Big Data

Unit-2

History of Hadoop:

Hadoop was created by Doug Cutting and Mike Cafarella in 2005. It was originally developed to support distribution for the Nutch search engine project. Doug, who was working at Yahoo! at the time and is now Chief Architect of Cloudera, named the project after his son's toy elephant. Cutting's son was 2 years old at the time and just beginning to talk. He called his beloved stuffed yellow elephant "Hadoop" (with the stress on the first syllable). Now 12, Doug's son often exclaims, "Why don't you say my name, and why don't I get royalties? I deserve to be famous for this!"

Apache Hadoop:

Apache Hadoop is an open source software framework for storage and large scale processing of data-sets on clusters of commodity hardware. Hadoop is an Apache top-level project being built and used by a global community of contributors and users. It is licensed under the Apache License 2.0.

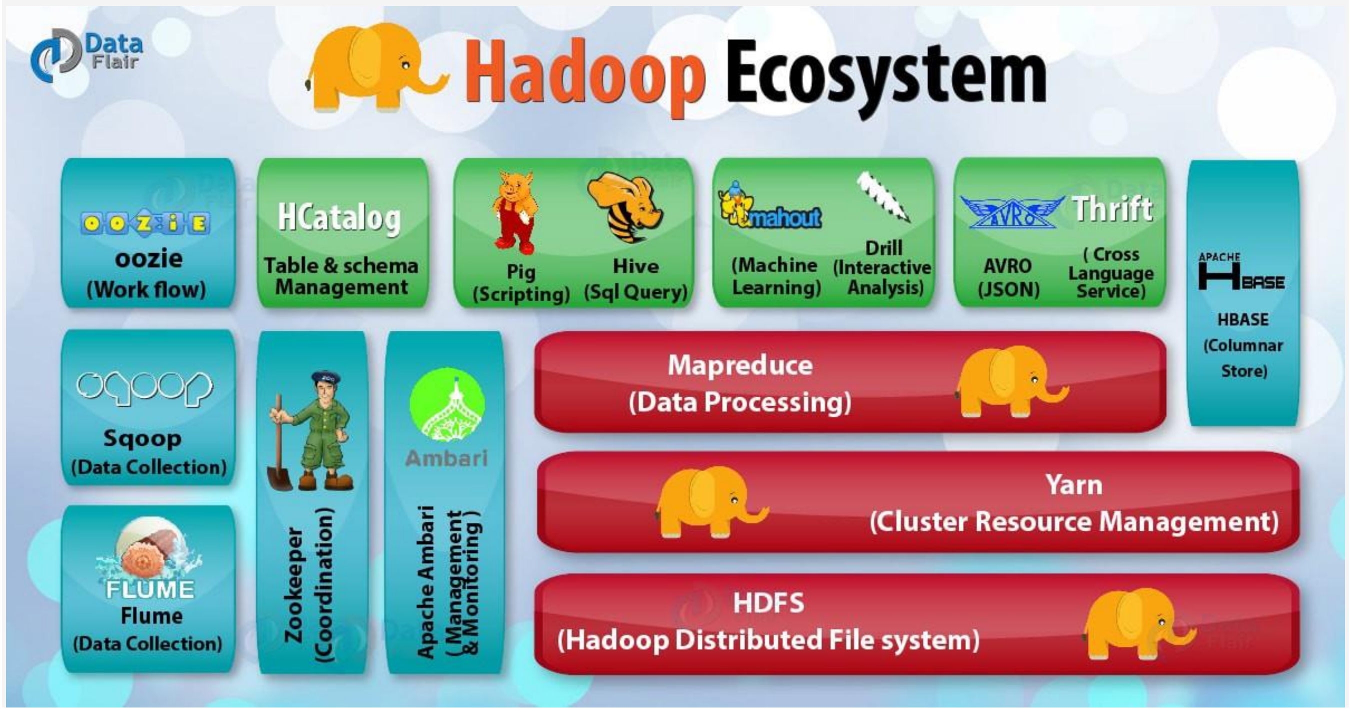
The Apache Hadoop framework is composed of the following modules

1. Hadoop Common: contains libraries and utilities needed by other Hadoop modules
2. Hadoop Distributed File System (HDFS): a distributed file-system that stores data on the commodity machines, providing very high aggregate bandwidth across the cluster
3. Hadoop YARN: a resource-management platform responsible for managing compute resources in clusters and using them for scheduling of users' applications
4. Hadoop MapReduce: a programming model for large scale data processing

All the modules in Hadoop are designed with a fundamental assumption that hardware failures (of individual machines, or racks of machines) are common and thus should be automatically handled in software by the framework. Apache Hadoop's MapReduce and HDFS components originally derived respectively from Google's MapReduce and Google File System (GFS) papers

Beyond HDFS, YARN and MapReduce, the entire Apache Hadoop "platform" is now commonly considered to consist of a number of related projects as well: Apache Pig, Apache Hive, Apache HBase, and others.

For the end-users, though MapReduce Java code is common, any programming language can be used with "Hadoop Streaming" to implement the "map" and "reduce" parts of the user's program. Apache Pig and Apache Hive, among other related projects, expose higher level user interfaces like Pig latin and a SQL variant respectively. The Hadoop framework itself is mostly written in the Java programming language, with some native code in C and command line utilities written as shell-scripts.


Data Flair
Hadoop Ecosystem
oozie (Work flow)
HCatalog Table & schema Management
Pig (Scripting)
Hive (Sql Query)
mahout (Machine Learning)
Drill (Interactive Analysis)
AVRO (JSON)
Thrift (Cross Language Service)
APACHE HBASE
HBASE (Columnar Store)
Sqoop (Data Collection)
Flume (Data Collection)
Zookeeper (Coordination)
Ambari
Apache Ambari (Management & Monitoring)
Mapreduce (Data Processing)
Yarn (Cluster Resource Management)
HDFS (Hadoop Distributed File system)

Hadoop to the Rescue

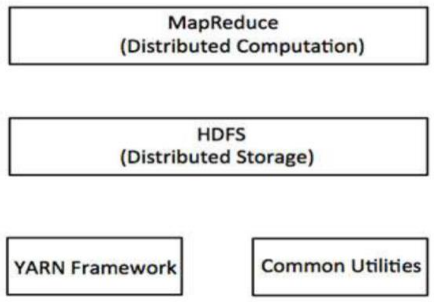
We have a savior to deal with Big Data challenges – its ***Hadoop***. Hadoop is an open source, Java-based programming framework that supports the storage and processing of extremely large data sets in a distributed computing environment. It is part of the Apache project sponsored by the Apache Software Foundation.

Hadoop with its distributed processing handles large volumes of structured and unstructured data more efficiently than the traditional enterprise data warehouse. Hadoop makes it possible to run applications on systems with thousands of commodity hardware nodes, and to handle thousands of terabytes of data. Organizations are adopting Hadoop because it is open source software and can run on commodity hardware (your personal computer). The initial cost savings are dramatic as commodity hardware is very cheap. As the organizational data increases, you need to add more & more commodity hardware on the fly to store it and hence, Hadoop proves to be economical. Additionally, Hadoop has a robust Apache community behind it that continues to contribute to its advancement.

Hadoop Architecture

At its core, Hadoop has two major layers namely –

- Processing/Computation layer (MapReduce), and
- Storage layer (Hadoop Distributed File System).



How Does Hadoop Work?

It is quite expensive to build bigger servers with heavy configurations that handle large scale processing, but as an alternative, you can tie together many commodity computers with single-CPU, as a single functional distributed system and practically, the clustered machines can read the dataset in parallel and provide a much higher throughput. Moreover, it is cheaper than one high-end server. So this is the first motivational factor behind using Hadoop that it runs across clustered and low-cost machines.

Hadoop runs code across a cluster of computers. This process includes the following core tasks that Hadoop performs –

- Data is initially divided into directories and files. Files are divided into uniform sized blocks of 128M and 64M (preferably 128M).
- These files are then distributed across various cluster nodes for further processing.
- HDFS, being on top of the local file system, supervises the processing.
- Blocks are replicated for handling hardware failure.
- Checking that the code was executed successfully.
- Performing the sort that takes place between the map and reduce stages.
- Sending the sorted data to a certain computer.
- Writing the debugging logs for each job.

Advantages of Hadoop

- Hadoop framework allows the user to quickly write and test distributed systems. It is efficient, and it automatic distributes the data and work across the machines and in turn, utilizes the underlying parallelism of the CPU cores.
- Hadoop does not rely on hardware to provide fault-tolerance and high availability (FTHA), rather Hadoop library itself has been designed to detect and handle failures at the application layer.

- Servers can be added or removed from the cluster dynamically and Hadoop continues to operate without interruption.
- Another big advantage of Hadoop is that apart from being open source, it is compatible on all the platforms since it is Java based.

Hadoop Distributed File System

The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. It is highly fault-tolerant and is designed to be deployed on low-cost hardware. It provides high throughput access to application data and is suitable for applications having large datasets.

Apart from the above-mentioned three core components, Hadoop –

1. Hadoop HDFS - Hadoop Distributed File System (HDFS) is the storage unit of Hadoop.
2. Hadoop MapReduce - Hadoop MapReduce is the processing unit of Hadoop.
3. Hadoop YARN - Hadoop YARN is a resource management unit of Hadoop.

With growing data velocity the data size easily outgrows the storage limit of a machine. A solution would be to store the data across a network of machines. Such filesystems are called *distributed filesystems*. Since data is stored across a network all the complications of a network come in.
This is where Hadoop comes in. It provides one of the most reliable filesystems. HDFS (Hadoop Distributed File System) is a unique design that provides storage for *extremely large files* with streaming data access pattern and it runs on *commodity hardware*. Let's elaborate the terms:

- ***Extremely large files***: Here we are talking about the data in range of petabytes(1000 TB).
- ***Streaming Data Access Pattern***: HDFS is designed on principle of *write-once and read-many-times*. Once data is written large portions of dataset can be processed any number times.
- ***Commodity hardware:*** Hardware that is inexpensive and easily available in the market. This is one of feature which specially distinguishes HDFS from other file system.

Nodes: Master-slave nodes typically forms the HDFS cluster.

1. **MasterNode:**
   - Manages all the slave nodes and assign work to them.
   - It executes filesystem namespace operations like opening, closing, renaming files and directories.
   - It should be deployed on reliable hardware which has the high config. not on commodity hardware.
2. **NameNode:**
   - Actual worker nodes, who do the actual work like reading, writing, processing etc.
   - They also perform creation, deletion, and replication upon instruction from the master.

- They can be deployed on commodity hardware.

HDFS deamons: Deamons are the processes running in background.

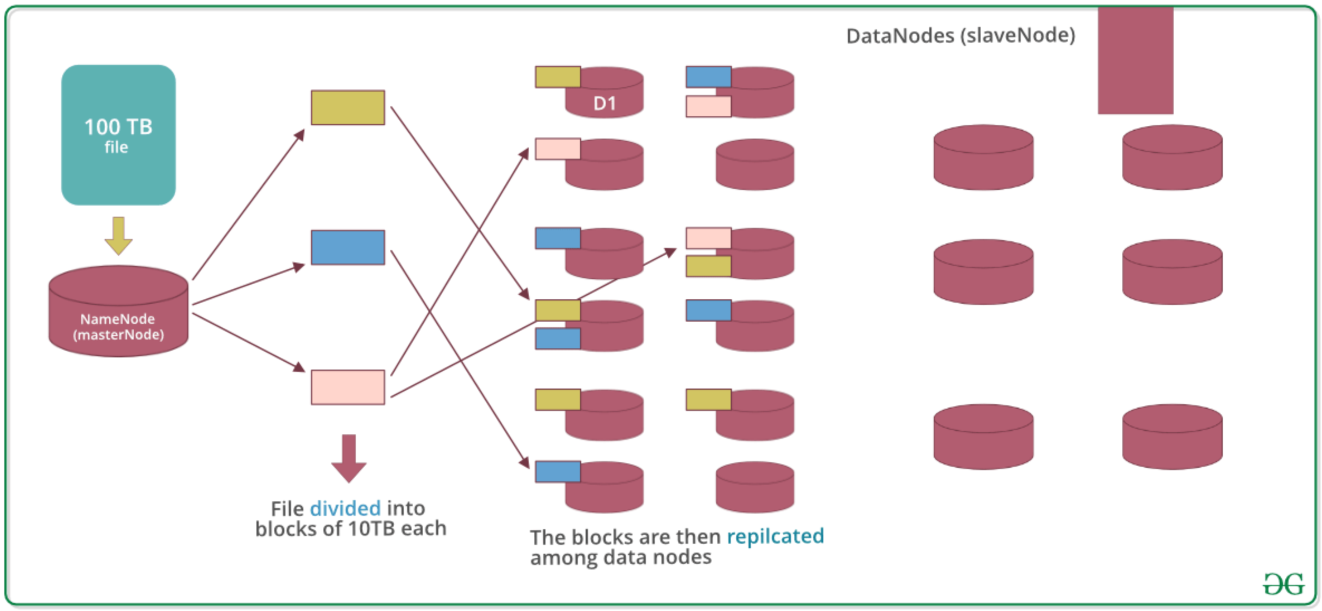
 Namenodes:

- Run on the master node.
- Store metadata (data about data) like file path, the number of blocks, block Ids. etc.
- Require high amount of RAM.
- Store meta-data in RAM for fast retrieval i.e to reduce seek time. Though a persistent copy of it is kept on disk.

 DataNodes:

- Run on slave nodes.
- Require high memory as data is actually stored here.

Data storage in HDFS: Now let's see how the data is stored in a distributed manner.


100 TB
file
NameNode
(masterNode)
File divided into blocks of 10TB each
D1
The blocks are then repilcated among data nodes
DataNodes (slaveNode)

Lets assume that 100TB file is inserted, then masternode(namenode) will first *divide* the file into blocks of 10TB (default size is *128 MB* in Hadoop 2.x and above). Then these blocks are stored across different datanodes(slavenode). Datanodes(slavenode)*replicate* the blocks among themselves and the information of what blocks they contain is sent to the master. Default replication factor is *3* means for each block 3 replicas are created (including itself). In hdfs.site.xml we can increase or decrease the replication factor i.e we can edit its configuration here.

Note: MasterNode has the record of everything, it knows the location and info of each and every single data nodes and the blocks they contain, i.e. nothing is done without the permission of masternode.

Why divide the file into blocks?

Answer: Let's assume that we don't divide, now it's very difficult to store a 100 TB file on a single machine. Even if we store, then each read and write operation on that *whole file* is going to take very high seek time. But if we have multiple blocks of size 128MB then its become easy to perform various read and write operations on it compared to doing it on a whole file at once. So we divide the file to have faster data access i.e. reduce seek time.

Why replicate the blocks in data nodes while storing?

Answer: Let's assume we don't replicate and only one yellow block is present on datanode D1. Now if the data node D1 crashes we will lose the block and which will make the overall data inconsistent and faulty. So we replicate the blocks to achieve *fault-tolerence.*

Terms related to HDFS:

- ***HeartBeat*** : It is the signal that datanode continuously sends to namenode. If namenode doesn't receive heartbeat from a datanode then it will consider it dead.
- ***Balancing*** : If a datanode is crashed the blocks present on it will be gone too and the blocks will be *under-replicated* compared to the remaining blocks. Here master node(namenode) will give a signal to datanodes containing replicas of those lost blocks to replicate so that overall distribution of blocks is balanced.
- ***Replication:***: It is done by datanode.

Note: No two replicas of the same block are present on the same datanode.

Features:

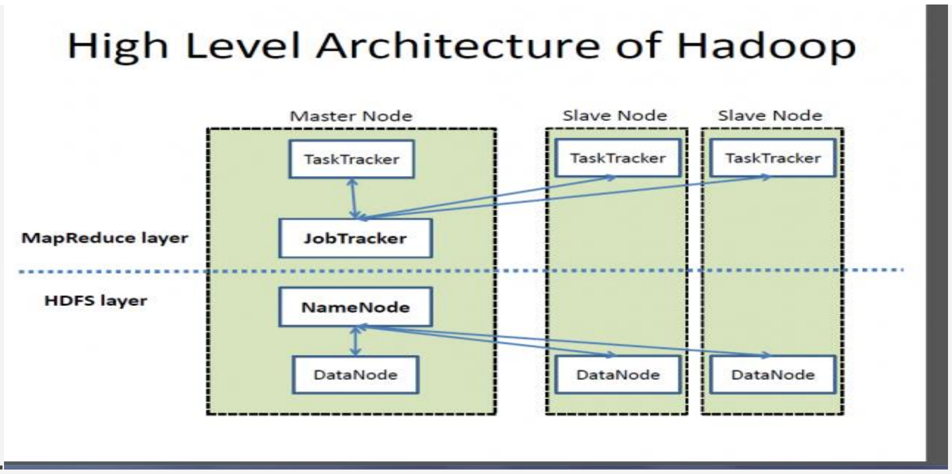
- Distributed data storage.
- Blocks reduce seek time.
- The data is highly available as the same block is present at multiple datanodes.
- Even if multiple datanodes are down we can still do our work, thus making it highly reliable.
- High fault tolerance.

Limitations: Though HDFS provide many features there are some areas where it doesn't work well.

- **Low latency data access**: Applications that require low-latency access to data i.e in the range of milliseconds will not work well with HDFS, because HDFS is designed keeping in mind that we need high-throughput of data even at the cost of latency.
- **Small file problem**: Having lots of small files will result in lots of seeks and lots of movement from one datanode to another datanode to retrieve each small file, this whole process is a very inefficient data access pattern.

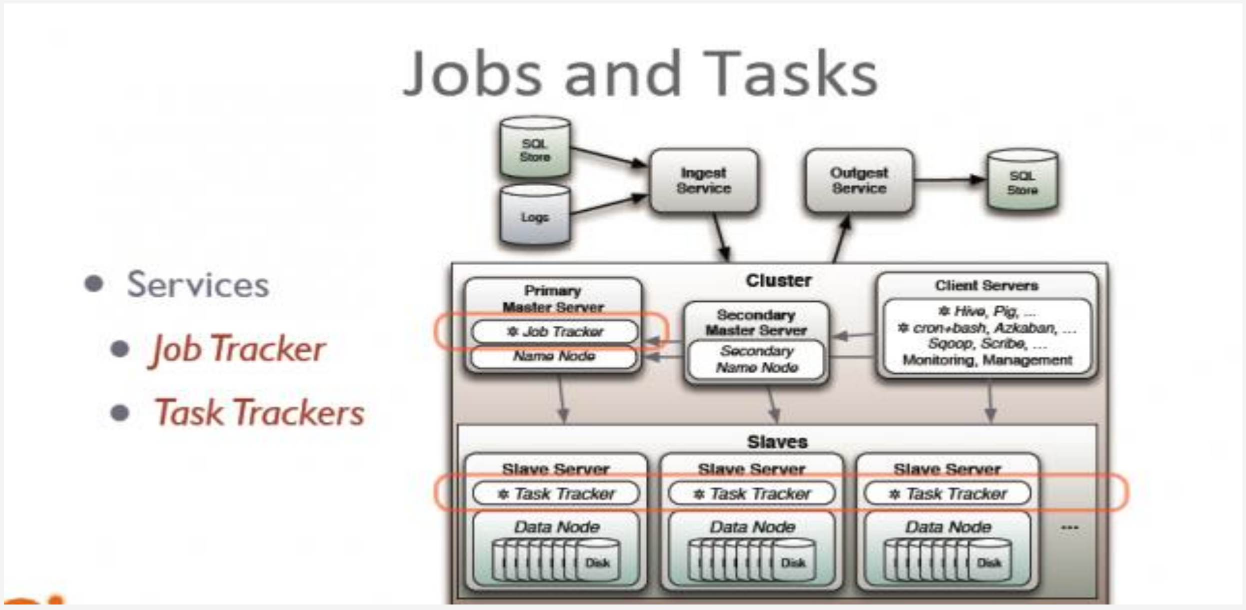
<u>MapReduce</u>

MapReduce is a parallel programming model for writing distributed applications devised at Google for efficient processing of large amounts of data (multi-terabyte data-sets), on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. The MapReduce program runs on Hadoop which is an Apache open-source framework.



Google.

JobTracker and TaskTracker: The MapReduce engine

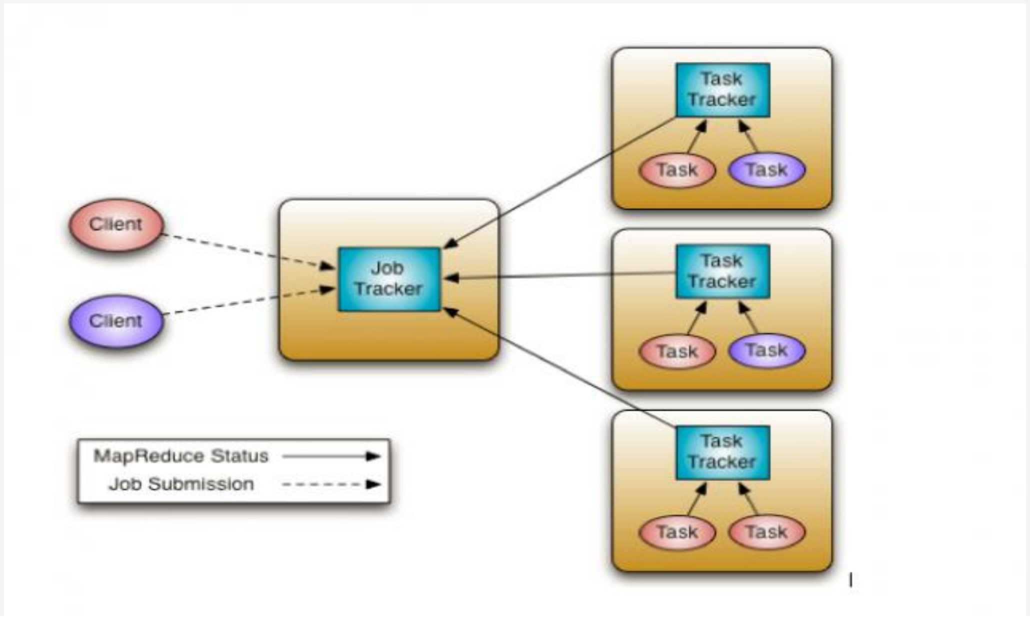


Above the file systems comes the MapReduce engine, which consists of one JobTracker, to which client applications submit MapReduce jobs. The JobTracker pushes work out to available TaskTracker nodes in the cluster, striving to keep the work as close to the data as possible.

With a rack-aware file system, the JobTracker knows which node contains the data, and which other machines are nearby. If the work cannot be hosted on the actual node where the data resides, priority is given to nodes in the same rack. This reduces network traffic on the main backbone network.

If a TaskTracker fails or times out, that part of the job is rescheduled. The TaskTracker on each node spawns off a separate Java Virtual Machine process to prevent the TaskTracker itself from failing if the running job crashes the JVM. A heartbeat is sent from the TaskTracker to the

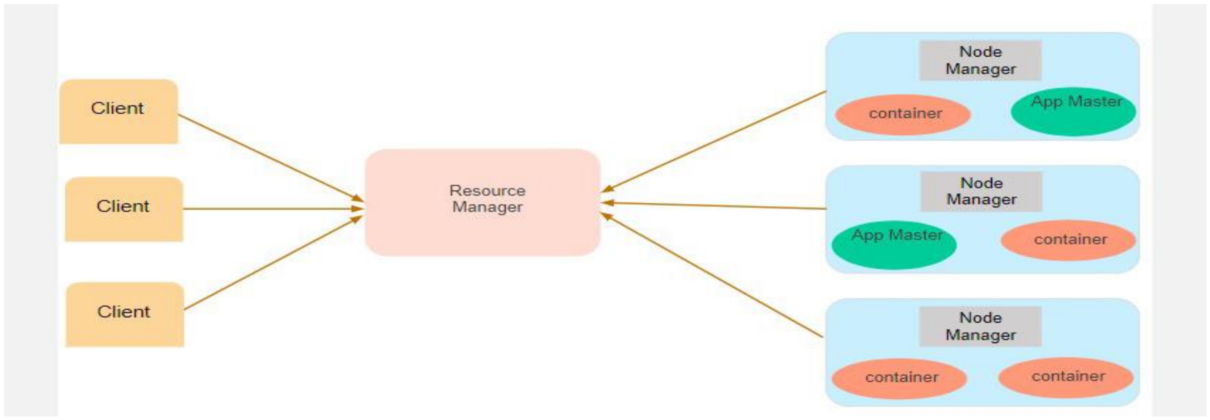
JobTracker every few minutes to check its status. The Job Tracker and TaskTracker status and information is exposed by Jetty and can be viewed from a web browser.



Hadoop YARN

Hadoop YARN stands for Yet Another Resource Negotiator. It is the resource management unit of Hadoop and is available as a component of Hadoop version 2.

- Hadoop YARN acts like an OS to Hadoop. It is a file system that is built on top of HDFS.
- It is responsible for managing cluster resources to make sure you don't overload one machine.
- It performs job scheduling to make sure that the jobs are scheduled in the right place



Suppose a client machine wants to do a query or fetch some code for data analysis. This job request goes to the resource manager (Hadoop Yarn), which is responsible for resource allocation and management.

In the node section, each of the nodes has its node managers. These node managers manage the nodes and monitor the resource usage in the node. The containers contain a collection of physical resources, which could be RAM, CPU, or hard drives. Whenever a job request comes in, the app master requests the container from the node manager. Once the node manager gets the resource, it goes back to the Resource Manager.

Types of Hadoop File Formats

Hive and Impala table in HDFS can be created using four different file formats:

- Text files
- Sequence File
- Avro data files
- Parquet file format

Text files

- A text file is the most basic and a human-readable file. It can be read or written in any programming language and is mostly delimited by comma or tab.
- The text file format consumes more space when a numeric value needs to be stored as a string. It is also difficult to represent binary data such as an image.

Sequence File

- The sequencefile format can be used to store an image in the binary format. They store key-value pairs in a binary container format and are more efficient than a text file. However, sequence files are not human- readable.

Avro Data Files

- The Avro file format has efficient storage due to optimized binary encoding. It is widely supported both inside and outside the Hadoop ecosystem.
- The Avro file format is ideal for long-term storage of important data. It can read from and write in many languages like Java, Scala and so on.
- Schema metadata can be embedded in the file to ensure that it will always be readable. Schema evolution can accommodate changes.
- The Avro file format is considered the best choice for general-purpose storage in Hadoop.

Parquet File Format

- Parquet is a columnar format developed by Cloudera and Twitter. It is supported in Spark, MapReduce, Hive, Pig, Impala, Crunch, and so on. Like Avro, schema metadata is embedded in the file.
- Parquet file format uses advanced optimizations described in Google's Dremel paper. These optimizations reduce the storage space and increase performance.
- This Parquet file format is considered the most efficient for adding multiple records at a time. Some optimizations rely on identifying repeated patterns.

SCALING OUT

To scale out, we need to store the data in a distributed file system, typically HDFS (which you'll learn about in the next chapter), to allow Hadoop to move the MapReduce computation to each machine hosting a part of the data
Scale out is a growth architecture or method that focuses on horizontal growth, or the addition of new resources instead of increasing the capacity of current resources (known as scaling up). In a system such as a cloud storage facility, following a scale-out growth would mean that new storage hardware and controllers would be added in order to increase capacity. This has two obvious pros – one is that storage capacity is increased and the second is traffic capacity is also increased because there is more hardware to share the load.

Scale out is a type of capacity expansion concentrating on the addition of new hardware resources instead of increasing the capacity of already available hardware resources such as storage or processing silos. This is often used in the context of storage because ideally, it is not just the storage capacity that needs to increase in such a system, but the controller and load balancing as well. In large cloud storage systems where multitenancy and scalability are required, increasing the capacity alone in a scale-up manner would not be sufficient to handle the increasing data traffic.

The scale-up approach was an older method for growth since hardware resources were expensive, so it made sense to make the most out of existing hardware and just increase capacity. But the decreasing hardware costs has made it easier to scale out, increasing all capacities in the process.

HADOOP STREAMING

Hadoop streaming is a utility that comes with the Hadoop distribution. This utility allows you to create and run Map/Reduce jobs with any executable or script as the mapper and/or the reducer.

How Streaming Works

When a script is specified for mappers, each mapper task will launch the script as a separate process when the mapper is initialized. As the mapper task runs, it converts its inputs into lines and feed the lines to the standard input (STDIN) of the process. In the meantime, the mapper collects the line-oriented outputs from the standard output (STDOUT) of the process and converts each line into a key/value pair, which is collected as the output of the mapper. By default, the prefix of a line up to the first tab character is the key and the rest of the line (excluding the tab character) will be the value. If there is no tab character in the line, then the entire line is considered as the key and the value is null. However, this can be customized, as per one need.

When a script is specified for reducers, each reducer task will launch the script as a separate process, then the reducer is initialized. As the reducer task runs, it converts its input key/values pairs into lines and feeds the lines to the standard input (STDIN) of the process. In the meantime, the reducer collects the line-oriented outputs from the standard output (STDOUT) of the process, converts each line into a key/value pair, which is collected as the output of the reducer. By default, the prefix of a line up to the first tab character is the key and the rest of the line (excluding the tab character) is the value. However, this can be customized as per specific requirements.

HADOOP PIPES

Hadoop Pipes is the name of the C++ interface to Hadoop Map Reduce. Unlike Streaming, which uses standard input and output to communicate with the map and reduce code, Pipes uses sockets as the channel over which the task tracker communicates with the process running the C++ map or reduce function. JNI is not used.

We'll rewrite the example running through the chapter in C++, and then we'll see how to run it using Pipes. Example shows the source code for the map and reduce functions in C++.

Example. Maximum temperature in C++

```
 #include <algorithm>
#include <limits>
#include <stdint.h>
#include <string>
#include "hadoop/Pipes.hh"
#include "hadoop/TemplateFactory.hh"
#include "hadoop/StringUtils.hh"
class MaxTemperatureMapper : public HadoopPipes::Mapper {
public:
MaxTemperatureMapper(HadoopPipes::TaskContext& context) {
}
void map(HadoopPipes::MapContext& context) {
std::string line = context.getInputValue()
std::string year = line.substr(15, 4);
std::string airTemperature = line.substr(87, 5);
std::string q = line.substr(92, 1);
if (airTemperature != "+9999" &&
(q == "0" || q == "1" || q == "4" || q == "5" || q == "9")) {
```

```
context.emit(year, airTemperature);
}
}
};
class MapTemperatureReducer : public HadoopPipes::Reducer {
public:
MapTemperatureReducer(HadoopPipes::TaskContext& context) {
}
void reduce(HadoopPipes::ReduceContext& context) {
int maxValue = INT_MIN;
while (context.nextValue()) {
maxValue = std::max(maxValue, HadoopUtils::toInt(context.getInputValue()));
}
context.emit(context.getInputKey(), HadoopUtils::toString(maxValue));
}
;
int main(int argc, char *argv[]) {
return HadoopPipes::runTask(HadoopPipes::TemplateFactory<MaxTemperatureMapper,
MapTemperatureReducer>());
}
```

The application links against the Hadoop C++ library, which is a thin wrapper for communicating with the tasktracker child process. The map and reduce functions are defined by extending the Mapper and Reducer classes defined in the HadoopPipes namespace and providing implementations of the map() and reduce() methods in each case. These methods take a context object (of type MapContext or ReduceContext), which provides the means for reading input and writing output, as well as accessing job configuration information via the JobConf class. The processing in this example is very similar to the Java equivalent.

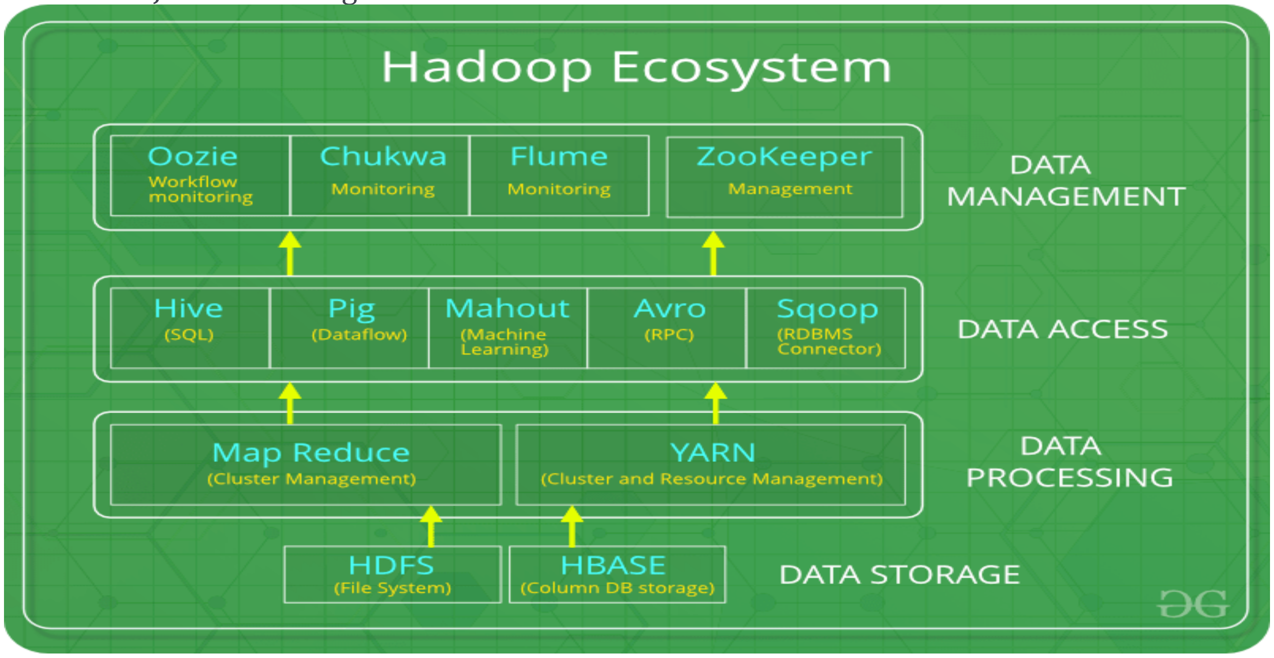
Unlike the Java interface, keys and values in the C++ interface are byte buffers, represented as Standard Template Library (STL) strings. This makes the interface simpler, although it does put a slightly greater burden on the application developer, who has to convert to and from richer domain-level types. This is evident in MapTempera tureReducer where we have to convert the input value into an integer (using a convenience method in HadoopUtils) and then the maximum value back into a string before it's written out. In some cases, we can save on doing the conversion, such as in MaxTem peratureMapper where the airTemperature value is never converted to an integer since it is never processed as a number in the map() method.

The main() method is the application entry point. It calls HadoopPipes::runTask, which connects to the Java parent process and marshals data to and from the Mapper or Reducer. The runTask() method is passed a Factory so that it can create instances of the Mapper or Reducer. Which one it creates is controlled by the Java parent over the socket connection. There are overloaded template factory methods for setting a combiner, partitioner, record reader, or record writer.

HADOOP ECOSYSTEM

Hadoop Ecosystem is a platform or a suite which provides various services to solve the big data problems. It includes Apache projects and various commercial tools and solutions. There are *four major elements of Hadoop* i.e. **HDFS, MapReduce, YARN, and Hadoop Common**. Most of the tools or solutions are used to supplement or support these major elements. All these tools work collectively to provide services such as absorption, analysis, storage and maintenance of data etc.
Following are the components that collectively form a Hadoop ecosystem:

- **HDFS:** Hadoop Distributed File System
- **YARN:** Yet Another Resource Negotiator
- **MapReduce:** Programming based Data Processing
- **Spark:** In-Memory data processing
- **PIG, HIVE:** Query based processing of data services
- **HBase:** NoSQL Database
- **Mahout, Spark MLLib:** Machine Learning algorithm libraries
- **Solar, Lucene:** Searching and Indexing
- **Zookeeper:** Managing cluster
- **Oozie:** Job Scheduling

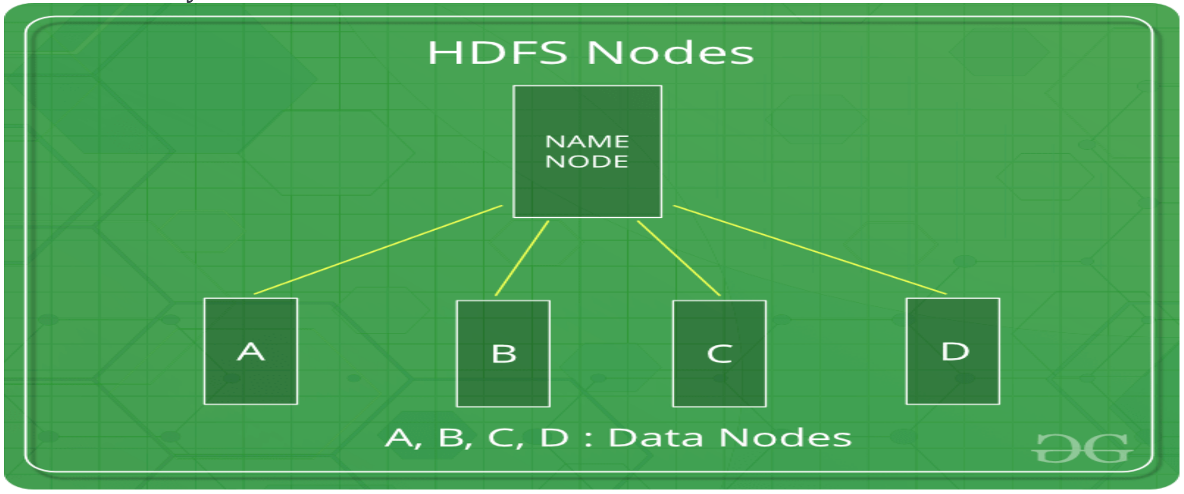


Note: Apart from the above-mentioned components, there are many other components too that are part of the Hadoop ecosystem.

All these toolkits or components revolve around one term i.e. *Data*. That's the beauty of Hadoop that it revolves around data and hence making its synthesis easier.

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 than the data nodes that stores the actual data. These data nodes are commodity hardware in the distributed environment. Undoubtedly, making Hadoop cost effective.
- 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.
  1. Resource Manager
  2. Nodes Manager
  3. 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 activities 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 om 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.