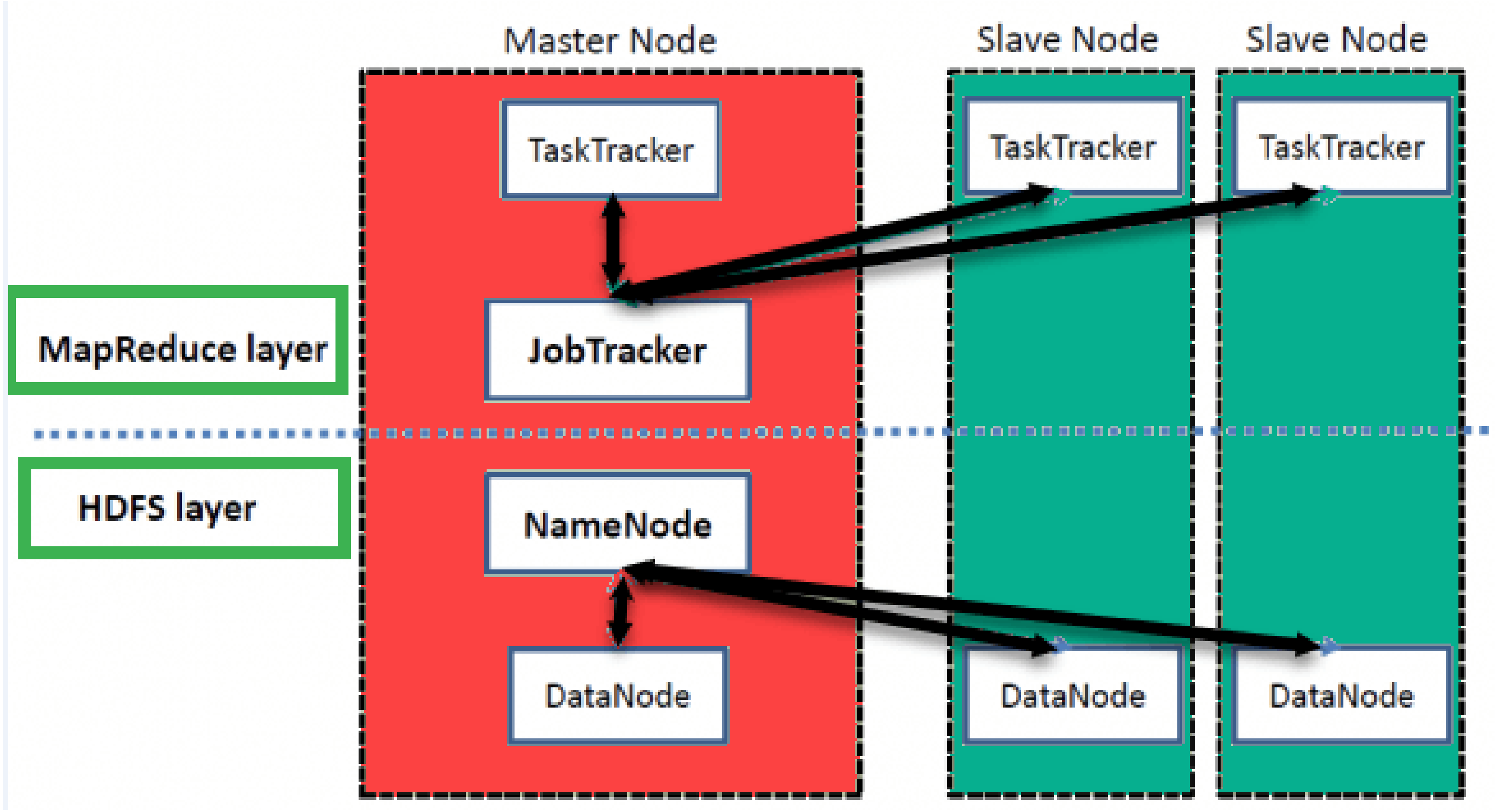
- Hadoop - HDFS Architecture
- Understanding 1.X core components
- Understanding 2.X core components
- Hadoop 1x vs 2x
- Hadoop 3x
- Hadoop 2x vs 3x

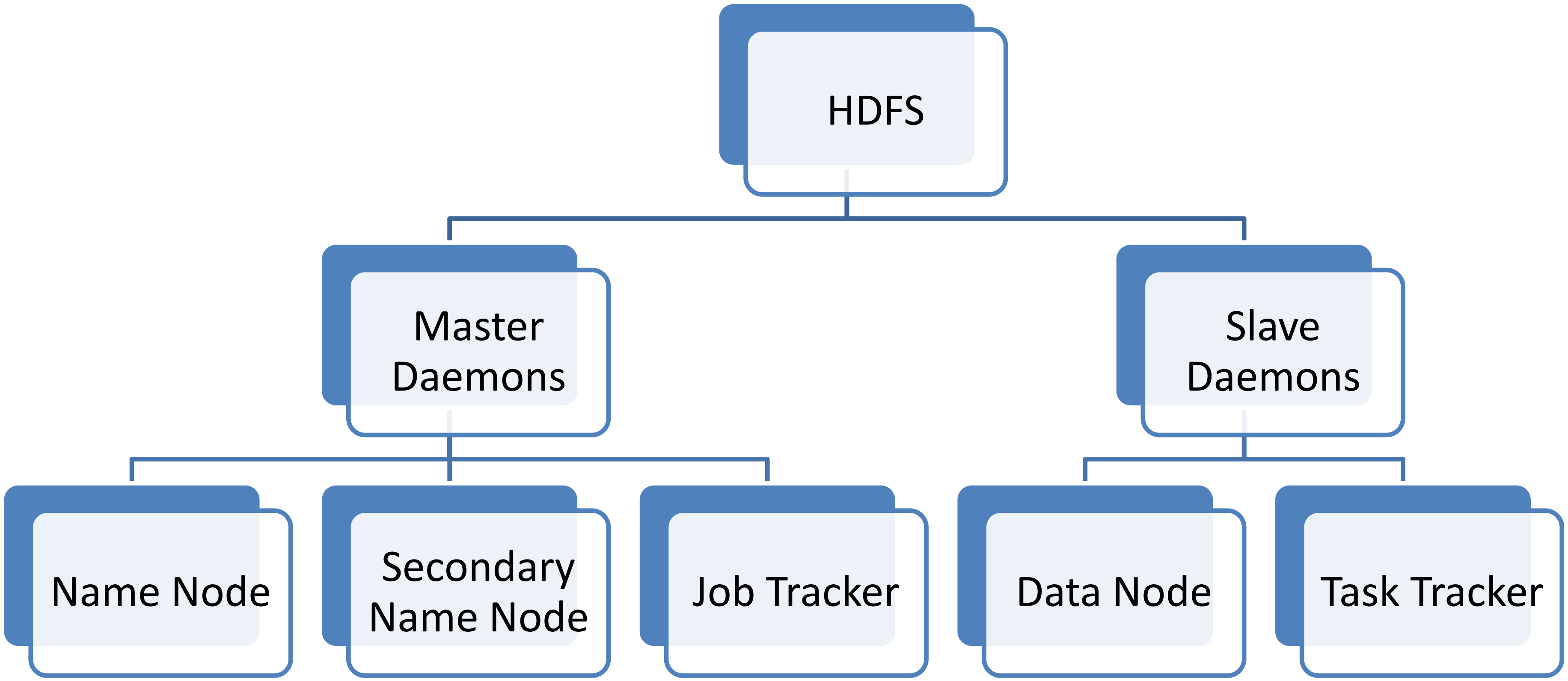
HDFS Architecture

Hadoop follows a Master-Slave architecture which is comprised of 2 daemons majorly.

A daemon is a back-ground service that runs on Hadoop.

Hadoop Architecture




HDFS
Master Daemons
Slave Daemons
Name Node
Secondary Name Node
Job Tracker
Data Node
Task Tracker

Name Node: -

- The Name Node is the center piece of an HDFS file system.
- The Name Node is major and a Master Node in Hadoop Architecture.
- Name Node keeps the directory tree of all files in the file system and tracks where the file is kept across the cluster

Name Node: -

- Name node does not store the any of these files data itself.
- Client applications talk to the Name Node whenever they wish to add/ copy/ move/ delete/ locate a file.
- The Name Node is responsible for maintaining the meta information of Hadoop file system.

Name Node

- The Name Node responds the successful requests by returning a list of relevant data Node servers where the data lives.
- The Name Node is a Single Point of Failure for the HDFS Cluster.
- When the Name Node goes down, the file system goes offline.

Name Node

- Name Node maintains the file system and the file system has the meta data for all files and directories.
- The Name node executes the file system called name space and all file operations.
- The Name Node will update two important permanent files.
  - Fsname space image
  - Edit log

Secondary Name Node:

- Secondary name node is deprecated and performs periodic check points of the name space.
- Secondary name node helps to keep the file size containing HDFS modifications log within certain limits at name node.
- Secondary name node replaced by check point node.

Secondary Name Node:

- Whenever the primary node is down, the Secondary name node will come into the picture.
- The Name Node stores modifications to the file system as a log appended to a native file, System file, edits.
- When a Name Node starts up, it reads HDFS state from an image file, fsimage and then applies edits from the edits log file.

Secondary Name Node:

- It then writes new HDFS state to the fsimage and starts normal operation with an empty edits file.
- The secondary Name Node merges the fsimage and the edits log files periodically and keeps edits log size within a limit.

Secondary Name Node:

- Secondary name node is not exactly replacement of primary node.
- It is usually run on a different machine than the primary Name Node since its memory requirements are on the same order as the primary Name Node.

Task Tracker:

- A Task Tracker is a node in the cluster that accepts tasks like Map, Reduce and Shuffle operations from a Job Tracker.
- Task tracker is responsible for instantiating & monitoring individual map and reduces work.

Task Tracker:

- Task tracker is responsible for instantiating & monitoring individual map and reduces work.
- Task Tracker is also known as s/w daemon for Hadoop architecture.

Task Tracker:

- Every Task Tracker is configured with a set of slots.
- The slot indicates the number of tasks that it can accept.

Task Tracker:

- The task manager primarily responsible for executing the tasks assigned by the job tracker in the form of MRJobs.

Task Tracker:

- In general, Task Manager will reside on top of data nodes.

Secondary Name Node:

- The start of the checkpoint process on the secondary Name Node is controlled by two Configuration parameters.
  - **fs.checkpoint.period**, set to 1 hour by default, specifies the maximum delay between two consecutive checkpoints.
  - **fs.checkpoint.size**, set to 64MB by default, defines the size of the edits log file that forces an urgent checkpoint even if the maximum checkpoint delay is not reached.

Secondary Name Node:

- The secondary Name Node stores the latest checkpoint in a directory which is structured the same way as the primary Name Node's directory.

Job Tracker:

- The Job Tracker is the service within Hadoop and farms out Map Reduce tasks to specific nodes in the cluster, ideally the nodes that have the data, or at least are in the same rack.
- Job tracker is responsible for scheduling and rescheduling and the tasks are in the form of Map reduce jobs.

Job Tracker:

- Job tracker will get the response/acknowledgement back from the task tracker.
- In general, Job tracker will reside on top of the Name Node.

Job Tracker:

- Job Tracker manages the map reduce tasks and distributes individual tasks to machine running the task tracker.
- Client applications submit jobs to the Job tracker.

Job Tracker:

- The Job Tracker talks to the Name Node to determine the data location.
- The Job Tracker locates Task Tracker nodes with available slots at or near the data.

Job Tracker:

- The Job Tracker submits the work to the chosen Task Tracker nodes.
- If they do not submit heartbeat signals often, a Task Tracker will notify the Job Tracker about the task failure.

Job Tracker:

- The Job Tracker decides what to do like it may resubmit the job elsewhere Or it may mark that specific record as something to avoid Or it may even blacklist the Task Tracker as unreliable.
- When the work is completed, the Job Tracker updates its status.

Job Tracker:

- Client applications can poll the Job Tracker for information.
- The Job Tracker is a point of failure for the Hadoop Map Reduce service. If it goes down, all running jobs are halted.

Data Node:

- A Data Node stores data in the Hadoop File System and is the place to hold the data.
- A functional file system has more than one Data Node with data distributed across the Data Nodes.

Data Node:

- Actual data in data nodes only in the form of HDFS blocks and by default, each block size is 64MB.
- In the beginning, Data Node connects to the Name Node and establishes the service.

Data Node:

- Then Data Node responds to requests from the Name Node for file system operations.
- Data Nodes are store and retrieve blocks, reporting name nodes.

Data Node:

- Client applications can talk directly to a Data Node by using the location of the data provided by Name Node.
- Task Tracker instances can indeed should be deployed on the same servers that host Data Node instance exists.

Data Node:

- A client accesses the file system on behalf of the user by communicating with data nodes.
- There is usually no need to use RAID storage for Data Node data.

Data Node:

- Data is designed to be replicated across multiple servers, rather than multiple disks on the same server.
- An ideal configuration is for a server to have a Data Node, a Task Tracker, and then physical disks one Task Tracker slot per CPU.
- This will allow every Task Tracker 100% of a CPU and separate disks to read and write data.

Hadoop 1.x

Hadoop 1.x Architecture is a history now because in most of the Hadoop applications are using ***Hadoop 2.x***

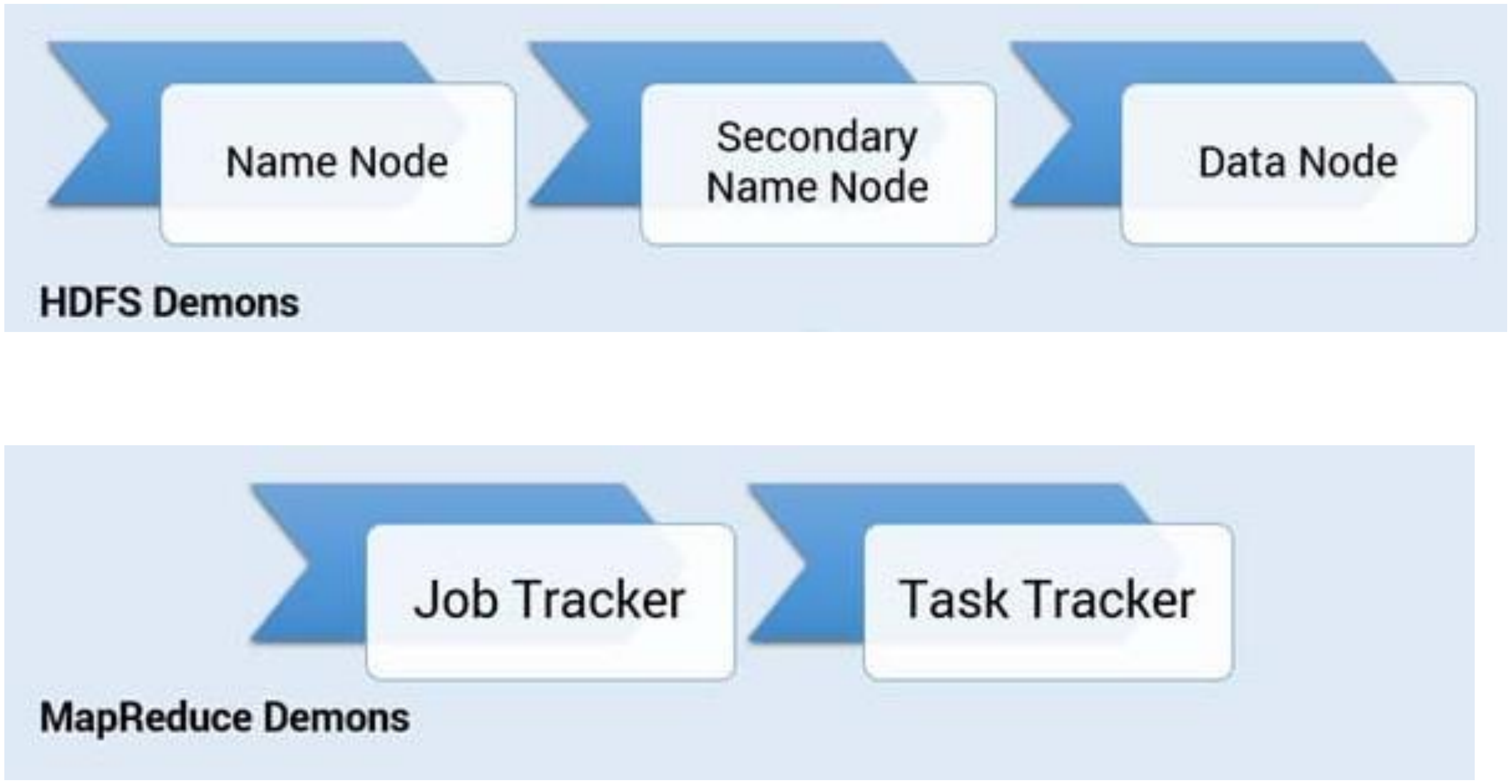


Fig. Hadoop 1x

Hadoop 1.x mainly has 3 daemons which are Name Node, Secondary Name Node and Data Node.

Name Node

- There is only single instance of this process runs on a cluster and that is on a master node
- It is responsible for manage metadata about files distributed across the cluster
- It manages information like location of file blocks across cluster and it's permission
- This process reads all the metadata from a file named `fsimage` and keeps it in memory
- After this process is started, it updates metadata for newly added or removed files in RAM
- It periodically writes the changes in one file called `edits` as edit logs
- This process is a heart of HDFS, if it is down HDFS is not accessible any more

Secondary Name Node

- For this also, only single instance of this process runs on a cluster
- This process can run on a master node (for smaller clusters) or can run on a separate node (in larger clusters) depends on the size of the cluster
- One misinterpretation from name is *"This is a backup Name Node"* but **IT IS NOT!!!!!**
- It manages the metadata for the Name Node. In the sense, it reads the information written in edit logs (by Name Node) and creates an updated file of current cluster metadata
- Than it transfers that file back to Name Node so that `fsimage` file can be updated
- So, whenever Name Node daemon is restarted it can always find updated information in `fsimage` file

Data Node

- There are many instances of this process running on various slave nodes(referred as Data nodes)
- It is responsible for storing the individual file blocks on the slave nodes in Hadoop cluster
- Based on the replication factor, a single block is replicated in multiple slave nodes(only if replication factor is > 1) to prevent the data loss
- Whenever required, this process handles the access to a data block by communicating with Name Node
- This process periodically sends heart bits to Name Node to make Name Node aware that slave process is running

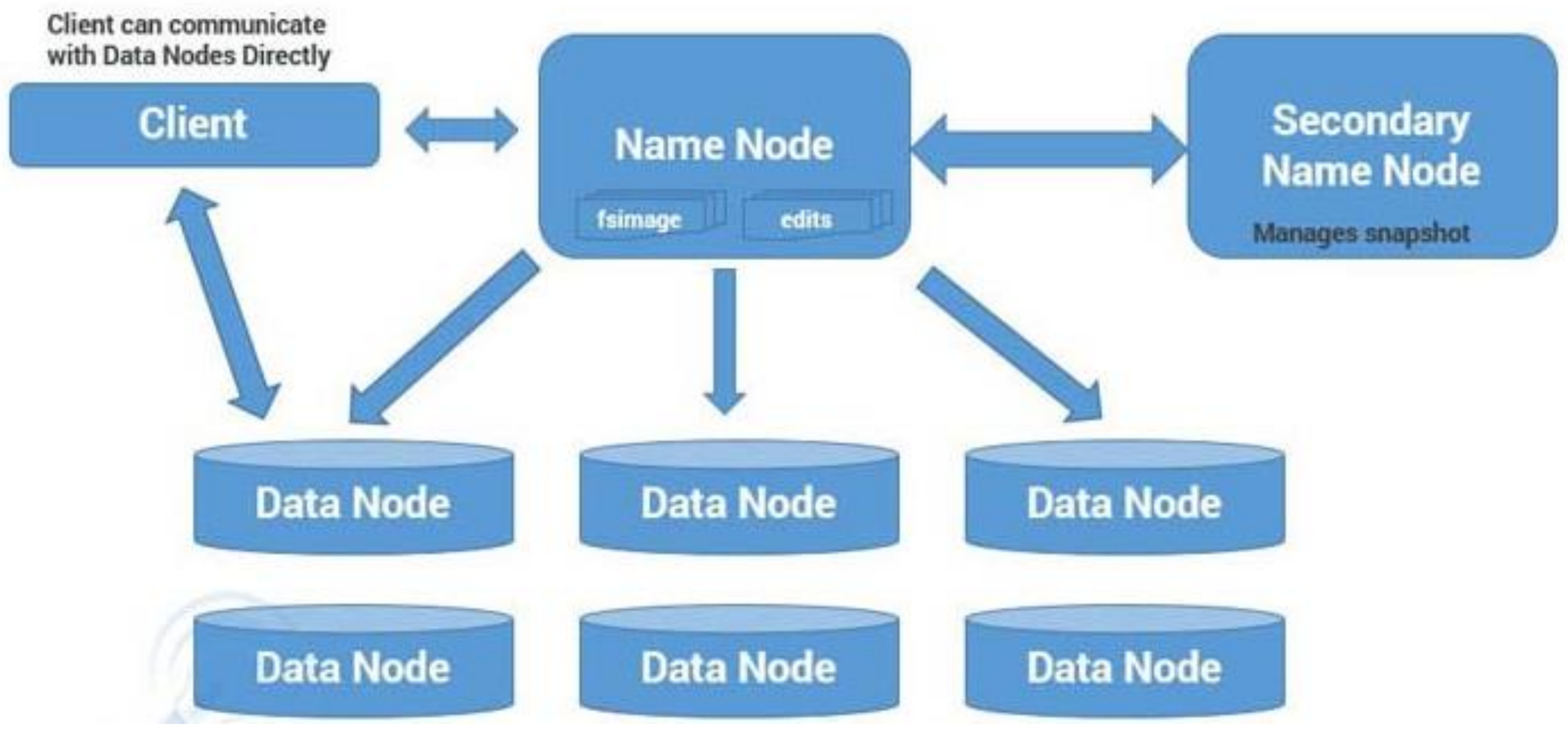


figure. Hadoop 1x architecture

Hadoop 1.x has a Single Point Of Failure

As per above description, HDFS has only one Name Node so if that process or machine goes down complete cluster goes down. That is why Name Node in Hadoop 1.x is considered to be a Single Point of Failure.

Hadoop 1.x Limitations

- It is only suitable for Batch Processing of Huge amount of Data, which is already in Hadoop System.
- It is not suitable for Real-time Data Processing.
- It is not suitable for Data Streaming.
- It supports upto **4000 Nodes** per Cluster.
- It has a single component : JobTracker to perform many activities like Resource Management, Job Scheduling, Job Monitoring, Re-scheduling Jobs etc

- JobTracker is the single point of failure.
- It supports only one Name Node and One Namespace per Cluster.
- It does not support Horizontal Scalability.
- It runs only Map/Reduce jobs.
- It follows Slots concept in HDFS to allocate Resources (Memory, RAM, CPU). It has static Map and Reduce Slots. That means once it assigns resources to Map/Reduce jobs, it cannot re-use them even though some slots are idle.
- For Example:- Suppose, 10 Map and 10 Reduce Jobs are running with 10 + 10 Slots to perform a computation. All Map Jobs are doing their tasks but all Reduce jobs are idle. We cannot use these Idle jobs for other purpose.

Understating 2.X core components

New features of Hadoop 2 x

- Hadoop 2.x has some common Hadoop API which can easily be integrated with any third party applications to work with Hadoop
- It has some new Java APIs and features in HDFS and MapReduce which are known as HDFS2 and MR2 respectively
- New architecture has added the architectural features like HDFS High Availability and HDFS Federation
- Hadoop 2.x not using Job Tracker and Task Tracker daemons for resource management now on-wards, it is using YARN (Yet Another Resource Negotiator) for Resource Management

HDFS High Availability (HA)

Problem: As you know in Hadoop 1.x architecture Name Node was a single point of failure, which means if your Name Node daemon is down somehow, you don’t have access to your Hadoop Cluster than after. How to deal with this problem?

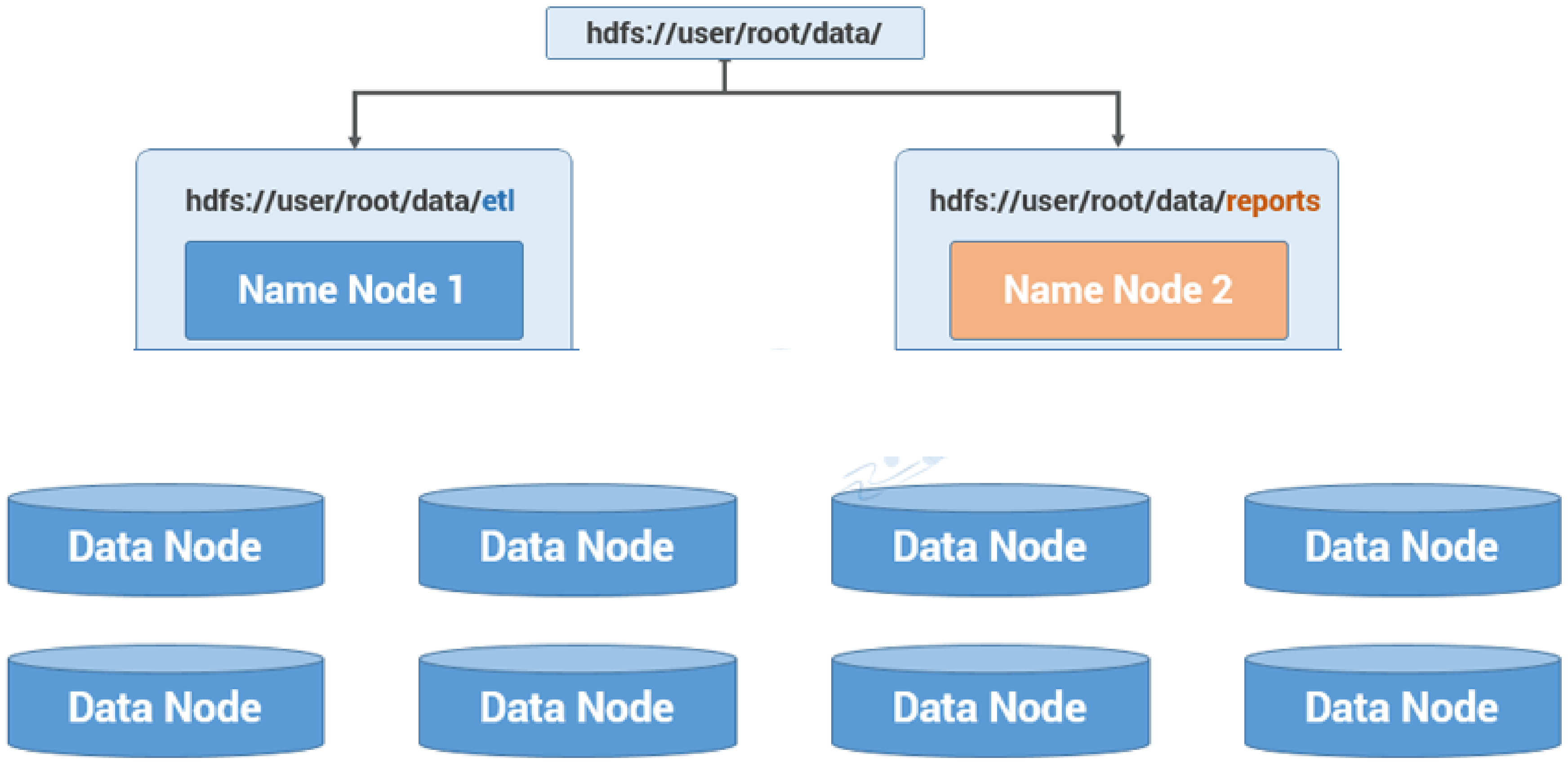
Solution: Hadoop 2.x is featured with Name Node HA which is referred as HDFS High Availability (HA).
Hadoop 2.x supports two Name Nodes at a time one node is active and another is standby node
Active Name Node handles the client operations in the cluster
StandBy Name Node manages metadata same as Secondary Name Node in Hadoop 1.x
When Active Name Node is down, Standby Name Node takes over and will handle the client operations then after
HDFS HA can be configured by two ways

- Using Shared NFS Directory
- Using Quorum Journal Manager

HDFS Federation

Problem: HDFS uses namespaces for managing directories, file and block level information in cluster.
Hadoop 1.x architecture was able to manage only single namespace in a whole cluster with the help of the Name Node (which is a single point of failure in Hadoop 1.x).
Once that Name Node is down you loose access of full cluster data. It was not possible for partial data availability based on name space.

Solution: Above problem is solved by HDFS Federation in Hadoop 2.x Architecture which allows to manage multiple namespaces by enabling multiple Name Nodes.
So on HDFS shell you have multiple directories available but it may be possible that two different directories are managed by two active Name Nodes at a time.



HDFS Federation

HDFS Federation by default allows single Name Node to manage full cluster (same as in Hadoop 1.x)

Hadoop 2.x Architecture

Hadoop2 Architecture has mainly 2 set of daemons

HDFS 2.x Daemons:

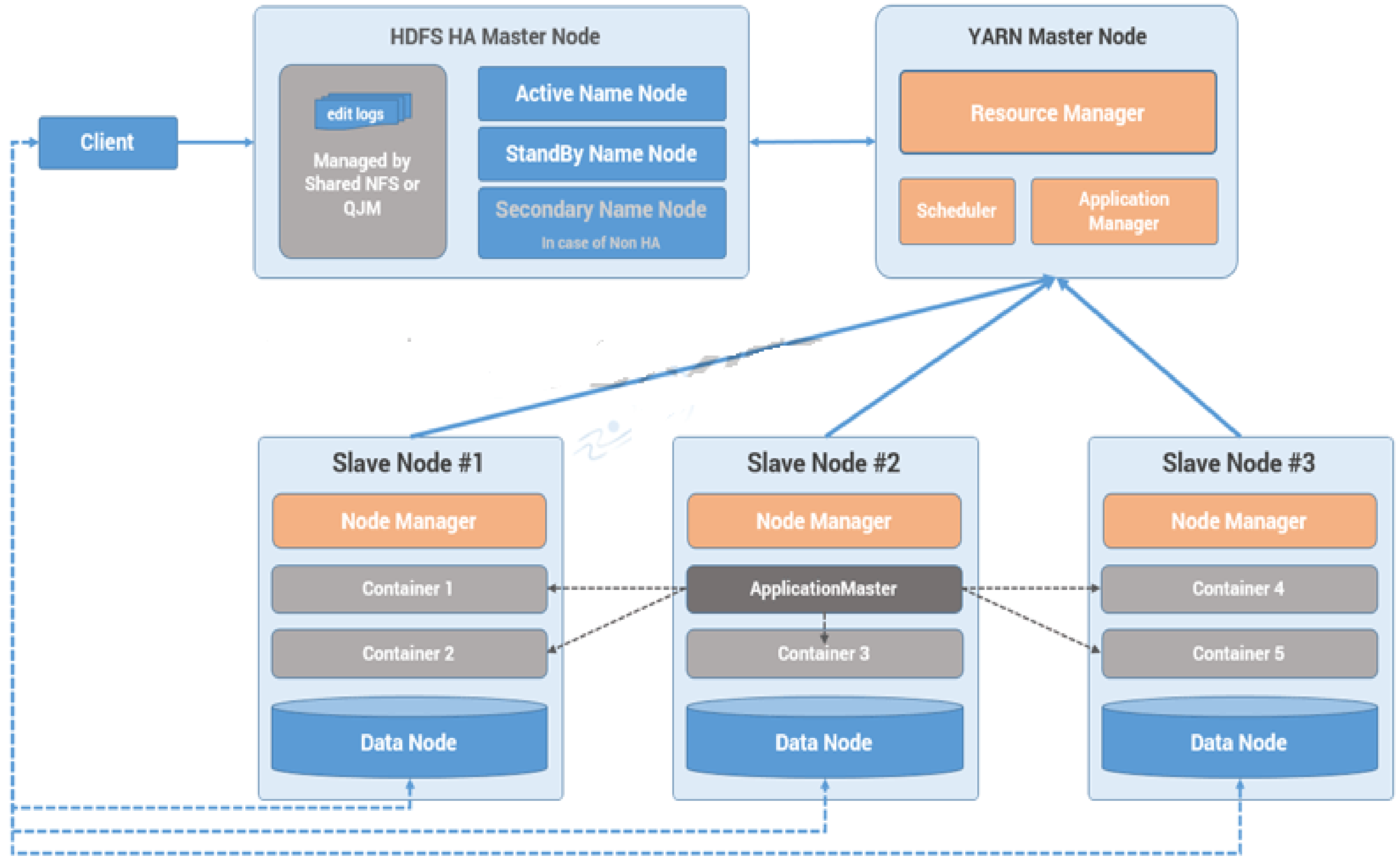
- Name Node,
- Secondary Name Node
- Data Nodes

MapReduce 2.x Daemons (YARN):

- Resource Manager
- Node Manager

HDFS 2.x Daemons- The working methodology of HDFS 2.x daemons is same as it was in Hadoop 1.x Architecture with following differences.

- Hadoop 2.x allows Multiple Name Nodes for HDFS Federation
- New Architecture allows HDFS High Availability mode in which it can have Active and StandBy Name Nodes (No Need of Secondary Name Node in this case)
- Hadoop 2.x Non HA mode has same Name Node and Secondary Name Node working same as in Hadoop 1.x architecture



Hadoop 2.x Architecture

Key-
Network File System (**NFS**) -allowing a user on a client computer to access files over a computer network much like local storage is accessed.
Quorum Journal Manager (**QJM**)- checks heartbeat / availability of nodes in cluster
HA- High Availibility

MapReduce 2.x Daemons (YARN)

- MapReduce2 has replace old daemon process Job Tracker and Task Tracker with YARN components Resource Manager and Node Manager respectively.
- These two components are responsible for executing distributed data computation jobs in Hadoop 2

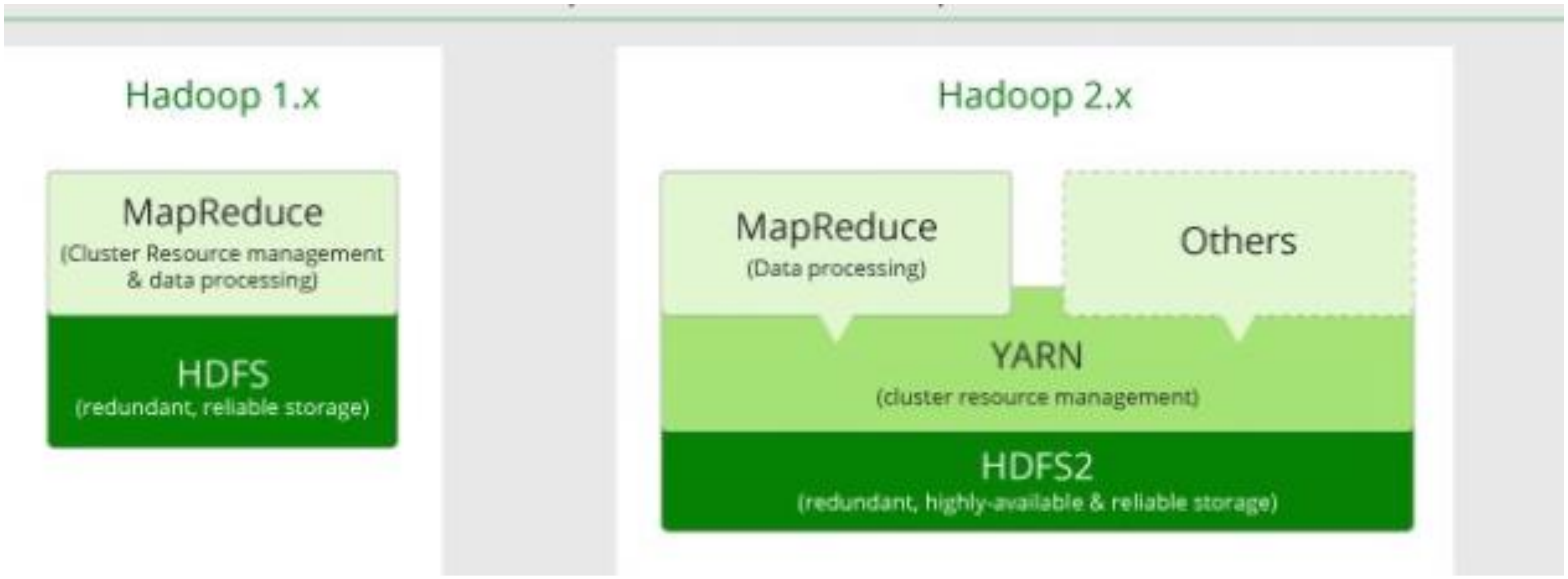
Resource Manager

- This daemon process runs on master node (may run on the same machine as name node for smaller clusters)
- It is responsible for getting job submitted from client and schedule it on cluster, monitoring running jobs on cluster and allocating proper resources on the slave node
- It communicates with Node Manager daemon process on the slave node to track the resource utilization
- It uses two other processes named *Application Manager* and *Scheduler* for MapReduce task and resource management

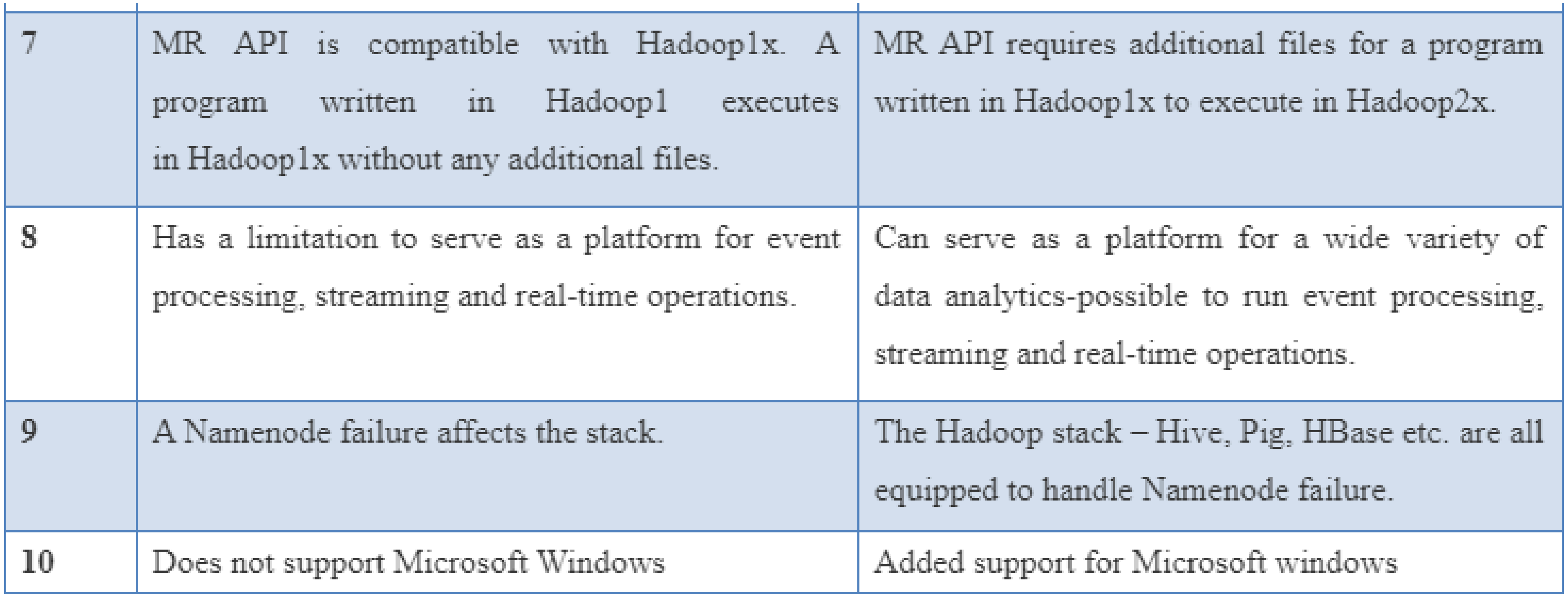
Node Manager

- This daemon process runs on slave nodes (normally on HDFS Data node machines)
- It is responsible for coordinating with Resource Manager for task scheduling and tracking the resource utilization on the slave node
- It also reports the resource utilization back to the Resource Manager
- It uses other daemon process like Application Master and Container for MapReduce task scheduling and execution on the slave node

Hadoop 1x vs 2x ?


Hadoop 1.x
MapReduce
(Cluster Resource management & data processing)
HDFS
(redundant, reliable storage)
Hadoop 2.x
MapReduce
(Data processing)
Others
YARN
(cluster resource management)
HDFS2
(redundant, highly-available & reliable storage)

Sl No	Hadoop1	Hadoop2
1	Supports MapReduce (MR) processing model only. Does not support non-MR tools	Allows to work in MR as well as other distributed computing models like Spark, Hama, Giraph, Message Passing Interface) MPI & HBase coprocessors.
2	MR does both processing and cluster-resource management.	YARN (Yet Another Resource Negotiator) does cluster resource management and processing is done using different processing models.
3	Has limited scaling of nodes. Limited to 4000 nodes per cluster	Has better scalability. Scalable up to 10000 nodes per cluster
4	Works on concepts of slots – slots can run either a Map task or a Reduce task only.	Works on concepts of containers. Using containers can run generic tasks.
5	A single Namenode to manage the entire namespace.	Multiple Namenode servers manage multiple namespaces.
6	Has Single-Point-of-Failure (SPOF) – because of single Namenode- and in the case of Namenode failure, needs manual intervention to overcome.	Has to feature to overcome SPOF with a standby Namenode and in the case of Namenode failure, it is configured for automatic recovery.



7	MR API is compatible with Hadoop1x. A program written in Hadoop1 executes in Hadoop1x without any additional files.	MR API requires additional files for a program written in Hadoop1x to execute in Hadoop2x.
8	Has a limitation to serve as a platform for event processing, streaming and real-time operations.	Can serve as a platform for a wide variety of data analytics-possible to run event processing, streaming and real-time operations.
9	A Namenode failure affects the stack.	The Hadoop stack – Hive, Pig, HBase etc. are all equipped to handle Namenode failure.
10	Does not support Microsoft Windows	Added support for Microsoft windows

Hadoop 3x

Why Hadoop 3x?

With Java 7 attaining end of life in 2015, there was a need to revise the minimum runtime version to Java 8 with a new Hadoop release so that the new release is supported by Oracle with security fixes and also will allow hadoop to upgrade its dependencies to modern versions.

With Hadoop 2.0 shell scripts were difficult to understand as hadoop developers had to read almost all the shell scripts to understand what is the correct environment variable to set an option and how to set it whether it is java.library.path or java classpath or GC options.

With support for only 2 NameNodes, Hadoop 2 did not provide maximum level of fault tolerance.

With the release of Hadoop 3.x there will be additional fault tolerance as it offers multiple NameNodes.

Replication is a costly affair in Hadoop 2 as it follows a 3x replication scheme leading to 200% additional storage space and resource overhead.

Hadoop 3.0 will incorporate **Erasure Coding** in place of replication consuming comparatively less storage space while providing same level of fault tolerance.

i) Minimum Runtime Version for Hadoop 3.0 is JDK 8

- With Oracle JDK 7 coming to its end of life in 2015, in Hadoop 3.0 JAR files are compiled to run on JDK 8 version.
- This gives Hadoop 3.0 a dependency upgrade to modern versions as most of the libraries only support Java 8 .

ii) Support for Erasure Coding in HDFS

- Considering the rapid growth trends in data and datacentre hardware, support for erasure coding in Hadoop 3.0 is an important feature in years to come.
- Erasure Coding is a 50 years old technique that lets any random piece of data to be recovered based on other piece of data i.e. metadata stored around it.
- Erasure Coding is more like an advanced RAID technique that recovers data automatically when hard disk fails.

In hadoop 2.0, HDFS inherits 3-way replication from GFS (Google File System).

The default replication factor is 3 meaning every piece of data is replicated thrice to ensure reliability of 99.999%.

Even with such a high precision of reliability, data reliability was still a matter of concern for many users.

A major problem with this approach is that replicating the data blocks to 3 data nodes incurs 200% additional storage overhead and network bandwidth when writing data.

With support for erasure coding in Hadoop 3.0, the physical disk usage will be cut by half (i.e. 3x disk space consumption will reduce to 1.5x) and the fault tolerance level will increase by 50%.

This new Hadoop 3.0 feature will save hadoop customers big bucks on hardware infrastructure as they can reduce the size of their hadoop cluster to half and store the same amount of data or continue to use the current hadoop cluster hardware infrastructure and store double the amount of data with HDFS EC

iii) Hadoop Shell Script Rewrite

- To address several long standing bugs, provide unifying behaviours and enhance the documentation and functionality- Hadoop shell scripts are rewritten in Hadoop 3.0. Some of the new features incorporated are –
- Allow hadoop developers to add build directories to the classpath to enable in source tree testing.
- The command to change ownership and permissions on many files 'hadoop distch' will now be executed through hadoop MapReduce jobs.
- To enable external log rotation, .out files will be appended in the new release unlike being overwritten in previous hadoop releases.
- In the earlier version of Hadoop, any unprotected shell errors would be displayed to the user, however after the shell scripts are rewritten in Hadoop 3.0 they would report error messages in a better way highlighting various states of the log and PID (process id) directories on daemon startup.
- With a new support/ debug option , shell scripts in hadoop 3.0 would report all the basic information on the construction of different environment variables, classpath, java options, etc. that will help in configuration debugging.

iv) MapReduce Task Level Native Optimization

- A new native implementation of the map output collector to perform sort, spill and IFile serialization in the native code as this will improve the performance of shuffle intensive jobs by 30%.

v) Support for Multiple NameNodes to maximize Fault Tolerance

- This new feature is just perfect for business critical deployments the need to run with high fault tolerance levels.

- Hadoop 3.0 supports 2 or more Standby nodes to provide additional fault tolerance unlike Hadoop 2.0 that supports only two NameNodes.

- Fault tolerance was limited in Hadoop 2.0 with as HDFS could run only a single standby and a single active NameNode. This limitation has been addressed in Hadoop 3.0 to enhance the fault tolerance in HDFS.

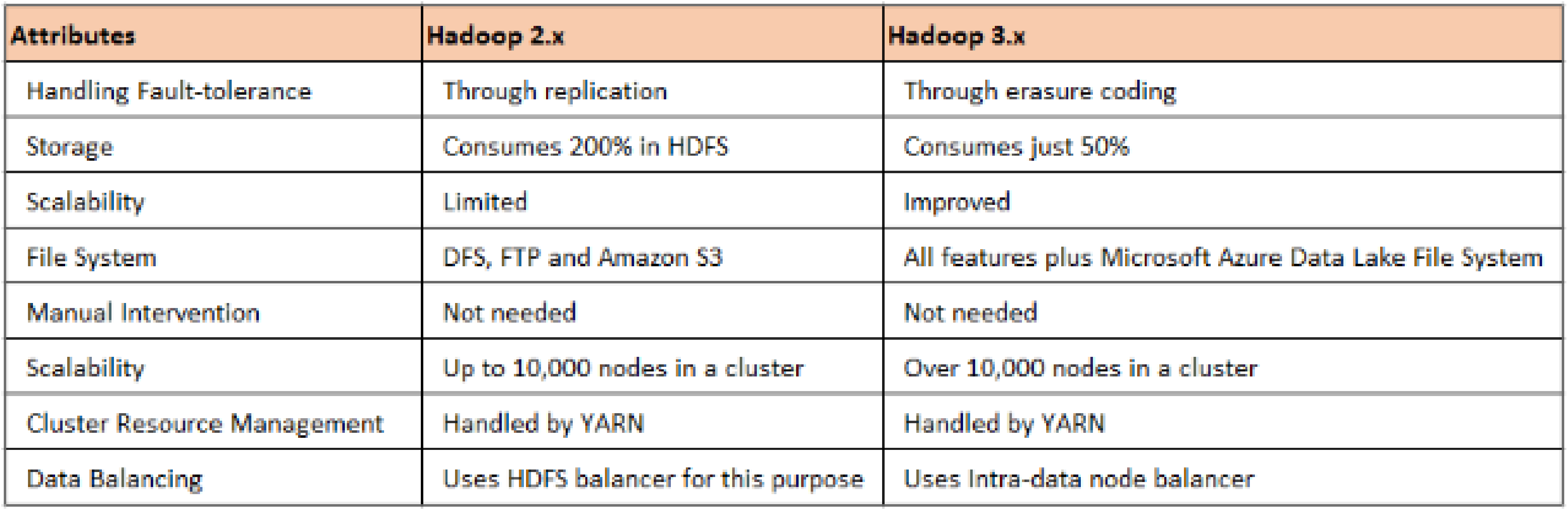
vi) More Powerful YARN in Hadoop 3.0

- Hadoop’s resource manager YARN was introduced in Hadoop 2.0 to make hadoop clusters run efficiently.
- In hadoop 3.0, YARN is coming off with multiple enhancements in the following areas –
- Support for long running services with the need to consolidate infrastructure.
- Better resource isolation for disk and network, resource utilization, user experiences, docker opportunities and elasticity.
- YARN in Hadoop 3.0 would be able to manage resources and services that run beyond the scope of a Hadoop cluster.

Change in Default Ports for Various Services and Addition of New Default Ports

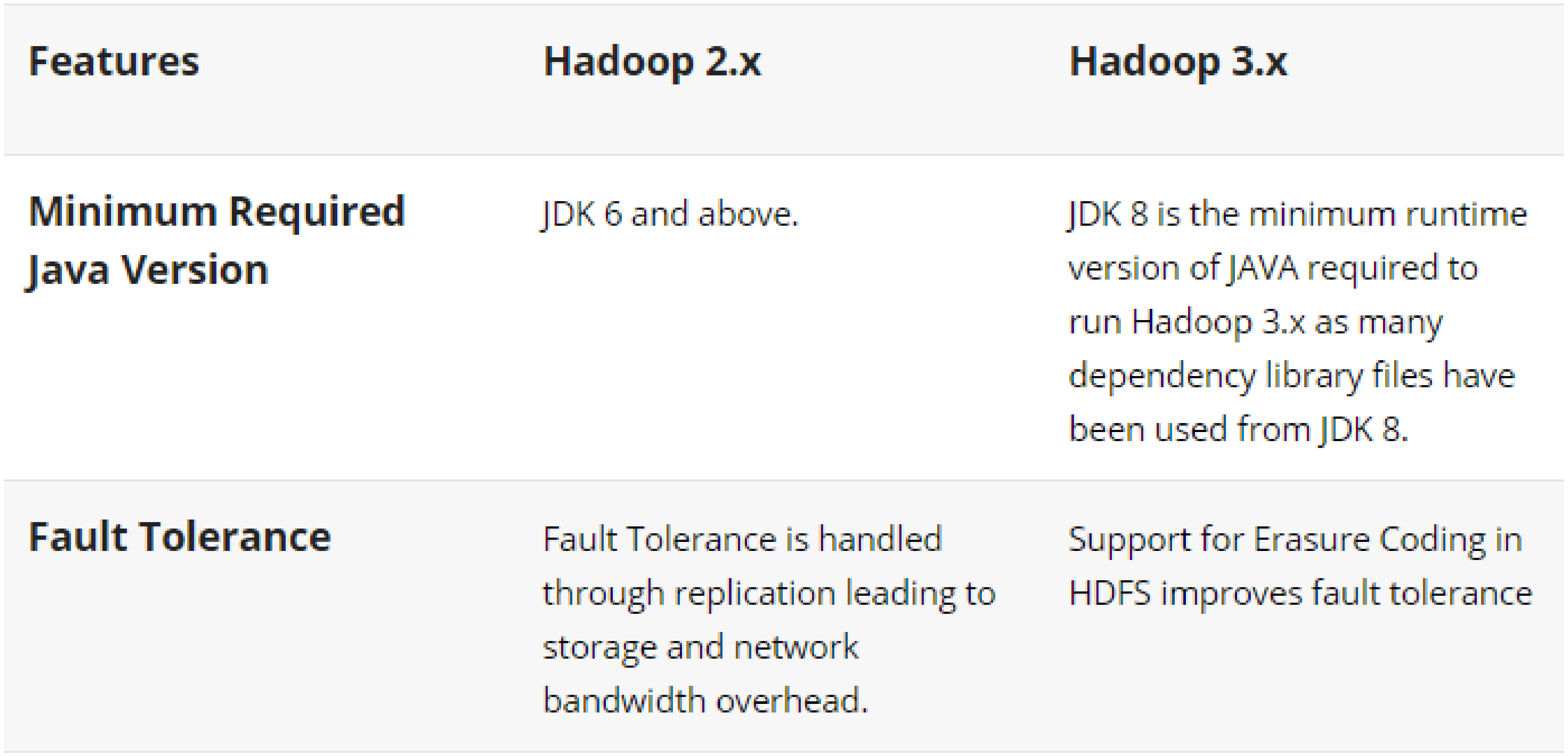
- The default ports for NameNode, DataNode, Secondary NameNode and KMS have been moved out of the Linux *ephemeral port range* (32768-61000) to avoid any bind errors on startup because of conflict with other application.
- This feature has been introduced to enhance the reliability of rolling restarts on large hadoop clusters.

Difference between Hadoop 2.x and Hadoop 3.x



Attributes	Hadoop 2.x	Hadoop 3.x
Handling Fault-tolerance	Through replication	Through erasure coding
Storage	Consumes 200% in HDFS	Consumes just 50%
Scalability	Limited	Improved
File System	DFS, FTP and Amazon S3	All features plus Microsoft Azure Data Lake File System
Manual Intervention	Not needed	Not needed
Scalability	Up to 10,000 nodes in a cluster	Over 10,000 nodes in a cluster
Cluster Resource Management	Handled by YARN	Handled by YARN
Data Balancing	Uses HDFS balancer for this purpose	Uses Intra-data node balancer

Hadoop 3 vs Hadoop 2 side by side



Features	Hadoop 2.x	Hadoop 3.x
Minimum Required Java Version	JDK 6 and above.	JDK 8 is the minimum runtime version of JAVA required to run Hadoop 3.x as many dependency library files have been used from JDK 8.
Fault Tolerance	Fault Tolerance is handled through replication leading to storage and network bandwidth overhead.	Support for Erasure Coding in HDFS improves fault tolerance

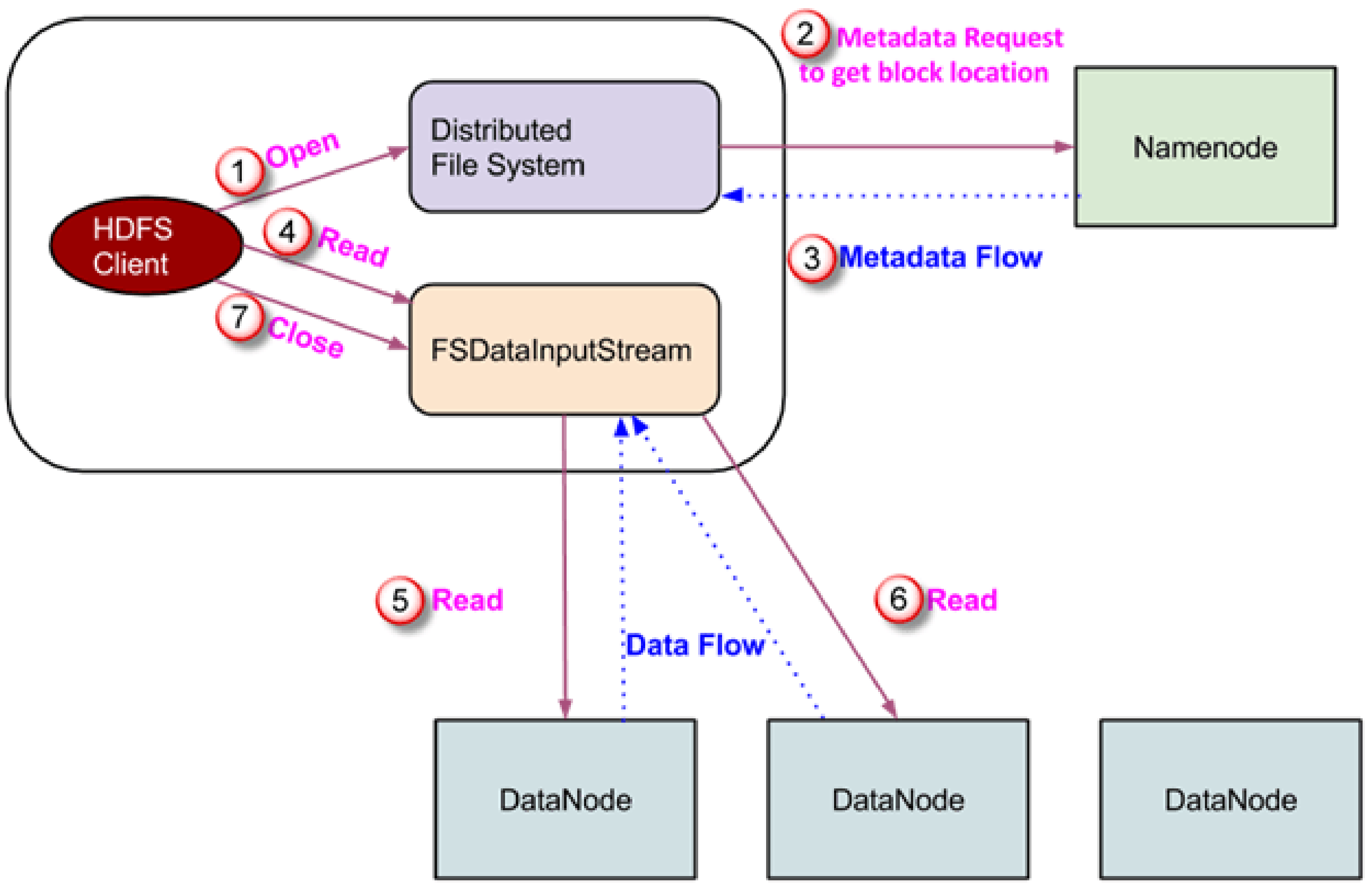
Storage Scheme	Follows a 3x Replication Scheme for data recovery leading to 200% storage overhead. For instance, if there are 8 data blocks then a total of 24 blocks will occupy the storage space because of the 3x replication scheme.	Storage overhead in Hadoop 3.0 is reduced to 50% with support for Erasure Coding. In this case, if here are 8 data blocks then a total of only 12 blocks will occupy the storage space.
Change in Port Numbers	Hadoop HDFS NameNode -8020 Hadoop HDFS DataNode -50010 Secondary NameNode HTTP -50091	Hadoop HDFS NameNode -9820 Hadoop HDFS DataNode -9866 Secondary NameNode HTTP -9869

YARN Timeline Service	YARN timeline service introduced in Hadoop 2.0 has some scalability issues.	YARN Timeline service has been enhanced with ATS v2 which improves the scalability and reliability.
Intra DataNode Balancing	HDFS Balancer in Hadoop 2.0 caused skew within a DataNode because of addition or replacement of disks.	Intra DataNode Balancing has been introduced in Hadoop 3.0 to address the intra-DataNode skews which occur when disks are added or replaced.
Number of NameNodes	Hadoop 2.0 introduced a secondary namenode as standby.	Hadoop 3.0 supports 2 or more NameNodes.

Heap Size	In Hadoop 2.0 , for Java and Hadoop tasks, the heap size needs to be set through two similar properties mapreduce.{map,reduce}.java. Opts and mapreduce.{map,reduce}.memory.mb	In Hadoop 3.0, heap size or mapreduce.*.memory.mb is derived automatically.

Read Operation In HDFS

Data read request is served by HDFS, NameNode, and DataNode. Let's call the reader as a 'client'


2 Metadata Request to get block location
1 Open
Distributed File System
Namenode
HDFS Client
4 Read
3 Metadata Flow
7 Close
FSDataInputStream
5 Read
6 Read
Data Flow
DataNode
DataNode
DataNode

1. A client initiates read request by calling **'open()'** method of FileSystem object; it is an object of type **DistributedFileSystem**.

2. This object connects to namenode using RPC and gets metadata information such as the locations of the blocks of the file. Please note that these addresses are of first few blocks of a file.

3. In response to this metadata request, addresses of the DataNodes having a copy of that block is returned back.

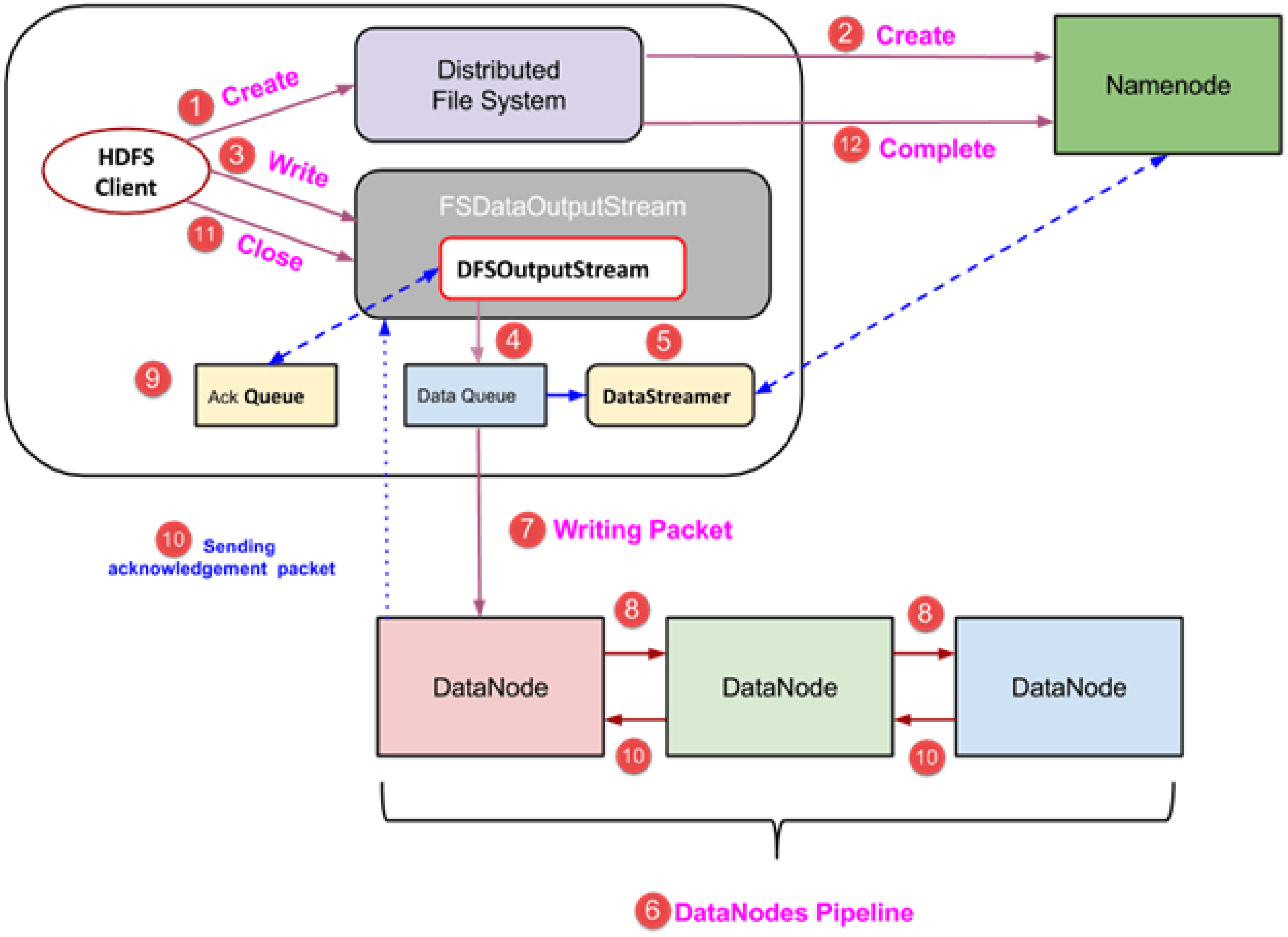
4. Once addresses of DataNodes are received, an object of type **FSDataInputStream** is returned to the client. **FSDataInputStream** contains **DFSInputStream** which takes care of interactions with DataNode and NameNode. In step 4 shown in the above diagram, a client invokes **'read()'** method which causes **DFSInputStream** to establish a connection with the first DataNode with the first block of a file.

5. Data is read in the form of streams wherein client invokes **'read()'** method repeatedly. This process of **read()** operation continues till it reaches the end of block.

6. Once the end of a block is reached, DFSInputStream closes the connection and moves on to locate the next DataNode for the next block

7. Once a client has done with the reading, it calls **a close()** method.

Write Operation In HDFS



1. A client initiates write operation by calling 'create()' method of DistributedFileSystem object which creates a new file - Step no. 1 in the above diagram.

2. DistributedFileSystem object connects to the NameNode using RPC call and initiates new file creation. However, this file creates operation does not associate any blocks with the file. It is the responsibility of NameNode to verify that the file (which is being created) does not exist already and a client has correct permissions to create a new file. If a file already exists or client does not have sufficient permission to create a new file, then **IOException** is thrown to the client. Otherwise, the operation succeeds and a new record for the file is created by the NameNode.

3. Once a new record in NameNode is created, an object of type FSDataOutputStream is returned to the client. A client uses it to write data into the HDFS. Data write method is invoked (step 3 in the diagram).

4. FSDataOutputStream contains DFSOutputStream object which looks after communication with DataNodes and NameNode. While the client continues writing data, **DFSOutputStream** continues creating packets with this data. These packets are enqueued into a queue which is called as **DataQueue**.

5. There is one more component called **DataStreamer** which consumes this **DataQueue**. DataStreamer also asks NameNode for allocation of new blocks thereby picking desirable DataNodes to be used for replication.

6. Now, the process of replication starts by creating a pipeline using DataNodes. In our case, we have chosen a replication level of 3 and hence there are 3 DataNodes in the pipeline.

7. The DataStreamer pours packets into the first DataNode in the pipeline.

8. Every DataNode in a pipeline stores packet received by it and forwards the same to the second DataNode in a pipeline.

9. Another queue, 'Ack Queue' is maintained by DFSOutputStream to store packets which are waiting for acknowledgment from DataNodes.

10. Once acknowledgment for a packet in the queue is received from all DataNodes in the pipeline, it is removed from the 'Ack Queue'. In the event of any DataNode failure, packets from this queue are used to reinitiate the operation.

11. After a client is done with the writing data, it calls a close() method (Step 9 in the diagram) Call to close(), results into flushing remaining data packets to the pipeline followed by waiting for acknowledgment.

12. Once a final acknowledgment is received, NameNode is contacted to tell it that the file write operation is complete.

Rack Awareness policy.

Rack– It the collection of machines around 40-50. All these machines are connected using the same network switch and if that network goes down then all machines in that rack will be out of service. Thus we say rack is down.

Rack Awareness was introduced by Apache Hadoop to overcome this issue.

In Rack Awareness, NameNode chooses the DataNode which is closer to the same rack or nearby rack. NameNode maintains **Rack ids** of each DataNode to achieve rack information.

Thus, this concept chooses Datanodes based on the rack information.

NameNode in hadoop makes ensures that all the replicas should not stored on the same rack or single rack. Rack Awareness Algorithm reduces latency as well as Fault Tolerance.

Default replication factor is 3.

Therefore according to Rack Awareness Algorithm:

- The first replica of the block will store on a local rack.
- The next replica will store on another datanode within the same rack.
- The third replica stored on the different rack.

Why Rack Awareness?

In **Big data** Hadoop, rack awareness is required for below reasons:

- To improve data **high availability** and reliability.
- **Improve the performance** of the cluster.
- To improve network bandwidth.
- Avoid losing data if entire rack fails though the chance of the rack failure is far less than that of node failure.
- To keep bulk data in the rack when possible.
- An assumption that in-rack id’s higher bandwidth, lower latency.

Replica Placement via Rack Awareness in Hadoop

Placement of replica is critical for ensuring high reliability and performance of HDFS. Optimizing replica placement via rack awareness distinguishes HDFS from other Distributed File System. Block Replication in multiple racks in HDFS is done using a policy as follows:

“No more than one replica is placed on one node. And no more than two replicas are placed on the same rack. This has a constraint that the number of racks used for block replication should be less than the total number of block replicas”.

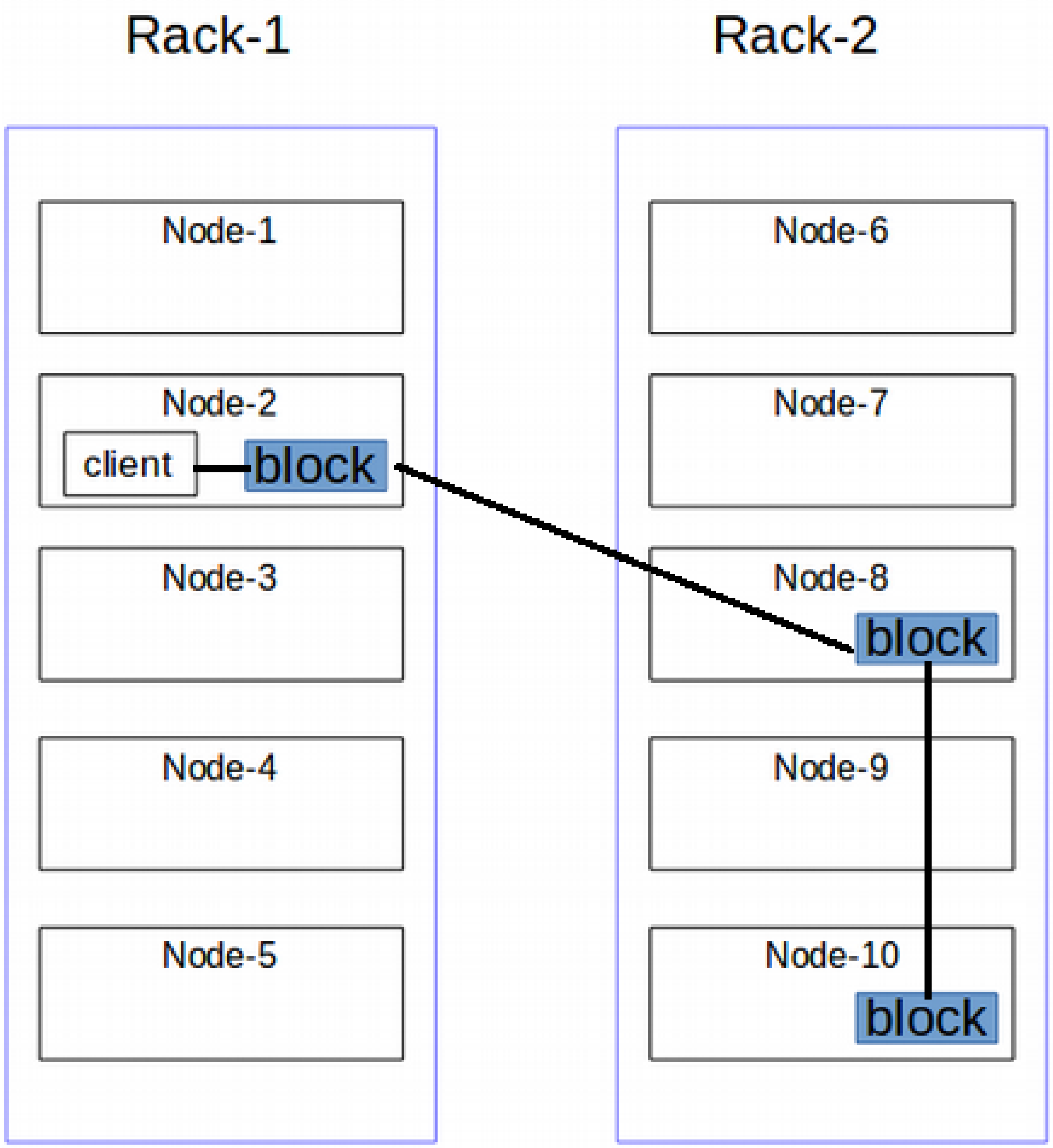
For Example:
When a new block is created: The First replica is placed on the local node. The Second one is placed on a different rack and the third one is placed on a different node at the local rack.

When re-replicating a block, if the number of an existing replica is one, place the second one on the different rack. If the number of an existing replica is two and if the two existing replicas are on the same rack, the third replica is placed on a different rack.

A simple but nonoptimal policy is to place replicas on the different racks. This prevents losing data when an entire rack fails and allows us to use bandwidth from multiple racks while reading the data.

The biggest drawback of this policy is that it will increase the cost of write operation because a writer needs to transfer blocks to multiple racks and communication between the two nodes in different racks has to go through switches.

In most cases, network bandwidth between machines in the same rack is greater than network bandwidth between machines in different racks. That’s why we use replica replacement policy.


Rack-1
Rack-2
Node-1
Node-2
client
block
Node-3
Node-4
Node-5
Node-6
Node-7
Node-8
block
Node-9
Node-10
block

What about performance?

Faster replication operation: Since the replicas are placed within the same rack it would use higher bandwidth and lower latency hence making it faster.

If **YARN** is unable to create a container in the same data node where the queried data is located it would try to create the container in a data node within the same rack. This would be more performing because of the higher bandwidth and lower latency of the data nodes inside the same rack.

Advantages of Implementing Rack Awareness

Minimize the writing cost and Maximize read speed –
Rack awareness places read/write requests to replicas on the same or nearby rack. Thus minimizing writing cost and maximizing reading speed.

Provide maximize network bandwidth and low latency –
Rack awareness maximizes network bandwidth by blocks transfer within a rack. Especially with rack awareness, the YARN is able to optimize MapReduceJob performance.

It assigns tasks to nodes that are 'closer' to their data in terms of network topology.

This is particularly beneficial in cases where tasks cannot be assigned to nodes where their data is stored locally.

Data protection against rack failure –
By default, the namenode assigns 2nd & 3rd replicas of a block to nodes in a rack different from the first replica. This provides data protection even against rack failure; however, this is possible only if Hadoop was configured with knowledge of its rack configuration.