Hadoop and Yarn

Md Sahil, roll: 001710501029
Priyank Lohariwal, roll: 001710501055
Jadavpur University

1 Introduction

Ever since the enhancement of technology, data has been growing every day. Increasingly cheaper and accessible technology along with the boom of the Internet resulted in the exponential increase in the data generation. With all this data available at hand, many sectors of the industry have changed their approach to be more and more efficient. For example:

- Manufacturers can more accurately manage supply chains and mitigate risk by analyzing internal data alongside external data sets such as weather, premium news, and shipping information to understand potential delays or revenue impact.
- In the Agriculture industry, farmers can leverage Internet of Things sensors to gather instant data and analysis around soil and crop health, yields, harvest timings and beyond to improve decision-making and returns. Information can also be passed to commodity traders, more closely aligning farmers with the financial services industry.
- In Healthcare, there are many applications from pharmaceuticals research to the World Health Organization, which leverages intelligence to continuously scan a vast database of documents, news and websites to detect disease outbreaks around the globe.
- In Financial Services, the rise of technology and analytics in trading, investing, reporting and managing risk have increased the demand for comprehensive market data across all asset classes. Whether instruments are traded and priced on-exchange or off, the demand for real-time information in equities, fixed income and commodities markets is rising.

These massive amounts of data generated is difficult to store and process using traditional database system. Traditional database management system is used to store and process relational and structured data only. However, in today's world there are lots of unstructured data getting generated like images, audio files, videos; hence traditional system will fail to store and process these kinds of data. Effective solution for this problem is Hadoop.

Apache Hadoop is a collection of open-source software utilities that facilitates using a network of many computers to solve problems involving massive amounts of data and computation. The Apache 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.

Apache Hadoop provides the following benefits in solving BigData Problems:

- **Opensource** : Apache Hadoop is Open Source BigData Solution with free license from Apache Software Foundation. (licensed under Apache License 2.0)
- **Highly Availability** : Hadoop Solution uses Replication Technique. By default it uses Replication factor = 3. If required, we can change this value. If one node is down for some reason, it will automatically pickup data from other near-by and available node. Hadoop System finds that failure node automatically and do the necessary things to up and running that node. So that it is highly available. So Apache Hadoop provides no downtime BigData Solutions.
- **Highly Scalable** : Hadoop is highly Scalable, because it can store and distribute very huge amount of Data across hundreds of thousands of commodity hardware that operates in parallel. We can scale it in Horizontally or Vertically based on our Project requirements.
- **Better Performance** : Hadoop shifts the processing to the location where the data is stored in HDFS. Thus, even though Hadoop uses commodity hardware, it distributes work into different nodes and perform those tasks parallel. So that it can process PB (Petabytes) or More amount of Data in just few minutes and gives better performance.
- **Handles Huge and Varied types of Data** : Hadoop handles very huge amount of variety of data by using Parallel computing technique.
- **Cost-Effective BigData Solutions** : Unlike Traditional Relational Databases and Tools, Hadoop uses very in-expensive and non-enterprise commodity hardware to setup Hadoop Clusters. We don't need to buy very Expensive, High-Capacity and High Performance Hardware to solve our BigData Problems. Hadoop uses Cheap Hardware and deliver very effective solutions.
- **Increases Profits** : By using very Cheap commodity hardware to construct Our BigData Network, it increases Profits. If we use Cloud Technology to solve BigData Problems, we can improve our profits a lot.
- **Very Flexible** : Hadoop can accept any kind of Data Formats from different data sources. We can integrate new Data Sources with Hadoop system and use them very easily.

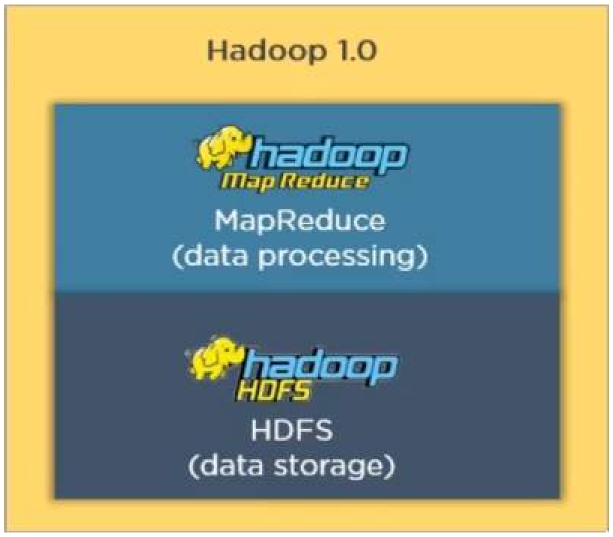

Figure 1: Hadoop 1.X

2 Hadoop 1.X

The two main components of Hadoop 1 are HDFS and MapReduce.

2.1 HDFS

HDFS is a distributed file system that handles large data sets running on commodity hardware. It is used to scale a single Apache Hadoop cluster to hundreds (and even thousands) of nodes. It is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS relaxes a few POSIX requirements to enable streaming access to file system data. HDFS was originally built as infrastructure for the Apache Nutch web search engine project. HDFS is now an Apache Hadoop sub-project.

2.1.1 Assumption and Goals of HDFS

- **Hardware Failure** : Hardware failure is the norm rather than the exception. An HDFS instance may consist of hundreds or thousands of server machines, each storing part of the file system's data. The fact that there are a huge number of components and that each component has a non-trivial probability of failure means that some component of HDFS is always non-functional. Therefore, detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS.

- **Streaming Data Access** : Applications that run on HDFS need streaming access to their data sets. They are not general purpose applications that typically run on general purpose file systems. HDFS is designed more for batch processing rather than interactive use by users. The emphasis is on high throughput of data access rather than low latency of data access. POSIX imposes many hard requirements that are not needed for applications that are targeted for HDFS. POSIX semantics in a few key areas has been traded to increase data throughput rates.

- **Large Data Sets** : Applications that run on HDFS have large data sets. A typical file in HDFS is gigabytes to terabytes in size. Thus, HDFS is tuned to support large files. It should provide high aggregate data bandwidth and scale to hundreds of nodes in a single cluster. It should support tens of millions of files in a single instance.

- **Simple Coherency Model** : HDFS applications need a write-once-read-many access model for files. A file once created, written, and closed need not be changed. This assumption simplifies data coherency issues and enables high throughput data access. A MapReduce application or a web crawler application fits perfectly with this model. There is a plan to support appending-writes to files in the future.

- **Moving Computation is Cheaper than Moving Data** A computation requested by an application is much more efficient if it is executed near the data it operates on. This is especially true when the size of the data set is huge. This minimizes network congestion and increases the overall throughput of the system. The assumption is that it is often better to migrate the computation closer to where the data is located rather than moving the data to where the application is running. HDFS provides interfaces for applications to move themselves closer to where the data is located.

- **Portability Across Heterogeneous Hardware and Software Platforms** : HDFS has been designed to be easily portable from one platform to another. This facilitates widespread adoption of HDFS as a platform of choice for a large set of applications.

2.1.2 Architecture

HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients.

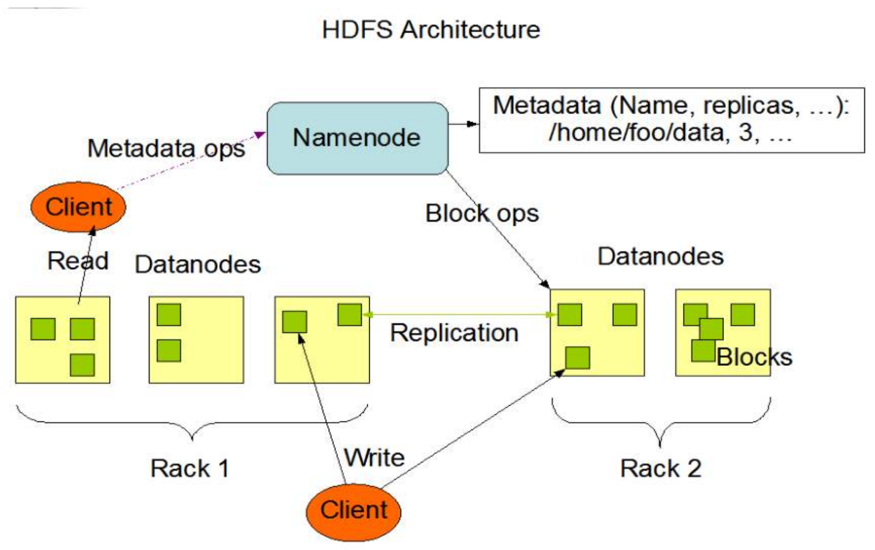

Figure 2: HDFS architecture

The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode. The NameNode and DataNode are pieces of software designed to run on commodity machines. These machines typically run a GNU/Linux operating system (OS). HDFS is built using the Java language; any machine that supports Java can run the NameNode or the DataNode software. Usage of the highly portable Java language means that HDFS can be deployed on a wide range of machines. A typical deployment has a dedicated machine that runs only the NameNode software. Each of the other machines in the cluster runs one instance of the DataNode software. The architecture does not preclude running multiple DataNodes on the same machine but in a real deployment that is rarely the case. The existence of a single NameNode in a cluster greatly simplifies the architecture of the system. The NameNode is the arbitrator and repository for all HDFS metadata. The system is designed in such a way that user data never flows through the NameNode.

2.1.3 Data replication

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time.

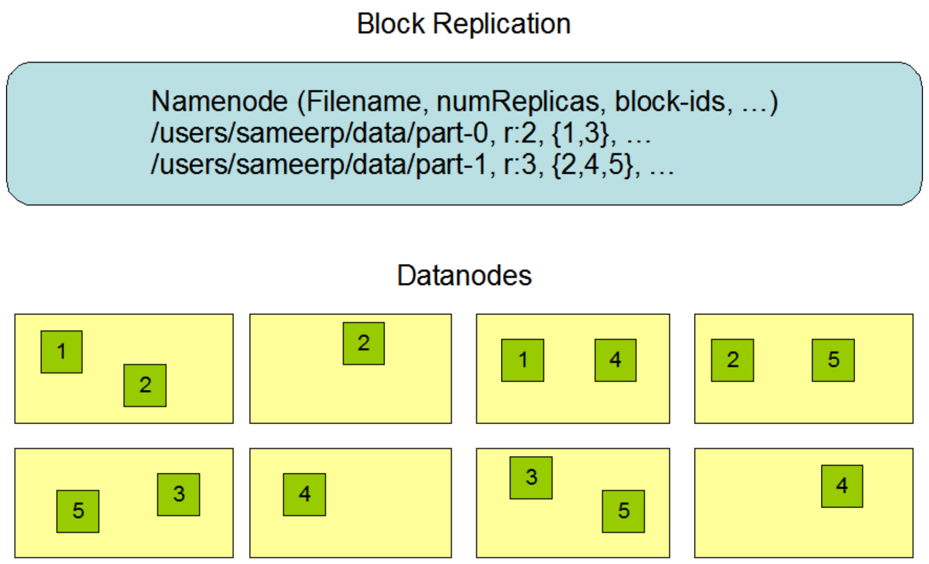
Block Replication
Namenode (Filename, numReplicas, block-ids, …)
/users/sameerp/data/part-0, r:2, {1,3}, …
/users/sameerp/data/part-1, r:3, {2,4,5}, …
Datanodes
1
2
2
1
4
2
5
5
3
4
3
5
4

Figure 3: File replication in HDFS

The NameNode makes all decisions regarding replication of blocks. It periodically receives a Heartbeat and a Blockreport from each of the DataNodes in the cluster. Receipt of a Heartbeat implies that the DataNode is functioning properly. A Blockreport contains a list of all blocks on a DataNode.

2.2 MapReduce

Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. A MapReduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks. Typically the processing nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster. The MapReduce framework consists of a single master JobTracker and one slave TaskTracker per cluster-node. The master is responsible for scheduling the jobs' component tasks on the slaves, monitoring them and re-executing the failed tasks. The slaves execute the tasks as directed by the master. Min-

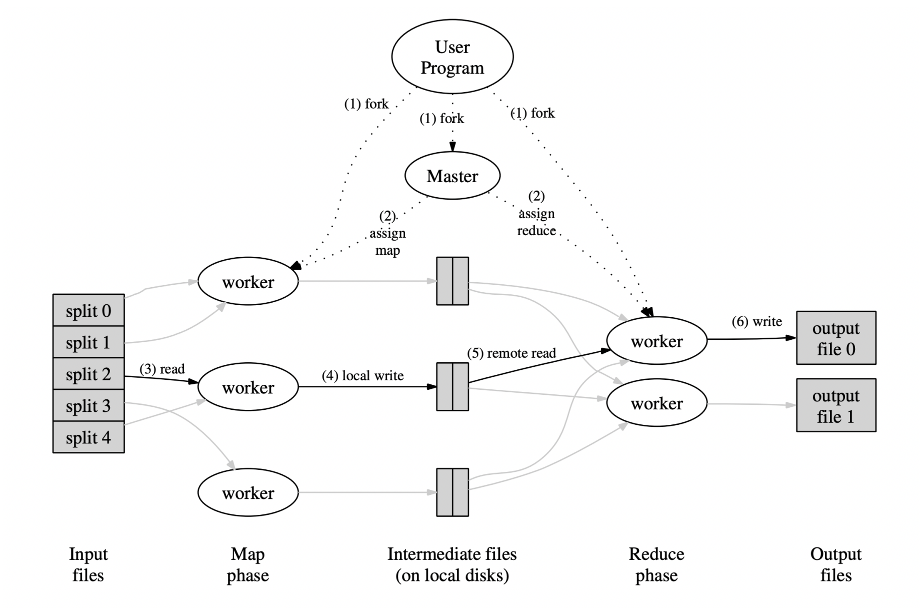

Figure 4: MapReduce Implementation Architecture

imally, applications specify the input/output locations and supply map and reduce functions via implementations of appropriate interfaces and/or abstract-classes. These, and other job parameters, comprise the job configuration. The Hadoop job client then submits the job (jar/executable etc.) and configuration to the JobTracker which then assumes the responsibility of distributing the software/configuration to the slaves, scheduling tasks and monitoring them, providing status and diagnostic information to the job-client.

2.3 Limitations of Hadoop 1.X

Hadoop 1.x has many limitations or drawbacks. Main drawback of Hadoop 1.x is that MapReduce Component in it's Architecture. That means it supports only MapReduce-based Batch/Data Processing Applications. The major drawbacks of Hadoop 1.X are as follows.

- Scalability: As there is a single JobTracker, scalability became a bottleneck. Hadoop 1.X cannot have a cluster with more than 4,000 nodes and it cannot run more than 40,000 concurrent tasks.
- Availability: JobTracker is a single point of failure. Any failure kills all queued and running jobs which required the users to resubmit the jobs.
- Resource Utilization: The number of map and reduce slots are predefined for each TaskTracker which causes some under utilization problems of resources at each node.

- Processing: Hadoop 1.X supports only MapReduce based batch driven data processing applications which does not fulfill the modern world processing requirements such as ad-hoc query processing, real-time data analysis, big data graph analysis etc.

3 Hadoop 2.X

Hadoop 2.X tries to incorporate the limitations of its previous generation and introduces a much more scalable and available framework that can utilize a multitude of data for several different forms of processing mechanisms such as real-time data analysis, graph analysis, ad-hoc query processing and much more. The main components of Hadoop 2.X are:

- HDFS Architecture for data storage
- YARN for resource management

3.1 HDFS in Hadoop 2.X

HDFS storage architecture used in Hadoop 2.X is also a master/slave architecture. An HDFS cluster consists of a master node that manages the namespaces of files and all the file operations. The master node is responsible for providing instructions to the slave nodes whenever a client requests an action over the data. It is also responsible to manage the metadata of file systems. This master node is known as the NameNode. The slave node that gets the instructions from namenode is known as the DataNode. The DataNode serves the read/write request from the client according to the instructions of the namenode. The concept of NameNode and DataNode is similar to the HDFS in Hadoop 1. Hadoop 2.X introduces certain components in its HDFS architecture to counter the failures of its predecessor which are mentioned below:

3.1.1 Secondary NameNode

Hadoop 2.X HDFS introduces a secondary namenode for data backup in case of cluster failures. A secondary namenode takes hourly backup of all the transactions and metadata available in the master namenode and stores the data as an fs-image file. In case of namenode failure, this fs-image file is used to create a different node to act as the master and the cluster is recovered.

3.1.2 Standby NameNode

As the name suggests, a standby namenode solves availability issues by acting as a standby for the master node. In case of active namenode failure, standby namenode takes over the working of the active namenode to keep the cluster running. The standby namenode keeps on taking the client requests and communicates with the datanodes for file read/write actions until the active namenode is recovered. This reduces the availability issues that Hadoop 1.X faces.

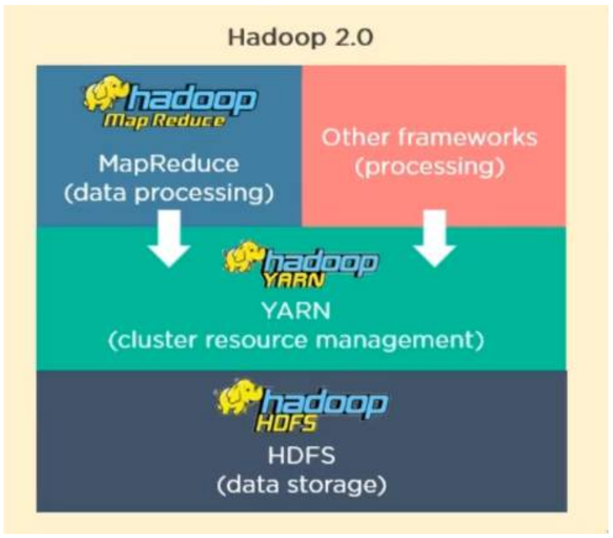
Hadoop 2.0
hadoop Map Reduce
MapReduce (data processing)
Other frameworks (processing)
hadoop YARN
YARN (cluster resource management)
hadoop HDFS
HDFS (data storage)

Figure 5: Hadoop 2.X

3.1.3 Federation

As there was only one master node in Hadoop 1.X, it was able to create only one file namespace. Hence, one cluster was able to manage a single file system only. The concept of Federation allows the possibility of multiple namenodes in a single cluster. Each namenode has its own standby namenode as well. This allows a cluster to handle multiple file systems. All the namenodes uses the same set of datanodes for file storage. A namenode accessing a file block must possess access rights to access that file block. This allows to keep the namespaces separate in the high level view.

3.1.4 Heartbeats

The NameNode requires to know the current active datanodes while processing a client request. It also needs to know the total storage and currently used storage capacity of a data node for maintaining the file distribution within the slaves. This is achieved by the concept of *Heartbeats*. Every datanode sends a signal known as the heartbeat in fixed intervals with information such as total storage capacity, used storage capacity and current actions being performed in the datanode. The interval of signals can be as frequent as 2-3 seconds. The master namenode uses these signals to maintain the filesystem and providing instructions for future client requests.

4 YARN - Yet Another Resource Negotiator

The fundamental idea of YARN is to split up the functionalities of resource management and job scheduling/monitoring into separate daemons. The idea is to have a global ResourceManager (RM) and per-application ApplicationMaster (AM). An application is either a single job or a Directed Acyclic Graph (DAG) of jobs.

The ResourceManager and the NodeManager form the data-computation framework. The ResourceManager is the ultimate authority that arbitrates resources among all the applications in the system. The NodeManager is the per-machine framework agent who is responsible for containers, monitoring their resource usage (cpu, memory, disk, network) and reporting the same to the ResourceManager/Scheduler.

The per-application ApplicationMaster is, in effect, a framework specific library and is tasked with negotiating resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the tasks.

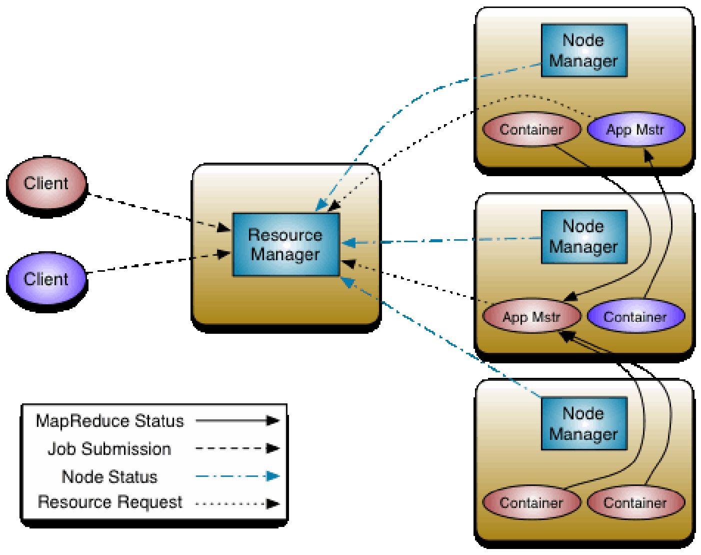

Figure 6: Yarn architecture

4.1 ResourceManager

It is the ultimate authority in resource allocation. On receiving the processing requests, it passes parts of requests to corresponding node managers accordingly, where the actual processing takes place. It is the arbitrator of the cluster resources and decides the allocation of the available resources for competing applications. Optimizes the cluster utilization like keeping all resources in use all the time against various constraints such as capacity guarantees and fairness. The ResourceManager has two main components:

- **Scheduler** : The Scheduler is responsible for allocating resources to the various running applications subject to familiar constraints of capacities, queues etc. The Scheduler is pure scheduler in the sense that it performs no monitoring or tracking of status for the application. Also, it offers no guarantees about restarting failed tasks either due to application failure or hardware failures. The Scheduler performs its scheduling function based on the resource requirements of the applications; it does so based on the abstract notion of a resource Container which incorporates elements such as memory, cpu, disk, network etc.

  The Scheduler has a pluggable policy which is responsible for partitioning the cluster resources among the various queues, applications etc. The current schedulers such as the CapacityScheduler and the FairScheduler would be some examples of plug-ins.

- **ApplicationsManager** : The ApplicationsManager is responsible for accepting job-submissions, negotiating the first container for executing the application specific ApplicationMaster and provides the service for restarting the ApplicationMaster container on failure.

The per-application ApplicationMaster has the responsibility of negotiating appropriate resource containers from the Scheduler, tracking their status and monitoring for progress.

4.2 NodeManager

It takes care of individual nodes in a Hadoop cluster and manages user jobs and workflow on the given node. Its primary goal is to manage application containers assigned to it by the resource manager. It keeps up-to-date with the Resource Manager. Application Master requests the assigned container from the Node Manager by sending it a Container Launch Context(CLC) which includes everything the application needs in order to run. The Node Manager creates the requested container process and starts it. It also monitors resource usage (memory, CPU) of individual containers, performs Log management and kills the container as directed by the Resource Manager.

An application is a single job submitted to the framework. Each such application has a unique **Application Master** associated with it which is a framework specific entity. It is the process that coordinates an application's execution in the cluster and also manages faults. Its task is to negotiate resources from the Resource Manager and work with the Node Manager to execute and monitor the component tasks. It is responsible for negotiating appropriate resource containers from the ResourceManager, tracking their status and monitoring progress. Once started, it periodically sends heartbeats to the Resource Manager to affirm its health and to update the record of its resource demands.

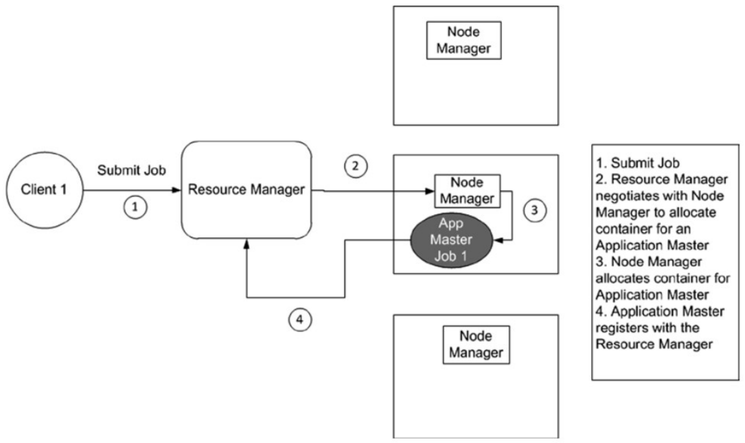
Node Manager
Client 1
Submit Job
1
Resource Manager
2
Node Manager
3
App Master Job 1
4
Node Manager
1. Submit Job
2. Resource Manager negotiates with Node Manager to allocate container for an Application Master
3. Node Manager allocates container for Application Master
4. Application Master registers with the Resource Manager

Figure 7: Application master startup

Container is a collection of physical resources such as RAM, CPU cores, and disks on a single node. YARN containers are managed by a container launch context which is container life-cycle(CLC). This record contains a map of environment variables, dependencies stored in a remotely accessible storage, security tokens, payload for Node Manager services and the command necessary to create the process. It grants rights to an application to use a specific amount of resources (memory, CPU etc.) on a specific host.

5 Anatomy of a Yarn Request

When a user submits a job to the Hadoop 2.x framework, the underlying YARN framework handles the request.

Here are the steps used:

1. A client program submits the application. The application type that in turn determines the Application Master is also specified.

2. The Resource Manager negotiates resources to acquire a container on a node to launch an instance of the Application Master.

3. The Application Master registers with the Resource Manager. This registration enables the client to query the Resource Manager for details about the Application Master. Thus the client will communicate with the Application Master it has launched through its own Resource Manager.

4. During its operation, the Application Master negotiates resources from the Resource Manager through resource requests. A resource request contains, among other things, the node on which containers are requested and the specifications of the container (CPU code and memory specifications).

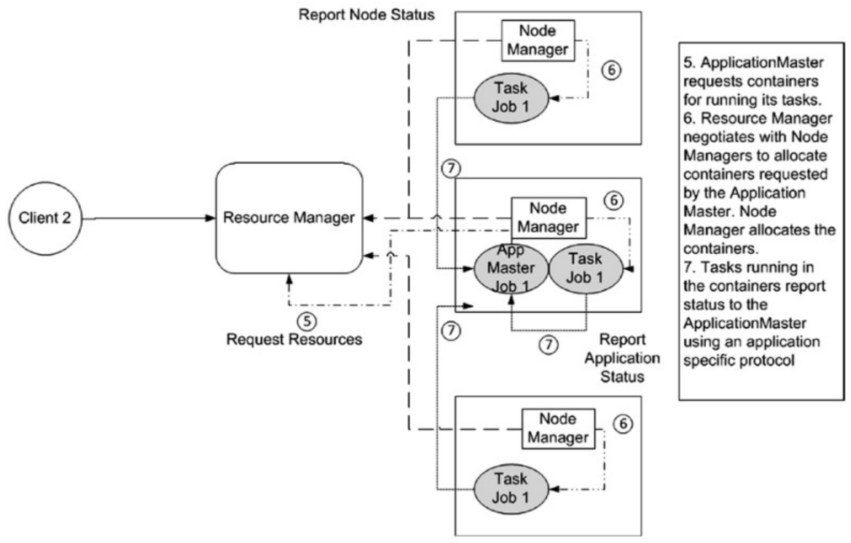
Report Node Status
Node Manager
Task Job 1
6
7
Client 2
Resource Manager
Node Manager
App Master Job 1
Task Job 1
5
Request Resources
Report Application Status
Node Manager
Task Job 1
5. ApplicationMaster requests containers for running its tasks.
6. Resource Manager negotiates with Node Managers to allocate containers requested by the Application Master. Node Manager allocates the containers.
7. Tasks running in the containers report status to the ApplicationMaster using an application specific protocol

Figure 8: Job resource allocation and execution

5. The application code executing in the launched container reports its progress to the Application Master (possibly remote) through an application-specific protocol.

6. The client program communicates with the Application Master through the application-specific protocol. The client references the Application Master through querying the Resource Manager it registered with in step 3.

Once the application completes execution, the Application Master deregisters with the Resource Manager, and the containers used are released back to the system.

6 Hadoop 2.X over Hadoop 1.X

The main idea behind Hadoop 2.X was to overcome the bottlenecks created by Hadoop 1.X that became a boulder for an established Hadoop architecture to expand further and to keep up the pace with the exponential data generation and their processing needs. The following list describes briefly how Hadoop 2.X solves the issues of its predecessor.

- The introduction of secondary and standby namenodes in the updated HDFS architecture decreases the chance of failures and reduces the time required for failure recoveries. This tolerance to failures in Hadoop 2.X also increases the cluster availability as a whole.

- The concept of Federation allows Hadoop 2.X to have multiple namespaces within a single cluster.
- YARN's resource management system allows a cluster to optimally utilise all the resources available in the slave nodes. This improves resource utilization. Better resource utilization results in more scalability as the number of distributed applications that can be run on a single cluster increases.
- The YARN architecture allows the use of different components which, in turn, allows a cluster to process multiple programming models. MapReduce, Interative, Streaming, Graph, Spark, Storm etc. are some of the different YARN components that can be used for different processing implementations.

7 Future of Hadoop

With the exponential increase in data generation speeds and massive boost to the big data processing, soon the current Hadoop 2.X will reach its limit. Improved data replication techniques, better fault tolerance and recovery mechanisms are needed. There are multiple cloud file systems present today. The count of these file systems are also increasing everyday. The Hadoop system needs to improve to incorporate every major file systems out there. This would lead to the need of much more scalable distributed cluster systems than can be provided by Hadoop 2.X architectures. The Hadoop 2.X community tried to focus on these issues and released Hadoop 3.X. Some of the notable features of Hadoop 3.X are mentioned below.

- Erasure coding is used instead of data replication in HDFS data nodes for fault tolerance. This reduces the storage overhead of the system to just 50% which was 200% for Hadoop 2.X.
- Intra-node balancer is used for data balancing in the data nodes. This allows more optimized control over the data balancing process.
- Yarn timeline service is improved to incorporate processing flows which are pretty common use case in the modern world.
- Hadoop 3.X supports FTP, Amazon S3, Windows Azure Storage Blob, Microsoft Azure Data Lake and Aliyun Object filesystems.
- Hadoop 3.X tries to improve cluster utilization by introducing the concept of ExecutionType where a application can be dispatched by the scheduler even if all the resources are not available at the time of scheduling.
- Hadoop 3.X introduces JournalNodes that reside at the level of master nodes. This aims to improve the fault tolerance of nodes by replicating edits to a quorum of JournalNodes.

8 References

- https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
- http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html
- https://www.journaldev.com/8808/hadoop1-architecture-and-how-major-components-works
- https://www.ibm.com/analytics/hadoop/hdfs
- https://link.springer.com/book/10.1007/978-1-4302-4864-4
- https://www.edureka.co/blog/hadoop-yarn-tutorial/
- https://hadoop.apache.org/docs/r3.0.0/

All sites last accessed at June 12th, 2021.