Unit 4

Part 1

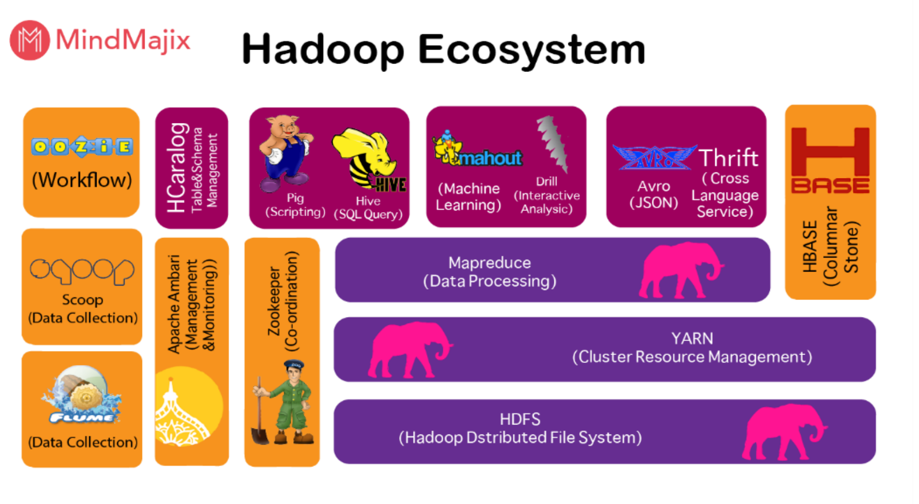
Hadoop Eco System and YARN: Hadoop ecosystem components, schedulers, fair and capacity, Hadoop 2.0 New Features - NameNode high availability, HDFS federation, MRv2, YARN, Running MRv1 in YARN.

Notes

Hadoop Ecosystem Components:

Hadoop ecosystem is a platform or framework which helps in solving the big data problems. It comprises of different components and services (ingesting, storing, analyzing, and maintaining) inside of it. Most of the services available in the Hadoop ecosystem are to supplement the main four core components of Hadoop which include HDFS, YARN, MapReduce and Common.

Hadoop ecosystem includes both Apache Open Source projects and other wide variety of commercial tools and solutions. Some of the well known open source examples include Spark, Hive, Pig, Sqoop and Oozie.



HDFS(Hadoop distributed file system)

The Hadoop distributed file system is a storage system which runs on Java programming language and used as a primary storage device in Hadoop applications. HDFS consists of two components, which are Namenode and Datanode; these applications are used to store large data across multiple nodes on the Hadoop cluster. First, let's discuss about the NameNode.

NameNode:

NameNode is a daemon which maintains and operates all DATA nodes (slave nodes).

It acts as the recorder of metadata for all blocks in it, and it contains information like size, location, source, and hierarchy, etc.

It records all changes that happen to metadata.

If any file gets deleted in the HDFS, the NameNode will automatically record it in EditLog.

NameNode frequently receives heartbeat and block report from the data nodes in the cluster to ensure they are working and live.

DataNode:

It acts as a slave node daemon which runs on each slave machine.

The data nodes act as a storage device.

It takes responsibility to serve read and write request from the user.

It takes the responsibility to act according to the instructions of NameNode, which includes deleting blocks, adding blocks, and replacing blocks.

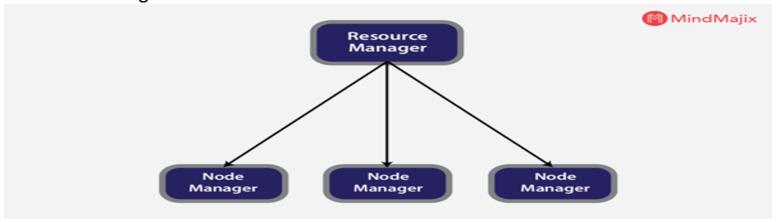
It sends heartbeat reports to the NameNode regularly and the actual time is once in every 3 seconds.

YARN:

YARN (Yet Another Resource Negotiator) acts as a brain of the Hadoop ecosystem. It takes responsibility in providing the computational resources needed for the application executions

YARN consists of two essential components. They are Resource Manager and Node Manager

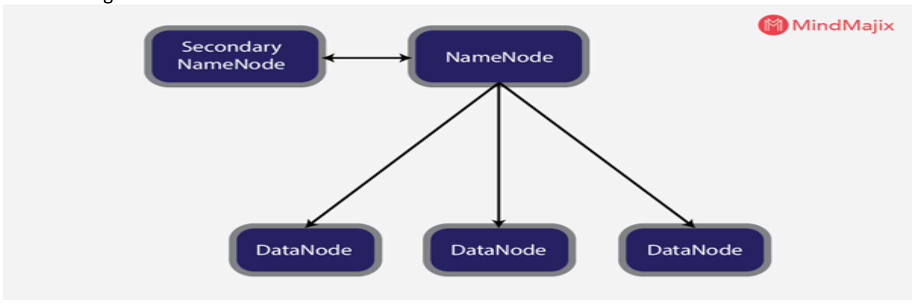
Resource Manager



It works at the cluster level and takes responsibility oforrunning the master machine.

It stores the track of heartbeats from the Node manager.

It takes the job submissions and negotiates the first container for executing an application.

It consists of two components: Application manager and Scheduler.
Node manager:


It works on node level component and runs on every slave machine.
It is responsible for monitoring resource utilization in each container and managing containers.
It also keeps track of log management and node health.
It maintains continuous communication with a resource manager to give updates.
MapReduce
MapReduce acts as a core component in Hadoop Ecosystem as it facilitates the logic of processing. To make it simple, MapReduce is a software framework which enables us in writing applications that process large data sets using distributed and parallel algorithms in a Hadoop environment.
Parallel processing feature of MapReduce plays a crucial role in Hadoop ecosystem. It helps in performing Big data analysis using multiple machines in the same cluster.
How does MapReduce work
In the MapReduce program, we have two Functions; one is Map, and the other is Reduce.

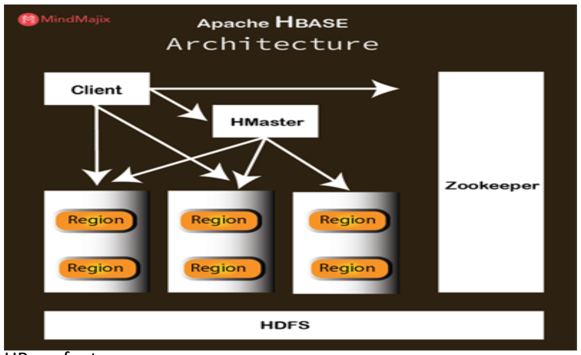
Map function: It converts one set of data into another, where individual elements are broken down into tuples. (key /value pairs).

Reduce function: It takes data from the Map function as an input. Reduce function aggregates & summarizes the results produced by Map function.
Apache Spark:
Apache Spark is an essential product from the Apache software foundation, and it is considered as a powerful data processing engine. Spark is empowering the big data applications around the world. It all started with the increasing needs of enterprises and where MapReduce is unable to handle them.
The growth of large unstructured amounts of data increased need for speed and to fulfill the real-time analytics led to the invention of Apache Spark.

Spark Features:
It is a framework for real-time analytics in a distributed computing environment.
It acts as an executor of in-memory computations which results in increased speed of data processing compared to MapReduce.
It is 100X faster than Hadoop while processing data with its exceptional in-memory execution ability and other optimization features.
Spark is equipped with high-level libraries, which support R, Python, Scala, Java etc. These standard libraries make the data processing seamless and highly reliable. Spark can process the enormous amounts of data with ease and Hadoop was designed to store the unstructured data which must be processed. When we combine these two, we get the desired results.

Hive:
Apache Hive is a data warehouse open source software built on Apache Hadoop for performing data query and analysis. Hive mainly does three functions; data summarization, query, and analysis. Hive uses a language called HiveQL(HQL), which is similar to SQL. Hive QL works as a translator which translates the SQL queries into MapReduce Jobs, which will be executed on Hadoop.
Main components of Hive are:
Metastore-   It serves as a storage device for the metadata. This metadata holds the information of each table such as location and schema. Metadata keeps track of data and replicates it, and acts as a backup store in case of data loss.
Driver- Driver receives the HiveQL instructions and acts as a Controller. It observes the progress and life cycle of various executions by creating sessions. Whenever HiveQL executes a statement, driver stores the metadata generated out of that action.
Compiler- The compiler is allocated with the task of converting the HiveQL query into MapReduce input. A compiler is designed with the process to execute the steps and functions needed to enable the HiveQL output, as required by the MapReduce.
H Base:

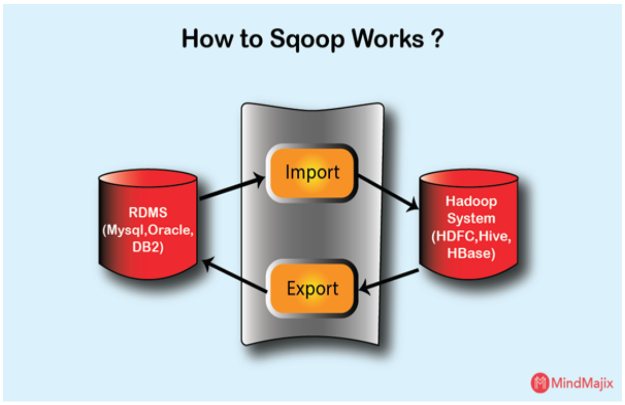
Hbase is considered as a Hadoop database, because it is scalable, distributed, and because NoSQL database that runs on top of Hadoop. Apache HBase is designed to store the structured data on table format which has millions of columns and billions of rows. HBase gives access to get the real-time data to read or write on HDFS.



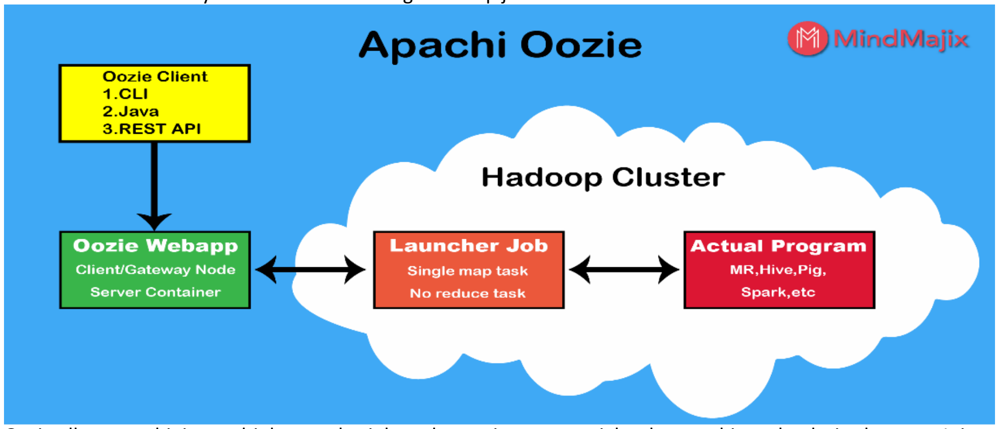
HBase features:
HBase is an open source, NoSQL database.
It is featured after Google's big table, which is considered as a distributed storage system designed to handle big data sets.
It has a unique feature to support all types of data. With this feature, it plays a crucial role in handling various types of data in Hadoop.
The HBase is originally written in Java, and its applications can be written in Avro, REST, and Thrift APIs.
Components of HBase:
There are majorly two components in HBase. They are HBase master and Regional server.

a) HBase master: It is not part of the actual data storage, but it manages load balancing activities across all RegionServers.
It controls the failovers.
Performs administration activities which provide an interface for creating, updating and deleting tables.
Handles DDL operations.
It maintains and monitors the Hadoop cluster.
b) Regional server: It is a worker node. It reads, writes, and deletes request from Clients. Region server runs on every node of Hadoop cluster. Its server runs on HDFS data nodes.

H Catalogue:
H Catalogue is a table and storage management tool for Hadoop. It exposes the tabular metadata stored in the hive to all other applications of Hadoop. H Catalogue accepts all kinds of components available in Hadoop such as Hive, Pig, and MapReduce to quickly read and write data from the cluster. H Catalogue is a crucial feature of Hive which allows users to store their data in any format and structure.
H Catalogue defaulted supports CSV, JSON, RCFile,ORC file from and sequenceFile formats.
Benefits of H Catalogue:
It assists the integration with the other Hadoop tools and provides read data from a Hadoop cluster or write data into a Hadoop cluster. It allows notifications of data availability.
It enables APIs and web servers to access the metadata from hive metastore.
It gives visibility for data archiving and data cleaning tools.
Apache Pig:
Apache Pig is a high-level language platform for analyzing and querying large data sets that are stored in HDFS. Pig works as an alternative language to Java programming for MapReduce and generates MapReduce functions automatically. Pig included with Pig Latin, which is a scripting language. Pig can translate the Pig Latin scripts into MapReduce which can run on YARN and process data in HDFS cluster.
Pig is best suitable for solving complex use cases that require multiple data operations. It is more like a processing language than a query language (ex:Java, SQL). Pig is considered as a highly customized one because the users have a choice to write their functions by using their preferred scripting language.
How does Pig work?
We use 'load' command to load the data in the pig. Then, we can perform various functions such as grouping data, filtering, joining, sorting etc. At last, you can dump the data on a screen, or you can store the result back in HDFS according to your requirement.
Apache Sqoop:
Sqoop works as a front-end loader of Big data. Sqoop is a front-end interface that enables in moving bulk data from Hadoop to relational databases and into variously structured data marts.
Sqoop replaces the function called 'developing scripts' to import and export data. It mainly helps in moving data from an enterprise database to Hadoop cluster to performing the ETL process.



What Sqoop does:
Apache Sqoop undertakes the following tasks to integrate bulk data movement between Hadoop and structured databases.
Sqoop fulfills the growing need to transfer data from the mainframe to HDFS.
Sqoop helps in achieving improved compression and light-weight indexing for advanced query performance.
It facilitates feature to transfer data parallelly for effective performance and optimal system utilization.
Sqoop creates fast data copies from an external source into Hadoop.
It acts as a load balancer by mitigating extra storage and processing loads to other devices.
Oozie:
Apache Ooze is a tool in which all sort of programs can be pipelined in a required manner to work in Hadoop's distributed environment. Oozie works as a scheduler system to run and manage Hadoop jobs.



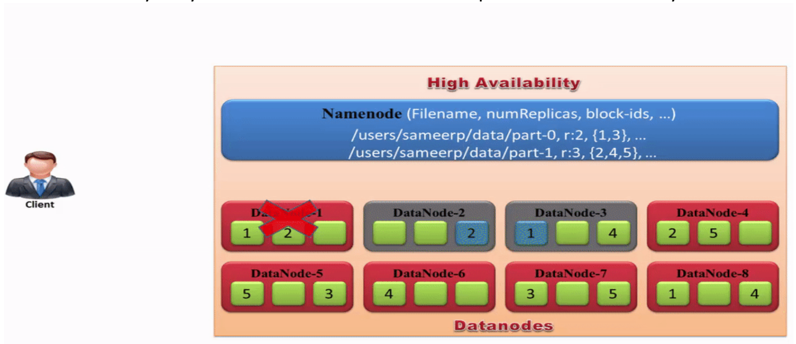
Oozie allows combining multiple complex jobs to be run in a sequential order to achieve the desired output. It is strongly integrated with Hadoop stack supporting various jobs like Pig, Hive, Sqoop, and system-specific jobs like Java, and Shell. Oozie is an open source Java web application.
Oozie consists of two jobs:
1. Oozie workflow: It is a collection of actions arranged to perform the jobs one after another. It is just like a relay race where one has to start right after one finish, to complete the race.
2. Oozie Coordinator: It runs workflow jobs based on the availability of data and predefined schedules.
Avro:
Apache Avro is a part of the Hadoop ecosystem, and it works as a data serialization system. It is an open source project which helps Hadoop in data serialization and data exchange. Avro enables big data in exchanging programs written in different languages. It serializes data into files or messages.
Avro Schema: Schema helps Avaro in serialization and deserialization process without code generation. Avro needs a schema for data to read and write. Whenever we store data in a file it's schema also stored along with it, with this the files may be processed later by any program.
Dynamic typing: it means serializing and deserializing data without generating any code. It replaces the code generation process with its statistically typed language as an optional optimization.
Avro features:
Avro makes Fast, compact, dynamic data formats.
It has Container file to store continuous data format.
It helps in creating efficient data structures.
Apache Drill :
The primary purpose of Hadoop ecosystem is to process the large sets of data either it is structured or unstructured. Apache Drill is the low latency distributed query engine which is designed to measure several thousands of nodes and query petabytes of data. The drill has a specialized skill to eliminate cache data and releases space.
Features of Drill:
It gives an extensible architecture at all layers.

Drill provides data in a hierarchical format which is easy to process and understandable.
The drill does not require centralized metadata, and the user doesn't need to create and manage tables in metadata to query data.
Apache Zookeeper:
Apache Zookeeper is an open source project designed to coordinate multiple services in the Hadoop ecosystem. Organizing and maintaining a service in a distributed environment is a complicated task. Zookeeper solves this problem with its simple APIs and Architecture. Zookeeper allows developers to focus on core application instead of concentrating on a distributed environment of the application.
Features of Zookeeper:
Zookeeper acts fast enough with workloads where reads to data are more common than writes.
Zookeeper acts as a disciplined one because it maintains a record of all transactions.
Apache Flume:
Flume collects, aggregates and moves large sets of data from its origin and send it back to HDFS. It works as a fault tolerant mechanism. It helps in transmitting data from a source into a Hadoop environment. Flume enables its users in getting the data from multiple servers immediately into Hadoop.
Apache Ambari:
Ambari is an open source software of Apache software foundation. It makes Hadoop manageable. It consists of software which is capable of provisioning, managing, and monitoring of Apache Hadoop clusters. Let's discuss each concept.
Hadoop cluster provisioning: It guides us with a step-by-step procedure on how to install Hadoop services across many hosts. Ambari handles configuration of Hadoop services across all clusters.
Hadoop Cluster management: It acts as a central management system for starting, stopping and reconfiguring of Hadoop services across all clusters.
Hadoop cluster monitoring: Ambari provides us with a dashboard for monitoring health and status.
The Ambari framework acts as an alarming system to notify when anything goes wrong. For example, if a node goes down or low disk space on node etc, it intimates us through notification.

2. SCHEDULERS:
Three schedulers are available in YARN: the FIFO, Capacity, and Fair Schedulers. The FIFO Scheduler places applications in a queue and runs them in the order of submission (first in, first out). Requests for the first application in the queue are allocated first; once its requests have been satisfied, the next application in the queue is served, and so on. The FIFO Scheduler has the merit of being simple to understand and not needing any configuration, but it's not suitable for shared clusters. Large applications will use all the resources in a cluster, so each application has to wait its turn. On a shared cluster it is better to use the Capacity Scheduler or the Fair Scheduler. Both of these allow longrunning jobs to complete in a timely manner, while still allowing users who are running concurrent smaller ad hoc queries to get results back in a reasonable time.
The difference between schedulers is illustrated , which shows that under the FIFO Scheduler (i) the small job is blocked until the large job completes. With the Capacity Scheduler, a separate dedicated queue allows the small job to start as soon as it is submitted, although this is at the cost of overall cluster utilization since the queue capacity is reserved for jobs in that queue. This means that the large job finishes later than when using the FIFO Scheduler. With the Fair Scheduler, there is no need to reserve a set amount of capacity, since it will dynamically balance resources between all running jobs. Just after the first (large) job starts, it is the only job running, so it gets all the resources in the cluster. When the second (small) job starts, it is allocated half of the cluster resources so that each job is using its fair share of resources. Note that there is a lag between the time the second job starts and when it receives its fair share, since it has to wait for resources to free up as containers used by the first job complete. After the small job completes and no longer requires resources, the large job goes back to using the full cluster capacity again. The overall effect is both high cluster utilization and timely small job completion.

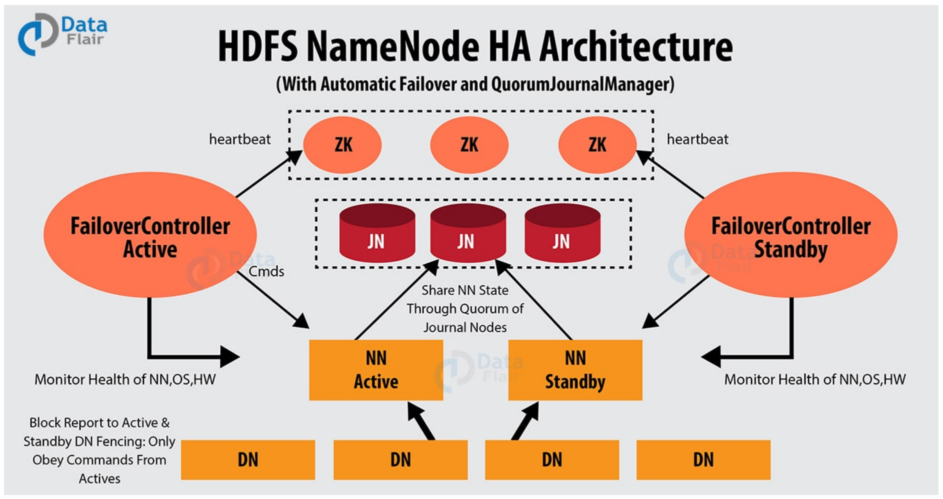
3. Hadoop 2.0 New Features - NameNode high availability
High Availability was a new feature added to Hadoop 2.x to solve the Single point of failure problem in the older versions of Hadoop.
As the Hadoop HDFS follows the master-slave architecture where the NameNode is the master node and maintains the filesystem tree. So HDFS cannot be used without NameNode. This NameNode becomes a bottleneck. HDFS high availability feature addresses this issue.

What is high availability?
High availability refers to the availability of system or data in the wake of component failure in the system.


High Availability
Namenode (Filename, numReplicas, block-ids, ...)
/users/sameerp/data/part-0, r:2, {1,3}, ...
/users/sameerp/data/part-1, r:3, {2,4,5}, ...
Client
DataNode-1
DataNode-2
DataNode-3
DataNode-4
DataNode-5
DataNode-6
DataNode-7
DataNode-8
Datanodes

High availability in Hadoop
The high availability feature in Hadoop ensures the availability of the Hadoop cluster without any downtime, even in unfavorable conditions like NameNode failure, DataNode failure, machine crash, etc.
It means if the machine crashes, data will be accessible from another path.

How Hadoop HDFS achieves High Availability?
As we know, HDFS (Hadoop distributed file system) is a distributed file system in Hadoop. HDFS stores users' data in files and internally, the files are split into fixed-size blocks. These blocks are stored on DataNodes. NameNode is the master node that stores the metadata about file system i.e. block location, different blocks for each file, etc.
1. Availability if DataNode fails
In HDFS, replicas of files are stored on different nodes.
DataNodes in HDFS continuously sends heartbeat messages to NameNode every 3 seconds by default.
If NameNode does not receive a heartbeat from DataNode within a specified time (10 minutes by default), the NameNode considers the DataNode to be dead.
NameNode then checks for the data in DataNode and initiates data replication. NameNode instructs the DataNodes containing a copy of that data to replicate that data on other DataNodes.
Whenever a user requests to access his data, NameNode provides the IP of the closest DataNode containing user data. Meanwhile, if DataNode fails, the NameNode redirects the user to the other DataNode containing a copy of the same data. The user requesting for data read, access the data from other DataNodes containing a copy of data, without any downtime. Thus cluster is available to the user even if any of the DataNodes fails.
2. Availability if NameNode fails
NameNode is the only node that knows the list of files and directories in a Hadoop cluster. **"The filesystem cannot be used without NameNode".**
The addition of the High Availability feature in Hadoop 2 provides a fast failover to the Hadoop cluster. The Hadoop HA cluster consists of two NameNodes (or more after Hadoop 3) running in a cluster in an active/passive configuration with a hot standby. So, if an active node fails, then a passive node becomes the active NameNode, takes the responsibility, and serves the client request.
This allows for the fast failover to the new machine even if the machine crashes.
Thus, data is available and accessible to the user even if the NameNode itself goes down.
Let us now study the NameNode High Availability in detail.
Before going to NameNode High Availability architecture, one should know the reason for introducing such architecture.
Reason for introducing NameNode High Availability Architecture
Prior to Hadoop 2.0, NameNode is the **single point of failure** in a Hadoop cluster. This is because:
1. Each cluster consists of only one NameNode. If the NameNode fails, then the whole cluster would go down. The cluster would be available only when we either restart the NameNode or bring it on a separate machine. These had limited availability in two ways:
The cluster would be unavailable if the machine crash until an operator restarts the NameNode.
Planned maintenance events such as software or hardware upgrades on the NameNode, results in downtime of the Hadoop cluster.
2. The time taken by NameNode to start from cold on large clusters with many files can be 30 minutes or more. This long recovery time is a problem.
To overcome these problems Hadoop High Availability architecture was introduced in Hadoop 2.
Hadoop NameNode High Availability Architecture


Data Flair
HDFS NameNode HA Architecture
(With Automatic Failover and QuorumJournalManager)
heartbeat
ZK
ZK
ZK
heartbeat
FailoverController Active
Cmds
JN
JN
JN
FailoverController Standby
Share NN State Through Quorum of Journal Nodes
NN Active
NN Standby
Monitor Health of NN,OS,HW
Monitor Health of NN,OS,HW
Block Report to Active & Standby DN Fencing: Only Obey Commands From Actives
DN
DN
DN
DN
HDFS NameNode High Availability Architecture
Hadoop 2.0 overcomes this **SPOF** by providing support for many NameNode. HDFS NameNode High Availability architecture provides the option of running two redundant NameNodes in the same cluster in an active/passive configuration with a hot standby.
Active NameNode – It handles all client operations in the cluster.
Passive NameNode – It is a standby namenode, which has similar data as active NameNode. It acts as a slave, maintains enough state to provide a fast failover, if necessary.
If Active NameNode fails, then passive NameNode takes all the responsibility of active node and cluster continues to work.
Issues in maintaining consistency in the HDFS High Availability cluster are as follows:
Active and Standby NameNode should always be in sync with each other, i.e. they should have the same metadata. This permit to reinstate the **Hadoop cluster** to the same namespace state where it got crashed. And this will provide us to have fast failover.

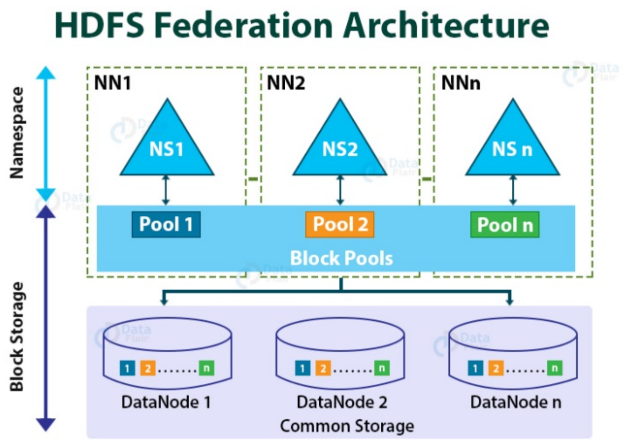
There should be only one NameNode active at a time. Otherwise, two NameNode will lead to corruption of the data. We call this scenario as a "**Split-Brain Scenario**", where a cluster gets divided into the smaller cluster. Each one believes that it is the only active cluster. "**Fencing**" avoids such scenarios. Fencing is a process of ensuring that only one NameNode remains active at a particular time.
After Hadoop High availability architecture let us have a look at the implementation of Hadoop NameNode high availability.

Implementation of NameNode High Availability Architecture

In HDFS NameNode High Availability Architecture, two NameNodes run at the same time. We can Implement the Active and Standby NameNode configuration in following two ways:
Using Quorum Journal Nodes
Using Shared Storage
A. Using Quorum Journal Nodes
QJM is an HDFS implementation. It is designed to provide edit logs. It allows sharing these edit logs between the active namenode and standby namenode.
For **High Availability**, standby namenode communicates and synchronizes with the active namenode. It happens through a group of nodes or daemons called "**Journal nodes**". The QJM runs as a group of journal nodes. There should be at least three journal nodes.
For N journal nodes, the system can tolerate at most (N-1)/2 failures and continue to function. So, for three journal nodes, the system can tolerate the failure of one {(3-1)/2} of them.
When an active node performs any modification, it logs modification to all journal nodes.
The standby node reads the edits from the journal nodes and applies to its own namespace in a constant manner. In the case of failover, the standby will ensure that it has read all the edits from the journal nodes before promoting itself to the Active state. This ensures that the namespace state is completely synchronized before a failure occurs.
To provide a fast failover, the standby node must have up-to-date information about the location of **data blocks** in the cluster. For this to happen, IP address of both the namenode is available to all the datanodes and they send block location information and heartbeats to both NameNode.
Fencing of NameNode
For the correct operation of an HA cluster, only one of the namenodes should active at a time. Otherwise, the namespace state would deviate between the two namenodes. So, fencing is a process to ensure this property in a cluster.
The journal nodes perform this fencing by allowing only one namenode to be the writer at a time.
The standby namenode takes the responsibility of writing to the journal nodes and prohibit any other namenode to remain active.
Finally, the new active namenode can perform its activities.
Learn: What is HDFS Disk Balancer
B. Using Shared Storage
Standby and active namenode synchronize with each other by using "shared storage device". For this implementation, both active namenode and standby namenode must have access to the particular directory on the shared storage device (.i.e. Network file system).
When active namenode perform any namespace modification, it logs a record of the modification to an edit log file stored in the shared directory. The standby namenode watches this directory for edits, and when edits occur, the standby namenode applies them to its own namespace. In the case of failure, the standby namenode will ensure that it has read all the edits from the shared storage before promoting itself to the Active state. This ensures that the namespace state is completely synchronized before failover occurs.
To prevent the "split-brain scenario" in which the namespace state deviates between the two namenode, an administrator must configure at least one fencing method for the shared storage.

4**. HDFS federation:**

The namenode keeps a reference to every file and block in the filesystem in memory, which means that on very large clusters with many files, memory becomes the limiting factor for scaling (see "How Much Memory Does a Namenode Need?" on page 294). HDFS federation, introduced in the 2.x release series, allows a cluster to scale by adding namenodes, each of which manages a portion of the filesystem namespace. For example, one namenode might manage all the files rooted under /user, say, and a second name- node might handle files under /share.
Under federation, each namenode manages a namespace volume, which is made up of the metadata for the namespace, and a block pool containing all the blocks for the files in the namespace. Namespace volumes are independent of each other, which means namenodes do not communicate with one another, and furthermore the failure of one namenode does not affect the availability of the namespaces managed by other namen- odes. Block pool storage is not partitioned, however, so datanodes register with each namenode in the cluster and store blocks from multiple block pools.
To access a federated HDFS cluster, clients use client-side mount tables to map file paths to namenodes. This is managed in configuration using ViewFileSystem and the viewfs:// URIs.
HDFS Federation Architecture
In HDFS Federation architecture, there are **multiple NameNodes** and **DataNodes**.
Each NameNode has its own namespace and **block pool**.
All the NameNodes uses DataNodes as the common storage.
Every NameNode is independent of the other and does not require any coordination amongst themselves.
Each Datanode gets registered to all the NameNodes in the cluster and store blocks for all the block pools in the cluster.
Also, DataNodes periodically send heartbeats and block reports to all the NameNode in the cluster and handles the instructions from the NameNodes.
Look at the figure below that shows the architecture design of the HDFS Federation.



In the above figure, which represents HDFS Federation architecture, there are multiple NameNodes which are represented as NN1, NN2, ..NNn.
NS1, NS2, and so on are the multiple namespaces managed by their respective NameNode (NS1 by NN1, NS2 by NN2, and so on).
Each namespace has its own block pool (NS1 has Pool1, NS2 has Pool2, and so on).
Each Datanode store blocks for all the block pools in the cluster.
For example, DataNode1 stores the blocks from Pool 1, Pool 2, Pool3, etc.
Let us now understand the block pool and namespace volume in detail.
Block pool
Block pool in HDFS Federation architecture is the collection of **blocks** belonging to the single namespace. HDFS Federation architecture has a collection of block pools, and each block pool is managed independently from each other. This allows the generation of the Block IDs for new blocks by the namespace, without any coordination with other namespaces.
Namespace Volume
Namespace with its block pool is termed as Namespace Volume. The HDFS Federation architecture has the collection of Namespace volume, which is a self-contained management unit. On deleting the NameNode or namespace, the corresponding block pool present in the DataNodes also gets deleted. On upgrading the cluster, each namespace volume gets upgraded as a unit.

Benefits of HDFS Federation

I. . Namespace Scalability

With federation, we can horizontally scale the namespace. This benefits the large clusters or cluster with too many small files because of more NameNode addition to the cluster.

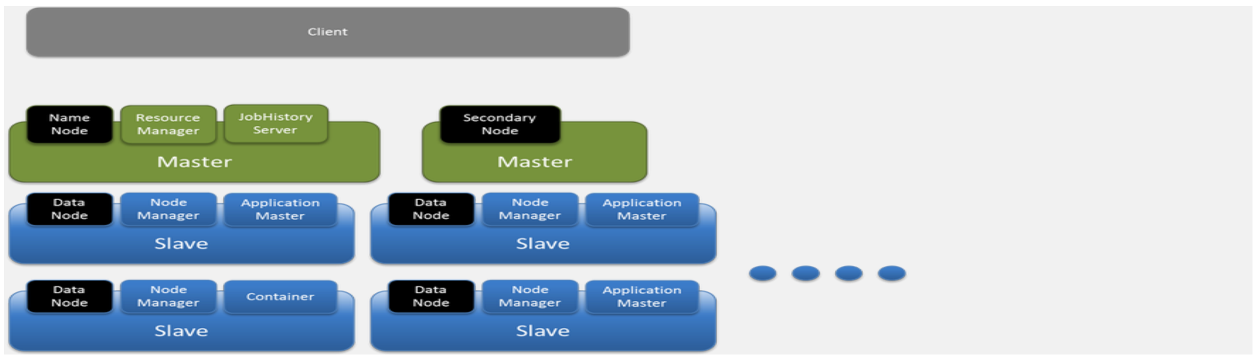
II. PerformanceS

It improves the performance of the filesystem as the filesystem operations are not limited by the throughput of a single NameNode.

III. Isolation

Due to multiple namespaces, it can provide isolation to the occupant organizations that are using the cluster.

4. MRV2:
MapReduce 2.0 has two components – YARN that has cluster resource management capabilities and MapReduce.



MapReduce 2.0

In MapReduce 2.0, the JobTracker is divided into three services:

- ResourceManager, a persistent YARN service that receives and runs applications on the cluster. A MapReduce job is an application.

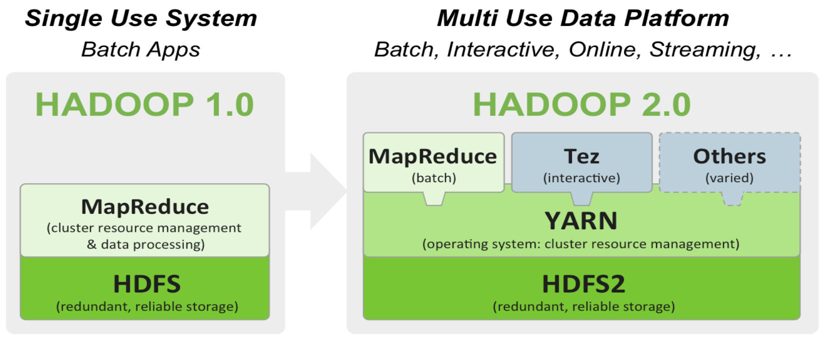
- JobHistoryServer, to provide information about completed jobs
- Application Master, to manage each MapReduce job and is terminated when the job completes.

Also, the TaskTracker has been replaced with the NodeManager, a YARN service that manages resources and deployment on a node. NodeManager is responsible for launching containers that could either be a map or reduce task.

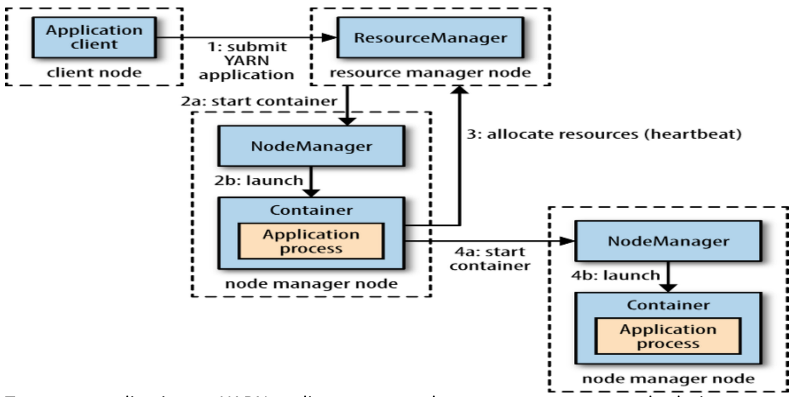
This new architecture breaks JobTracker model by allowing a new ResourceManager to manage resource usage across applications, with ApplicationMasters taking the responsibility of managing the execution of jobs. This change removes a bottleneck and let Hadoop clusters scale up to larger configurations than 4000 nodes. This architecture also allows simultaneous execution of a variety of programming models such as graph processing, iterative processing, machine learning, and general cluster computing, including the traditional MapReduce.

5. YARN:

Apache YARN (Yet Another Resource Negotiator) is Hadoop's cluster resource man- agement system. YARN was introduced in Hadoop 2 to improve the MapReduce im- plementation, but it is general enough to support other distributed computing para- digms as well.
YARN provides APIs for requesting and working with cluster resources, but these APIs are not typically used directly by user code. Instead, users write to higher-level APIs provided by distributed computing frameworks, which themselves are built on YARN and hide the resource management details from the user. The situation is illustrated in Figure, which shows some distributed computing frameworks (MapReduce, Spark, and so on) running as YARN applications on the cluster compute layer (YARN) and the cluster storage layer (HDFS and HBase).


Single Use System
Batch Apps
Multi Use Data Platform
Batch, Interactive, Online, Streaming, ...
HADOOP 1.0
MapReduce
(cluster resource management & data processing)
HDFS
(redundant, reliable storage)
HADOOP 2.0
MapReduce
(batch)
Tez
(interactive)
Others
(varied)
YARN
(operating system: cluster resource management)
HDFS2
(redundant, reliable storage)

Anatomy of a YARN Application Run
YARN provides its core services via two types of long-running daemon: a resource manager (one per cluster) to manage the use of resources across the cluster, and node managers running on all the nodes in the cluster to launch and monitor containers. A container executes an application-specific process with a constrained set of resources (memory, CPU, and so on). Depending on how YARN is configured.



To run an application on YARN, a client contacts the resource manager and asks it to run an application master process. The resource manager then finds a node manager that can launch the application master in a container. Precisely what the application master does once it is running depends on the application. It could simply run a computation in the container it is running in and return the result to the client. Or it could request more containers from the resource managers (step 3), and use them to run a distributed computation
6. YARN Compared to MapReduce 1
In MapReduce 1, there are two types of daemon that control the job execution process: a jobtracker and one or more tasktrackers. The jobtracker coordinates all the jobs run on the system by scheduling tasks to run on tasktrackers. Tasktrackers run tasks and send progress reports to the jobtracker, which keeps a record of the overall progress of each job. If a task fails, the jobtracker can reschedule it on a different tasktracker.
In MapReduce 1, the jobtracker takes care of both job scheduling (matching tasks with tasktrackers) and task progress monitoring (keeping track of tasks, restarting failed or slow tasks, and doing task bookkeeping, such as maintaining counter totals). By con- trast, in YARN these

responsibilities are handled by separate entities: the resource man- ager and an application master (one for each MapReduce job). The jobtracker is also responsible for storing job history for completed jobs, although it is possible to run a job history server as a separate daemon to take the load off the jobtracker. In YARN, the equivalent role is the timeline server, which stores application history.

YARN was designed to address many of the limitations in MapReduce 1. The benefits to using YARN include the following:

Scalability

YARN can run on larger clusters than MapReduce 1. MapReduce 1 hits scalability bottlenecks in the region of 4,000 nodes and 40,000 tasks,6 stemming from the fact that the jobtracker has to manage both jobs and tasks. YARN overcomes these limitations by virtue of its split resource manager/application master architecture: it is designed to scale up to 10,000 nodes and 100,000 tasks.

Availability

High availability (HA) is usually achieved by replicating the state needed for another daemon to take over the work needed to provide the service, in the event of the service daemon failing. However, the large amount of rapidly changing complex state in the jobtracker's memory (each task status is updated every few seconds, for example) makes it very difficult to retrofit HA into the jobtracker service.

With the jobtracker's responsibilities split between the resource manager and ap- plication master in YARN, making the service highly available became a divideand-conquer problem: provide HA for the resource manager, then for YARN ap- plications (on a per-application basis). And indeed, Hadoop 2 supports HA both for the resource manager and for the application master for MapReduce jobs.

Utilization

In MapReduce 1, each tasktracker is configured with a static allocation of fixed-size "slots," which are divided into map slots and reduce slots at configuration time. A map slot can only be used to run a map task, and a reduce slot can only be used for a reduce task.

In YARN, a node manager manages a pool of resources, rather than a fixed number of designated slots. MapReduce running on YARN will not hit the situation where a reduce task has to wait because only map slots are available on the cluster, which can happen in MapReduce 1. If the resources to run the task are available, then the application will be eligible for them.

Multitenancy

In some ways, the biggest benefit of YARN is that it opens up Hadoop to other types of distributed application beyond MapReduce. MapReduce is just one YARN ap- plication among many.

It is even possible for users to run different versions of MapReduce on the same YARN cluster, which makes the process of upgrading MapReduce more manage- able.