Hadoop Architecture and HDFS

Part 1

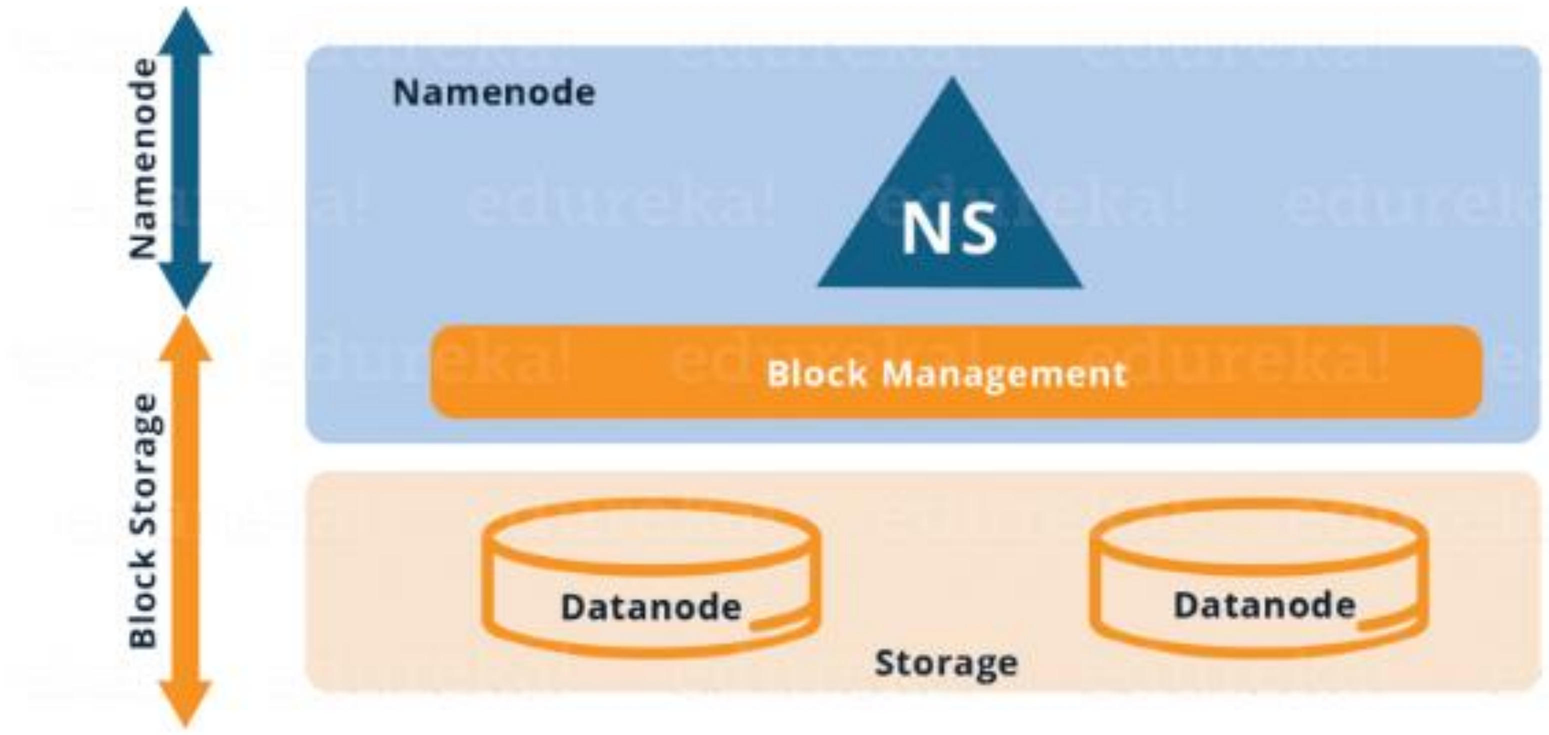
Dr. S. Srivastava

- Hadoop 2.x cluster architecture
- Federation Architecture,
- High Availability Architecture
- Modes of hadoop cluster
- Understading basis hadoop commands

Hadoop 2.0 Cluster Architecture

- ***HDFS Architecture*** follows Master/Slave Topology where NameNode acts as a master daemon and is responsible for managing other slave nodes called DataNodes.
- In this ecosystem, this single Master Daemon or NameNode becomes a bottleneck and on the contrary, companies need to have NameNode which is highly available. This very reason became the foundation of HDFS Federation Architecture and ***HA (High Availability) Architecture***.

Overview of Current HDFS Architecture:



the current HDFS has two layers

HDFS Namespace (NS):

- This layer is responsible for managing the directories, files and blocks.
- It provides all the File System operation related to Namespace like creating, deleting or modifying the files or the file directories.

Storage Layer: It comprises two basic components.

Block Management: It performs the following operations:

- Checks heartbeats of DataNodes periodically and it manages DataNode membership to the cluster.
- Manages the block reports and maintains block location.
- Supports block operations like creation, modification, deletion and allocation of block location.
- Maintains replication factor consistent throughout the cluster.

Physical Storage:

- It is managed by DataNodes which are responsible for storing data and thereby provides Read/Write access to the data stored in HDFS.

The current HDFS Architecture allows you to have a single namespace for a cluster.

In this architecture, a single NameNode is responsible for managing the namespace.

This architecture is very convenient and easy to implement.

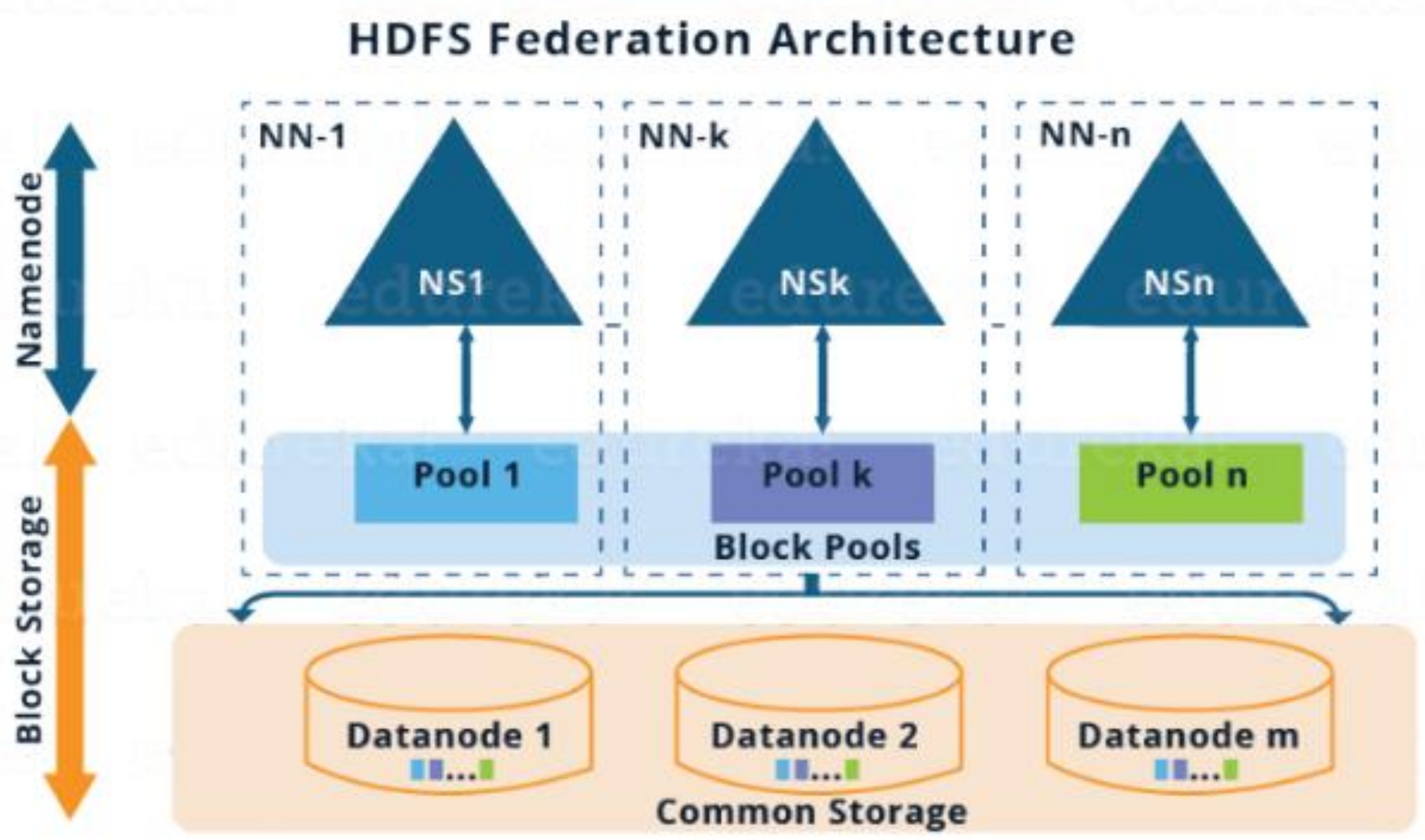
It provides sufficient capability to cater the needs of the small production cluster.

Limitations of Current HDFS:

- The namespace is **not scalable** like DataNodes. Hence, we can have only that number of DataNodes in the cluster that a single NameNode can handle.
- The two layers, i.e. Namespace layer and storage layer are **tightly coupled** which makes the alternate implementation of NameNode very difficult.
- The performance of the entire Hadoop System depends on the **throughput** of the NameNode. Therefore, entire performance of all the HDFS operations depends on how many tasks the NameNode can handle at a particular time.
- The NameNode stores the entire namespace in RAM for fast access. This leads to limitations in terms of **memory size** i.e. The number of namespace objects (files and blocks) that a single namespace server can cope up with.
- Many of the organizations (vendor) having HDFS deployment, allows multiple organizations (tenant) to use their cluster namespace. So, there is no separation of namespace and therefore, there is **no isolation** among tenant organization that are using the cluster.

HDFS Federation Architecture:

- Federation in Hadoop uses multiple independent Namenode/namespaces to scale the name service horizontally.
- In HDFS Federation Architecture, at the bottom, datanodes are present. And datanodes are used as common storage for blocks by all the namenodes.
- Each datanodes registers with all the namenodes in the cluster. These datanodes send periodic heartbeats, block report and handle command from the namenodes.



Many namenodes (NN1, NN2…, NNn) manages many namespaces (NS1, NS2…, NSn) respectively.

Each namespace has its own block pool (NS1 Has pool 1and so on). Block from pool 1 is stored on datanode 1 and so on.

Block pool

- Set of blocks is **Block pool** that belongs to a single namespace.
- There is a collection of pools in HDFS federation architecture. And each block is managed independently from other.
- This allows a namespace to create *Block ID* for new blocks without coordination with another namespace.
- All Datanodes stores data blocks present in all block pool.

Namespace volume

- Namespace along with its block pool is **Namespace volume**.
- Many namespace volumes are there in HDFS federation. Each namespace volume works independently.
- When we delete namenode or namespace, then corresponding block pool present on the datanodes will also be deleted.

Benefits of HDFS Federation

- **Isolation –** There is no isolation in single namenode in a multi-user environment. In HDFS federation different categories of application and users can be isolated to different namespaces by using many namenodes.
- **Namespace Scalability –** In federation many namenodes horizontally scales up in the filesystem namespace.
- **Performance –** We can improve Read/write operation throughput by adding more namenodes.

HDFS 2.x High Availability Cluster Architecture

The concept of High Availability cluster was introduced in Hadoop 2.x to solve the single point of failure problem in Hadoop 1.x.

- ***HDFS Architecture*** follows Master/Slave Topology where NameNode acts as a master daemon and is responsible for managing other slave nodes called DataNodes.
- This single Master Daemon or NameNode becomes a bottleneck.
- Although, the introduction of Secondary NameNode did prevent us from data loss and offloading some of the burden of the NameNode but, it did not solve the availability issue of the NameNode.

NameNode Availability:

- If you consider the standard configuration of HDFS cluster, the NameNode becomes a **single point of failure**.
- It happens because the moment the NameNode becomes unavailable, the whole cluster becomes unavailable until someone restarts the NameNode or brings a new one

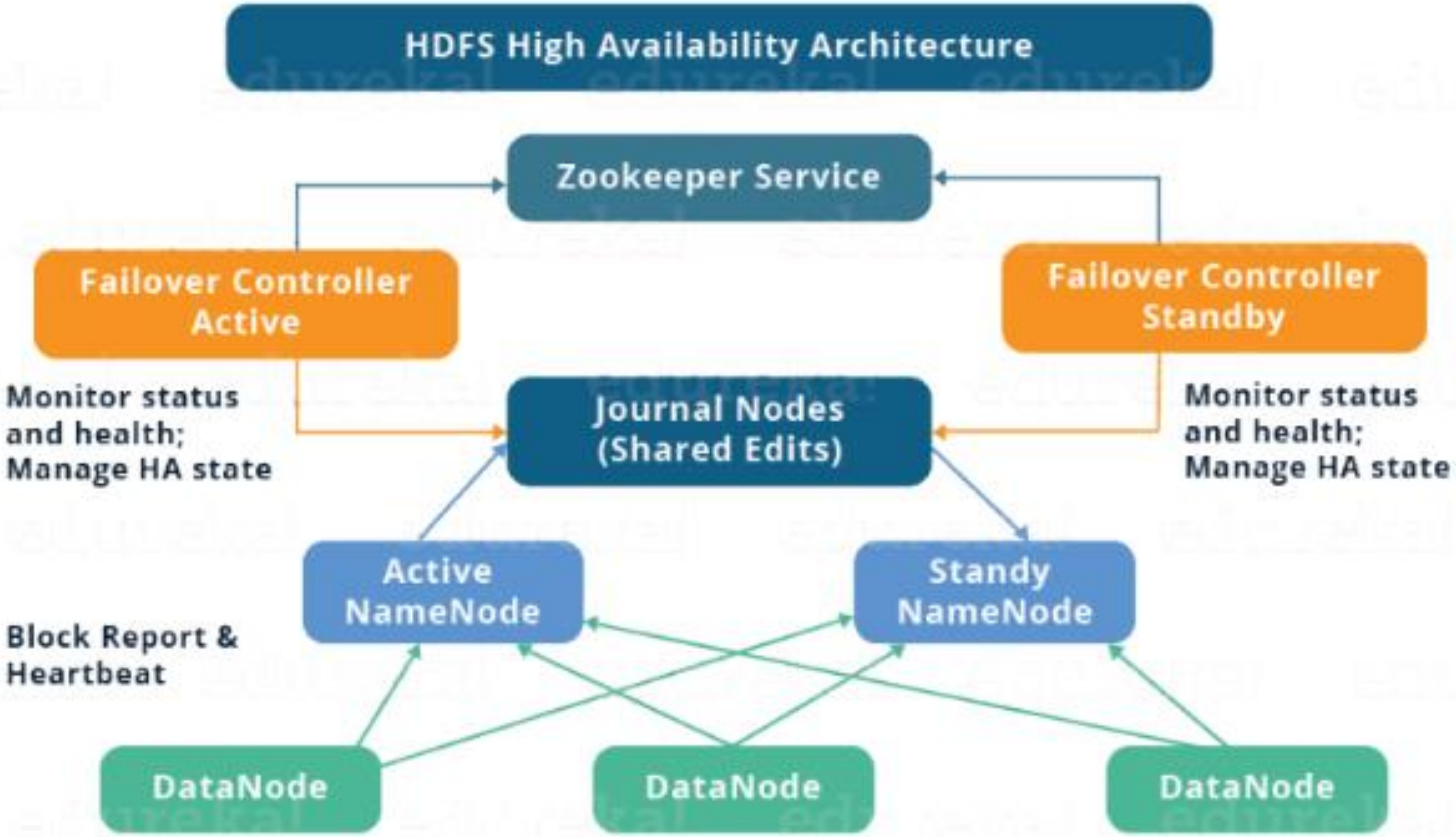
The reasons for unavailability of NameNode can be:

- A planned event like maintenance work such has upgradation of software or hardware.
- It may also be due to an unplanned event where the NameNode crashes because of some reasons.

The HA architecture solved this problem of NameNode availability by allowing us to have two NameNodes in an active/passive configuration.

- Active NameNode
  - Standby/Passive NameNode.

- If one NameNode goes down, the other NameNode can take over the responsibility and therefore, reduce the cluster down time.
- The standby NameNode serves the purpose of a backup NameNode (unlike the Secondary NameNode) which incorporate failover capabilities to the Hadoop cluster.
- Therefore, with the StandbyNode, we can have automatic failover whenever a NameNode crashes (unplanned event) or we can have a graceful (manually initiated) failover during the maintenance period.


HDFS High Availability Architecture
Zookeeper Service
Failover Controller Active
Failover Controller Standby
Monitor status and health; Manage HA state
Journal Nodes (Shared Edits)
Monitor status and health; Manage HA state
Active NameNode
Standy NameNode
Block Report & Heartbeat
DataNode
DataNode
DataNode

Issues in maintaining consistency in the HDFS High Availability cluster:

- **Active and Standby NameNode should always be in sync with each other, i.e. They should have the same metadata.**
  - This will allow us to restore the Hadoop cluster to the same namespace state where it got crashed and therefore, will provide us to have fast failover.
- **There should be only one active NameNode at a time because two active NameNode will lead to corruption of the data.**
  - *This kind of scenario is termed as a split-brain scenario where a cluster gets divided into smaller cluster, each one believing that it is the only active cluster.*
    - To avoid such scenarios fencing is done. Fencing is a process of ensuring that only one NameNode remains active at a particular time.

Implementation of HA Architecture:

- we can implement the Active and Standby NameNode configuration in following two ways:
  - Using Quorum Journal Nodes
  - Shared Storage using NFS

1. Using Quorum Journal Nodes:

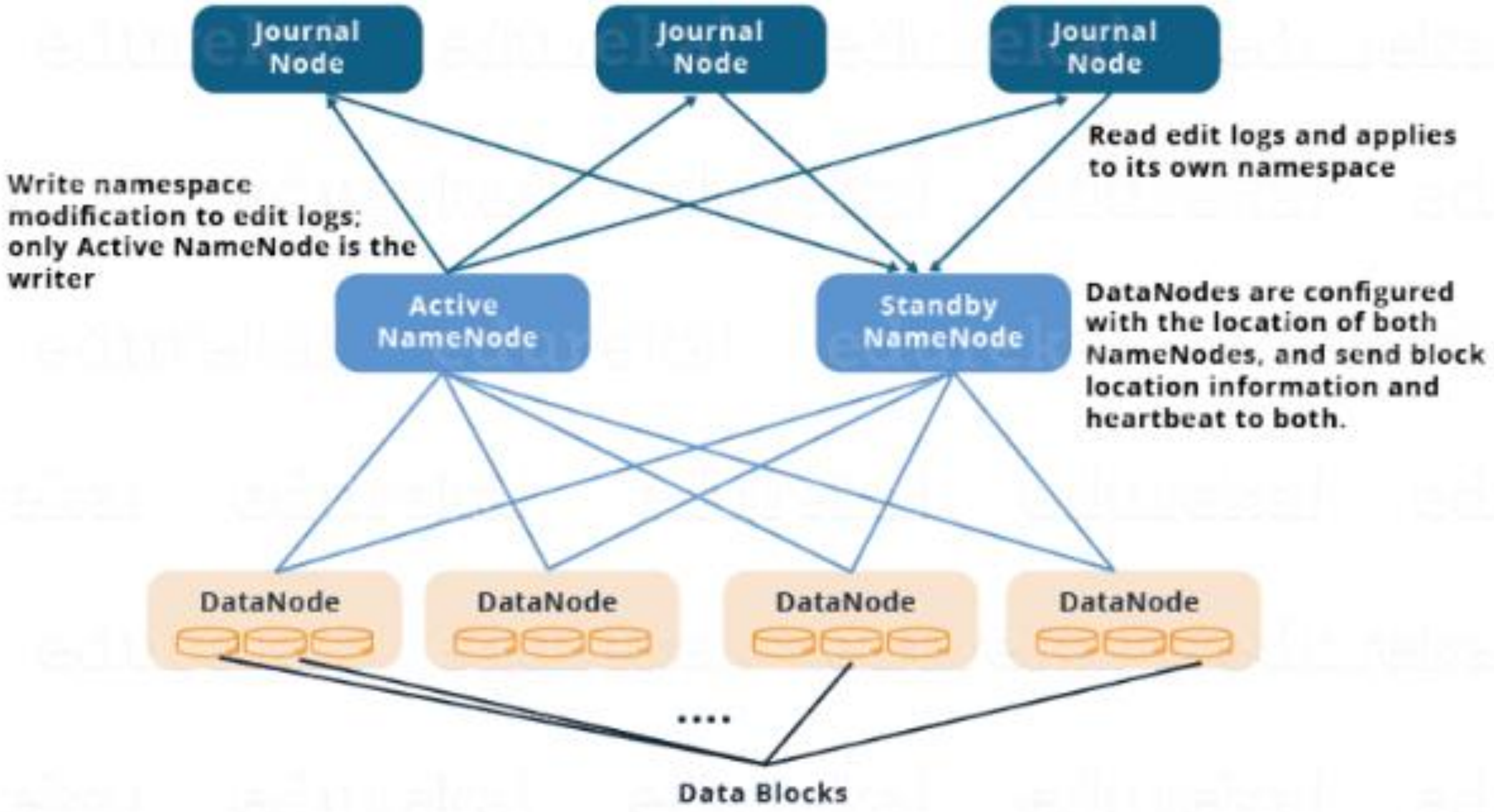
- The standby NameNode and the active NameNode keep in sync with each other through a separate group of nodes or daemons -called **JournalNodes**.
- The JournalNodes follows the ring topology where the nodes are connected to each other to form a ring. The JournalNode serves the request coming to it and copies the information into other nodes in the ring.
- This provides fault tolerance in case of JournalNode failure.
- The active NameNode is responsible for updating the EditLogs (metadata information) present in the JournalNodes.

- The StandbyNode reads the changes made to the EditLogs in the JournalNode and applies it to its own namespace in a constant manner.

- During failover, the StandbyNode makes sure that it has updated its meta data information from the JournalNodes before becoming the new Active NameNode. This makes the current namespace state synchronized with the state before failover.

- The IP Addresses of both the NameNodes are available to all the DataNodes and they send their heartbeats and block location information to both the NameNode.

- This provides a fast failover (less down time) as the StandbyNode has an updated information about the block location in the cluster.


Journal Node
Journal Node
Journal Node
Read edit logs and applies to its own namespace
Write namespace modification to edit logs; only Active NameNode is the writer
Active NameNode
Standby NameNode
DataNodes are configured with the location of both NameNodes, and send block location information and heartbeat to both.
DataNode
DataNode
DataNode
DataNode
....
Data Blocks

Fencing of NameNode:

- It is very important to ensure that there is only one Active NameNode at a time. So, fencing is a process to ensure this very property in a cluster.
- The JournalNodes performs this fencing by allowing only one NameNode to be the writer at a time.
- The Standby NameNode takes over the responsibility of writing to the JournalNodes and forbid any other NameNode to remain active.
- Finally, the new Active NameNode can perform its activities safely.

2. Using Shared Storage:

- The StandbyNode and the active NameNode keep in sync with each other by using a **shared storage device**.
- The active NameNode logs the record of any modification done in its namespace to an EditLog present in this shared storage.
- The StandbyNode reads the changes made to the EditLogs in this shared storage and applies it to its own namespace.
- Now, in case of failover, the StandbyNode updates its metadata information using the EditLogs in the shared storage at first.
- Then, it takes the responsibility of the Active NameNode. This makes the current namespace state synchronized with the state before failover.
- The administrator must configure at least one fencing method to avoid a split-brain scenario.

- The system may employ a range of fencing mechanisms. It may include killing of the NameNode’s process and revoking its access to the shared storage directory.

- As a last resort, we can fence the previously active NameNode with a technique known as STONITH, or “shoot the other node in the head”. STONITH uses a specialized power distribution unit to forcibly power down the NameNode machine.

Automatic Failover:
Failover is a procedure by which a system automatically transfers control to secondary system when it detects a fault or failure. There are two types of failover:

Graceful Failover: In this case, we manually initiate the failover for routine maintenance.

Automatic Failover: In this case, the failover is initiated automatically in case of NameNode failure (unplanned event).

Apache Zookeeper is a service that provides the automatic failover capability in HDFS High Availabilty cluster.

It maintains small amounts of coordination data, informs clients of changes in that data, and monitors clients for failures.

Zookeeper maintains a session with the NameNodes. In case of failure, the session will expire and the Zookeeper will inform other NameNodes to initiate the failover process.

In case of NameNode failure, other passive NameNode can take a lock in Zookeeper stating that it wants to become the next Active NameNode.

The ZookeerFailoverController (ZKFC) is a Zookeeper client that also monitors and manages the NameNode status.

Each of the NameNode runs a ZKFC also. ZKFC is responsible for monitoring the health of the NameNodes periodically.

Modes of hadoop cluster

1. Local Mode or Standalone Mode

- Standalone mode is the default mode in which Hadoop run. Standalone mode is mainly used for debugging where HDFS is not required.
- input and output files can be used as a local file system in standalone mode.
- Any custom configuration in the files- **mapred-site.xml, core-site.xml, hdfs-site.xml** is not required
- Standalone mode is usually the fastest Hadoop modes as it uses the local file system for all the input and output.

Summary-

- Default mode of hadoop
- HDFS is not utilized in this mode
- Local file system is used Inputs and Outputs
- Used for Debugging/Testing purpose
- No custom configuration is required such as (core-site.xml, hdfs-site.xml, mapred-site.xml, masters & slaves)
- Standalone mode is much faster than Pseudo-distributed mode.

2. Pseudo-distributed Mode

- The **pseudo-distribute mode** is also known as a **single-node cluster** where both NameNode and DataNode will reside on the same machine.
- All the Hadoop daemons will be running on a single node.
- Such configuration is mainly used while testing when we don’t need to think about the resources and other users sharing the resource.
- In this architecture, a separate JVM is spawned for every Hadoop components as they could communicate across network sockets, effectively producing a fully functioning and optimized mini-cluster on a single host.

Summary

- Custom configuration is required.
- Replication factor is set to one for HDFS.
- Used for real code to test in HDFS.
- HDFS is used for Inputs and outputs.
- All the demons should run in background such as (NameNode, JobTracker, DataNode, TaskTracker & Secondary NameNode)
- Good for testing, debugging and prototyping.

Fully-Distributed Mode (Multi-Node Cluster)

- This is the **production mode of Hadoop** where multiple nodes will be running. Here data will be distributed across several nodes and processing will be done on each node.
- Master and Slave services will be running on the separate nodes in fully-distributed Hadoop Mode.

Summary

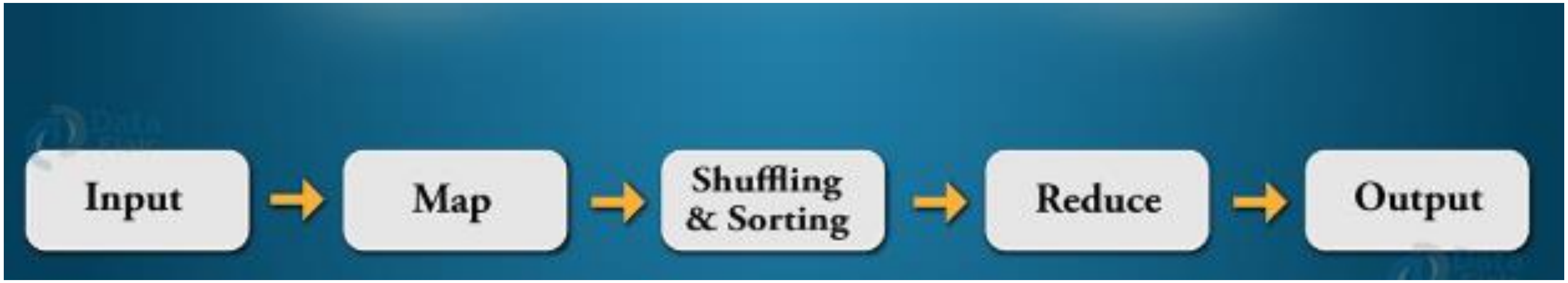
- This is a production phase.
- Replication factor is three.
- Data are used and distributed across many nodes.
- Different Nodes will be used as NameNode, JobTracker, DataNode & Secondary NameNode.
- NameNode, JobTracker, Secondary NameNode demons are run on master nodes.
- DataNode & TaskTracker demons runs on the slave nodes.

Mapreduce flow chart

1. Objective

- **Hadoop** MapReduce processes a huge amount of data in parallel by dividing the job into a set of independent tasks (sub-job).
- In Hadoop, MapReduce works by breaking the processing into phases: Map and Reduce.

Hadoop MapReduce Flow



2. What is MapReduce?

- **MapReduce** is the data processing layer of Hadoop.
- MapReduce is a programming paradigm designed for processing huge volumes of data in parallel by dividing the job (submitted work) into a set of independent tasks (sub-job).

3. How Hadoop MapReduce Works?

- **MapReduce** is the heart of **Hadoop**. It is a programming model designed for processing huge volumes of data (both structured as well as unstructured) in parallel by dividing the work into a set of independent sub-work (tasks).

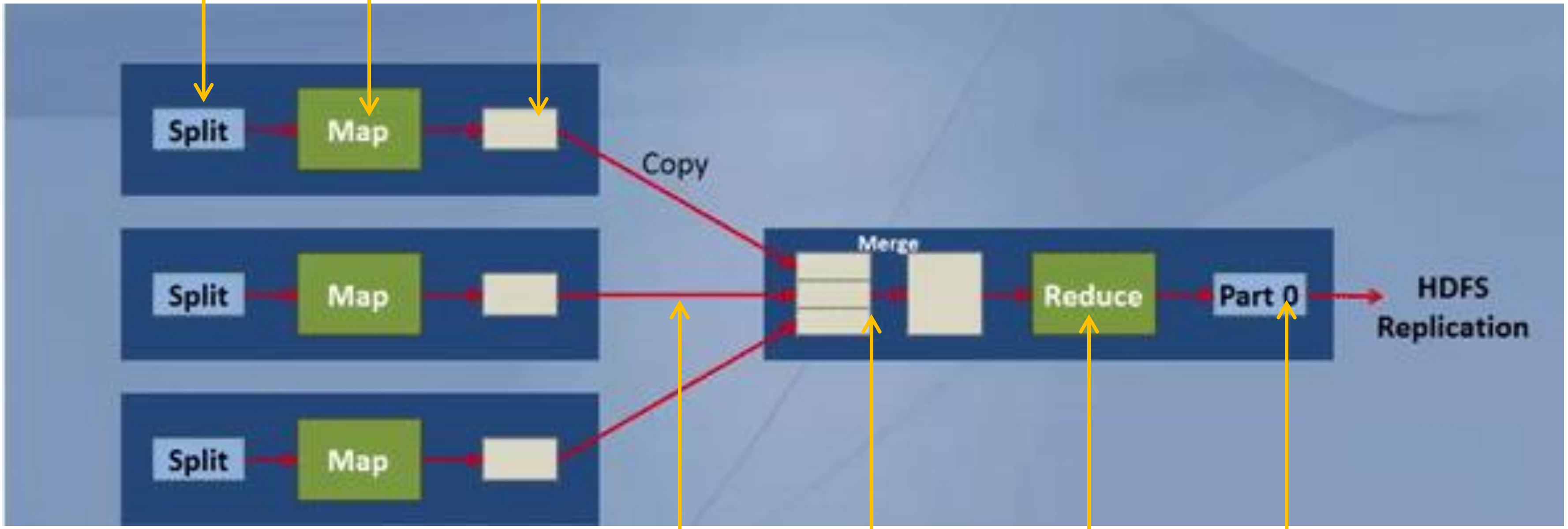
3.1. MapReduce Internals

- MapReduce is the combination of two different processing idioms called **Map**and **Reduce**, where we can specify our custom business logic.
- The map is the first phase of processing, where we specify all the complex logic/business rules/costly code.
- On the other hand, Reduce is the second phase of processing, where we specify light-weight processing.

Data Block stored in HDFS

First phase of processing

Intermediate output stored on local disc



Output stored on HDFS

Second Phase of processing

Data shuffling from mapper node to reduce node

Merge intermediate output from all mapper

3.2. Step of MapReduce Job Flow

- The data processed by MapReduce should be stored in **HDFS**, which divides the data into blocks and store in a distributed system

Step 1: One block is processed by one **mapper** at a time.

In the mapper, a developer can specify his own business logic as per the requirements.

In this manner, Map runs on all the nodes of the cluster and process the data blocks in parallel.

Step 2: Output of Mapper also known as intermediate output is written to the local disk.

An output of mapper is not stored on HDFS as this is temporary data and **writing on HDFS** will create unnecessary many copies.

Step 3: Output of mapper is shuffled to **reducer** node (which is a normal slave node but reduce phase will run here hence called as reducer node).

The shuffling/copying is a physical movement of data which is done over the network.

Step 4: Once all the mappers are finished and their output is shuffled on reducer nodes then this intermediate output is merged & sorted.

Which is then provided as input to reduce phase.

Step 5: Reduce is the second phase of processing where the user can specify his own custom business logic as per the requirements.

An input to a reducer is provided from all the mappers.

An output of reducer is the final output, which is written on HDFS.

Hence, in this manner, a map-reduce job is executed over the cluster. All the complexities of distributed processing are handled by the framework. The user just needs to concentrate on his own business requirements and write his custom code at specified phases (map and reduce).

3.3. Data Locality

- **Data locality** is one of the most innovative principles which says move the algorithm close to the data rather than moving the data.
- Since data is in the range of Petabytes. Movement of petabytes of data is not workable, hence algorithms / user-codes are moved to the location where data is present.
- If we summarize data locality – Movement of computation is cheaper than the movement of data.