Unit 3

Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to **scale up from single servers to thousands of machines**, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the **library itself** is designed to **detect and handle failures at the application layer**, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

Features of Hadoop Which Makes It Popular

1. Open Source:

Hadoop is open-source, which means it is free to use. Since it is an open-source project the source-code is available online for anyone to understand it or make some modifications as per their industry requirement.

2. Highly Scalable Cluster:

Hadoop is a highly scalable model. A large amount of data is divided into multiple inexpensive machines in a cluster which is processed parallelly. the number of these machines or nodes can be increased or decreased as per the enterprise's requirements. In traditional RDBMS(Relational DataBase Management System) the systems can not be scaled to approach large amounts of data.

3. Fault Tolerance is Available:

Hadoop uses commodity hardware(inexpensive systems) which can be crashed at any moment. In Hadoop data is replicated on various DataNodes in a Hadoop cluster which ensures the availability of data if somehow any of your systems got crashed. You can read all of the data from a single machine if this machine faces a technical issue data can also be read from other nodes in a Hadoop cluster because the data is copied or replicated by default. By default, Hadoop makes 3 copies of each file block and stored it into different nodes. This replication factor is configurable and can be changed by changing the replication property in the hdfs-site.xml file.

4. High Availability is Provided:

Fault tolerance provides High Availability in the Hadoop cluster. High Availability means the availability of data on the Hadoop cluster. Due to fault tolerance in case if any of the DataNode goes down the same data can be retrieved from any other node where the data is replicated. The High available Hadoop cluster also has 2 or more than two Name Node i.e. Active NameNode and Passive NameNode also known as stand by NameNode. In case if Active NameNode fails then the Passive node will take the responsibility of Active Node and provide the same data as that of Active NameNode which can easily be utilized by the user.

5. Cost-Effective:

Hadoop is open-source and uses cost-effective commodity hardware which provides a cost-efficient model, unlike traditional Relational databases that require expensive hardware and high-end processors to deal with Big Data. The problem with traditional Relational databases is that storing the Massive volume of data is not cost-effective, so the company's started to remove the Raw data. which may not result in the correct

scenario of their business. Means Hadoop provides us 2 main benefits with the cost one is it's open-source means free to use and the other is that it uses commodity hardware which is also inexpensive.

6. Hadoop Provide Flexibility:

Hadoop is designed in such a way that it can deal with any kind of dataset like structured(MySql Data), Semi-Structured(XML, JSON), Un-structured (Images and Videos) very efficiently. This means it can easily process any kind of data independent of its structure which makes it highly flexible. It is very much useful for enterprises as they can process large datasets easily, so the businesses can use Hadoop to analyze valuable insights of data from sources like social media, email, etc. With this flexibility, Hadoop can be used with log processing, Data Warehousing, Fraud detection, etc.

7. Easy to Use:

Hadoop is easy to use since the developers need not worry about any of the processing work since it is managed by the Hadoop itself. Hadoop ecosystem is also very large comes up with lots of tools like Hive, Pig, Spark, HBase, Mahout, etc.

8. Hadoop uses Data Locality:

The concept of Data Locality is used to make Hadoop processing fast. In the data locality concept, the computation logic is moved near data rather than moving the data to the computation logic. The cost of Moving data on HDFS is costliest and with the help of the data locality concept, the bandwidth utilization in the system is minimized.

9. Provides Faster Data Processing:

Hadoop uses a distributed file system to manage its storage i.e. HDFS(Hadoop Distributed File System). In DFS(Distributed File System) a large size file is broken into small size file blocks then distributed among the Nodes available in a Hadoop cluster, as this massive number of file blocks are processed parallelly which makes Hadoop faster, because of which it provides a High-level performance as compared to the traditional DataBase Management Systems.

10. Support for Multiple Data Formats:

Hadoop supports multiple data formats like CSV, JSON, Avro, and more, making it easier to work with different types of data sources. This makes it more convenient for developers and data analysts to handle large volumes of data with different formats.

11. High Processing Speed:

Hadoop's distributed processing model allows it to process large amounts of data at high speeds. This is achieved by distributing data across multiple nodes and processing it in parallel. As a result, Hadoop can process data much faster than traditional database systems.

12. Machine Learning Capabilities:

Hadoop offers machine learning capabilities through its ecosystem tools like Mahout, which is a library for creating scalable machine learning applications. With these tools, data analysts and developers can build machine learning models to analyze and process large datasets.

13. Integration with Other Tools:

Hadoop integrates with other popular tools like Apache Spark, Apache Flink, and Apache Storm, making it easier to build data processing pipelines. This integration allows developers and data analysts to use their favorite tools and frameworks for building data pipelines and processing large datasets.

14. Secure:

Hadoop provides built-in security features like authentication, authorization, and encryption. These features help to protect data and ensure that only authorized users have access to it. This makes Hadoop a more secure platform for processing sensitive data.

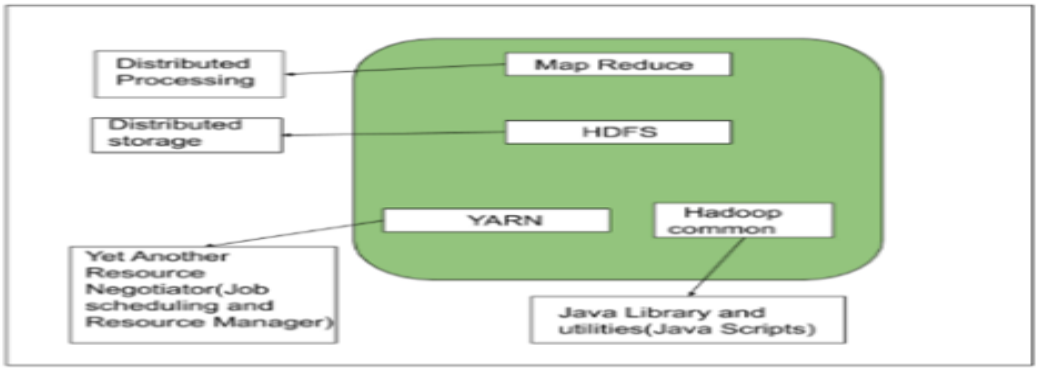
15. Community Support:

Hadoop has a large community of users and developers who contribute to its development and provide support to users. This means that users can access a wealth of resources and support to help them get the most out of Hadoop.

Hadoop is a framework **written in Java** that utilizes a large cluster of commodity hardware to **maintain and store big size data**. Hadoop **works** on **MapReduce Programming Algorithm** that was introduced by Google.

The Hadoop Architecture Mainly consists of 4 components.

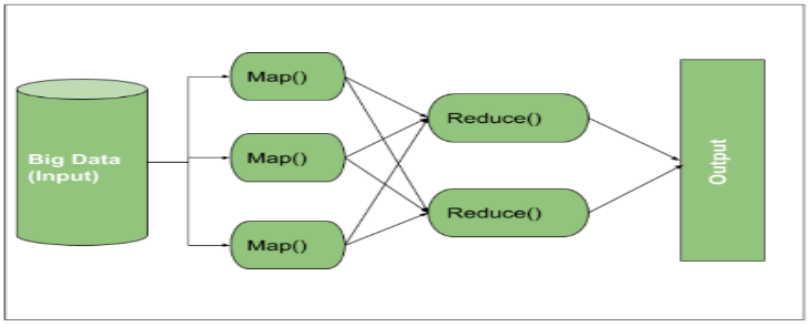
- MapReduce
- HDFS(Hadoop Distributed File System)
- YARN(Yet Another Resource Negotiator)
- Common Utilities or Hadoop Common



1. MapReduce

MapReduce nothing but **just like** an **Algorithm or a data structure** that is based on the **YARN framework**. The major feature of MapReduce is to perform the **distributed processing in parallel** in a Hadoop cluster which Makes Hadoop working so fast. When you are dealing with Big Data, **serial processing is no more** of any use. MapReduce has mainly 2 tasks which are divided phase-wise:

In first phase, Map is utilized and in next phase Reduce is utilized.



Here, we can see that the Input is provided to the Map() function then it's output is used as an input to the Reduce function and after that, we receive our final output. Let's understand What this Map() and Reduce() does.
As we can see that an Input is provided to the Map(), now as we are using Big Data. **The Input is a set of Data.** The Map() function here **breaks this DataBlocks into Tuples** that are nothing but a **key-value pair.** These key-value pairs are now sent as input to the Reduce(). The Reduce() function then **combines this broken Tuples or key-value pair based on its Key value and form set of Tuples**, and perform some operation like sorting, summation type job, etc. which is then sent to the final Output Node. Finally, the Output is Obtained.
The data processing is always done in Reducer depending upon the business requirement of that industry. This is How First Map() and then Reduce is utilized one by one.

Let's understand the Map Task and Reduce Task in detail.

Map Task:

- **RecordReader** The purpose of *recordreader* is to **break the records**. It is responsible for providing key-value pairs in a Map() function. The key is actually is its locational information and value is the data associated with it.
- **Map:** A map is nothing but a **user-defined function** whose work is to **process the Tuples** obtained from **record reader**. The Map() function either does not generate any key-value pair or generate multiple pairs of these tuples.
- **Combiner:** Combiner is used for grouping the data in the Map workflow. It is similar to a Local reducer. The **intermediate key-value that are generated in the Map is combined with the help of this combiner**. Using a combiner is **not necessary** as it is **optional.**
- **Partitionar:** Partitional is responsible for **fetching key-value** pairs **generated in the Mapper Phases**. The partitioner generates the shards (Broken pieces) corresponding to each reducer. Hashcode of each key is also fetched by this partition. Then partitioner performs it's(Hashcode) modulus with the number of reducers(*key.hashcode()%(number of reducers)).*

Reduce Task
Shuffle and Sort: The Task of Reducer starts with this step, the process in which the Mapper generates the intermediate key-value and transfers them to the Reducer task is known as ***Shuffling*. Using the Shuffling process the system can sort the data using its key value.**

Once some of the Mapping tasks are done Shuffling begins that is why it is a faster process and does not wait for the completion of the task performed by Mapper.

- **Reduce:** The main function or task of the Reduce is to **gather the Tuple generated** from Map and then perform some **sorting and aggregation sort of process on those key-value depending on its key element**.
- **OutputFormat:** Once all the operations are performed, the **key-value pairs are written into the file with** the help of **record writer**, each record in a new line, and the key and value in a space-separated manner.

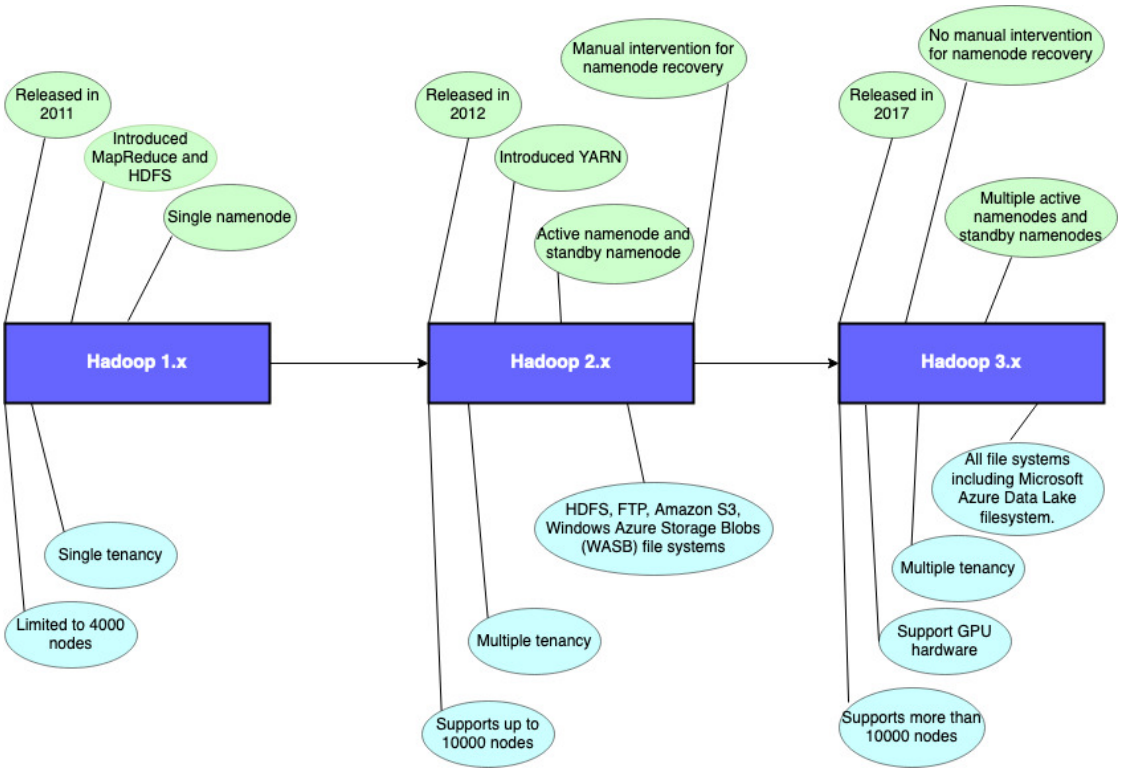
Comparison between Hadoop 1.X vs Hadoop 2.X vs Hadoop 3.X

Hadoop 1.X	**Hadoop 2.X**	**Hadoop 3.X**
Hadoop 1.x was released in 2011	Hadoop 2.x released in 2012	Hadoop 3.x released in 2017
It introduced MapReduce and HDFS. That is to say, the MapReduce frameowrk is used as data processing and for resource management also.	YARN (Yet another resource negotiator) added for better resource management. As a result, it enabled multi-tenancy. Therefore, the same cluster can be used by MapReduce as well as by some other processes using YARN.	In Hadoop 3.x, the YARN resource model is generalized to support user-defined resource types beyond CPU and memory. For example, the administrator can define resources like GPUs, software licenses, or locally-attached storage. YARN tasks can then be scheduled based on the availability of these resources.
Supports single tenancy only	Supports multiple tenants using YARN	Multiple tenants are supported here.
Hadoop 1.x uses Master-Slave architecture that consists of a single master and multiple slaves. So, in case the master node gets failed then the entire clusters become unavailable.	Hadoop 2.x is also a Master-Slave architecture. However, this consists of multiple masters that includes active namenode and standby namenode. So, in this case if master node get failed then the standby master node will take over it. As a result, hadoop 2.x fixes the problem of a single point of failure.	It added supports for multiple active namenodes
Hadoop 1.x is limited to 4000 nodes per cluster.	It supports up to 10000 nodes in a cluster.	The scalability is improved in Hadoop 3.x and it can have more than 10000 nodes in one cluster.
	Manual intervention is needed	We don’t need manual

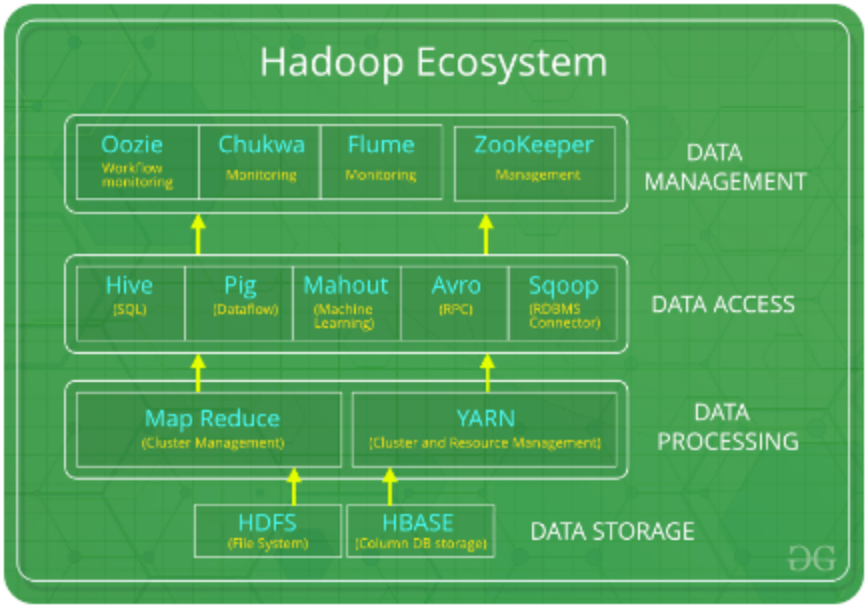
	for namenode recovery.	intervention for namenode recovery.
	Java 7 is the minimum supported version	Java 8 is the minimum supported version.
	It supports HDFS(default), FTP, Amazon S3 and Windows Azure Storage Blobs (WASB) file systems.	All file systems including Microsoft Azure Data Lake filesystem is compatible with Hadoop 3.x.
	It uses 3x replication scheme that results in 200% storage overhead.	Hadoop 3 uses eraser encoding in HDFS that helps to reduce the storage overhed. It has 50% storage overhead only.
		It added support for GPU hardware that can be used to execute deep leanring algorithms on a Hadoop cluster.

Comparison between Hadoop 1, Hadoop 2 and Hadoop 3

Now, to summarize the above points, we can have a look at the below image:



Hadoop Ecosystem



HDFS:

HDFS is the primary or major component of Hadoop ecosystem and is responsible for **storing large data sets** of structured or unstructured data across various nodes and thereby **maintaining the metadata** in the form of **log files.**

HDFS consists of **two core components** i.e.

1. Name node
2. Data Node

- **Name Node** is the prime node which **contains metadata** (data about data) requiring comparatively **fewer resources (Maintain Index)** than the data nodes that stores the **actual data**. These data nodes are service hardware in the distributed environment. Undoubtedly, making Hadoop cost effective.

  Datanodes do not know the **name of a file** and namenode does not know **what is inside a file.**

- HDFS maintains all the **coordination between the clusters and hardware**, thus working at the heart of the system.

YARN:

Yet Another Resource Negotiator, as the name implies, YARN is the one who helps to **manage the resources across the clusters**. In short, it performs **scheduling and resource allocation** for the Hadoop System.

Consists of three major components i.e.

- Resource Manager
- Nodes Manager
- Application Manager

Resource manager has the privilege of **allocating resources for the applications** in a system whereas **Node managers work**. on the allocation of resources such as CPU, memory, bandwidth per machine and later on acknowledges the resource manager.

Application manager works as an **interface between** the resource manager and node manager and **performs negotiations** as per the requirement of the two.

MapReduce:

By making the use of distributed and parallel algorithms, MapReduce makes it possible to carry over the processing's logic and helps to write applications which transform big data sets into a manageable one.

- MapReduce makes the use of two functions i.e. Map() and Reduce() whose task is:
  1. Map() performs sorting and filtering of data and thereby organizing them in the form of group. Map generates a key-value pair based result which is later on processed by the Reduce() method.
  2. Reduce(), as the name suggests does the summarization by aggregating the mapped data. In simple, Reduce() takes the output generated by Map() as input and combines those tuples into smaller set of tuples.

PIG:

- Pig was basically developed by Yahoo which works on a pig Latin language, which is **Query based language similar to SQL**.
- It is a **platform** for structuring the **data flow**, **processing and analyzing huge data sets.**
- Pig does the work of **executing commands and in the background**, all the ac**tivities of MapReduce are taken care of**. After the **processing**, pig **stores the result in HDFS**.
- **Pig Latin language** is specially designed for this framework which runs on **Pig Runtime**. Just the way Java runs on the JVM.
- Pig helps to achieve ease of programming and optimization and hence is a major segment of the Hadoop Ecosystem.

HIVE:

- With the help of SQL methodology and interface, **HIVE performs reading and writing of large data sets**. However, its query language is called as HQL (**Hive Query Language**).
- It is highly scalable as it allows real-time processing and batch processing both. Also, **all the SQL datatypes are supported by Hive** thus, making the query processing easier.
- Similar to the Query Processing frameworks, HIVE too comes with two components: **JDBC Drivers and HIVE Command Line**.
- JDBC, along with ODBC drivers work on **establishing the data storage permissions and connection whereas HIVE Command line helps in the processing of queries.**

Mahout:

- Mahout, allows Machine Learnability to a system or application. Machine Learning, as the name suggests helps the system to develop itself based on some patterns, user/environmental interaction or on the basis of algorithms.
- It provides various libraries or functionalities such as collaborative filtering, clustering, and classification which are nothing but concepts of Machine learning. It allows invoking algorithms as per our need with the help of its own libraries.

Apache Spark:

- It's a platform that handles all the process consumptive tasks like batch processing, interactive or iterative real-time processing, graph conversions, and visualization, etc.

- It consumes in memory resources hence, thus being faster than the prior in terms of optimization.
- Spark is best suited for real-time data whereas Hadoop is best suited for structured data or batch processing, hence both are used in most of the companies interchangeably.

Apache HBase:

- It's a NoSQL database which supports all kinds of data and thus capable of handling anything of Hadoop Database. It provides capabilities of Google's BigTable, thus able to work on Big Data sets effectively.
- At times where we need to search or retrieve the occurrences of something small in a huge database, the request must be processed within a short quick span of time. At such times, HBase comes handy as it gives us a tolerant way of storing limited data

Other Components: Apart from all of these, there are some other components too that carry out a huge task in order to make Hadoop capable of processing large datasets. They are as follows:

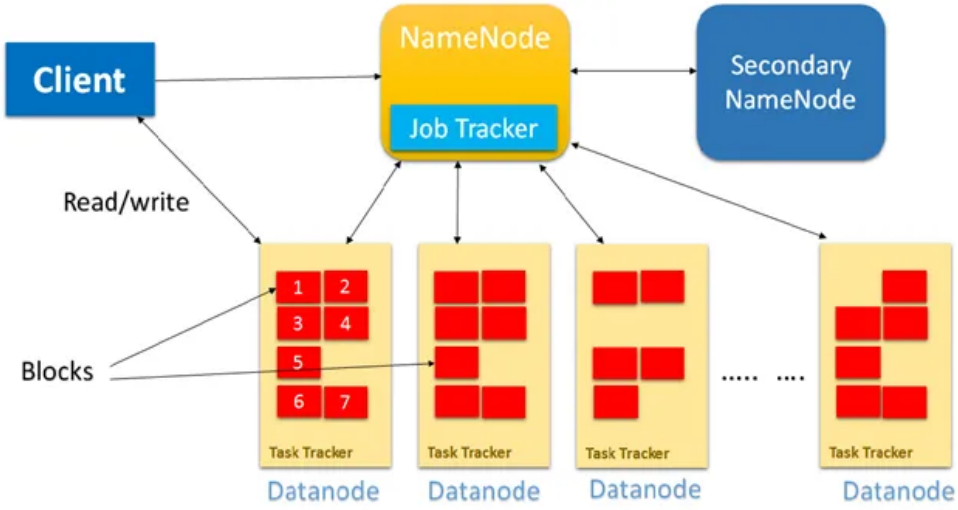
Solr, Lucene: These are the two services that perform the task of searching and indexing with the help of some java libraries, especially Lucene is based on Java which allows spell check mechanism, as well. However, Lucene is driven by Solr.

- **Zookeeper:** There was a huge issue of management of coordination and synchronization among the resources or the components of Hadoop which resulted in inconsistency, often. Zookeeper overcame all the problems by **performing synchronization, inter-component based communication, grouping, and maintenance**.
- **Oozie:** Oozie simply performs the task of a **scheduler**, thus **scheduling jobs and binding them together as a single unit**. There is two kinds of jobs .i.e **Oozie workflow and Oozie coordinator** jobs. Oozie workflow is the jobs that need to be **executed in a sequentially** ordered manner whereas Oozie Coordinator jobs **are those that are triggered when some data or external stimulus is given to it.**

The components associated with 1.0 architecture cluster are NameNode(Master) ,Backup Node and a collection of DataNodes(slaves).

Internal Process of 1.0 architecture

When a job is triggered via client the NameNode is intimated about it which internally uses Job tracker in order to check the availability of DataNodes. Based on the availability it breaks down the job into chunks of tasks and assigns it to DataNodes which in turn distributes it amongst the Task-tracker. The Task-tracker then performs the required operations that's parallel processing(***MapReduce algorithm***).



Job Distribution into DataNodes

During this process the NameNode checks for any failure of DataNode and replace them accordingly, based on the ***Heartbeat*** signal that is received by NameNode from the working DataNode (DataNodes sends heartbeat signal at regular intervals of time).

The default number of replication Factor for each block stored into HDFS is 3 and the size of each data block is 64MB. The three sets of copies are

usually stored in different DataNodes as part of different racks for fault tolerance.

When the data is being processed, the NameNode at regular intervals takes a snapshot of an intermediary state (snapshot) known as ***FS Image*** and stores the image in secondary NameNode In case the NameNode goes down.

Pro's and Con's involved in this architecture

Pros:

- Batch processing was made possible with large chunks of data
- Efficient storage with fault tolerance

Con's :

- Scalability - You cannot increase the number of Name Nodes.
- If a NameNode failure occurs then a manual intervention(boot up) is required in order to get the secondary NameNode up and running.
- As there is a single Job tracker if there are a number of tasks it can be overloaded (MapReduce processing) resulting in a delay in performance.
- Ample amount of time required in order to get secondary NameNode running.

Hadoop 2.0 broadly consists of two components **Hadoop Distributed File System(HDFS)** which can be used to store large volumes of data and **Yet Another Resource Negotiator(YARN)** which provides resource management and scheduling for running jobs. YARN supports various processing frameworks such as MapReduce, Spark, Storm etc.

HDFS

HDFS is a highly fault-tolerant distributed file system designed to store large volumes of data across upto thousands of commodity hardware machines. **HDFS has a master-slave architecture** and comprises of mainly three components which are **Namenode**, **Secondary Namenode**, **Datanodes**.

Datanodes- Datanodes are the nodes where the data is stored. A single cluster of Hadoop 2.x may accommodate upto 10,000 datanodes. **Datanodes are slave nodes** in Hadoop cluster. A large volume of data is divided into smaller blocks of data of defined size **(default 64 MB or 128 MB)** and their **replicas** are also produced (default replication factor is **3**). These blocks are then stored in different datanodes according to **topology of datanodes.**

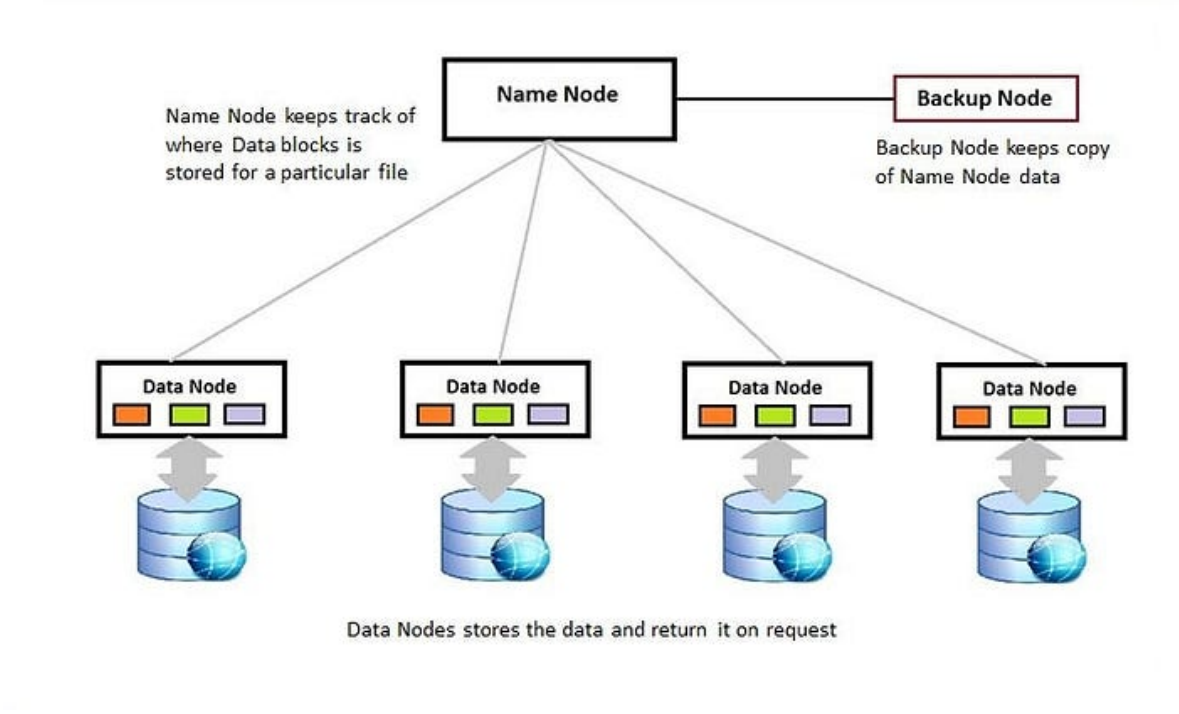
Namenode- Namenode is the **master node** in Hadoop cluster. Namenode stores the **metadata of each data block** present in datanodes and **status of datanodes**. Metadata is stored in namenode objects which **includes location, permissions, creation time, access time etc**. of data block. Namenode objects are file inodes and blocks and each object consumes **approx. 150 Bytes**. A namenode block stores the metadata of one data block. Consider an example of a **192 MB file** which is divided into two data blocks of 128 MB and 64 MB and stored in datanodes. Corresponding to this file three objects will form in namenode(1 file inode and 2 blocks) and consumes approx. 3 x 150 = 450 Bytes of memory in namenode.

Prior to Hadoop 2.0 one hadoop cluster could have **only one namenode** and it used to be a single point of **failure for whole cluster** as it contains all the metadata information. But Hadoop 2.0 overcome this issue by introducing the concept of **HDFS Federation.** In this concept **one cluster** can have **multiple namenodes** and they are scaled horizontally. The namenodes are independent and **do not** require any **coordination with each other**. Each **datanode registers** with all the **namenodes**

of the cluster. Datanodes sends **periodic signals**(heartbeats) to **namenodes** to **confirm their active status.**

When in use namenode stores all metadata in **main memory** but this information is also stored on disk for persistence storage. Metadata is **stored on disk** in **two files** called ***fsimage*** and ***edit logs*****. fsimage** contains the **snapshot of filesystem** when namenode started and **editlogs** contains the changes made to filesystem after namenode started. When **namenode restarts** editlogs are applied to fsimage to update it and editlogs are cleared.

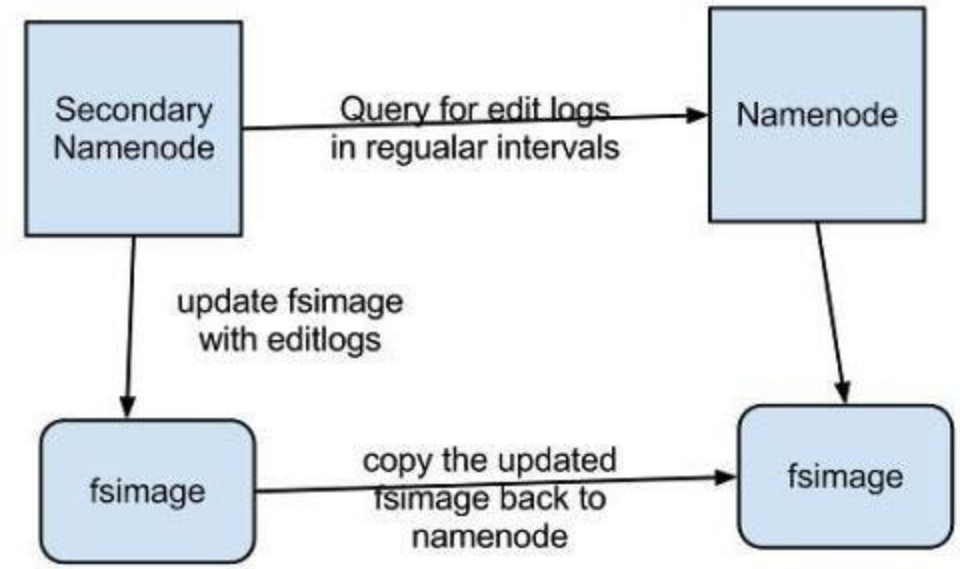
When **HDFS client receives the request for writing into HDFS**, it first asks the Namenode to choose datanodes to host replicas of the first block of the file. The client organizes a pipeline from node to node and sends the data. A **new pipeline** is organized **for each block** of data to be sent.



Secondary Namenode- As fsimage is updated **during namenode restart** so there may be a possibility of **edit logs** become very large and namenode may take **large time to apply them** and to get restarted. Also in case of **namenode crash,** a large **amount of metadata can be lost as fsimage** is too old.

To overcome these problems Secondary Namenode is used. Secondary namenode keep **updating fsimage of namenode using editlogs at regular intervals** also called

checkpoints. Once it has new fsimage, it copies back it to namenode which is used by namenode at restart time.



Please note that *Secondary Namenode is not the backup or replacement for Namenode*. Secondary namenode can never become namenode if namenode crashes. It is also called **Checkpoint Node.**

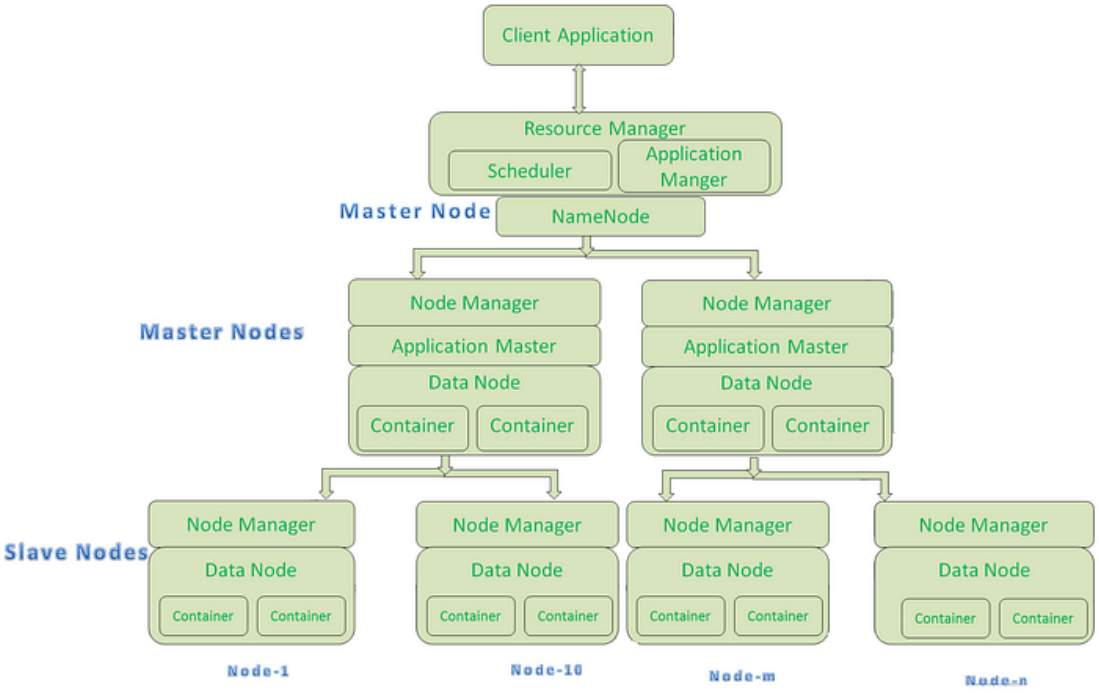
YARN

YARN was introduced in Hadoop 2.0. It is the resource management and scheduling layer of Hadoop 2.x. It is not constrained to only MapReduce but also supports many other processing frameworks such as Spark, Storm etc. In Hadoop 1.x Jobtracker has all the responsibilities of resource management and job scheduing/monitoring. But Hadoop 2.0 split up these functionalities into separate demons. YARN broadly consists of three components **Resource Manager, Node Manager, Application Master.** Resource manager resides **at global master node**. Every data node has its node manager. There is a per application/job appication master which also resides on one of the datanode.

Resource manager consists of two components **Scheduler** and **Application Manager. Scheduler allocate** resources to various **running applications**. Scheduler do not monitor or track the status of applications. **Application manager** has the responsibilities of **job submissions** from client, initialisation of job specific application master and tracking the status of application masters. The per application

master has the responsibilities of requesting resource containers from the scheduler, tracking their status and monitor their progress.

The whole process of job execution goes like this- Job is submitted to application manager. Application manager then makes the container at one datanode to start application master for that job. *Application master is like jobtracker for that individual job*. Application master then requests resource containers from scheduler to execute the task. Containers contain certain amount of resources(memory, CPU etc.) on a datanode. Application master notifies the node manager on which the container needs to be launched and then code is executed within these containers. After all processing application master unregisters itself from scheduler and result is given back to the client. This is how total load of jobtracker in Hadoop 1.0 is divided among resource manager and per job application master in Hadoop 2.0.



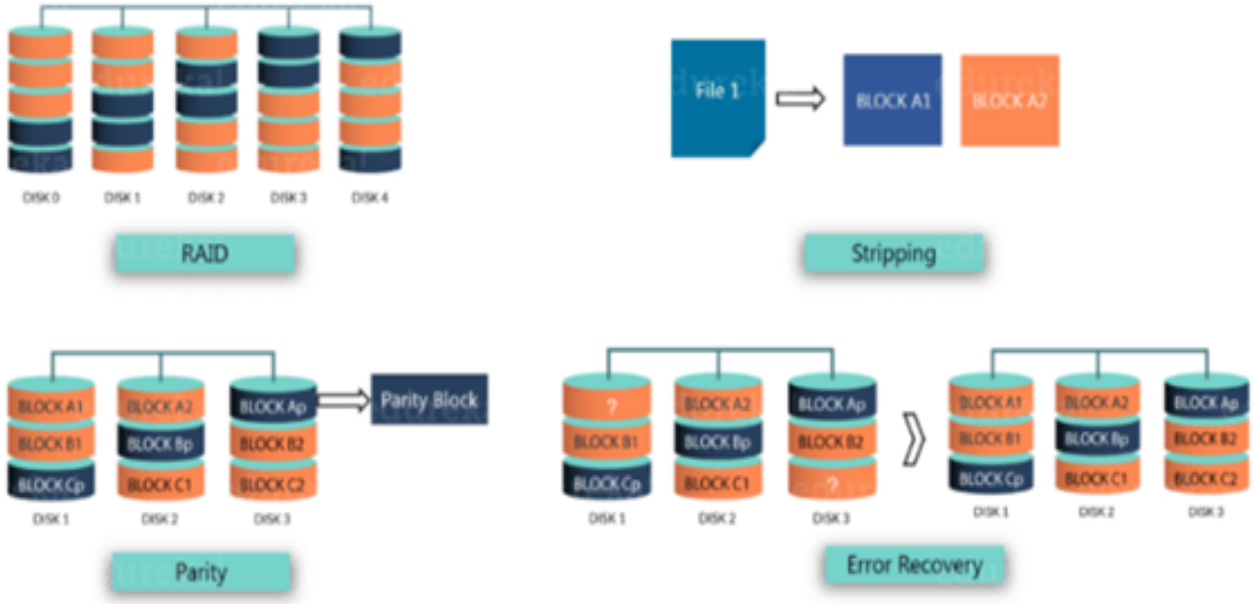
Hadoop 2.x In-Detail Architecture

Hadoop 3.0

1. Minimum Required Java Version in Hadoop 3 is Increased from 7 to 8

In Hadoop 3, all Hadoop JARs are compiled targeting a runtime version of Java 8. One of the important enhancement of Hadoop 3, i.e. Erasure(cut) Encoding, which will reduce the storage overhead while providing the same level of fault tolerance.

2. Support for Erasure Encoding in HDFS

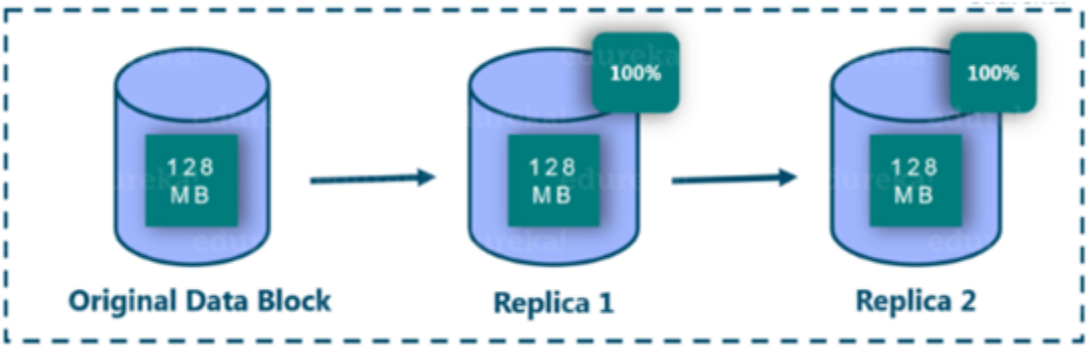


Generally, in storage systems, Erasure Coding is mostly used in ***Redundant Array of Inexpensive Disks (RAID)***.

As you can see in the above image, RAID implements EC through ***striping***, in which the logically sequential data (such as a file) is divided into smaller units (such as bit, byte, or block) and stores consecutive units on different disks.

Then for each stripe of original data cells, a certain number of ***parity cells*** are calculated and stored. This process is called ***encoding***. *The error on any striping cell can be recovered through decoding calculation based on surviving data cells and parity cells*.

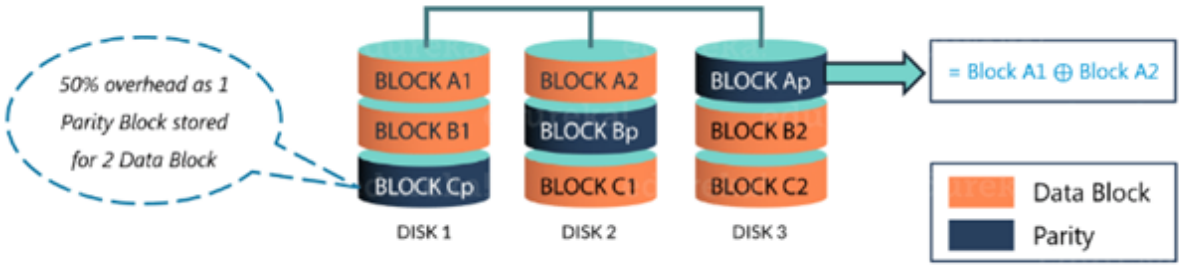
As we got an idea of Erasure coding, now let us first go through the earlier scenario of replication in Hadoop 2.x.



The default replication factor in ***HDFS*** is 3 in which one is the original data block and the other 2 are replicas which require 100% storage overhead each. So that makes ***200% storage overhead*** and it consumes other resources like network bandwidth.

However, the replicas of cold datasets which have low I/O activities are rarely accessed during normal operations, but still, consume the same amount of resources as the original dataset.

Erasure coding stores the data and provide fault tolerance with less space overhead as compared to HDFS replication. Erasure Coding (EC) can be used in place of replication, which will provide the same level of fault-tolerance with less storage overhead.



Integrating EC with HDFS can maintain the same fault-tolerance with improved storage efficiency. As an example, a 3x replicated file with 6 blocks will consume 6*3 = 18 blocks of disk space. But with EC (6 data, 3 parity) deployment, it will only consume 9 blocks (6 data blocks + 3 parity blocks) of disk space. This only requires the storage overhead up to 50%.

Since Erasure coding requires additional overhead in the reconstruction of the data due to performing remote reads, thus it is generally used for storing less frequently accessed data. Before deploying Erasure code, users should consider all the overheads like storage, network and CPU overheads of erasure coding.

From this **Big Data Course** designed by a Big Data professional, you will get 100% real-time project experience in Hadoop tools, commands and concepts.

Now to support the Erasure Coding effectively in HDFS they made some changes in the architecture. Lets us take a look at the architectural changes.

HDFS Erasure Encoding: Architecture

- **NameNode Extensions** – The HDFS files are striped into block groups, which have a certain number of internal blocks. Now to reduce NameNode memory consumption from these additional blocks, a new hierarchical ***block naming protocol*** was introduced. The ID of a block group can be deduced from the ID of any of its internal blocks. This allows management at the level of the block group rather than the block.

- **Client Extensions** – After implementing Erasure Encoding in HDFS, NameNode works on block group level & the client read and write paths were enhanced to work on multiple internal blocks in a block group in *parallel*.
    - On the output/write path, *DFSStripedOutputStream* manages a set of data streamers, one for each DataNode storing an internal block in the current block group. A coordinator takes charge of operations on the entire block group, including ending the current block group, allocating a new block group, etc.
    - On the input/read path, *DFSStripedInputStream* translates a requested logical byte range of data as ranges into internal blocks stored on DataNodes. It then issues read requests in parallel. Upon failures, it issues additional read requests for decoding.

- **DataNode Extensions –** The DataNode runs an additional ErasureCodingWorker (ECWorker) task for background recovery of failed erasure coded blocks. Failed EC blocks are detected by the NameNode, which then chooses a DataNode to do the recovery work. Reconstruction performs three key tasks:
    1. Read the data from source nodes and reads only the minimum number of input blocks & parity blocks for reconstruction.
    2. New data and parity blocks are decoded from the input data. All missing data and parity blocks are decoded together.
    3. Once decoding is finished, the recovered blocks are transferred to target DataNodes.

- **ErasureCoding policy** – To accommodate heterogeneous workloads, we allow files and directories in an HDFS cluster to have different replication and EC policies. Information about encoding & decoding files is encapsulated in an ErasureCodingPolicy class. It contains 2 pieces of information, i.e. the *ECSchema & the size of a stripping cell.*

The second most important enhancement in Hadoop 3 is YARN Timeline Service version 2 from YARN version 1 (in Hadoop 2.x). They are trying to make many upbeat changes in YARN Version 2.

3. YARN Timeline Service v.2

Hadoop is introducing a major revision of YARN Timeline Service i.e. v.2. YARN Timeline Service. It is developed to address two major challenges:

1. *Improving scalability and reliability of Timeline Service*
2. *Enhancing usability by introducing flows and aggregation*

YARN Timeline Service v.2 can be tested by the developers to provide feedback and suggestions. It should be exploited only in a test capacity. The security is not enabled in YARN Timeline Service v.2.

So, let us first discuss scalability and then we will discuss flows and aggregations.

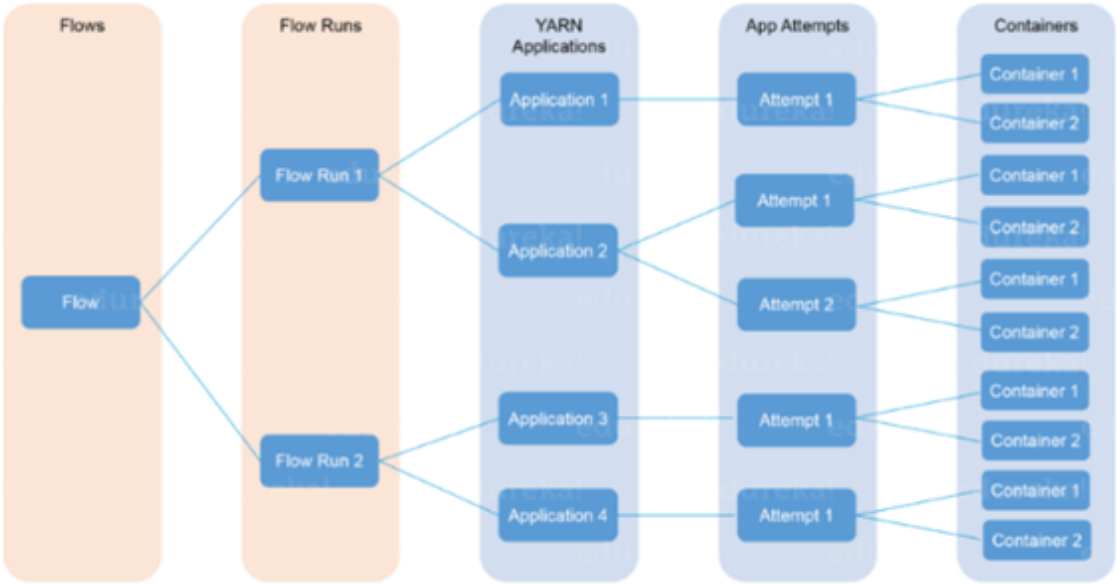
YARN Timeline Service v.2: Scalability

YARN version 1 is limited to a single instance of writer/reader and does not scale well beyond small clusters. Version 2 uses a more scalable distributed writer architecture and a scalable backend storage. It separates the collection (writes) of data from serving (reads) of data. It uses distributed collectors, essentially one collector for each YARN application. The readers are separate instances that are dedicated to serving queries via REST API.

YARN Timeline Service v.2 chooses Apache HBase as the primary backing storage, as Apache HBase scales well to a large size while maintaining good response times for reads and writes.

YARN Timeline Service v.2: Usability Improvements

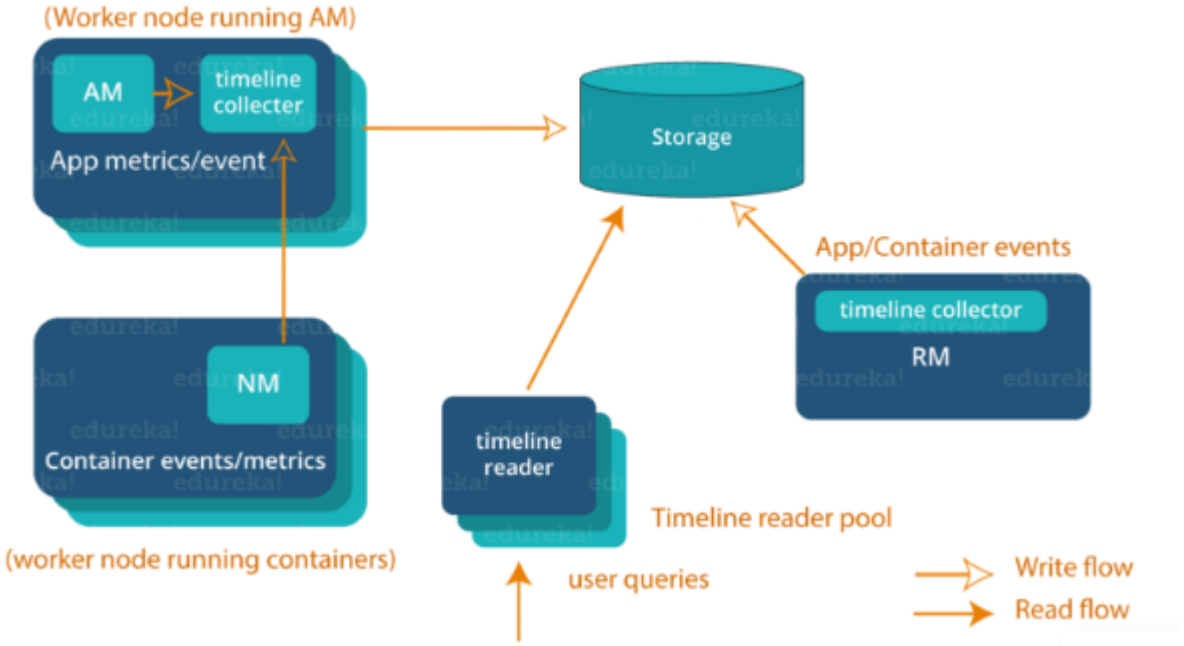
Now talking about usability improvements, in many cases, users are interested in the information at the level of "flows" or logical groups of YARN applications. It is much more common to launch a set or series of YARN applications to complete a logical application. Timeline Service v.2 supports the notion of flows explicitly. In addition, it supports aggregating metrics at the flow level as you can see in the below diagram.



Now lets us look at the architectural level, how YARN version 2 works.

YARN Timeline Service v.2: Architecture

YARN Timeline Service v.2 uses a set of collectors (writers) to write data to the backend storage. The collectors are distributed and co-located with the application masters to which they are dedicated, as you can see in the below image. All data that belong to that application are sent to the application level timeline collectors with the exception of the resource manager timeline collector.



For a given application, the application master can write data for the application to the co-located timeline collectors. In addition, node managers of other nodes that are running the containers for the application also write data to the timeline collector on the node that is running the application master.

The resource manager also maintains its own timeline collector. It emits only YARN-generic life cycle events to keep its volume of writes reasonable.

The timeline readers are separate daemons separate from the timeline collectors, and they are dedicated to serving queries via REST API.

4. Shell Script Rewrite

The Hadoop shell scripts have been rewritten to fix many bugs, resolve compatibility issues and change in some existing installation. It also incorporates some new features. So I will list some of the important ones:

- All Hadoop shell script subsystems now execute hadoop-env.sh, which allows for all of the environment variables to be in one location.
- Daemonization has been moved from *-daemon.sh to the bin commands via the –daemon option. In Hadoop 3 we can simply use –daemon start to start a daemon, –daemon stop to stop a daemon, and –daemon status to set $? to the daemon's status. For example, 'hdfs –daemon start namenode'.
- Operations which trigger ssh connections can now use pdsh if installed.
- ${HADOOP_CONF_DIR} is now properly honored everywhere, without requiring symlinking and other such tricks.
- Scripts now test and report better error messages for various states of the log and pid dirs on daemon startup. Before, unprotected shell errors would be displayed to the user.

There are many more features you will know when Hadoop 3 will be in the beta phase. Now let us discuss the shaded client jar and know their benefits.

5. Shaded Client Jars

The *hadoop-client* available in Hadoop 2.x releases pulls Hadoop's transitive dependencies onto a Hadoop application's classpath. This can create a problem if the versions of these transitive dependencies conflict with the versions used by the application.

So in Hadoop 3, we have new hadoop-client-api and hadoop-client-runtime artifacts that shade Hadoop's dependencies into a single jar. *hadoop-client-api* is compile scope & *hadoop-client-runtime* is runtime scope, which contains relocated third party dependencies from *hadoop-client*. So, that you can bundle the dependencies into a jar and test the whole jar for version conflicts. This avoids leaking Hadoop's dependencies onto the application's classpath. For example, HBase can use to talk with a Hadoop cluster without seeing any of the implementation dependencies.

Now let us move ahead and understand one more new feature, which has been introduced in Hadoop 3, i.e. opportunistic containers.

6. Support for Opportunistic Containers and Distributed Scheduling

A new ExecutionType has been introduced, i.e. ***Opportunistic containers***, which can be dispatched for execution at a NodeManager even if there are no resources available at the moment of scheduling. In such a case, these containers will be queued at the NM, waiting for resources to be available for it to start. Opportunistic containers are of lower priority than the default Guaranteed containers and are therefore preempted, if needed, to make room for Guaranteed containers. This should improve cluster utilization.

Guaranteed containers correspond to the existing YARN containers. They are allocated by the Capacity Scheduler, and once dispatched to a node, it is guaranteed that there are available resources for their execution to start immediately. Moreover, these containers run to completion as long as there are no failures.

Opportunistic containers are by default allocated by the central RM, but support has also been added to allow opportunistic containers to be allocated by a distributed scheduler which is implemented as an AMRMProtocol interceptor.

Now moving ahead, let us take a look how MapReduce performance has been optimized.

7. MapReduce Task-Level Native Optimization

In Hadoop 3, a native Java implementation has been added in MapReduce for the map output collector. For shuffle-intensive jobs, this improves the performance by 30% or more.

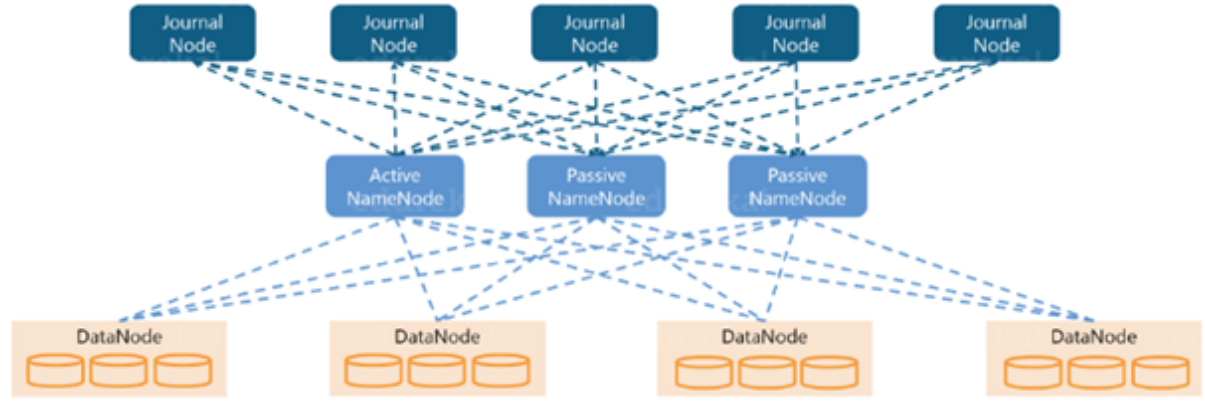
They added a native implementation of the map output collector. For shuffle-intensive jobs, this may provide speed-ups of 30% or more. They are working on native optimization for MapTask based on JNI. The basic idea is to add a NativeMapOutputCollector to handle key value pairs emitted by the mapper, therefore sort, spill, IFile serialization can all be done in native code. They are still working on the Merge code.

Now let us take a look, how Apache community is trying to make Hadoop 3 more fault tolerant.

8. Support for More than 2 NameNodes

In Hadoop 2.x, HDFS NameNode high-availability architecture has a single active NameNode and a single Standby NameNode. By replicating edits to a quorum of three JournalNodes, this architecture is able to tolerate the failure of any one NameNode.

However, business critical deployments require higher degrees of fault-tolerance. So, in Hadoop 3 allows users to run multiple standby NameNodes. For instance, by configuring three NameNodes (1 active and 2 passive) and five JournalNodes, the cluster can tolerate the failure of two nodes.



Next, we will look at default ports of Hadoop services that have been changed in Hadoop 3.

9. Default Ports of Multiple Services have been Changed

Earlier, the default ports of multiple Hadoop services were in the Linux *ephemeral port range* (32768-61000). Unless a client program explicitly requests a specific port number, the port number used is an ***ephemeral*** port number. So at startup, services would sometimes fail to bind to the port due to a conflict with another application.

Thus the conflicting ports with ephemeral range have been moved out of that range, affecting port numbers of multiple services, i.e. the NameNode, Secondary NameNode, DataNode, etc. Some of the important ones are:

Daemon	App	Hadoop 2.x Port	Hadoop 3 Port
NameNode Ports	Hadoop HDFS NameNode	8020	9820
	Hadoop HDFS NameNode HTTP UI	50070	9870
	Hadoop HDFS NameNode HTTPS UI	50470	9871
Secondary NN ports	Secondary NameNode HTTP	50091	9869
	Secondary NameNode HTTP UI	50090	9868
Datanode ports	Hadoop HDFS DataNode IPC	50020	9867
	Hadoop HDFS DataNode	50010	9866
	Hadoop HDFS DataNode HTTP UI	50075	9864
	Hadoop HDFS DataNode HTTPS UI	50475	9865

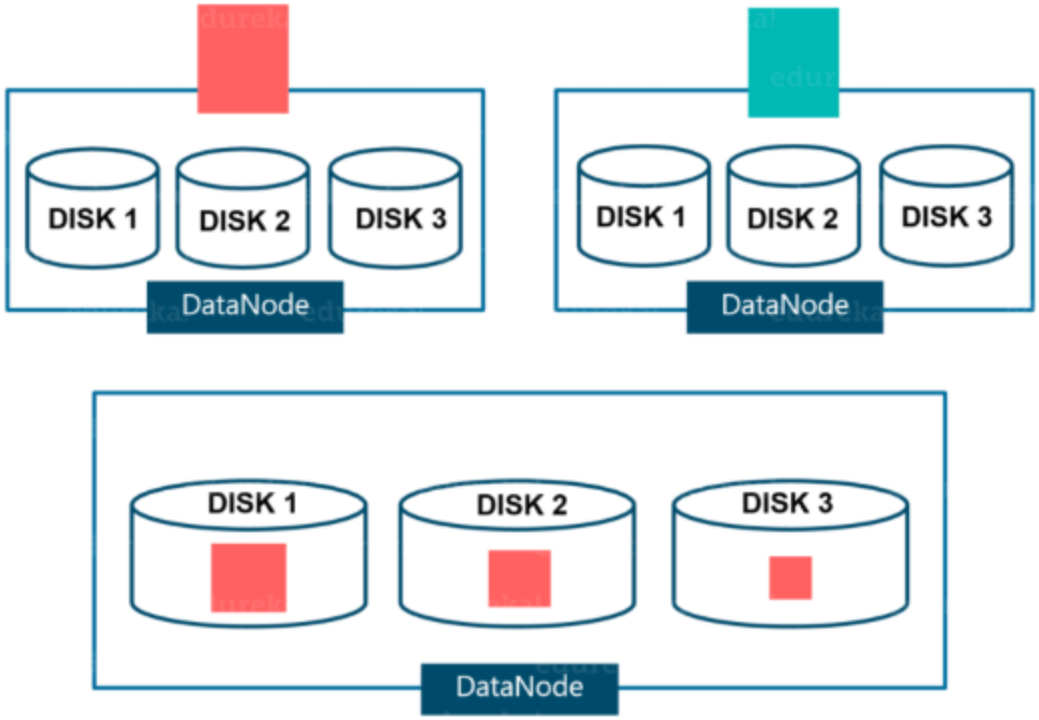
There are few more which are expected. Now moving on, let us know what are new Hadoop 3 file system connecters.

10. Support for Filesystem Connector

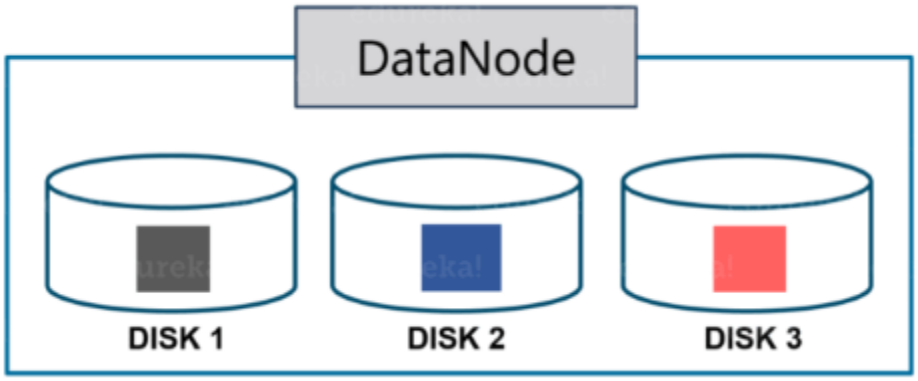
Hadoop now supports integration with Microsoft Azure Data Lake and Aliyun Object Storage System. It can be used as an alternative Hadoop-compatible filesystem. First Microsoft Azure Data Lake was added and then they added Aliyun Object Storage System as well. You might expect some more.

Let us understand how ***Balancer*** have been improved within multiple disks in a Data Node.

11. Intra-DataNode Balancer



A single DataNode manages multiple disks. During a normal write operation, data is divided evenly and thus, disks are filled up evenly. But adding or replacing disks leads to skew within a DataNode. This situation was earlier not handled by the existing HDFS balancer. This concerns intra-DataNode skew.



Now Hadoop 3 handles this situation by the new intra-DataNode balancing functionality, which is invoked via the hdfs diskbalancer CLI.

Now let us take a look how various memory managements have taken place.

12. Reworked Daemon and Task Heap Management

A series of changes have been made to heap management for Hadoop daemons as well as MapReduce tasks.

- New methods for configuring daemon heap sizes. Notably, auto-tuning is now possible based on the memory size of the host, and the HADOOP_HEAPSIZE variable has been deprecated.In its place, HADOOP_HEAPSIZE_MAX and HADOOP_HEAPSIZE_MIN have been

introduced to set Xmx and Xms, respectively. All global and daemon-specific heap size variables now support units. If the variable is only a number, the size is assumed to be in megabytes.

- Simplification of the configuration of map and reduce task heap sizes, so the desired heap size no longer needs to be specified in both the task configuration and as a Java option. Existing configs that already specify both are not affected by this change.

I hope this blog was informative and added value to you. Apache community is still working on multiple enhancements which might come up until beta phase. We will keep you updated and come up with more blogs and videos on Hadoop 3.

Now that you know the expected changes in Hadoop 3, check out the Big Data training in Chennai by Edureka, a trusted online learning company with a network of more than 250,000 satisfied learners spread across the globe. The Edureka Big Data Hadoop Certification Training course helps learners become expert in HDFS, Yarn, MapReduce, Pig, Hive, HBase, Oozie, Flume and Sqoop using real-time use cases on Retail, Social Media, Aviation, Tourism, Finance domain.