Big Data Analytics
Module 2

Introduction to Hadoop

Programming Model

1. Centralized Model
2. Distributed Model

Centralized Model:

- The data is transferred from multiple distributed data sources to a central server.
- Analyzing, reporting, visualizing, business-intelligence tasks compute centrally.
- Data are inputs to the central server.
- Assume that a centralized server does the function of collection, storing and analyzing.
- An ACVM Company enterprise server. The data at the server gets collected from a large number of ACVMs which the company locates in multiple cities, areas and locations. The server also receives data from social

Applications running at the server does the following analysis:

1. Suggests a strategy for filling the machines at minimum cost of logistics
2. Finds locations of high sales such as gardens, playgrounds etc.
3. Finds days or periods of high sales such as Xmas etc.
4. Finds children's preferences for specific chocolate flavors
5. Finds the potential region of future growth
6. Identifies unprofitable machines
7. Identifies need of replicating the number of machines at specific locations.

Distributed Model

- Distributed computing that uses the databases at multiple computing nodes with data sharing between the nodes during computation.
- This model requires the cooperation (sharing) between the DBs in a transparent manner.
- Transparent means that each user within the system may access all the data within all databases as if they were a single database.
- A second requirement is location independence. Analysis results should be independent of geographical locations.
- The access of one computing node to other nodes may fail due to a single link failure.

Distributed pieces of codes as well as the data at the computing nodes

Transparency between data nodes at computing nodes do not fulfil for Big Data when distributed computing takes place using data sharing between local and remote. For the following are the reasons:

- Distributed data storage systems do not use the concept of joins.
- Data need to be fault-tolerant and data stores should take into account the possibilities of network failure. When data need to be partitioned into data blocks and written at one set of nodes, then those blocks need replication at multiple nodes. This takes care of possibilities of network faults. When a network fault occurs, then replicated node makes the data available.
- Big Data follows a theorem known as the CAP theorem. The CAP states that out of three properties (consistency, availability and partitions), two must at least be present for applications, services and processes.

Big Data Store Model

- Data store in file system consisting of data blocks (physical division of data).
- The data blocks are distributed across multiple nodes.
- Data nodes are at the racks of a cluster. Racks are scalable.
- A Rack has multiple data nodes (data servers), and each cluster is arranged in a number of racks.
- Hadoop system uses the data store model in which storage is at clusters, racks, data nodes and data blocks.
- Data blocks replicate at the DataNodes such that a failure of link leads to access of the data block from the other nodes replicated at the same or other racks.

Big Data Programming Model

Big Data programming model is that application in which application jobs and tasks (or sub-tasks) is scheduled on the same servers which store the data for processing.

Job means running an assignment of a set of instructions for processing. For example, processing the queries in an application and sending the result back to the application is a job.

Job scheduling means assigning a job for processing following a schedule. For example, scheduling after a processing unit finishes the previously assigned job, scheduling as per specific sequence or after a specific.

Hadoop system uses the programming model, where jobs or tasks are assigned and scheduled on the same servers which hold the data.

Cluster Computing refers to computing, storing and analyzing huge amounts of unstructured or structured data in a distributed computing environment.

- Each cluster forms by a set of loosely or tightly connected computing nodes that work together and many of the operations can be timed (scheduled) and can be realized as if from a single computing system.

- Clusters improve the performance, provide cost-effective and improved node acessibility compared to a single computing node.

- Each node of the computational cluster is set to perform the same task and sub-tasks, such as MapReduce, which software control and schedule.

Data Flow (DF) refers to flow of data from one node to another. For example, transfer of output data after processing to input of application.

Data Consistency means all copies of data blocks have the same values.

Data Availability means at least one copy is available in case a partition becomes inactive or fails. For example, in web applications, a copy in the other partition is available.

Resources means computing system resources, i.e., the physical or virtual components or devices, made available for specified or scheduled periods within the system. Resources refer to sources such as files, network connections and memory blocks.

Resource management refers to managing resources such as their creation, deletion and controlled usages. The manager functions includes managing the (i) availability for specified or scheduled periods, (ii) prevention of resource unavailability after a task finishes and (ii) resources allocation when multiple tasks attempt to use the same set of resources.

Horizontal scalability means increasing the number of systems working in coherence. For example, using MPPs or number of servers as per the size of the dataset. Processing different datasets of a large data store running similar application deploys the horizontal scalability.

Vertical scalability means scaling up using the giving system resources and increasing the number of tasks in the system.

For example, extending analytics processing by including the reporting, business processing (BP), business intelligence (B), data visualization, knowledge discovery and machine learning (ML) capabilities which require additional ways to solve problems of greater complexities and greater processing, storage and inter-process communication among the resources. Processing different datasets of a large data store running multiple application tasks deploys vertical scalability.

Ecosystem refers to a system made up of multiple computing components, which work together.

Distributed File System means a system of storing files. Files can be for the set of data records, key- value pairs, hash key-value pairs, relational database or NoSQL database at the distributed computing nodes, accessible after referring to their resource-pointer using a master directory service, look-up tables or name-node server.

Hadoop Distributed File System means a system of storing files at distributed computing nodes according to Hadoop architecture and accessibility of data blocks after finding reference to their racks and cluster. NameNode servers enable referencing to data blocks.

Scalability of storage and processing means the execution using varying number of servers according to requirements.

HADOOP AND ITS ECOSYSTEM

⇒ Hadoop is a computing environment in which input data stores, processes, and stores the results.

⇒ The environment consists of clusters which distribute at the cloud or set of servers. Each cluster consists of a string of data files constituting data blocks.

⇒ The toy named Hadoop consisted of a stuffed elephant.

⇒ The Hadoop system cluster stuffs files in data blocks.

⇒ The complete system consists of a scalable distributed set of clusters.

⇒ Infrastructure consists of cloud for clusters. A cluster consists of sets of computers or PCs.

⇒ The Hadoop platform provides a low-cost Big Data platform, which is open source and uses cloud services. Tera Bytes of data processing takes just few minutes. Hadoop enables distributed processing of large datasets across clusters of computers using a programming model called MapReduce.

⇒ The system characteristics are scalable, self-manageable, self-healing and distributed file system.

⇒ Scalable means can be scaled up (enhanced) by adding storage and processing units as per the requirements.

⇒ Self-manageable means creation of storage and processing resources which are used, scheduled, and reduced or increased with the help of the system itself.

⇒ Self-healing means that in case of faults, they are taken care of by the system itself. Self-healing enables functioning and resources availability. Software detects and handle
failures at the task level.

⇒ Software enable the service or task execution even in case of communication or node failure.

Hadoop Core Components

The Hadoop core components of the framework are:

1. **Hadoop Common** - The common module contains the libraries and utilities that are required by the other modules of Hadoop. For example, Hadoop common provides various components and interfaces for distributed file system and general input/output. This includes serialization, Java RPC and file-based data structures.
2. **Hadoop Distributed File System (HDFS)** - A Java-based distributed file system which can store all kinds of data on the disks at the clusters.

3. **MapReduce v**l - Software programming model in Hadoop I using Mapper and Reducer. The vl processes large sets of data in parallel and in batches.

4. **YARN**-Software for managing resources for computing. The user application tasks, or sub-tasks run in parallel at the Hadoop, uses scheduling and handles the requests for the resources in distributed running of the tasks.

5. **MapReduce v2** - Hadoop 2 YARN-based system for parallel processing of large datasets and distributed processing of the application tasks.

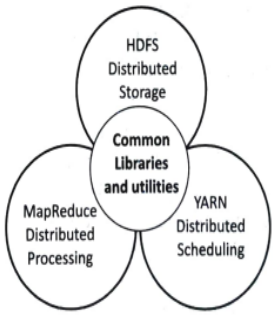


Figure: Core components of Hadoop

Spark

Spark is an open-source cluster-computing framework of Apache Software Foundation. Hadoop deploys data at the disks.

Spark provisions for in-memory analytics and enables OLAP and real-time processing. Spark does faster processing of Big Data.

Spark has been adopted by large organizations, such as Amazon, eBay and Yahoo. Several organizations run Spark on clusters with thousands of nodes.

Features of Hadoop

1. **Fault-efficient scalable, flexible and modular design** which uses simple and modular programming model. The system provides servers at high scalability. The system is scalable by adding new nodes to handle larger data. Hadoop proves very helpful in storing, managing. processing and analyzing Big Data. Modular functions make the system flexible. One can add or replace components at ease. Modularity allows replacing its components for a different software tool.

2. **Robust design of HDFS**: Execution of Big Data applications continue even when an individual server or cluster fails. This is because of Hadoop provisions for backup and a data recovery mechanism. HDFS thus has high reliability.

3. **Store and process Big Data**: Processes Big Data of 3V characteristics.

4. **Distributed clusters computing model with data locality:** Processes Big Data at high speed as the application tasks and sub-tasks submit to the DataNodes. One can achieve more computing power by increasing the number of computing nodes. The processing splits across multiple DataNodes (servers), and thus fast processing and aggregated results.

5. **Hardware fault-tolerant**: A fault does not affect data and application processing. If a node goes down, the other nodes take care of the residue. This is due to multiple copies of all data blocks which replicate automatically. Default is three copies of data blocks.

6. **Open-source framework**: Open source access and cloud services enable large data store. Hadoop uses a cluster of multiple inexpensive servers or the cloud.

7. **Java and Linux based:** Hadoop uses Java interfaces. Hadoop base is Linux but has its own set of shell common support.

Hadoop Ecosystem Components

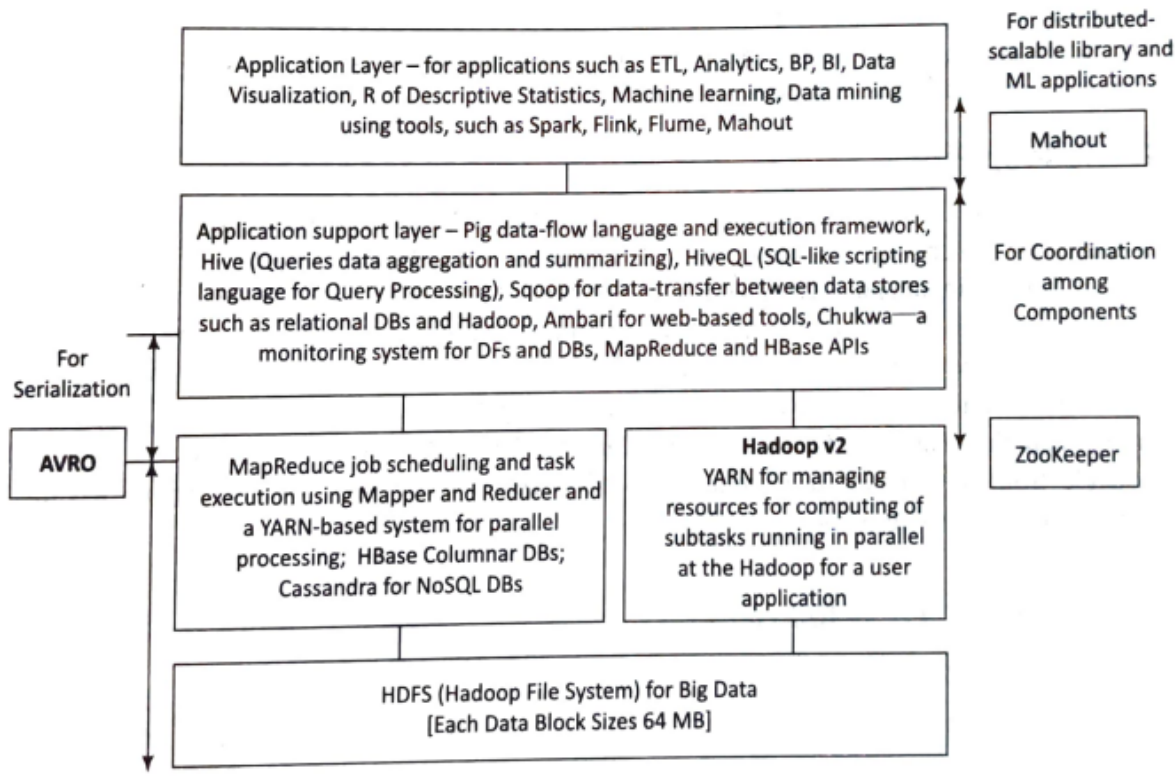


Figure: Hadoop main components and ecosystem components

Hadoop ecosystem refers to a combination of technologies. Hadoop ecosystem consists of own family of applications which tie up together with the Hadoop. The system components support the storage, processing, access, analysis, governance, security and operations for Big Data.

The system includes the application support layer and application layer components- AVRO, ZooKeeper, Pig, Hive, Sqoop, Ambari, Chukwa, Mahout, Spark, Flink and Flume.

The four layers are as follows:

1. Distributed storage layer
2. Resource-manager layer for job or application sub-tasks scheduling and execution
3. Processing-framework layer, consisting of Mapper and Reducer for the MapReduce process-flow
4. APls at application support layer

⇒ The codes communicate run using MapReduce or YARN at processing framework layer.

⇒ Reducer output communicate to APls

⇒ AVRO enables data serialization between the layers.

⇒ Zookeeper enables coordination among layer components.

⇒ Client hosts run applications using Hadoop ecosystem projects, such as Pig, Hive and Mahout.

⇒ Hadoop uses Java programming. Such Hadoop programs run on any platform with the
⇒ Java virtual-machine deployment model.

⇒ HDFS is a Java-based distributed file system that can store various kinds of data on the computers.

Hadoop Streaming

HDFS with MapReduce and YARN-based system enables parallel processing of large datasets.

Spark provides in-memory processing of data, thus improving the processing speed. Spark and Flink technologies enable in-stream processing.

The two lead stream processing systems and are more useful for procesing a large volume of data. Spark includes security features.

Flink is emerging as a powerful tool. Flink improves the overall performance as it provides single run-time for streaming as well as batch processing. Simple and flexible architecture of Flink is suitable for streaming data.

Hadoop Pipes

Hadoop Pipes is the name of the C++ interface to Hadoop MapReduce. 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

Apache Hadoop provides an adapter layer, which processes in pipes. A pipe means data streaming into the system at Mapper input and aggregated results flowing out at outputs. The adapter layer enables running of application tasks in C++ coded MapReduce programs. Applications which require faster numerical computations can achieve higher throughput using C++ when used through the pipes, as compared to Java.

HADOOPDISTRIBUTED FILE SYSTEM

HDFS is a core component of Hadoop. HDFS is designed to run on a cluster of computers and servers at cloud-based utility services. HDFS stores Big Data which may range from GBs (1 GB = 2^{30} B) to PBs (1 PB = 10^{15} Nearly the 2^{50} B), HDES stores the data in a distributed manner in order to compute fast. The distributed data store in HDFS stores data in any format regardless of schema. HDFS provides high throughput access to data-centric applications that require large-scale data processing workloads.

HDFS Data Storage

Hadoop stores the data in clusters.

Each cluster has a number of data stores, called racks. Each rack stores a number of DataNodes.

Each DataNode has a large number of data blocks. The racks distribute across a cluster. The nodes have processing and storage capabilities. The nodes have the data in data blocks to run the application tasks.

The data blocks replicate by default at least on three DataNodes in same or remote nodes. Data at the stores enable running the distributed applications including analytics, data mining, OLAP using the clusters. A file, containing the data divides into data blocks.

A data block default size is 64 MBs

Hadoop HDFS features are as follows:

I. Create, append, delete, rename, and attribute modification functions
II. Content of individual file cannot be modified or replaced but appended with new data at the end of the file
III. Write once but read many times during usages and processing
IV. Average file size can be more than 500 MB.

Hadoop Physical Organization

The conventional file system uses directories. A directory consists of folders. A folder consists of files. When data processes, the data sources identify by pointers for the resources. A data-dictionary stores the resource pointers. Master tables at the dictionary store at a central location.

The centrally stored tables enable administration easier when the data sources change during processing.

The files, DataNodes and blocks need the identification during processing at HDFS.

HDFS use the NameNodes and DataNodes.

A NameNode stores the file's meta data. Meta data gives information about the file of user application, but does not participate in the computations.

The DataNode stores the actual data files in the data blocks.

Few nodes in a Hadoop cluster act as NameNodes. These nodes are termed as MasterNodes or simply masters.

The masters have a different configuration supporting high DRAM and processing power. The
masters have much less local storage.

Majority of the nodes in Hadoop cluster act as DataNodes and TaskTrackers. These nodes are referred to as slave nodes or slaves.

The slaves have lots of disk storage and moderate amounts of processing capabilities and DRAM. Slaves are responsible to store the data and process the computation tasks submitted by the clients.

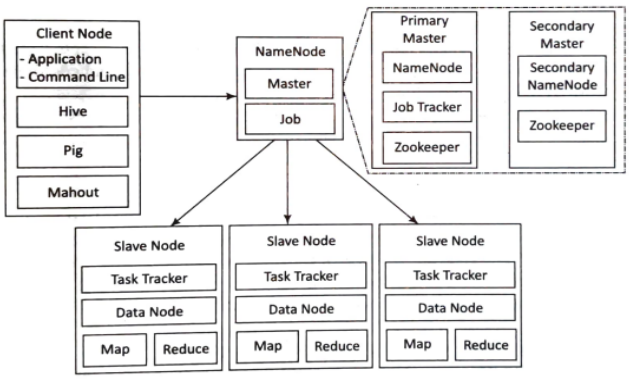


Figure: The client, master NameNode, MasterNodes and slave nodes

Apache Hive is a data warehouse and which provides an SQL-like interface between the user and the Hadoop distributed file system (HDFS) which integrates Hadoop. Difference between Pig and Hive : ... Pig is a Procedural Data Flow Language. Hive is a Declarative SQLish Language.

Clients as the users run the application with the help of Hadoop ecosystem projects. A single MasterNode provides HDFS, MapReduce and Hbase using threads in small to medium sized clusters. When the cluster size is large, multiple servers are used, such as to balance the load.

The secondary NameNode provides NameNode management services and Zookeeper is used by HBase for metadata storage.

The MasterNode fundamentally plays the role of a coordinator. The MasterNode receives client
connections, maintains the description of the global file system namespace, and the allocation of file blocks. It also monitors the state of the system in order to detect any failure. The Masters consists of three components NameNode, Secondary NameNode and JobTracker. The NameNode stores all the file system related information such as:

I. The file section is stored in which part of the cluster
II. Last access time for the files
III. User permissions like which user has access to the file.

Secondary NameNode is an alternate for NameNode. Secondary node keeps a copy of NameNode meta data. Stored meta data can be rebuilt easily, in case of NameNode failure.

The JobTracker coordinates the parallel processing of data.

Masters and slaves, and Hadoop client (node) load the data into cluster, submit the processing job and then retrieve the data to see the response after the job completion.

Hadoop 2

Hadoop 2 provides the multiple NameNodes. This enables higher resource availability. Each MN has the following components:

⇒ An associated NameNode
⇒ Zookeeper coordination client (an associated NameNode), functions as a centralized repository for distributed applications. Zookeeper uses synchronization, serialization and coordination activities. It enables functioning of a distributed system as a single function.
⇒ Associated JournalNode (JN). The JN keeps the records of the state, resources assigned, and intermediate results or execution of application tasks. Distributed applications can write and read data from a JN.

One set of resources is in active state. The other one remains in standby state.

Two masters, one MN1 is in active state and other MN2 is in secondary state.

This ensures the availability in case of network fault of an active NameNode NM1.

The Hadoop system then activates the secondary NameNode NM2 and creates a secondary in another MasterNode MN3 unused earlier.

The entries copy from JN1 in MNI into the JN2 which is at newly active MasterNode MN2. Therefore, the application runs uninterrupted and resources are available uninterrupted.

HDFS Commands

The HDFS shell is not compliant with the POSIX. Thus, the shell cannot interact Similar to Unix or Linux. Commands for interacting with the files in HDES require /bin/hdfs dfs <args>,
where args stands for the command arguments. All Hadoop commands are invoked by the bin/Hadoop script. Hadoop fsck / -files -blocks.

HDFS shell command	Example of usage
-mkdir	Assume stu_filesdir is a directory of student files in Example 2.2. Then command for creating the directory is `$Hadoop hdfs-mkdir/user/stu_filesdir` creates the directory named stu_files_dir
-put	Assume file stuData_id96 to be copied at stu_filesdir directory in Example 2.2. Then `$Hadoop hd-fs-put stuData_id96 /user/ stu_filesdir` copies file for student of id96 into stu_filesdir directory
-ls	Assume all files to be listed. Then `$hdfs hdfs dfs-ls` command does provide the listing.
-cp	Assume stuData_id96 to be copied from stu_filesdir to new students' directory newstu_filesDir. Then `$Hadoop hdfs-cp stuData_id96 /user/stu_filesdir newstu_filesDir` copies file for student of ID 96 into stu_filesdir directory

MapReduce FRAMEWORK AND PROGRAMMING MODEL

MapReduce is a programming model for distributed computing.

Mapper means software for doing the assigned task after organizing the data blocks imported using the keys.

A key specifies in a command line of Mapper. The command maps the key to the data, which an application uses.

Reducer means software for reducing the mapped data by using the aggregation, query or user-specified function. The reducer provides a concise cohesive response for the application.

Aggregation function means the function that groups the values of multiple rows together to result a single value of more significant meaning or measurement. For example, function such as count, sum, maximum, minimum, deviation and standard deviation.

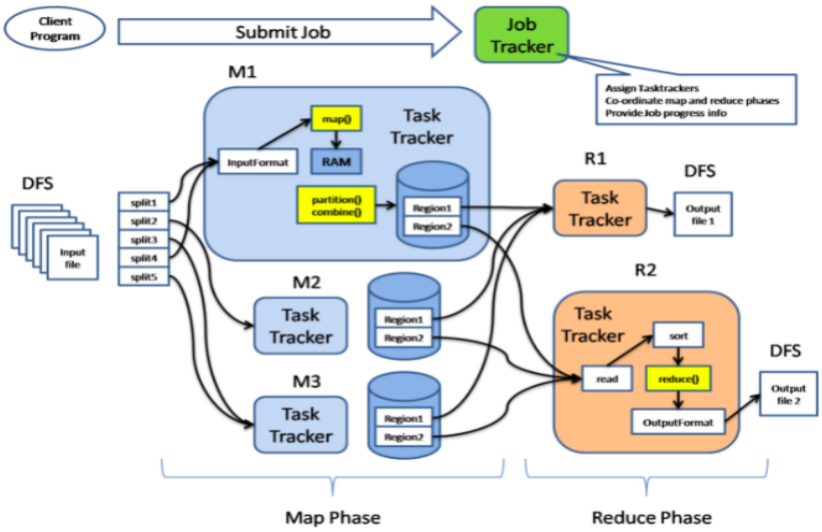
Querying function means a function that finds the desired values. For example, function for finding a best student of a class who has shown the best performance in examination.

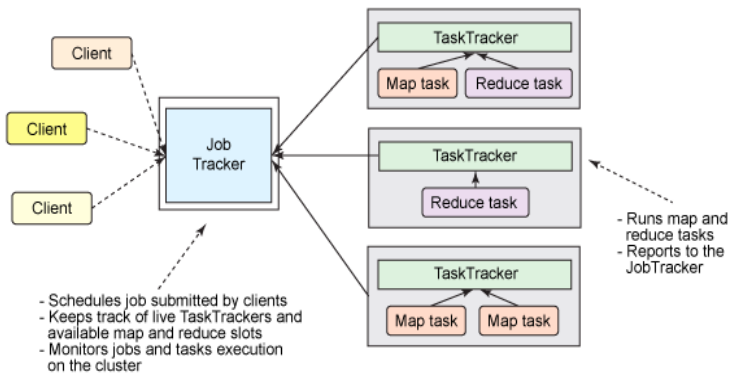
Features of MapReduce framework are as follows:

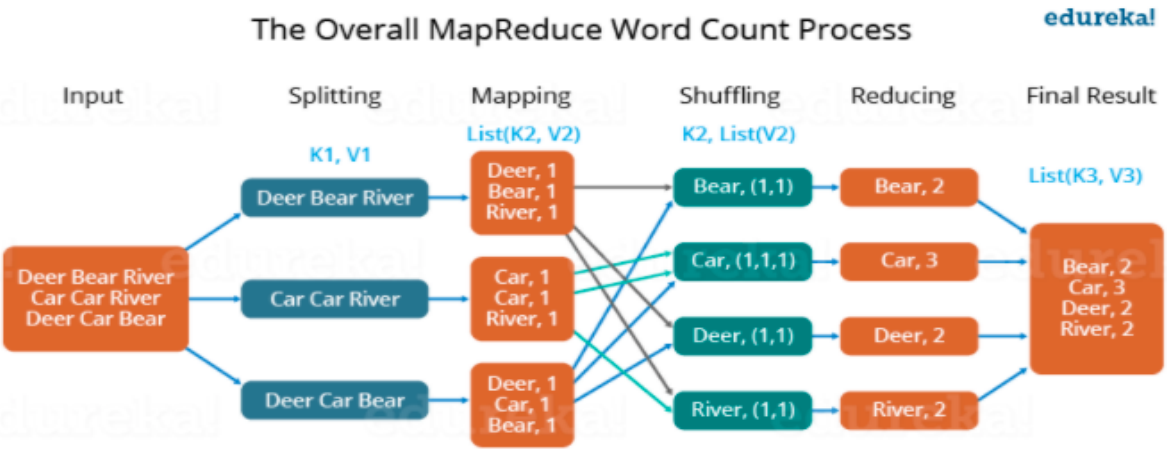
1. Provides automatic parallelization and distribution of computation based on several processors

2. Processes data stored on distributed clusters of DataNodes and racks
3. Allows processing large amount of data in parallel
4. Provides scalability for usages of large number of servers
5. Provides MapReduce batch-oriented programming model in Hadoop version 1
6. Provides additional processing modes in Hadoop 2 YARN-based system and enables required parallel processing. For example, for queries, graph databases, streaming data, messages, real-time OLAP and ad hoc analytics with Big Data 3V characteristics.

Hadoop MapReduce Framework







MapReduce provides two important functions.

⇒ The distribution of job based on client application task or users query to various nodes within a cluster is one function.
⇒ The second function is organizing and reducing the results from each node into a cohesive response to the application or answer to the query.

The processing tasks are submitted to the Hadoop. The Hadoop framework in turns manages the task of issuing jobs, job completion, and copying data around the cluster between the DataNodes with the help of JobTracker.

Daemon refers to a highly dedicated program that runs in the background in a system. The user does not control or interact with that. An example is MapReduce in Hadoop system.

MapReduce runs as per assigned Job by JobTracker, which keeps track of the job submitted for execution and runs TaskTracker for tracking the tasks.

MapReduce programming enables job scheduling and task execution as follows:

A client node submits a request of an application to the JobTracker. A JobTracker is a Hadoop daemon (background program).

The following are the steps on the request to MapReduce:

(i) estimate the need of resources for processing that request,
(ii) analyze the states of the slave nodes,
(iii) place the mapping tasks in queue,
(iv) monitor the progress of task, and on the failure, restart the task on slots of time available.

The job execution is controlled by two types of processes in MapReduce:

1. The Mapper deploys map tasks on the slots. Map tasks assign to those nodes where the data for the application is stored. The Reducer output transfers to the client node after the data serialization using AVRO.
2. The Hadoop system sends the Map and Reduce jobs to the appropriate servers in the cluster. The Hadoop framework in turns manages the task of issuing jobs, job completion and copying data around the cluster between the slave nodes. Finally, the cluster collects and reduces the data to obtain the result and sends it back to the Hadoop server after completion of the given tasks.

The job execution is controlled by two types of processes in MapReduce. A single master process called JobTracker is one. This process coordinates all jobs running on the cluster and assigns map and reduce tasks to run on the TaskTrackers.

The second is a number of subordinate processes called TaskTrackers. These processes run assigned tasks and periodically report the progress to the JobTracker.

MapReduce Programming Model

MapReduce program can be written in any language including JAVA, C++ PIPEs or Python.

Map function of MapReduce program do mapping to compute the data and convert the data into other data sets.

After the Mapper computations finish, the Reducer function collects the result of map and generates the final output result.

MapReduce program can be applied to any type of data, i.e., structured or unstructured stored in HDFS.

The input data is in the form of file or directory and is stored in the HDFS. The MapReduce program performs two jobs on this input data, the Map job and the Reduce job.

Map phase and Reduce phase. The map job takes a set of data and converts it into another set of data. The individual elements are broken down into tuples (key/value pairs) in the resultant set of data.

The reduce job takes the output from a map as input and combines the data tuples into a smaller set of tuples.

Map and reduce jobs run in isolation from one another. As the sequence of the name MapReduce implies, the reduce job is always performed after the map job.

The MapReduce v2 uses YARN based resource scheduling which simplifies the software development.

HADOOP YARN(Yet Another Resource Negotiator):

YARN is a resource management platform which manages computer resources.

The platform is responsible for providing the computational resources, such as CPUs, memory, network I/O which are needed when an application executes.

YARN manages the schedules for running of the sub-tasks. Each sub-task uses the resources in allotted time intervals.

YARN separates the resource management and processing components.

An application consists of a number of tasks. Each task can consist of a number of sub-tasks (threads), which run in parallel at the nodes in the cluster.

YARN enables running of multi-threaded applications.

YARN manages and allocates the resources for the application sub-tasks and submits the resources for them at the Hadoop system.

Hadoop 2 Execution Model

YARN components are:

Client,
Resource Manager (RM),
Node Manager (NM),
Application Master (AM) and Containers.

List of actions of YARN resource allocation and scheduling functions is as follows:

- ⇒ A MasterNode has two components: (i) Job History Server and (i) Resource Manager(RM).

- ⇒ A Client Node submits the request of an application to the RM. The RM is the master. One RM exists per cluster. The RM keeps information of all the slave NMs. Information is about the location and the number of resources (data blocks and servers) they have. The RM also renders the Resource Scheduler service that decides how to assign the resources. It, therefore, performs resource management as well as scheduling.

- ⇒ Multiple NMs are at a cluster. An NM creates an AM instance (AMI) and starts up. The AMI initializes itself and registers with the RM. Multiple AMIs can be created in an AM.

- ⇒ The AMI performs role of an Application Manager (ApplM), that estimates the resources requirement for running an application program or sub-task. The ApplMs send their requests for the necessary resources to the RM. Each NM includes several containers for uses by the subtasks of the application.

- ⇒ NM is a slave of the infrastructure. It signals whenever it initializes. All active NMs send the controlling signal periodically to the RM signaling their presence.

⇒ Each NM assigns a container(s) for each AMI. The container(s) assigned at an instance may be at same NM or another NM. ApplM uses just a fraction of the resources available. The ApplM at an instance uses the assigned container(s) for running the application sub-task.

⇒ RM allots the resources to AM, and thus to ApplMs for using assigned containers on the same or other NM for running the application subtasks in parallel.

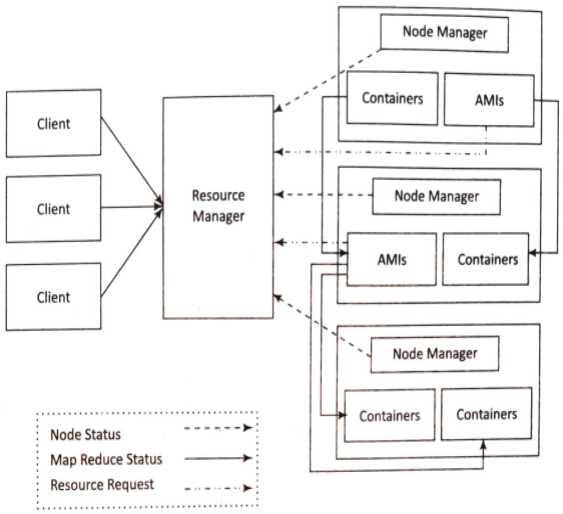


Figure: YARN-based execution model

HADOOPECOSYSTEM TOOLS

1. Zookeeper
2. Oozie
3. Sqoop
4. Flume
5. Ambari

Zookeeper:

Apache is a coordination service that enables synchronization across a cluster in distributed applications.

Zookeeper in Hadoop behaves as a centralized repository where distributed applications can write data at a node called JournalNode and read the data out of it.

It uses synchronization, serialization and coordination activities. It enables functioning of a distributed system as a single function.

ZooKeeper's main coordination services are:

Name service - A name service maps a name to the information associated with that name. For example, DNS service is a name service that maps a domain name to an IP address. Similarly, name keeps a track of servers or services those are up and running, and looks up their status by name in name service.

Concurrency control - Concurrent access to a shared resource may cause inconsistency of the resource. A concurrency control algorithm accesses shared resource in the distributed system and controls concurrency.

Configuration management - A requirement of a distributed system is a central configuration manager. A new joining node can pick up the up-to-date centralized configuration from the ZooKeeper coordination service as soon as the node joins the system.

Failure - Distributed systems are susceptible to the problem of node failures. This requires implementing an automatic recovery strategy by selecting some alternate node for processing

Oozie

Apache Oozie is an open-source project of Apache that schedules Hadoop jobs. An efficient process for job handling is required.

Analysis of Big Data requires creation of multiple jobs and sub-tasks in a process.

Oozie design provisions the scalable processing of multiple jobs. Oozie provides a way to package and bundle multiple coordinator and workflow jobs, and manage the lifecycle of those jobs.

The two basic Oozie functions are:

1. Oozie workflow jobs are represented as Directed Acrylic Graphs (DAGs), specifying a sequence of actions to execute.
2. Oozie coordinator jobs are recurrent Oozie workflow jobs that are triggered by time and data availability.

Oozie provisions for the following:

1. Integrates multiple jobs in a sequential manner
2. Stores and supports Hadoop jobs for MapReduce, Hive, Pig, and Sqoop
3. Runs workflow jobs based on time and data triggers
4. Manages batch coordinator for the applications
5. Manages the timely execution of tens of elementary jobs lying in thousands of workflows in a Hadoop luster

Sqoop

Apache Sqoop is a tool that is-built for loading efficiently the voluminous amount of data between Hadoop and external data repositories that resides on enterprise application servers or relational databases.

Sqoop works with relational databases such as Oracle, MySQL, PostgreSQL and DB2.

Sqoop provides the mechanism to import data from external Data Stores into HDFS.

Sqoop relates to Hadoop eco-system components, such as Hive and HBase. Sqoop can extract data from Hadoop or other ecosystem components.

Sqoop provides command line interface to its users.

The tool allows defining the schema of the data for import. Sqoop exploits MapReduce framework to import and export the data, and transfers for parallel processing of sub-tasks.

Sqoop provisions for fault tolerance.
Sqoop initially parses the arguments passed in the command line and prepares the map task.

The map task initializes multiple Mappers depending on the number supplied by the user in the command line.

Sqoop distributes the input data equally among the Mappers. Then each Mapper creates a connection with the database using JDBC and fetches the part of data assigned by Sqoop and writes it into HDFS/Hive/HBase as per the choice provided in the command line.

Flume

Apache Flume provides a distributed, reliable, and available service.

Flume efficiently collects, aggregates and transfers a large amount of streaming data into HDFS.

Flume enables upload of large files into Hadoop clusters.

The features of flume include robustness and fault tolerance. Flume provides data transfer which is reliable and provides for recovery in case of failure.

Flume is useful for transferring a large amount of data in applications related to logs of network traffic, sensor data, geo-location data, e-mails and social-media messages.

Apache Flume has the following four important components:

1. Sources which accept data from a server or an application.
2. Sinks which receive data and store it in HDFS repository or transmit the data to another source. Data units that are transferred over a channel from source to sink are called events.
3. Channels connect between sources and sink by queuing event data for transactions. The size of events data is usually 4 KB. The data source is considered to be a source of various set of events. Sources listen for events and write events to a channel. Sinks basically write event data to a target and remove the event from the queue.
4. Agents run the sinks and sources in Flume. The interceptors drop the data or transfer data as it flows into the system.

Ambari

Apache Ambari is a management platform for Hadoop. It is open source.

Ambari enables an enterprise to plan, securely install, manage and maintain the clusters in the Hadoop.

Ambari provisions for advanced cluster security capabilities, such as Kerberos Ambari.

Features of Ambari and associated components are as follows:

1. Simplification of installation, configuration and management
2. Enables easy, efficient, repeatable and automated creation of clusters
3. Manages and monitors scalable clustering
4. Provides an intuitive Web User Interface and REST API. The provision enables automation cluster operations.
5. Visualizes the health of clusters and critical metrics for their operations
6. Enables detection of faulty node links
7. Provides extensibility and customizability.

Hadoop Administration

Administrator procedures enable managing and administering Hadoop clusters, resources, and associated Hadoop ecosystem components.

Administration includes installing and monitoring clusters.

Ambari provides a centralized setup for security.

Ambari helps automation of the setup and configuration of Hadoop using Web User Interface and REST APIs.

The console is similar to web UI at Ambari. The console enables visualization of the cluster health, HDFS directory structure, status of MapReduce tasks review of log records and access application status.

Single harmonized view on console makes administering the task easier. Visualization can be up to individual components level on drilling down. Nodes addition and deletion are easy using the console.

HBase

HBase is an Hadoop system database. HBase was created for large tables. HBase is an open-source, distributed, versioned and non-relational (NoSQL) database.

Features of HBase features are:

1. Uses a partial columnar data schema on top of Hadoop and HDFS.
2. Supports a large table of billions of rows and millions of columns.

3. Provides small amounts of information, called sparse data taken from large data sets which are storing empty or presently not-required data. For example, yearly sales data of KitKats from the data of hourly, daily and monthly sales.
4. Supports data compression algorithms.
5. Provisions in-memory column-based data transactions.
6. Accesses rows serially and does not provision for random accesses and write into the rows.
7. Provides random, real-time read/write access to Big Data.
8. Fault tolerant storage due to automatic failure support between DataNodes servers.
9. Similarity with Google BigTable.

⇒ HBase is written in Java. It stores data in a large structured table.

⇒ HBase provides scalable distributed Big Data Store.

⇒ HBase data store as key-value pairs.

⇒ HBase system consists of a set of tables. Each table contains rows and columns, similar to a traditional database.

⇒ HBase provides a primary key as in database table.

⇒ HBase applies a partial columnar scheme on top of the Hadoop and HDFS.

Hive

Apache Hive is an open-source data warehouse software. Hive facilitates reading, writing and managing large datasets which are at distributed Hadoop files.

Hive uses SQL

Hive design provisions for batch processing of large sets of data.

Hive does not process real time queries and does not update row-based data tables.

Hive enables data serialization/deserialization and increase flexibility in schema design by including a system catalog called Hive Metastore.

Hive supports different storage types like text files, sequence files, RC Files, ORC Files and HBase.

Pig

Apache Pig is an open source, high-level language platform. Pig was developed for analyzing large-data sets.

Pig executes queries on large datasets that are stored in HDFS using Apache Hadoop.

Pig Latin language is similar to SQL query language but applies on Pig are as follows: larger datasets.

Additional features of Pig are as follows:

I. Loads the data after applying the required filters and dumps the data in the desired format.
II. Requires Java runtime environment for executing Pig Latin programs.
III. Converts all the operations into map and reduce tasks. The tasks run on Hadoop.
IV. Allows concentrating upon the complete operation, Reducer functions to irrespective of the individual Mapper and reducer functions to produce the output results.

Mahout

Mahout is a Java library Implementing Machine Learning techniques for clustering, classification, and recommendation.

Apache Mahout features are:

⇒ Collaborative data-filtering that mines user behavior and makes product recommendations.
⇒ Clustering that takes data items in a particular class, and organizes them into naturally occurring groups, such that items belonging to the same group are similar to each other.
⇒ Classification that means learning from existing categorizations and then assigning the future items to the best category.
⇒ Frequent item-set mining that analyzes items in a group and then identifies which items
usually occur together.

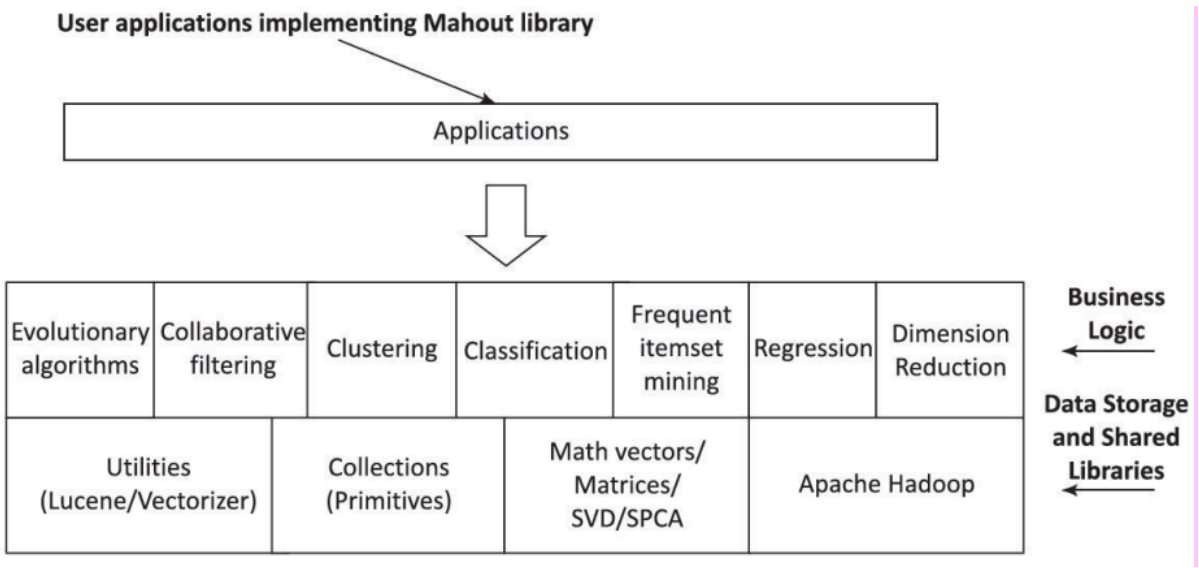


Figure: Mahout architecture

Hadoop Distributed File System Basics

The Hadoop Distributed File System (HDFS) was designed for Big Data processing.

HDFS rigorously restricts data writing to one user at a time.

All additional writes are "append-only," and there is no random writing to HDFS files.

The design of HDFS is based on the design of the Google File System (GFS).

HDFS is designed for data streaming where large amounts of data are read from disk in bulk.

The HDFS block size is typically 64MB or 128MB.

In HDFS there is no local caching mechanism. The large block and file sizes make it more efficient to reread data from HDFS than to try to cache the data.

Important feature of HDFS is **data locality.**

MapReduce is the emphasis on moving the computation to the data rather than moving the data to the computation.

The following points are the important aspects of HDFS:

- ⇒ The write-once/read-many design is intended to facilitate streaming reads.
- ⇒ Files may be appended, but random seeks are not permitted. There is no caching of data.
- ⇒ Converged data storage and processing happen on the same server nodes.
- ⇒ "Moving computation is cheaper than moving data."
- ⇒ A reliable file system maintains multiple copies of data across the cluster. Consequently, failure of a single node (or even a rack in a large cluster) will not bring down the file system.
- ⇒ A specialized file system is used, which is not designed for general use.

HDFS Components

HDFS consists of two main components:

A NameNode
Multiple DataNodes

A single NameNode manages all the metadata needed to store and retrieve the actual data from the DataNodes.

The design is a master/slave architecture in which the master (NameNode) manages the file system namespace and regulates access to files by clients.

File system namespace operations such as opening, closing, and renaming files and directories are all managed by the NameNode.

The NameNode also determines the mapping of blocks to DataNodes and handles DataNode failures.

The slaves (DataNodes) are responsible for serving read and write requests from the file system to the clients.

The NameNode manages block creation, deletion, and replication.

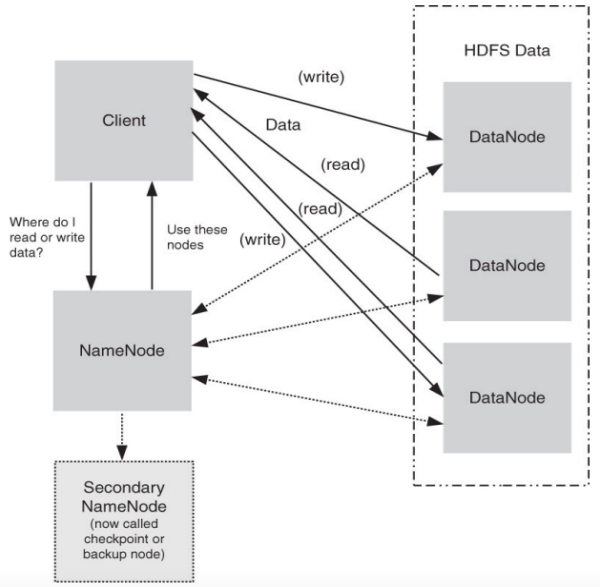


Figure: Various system roles in an HDFS deployment

When a client writes data, it first communicates with the NameNode and requests to create a file. The NameNode determines how many blocks are needed and provides the client with the DataNodes that will store the data.

As part of the storage process, the data blocks are replicated after they are written to the assigned node.

Depending on how many nodes are in the cluster, the NameNode will attempt to write replicas of the data blocks on nodes that are in other separate racks. If there is only one rack, then the replicated blocks are written to other servers in the same rack.

After the DataNode acknowledges that the file block replication is complete, the client closes the file and informs the NameNode that the operation is complete.

The NameNode does not write any data directly to the DataNodes. It give the client a limited amount of time to complete the operation. If it does not complete in the time period, the operation is canceled.

Reading data happens in a similar fashion.

The client requests a file from the NameNode, which returns the best DataNodes from which to read the data.

The client then accesses the data directly from the DataNodes. once the metadata has been delivered to the client, the NameNode steps back and lets the conversation between the client and the DataNodes proceed.

While data transfer is progressing, the NameNode also monitors the DataNodes by listening for heartbeats sent from DataNodes for detecting the failure.

If the DataNodes fail the NameNode will route around the failed DataNode and begin re-replicating the now-missing blocks.

The block reports are sent every 10 heartbeats. The reports enable the NameNode to keep an up-to-date account of all data blocks in the cluster.

The purpose of the SecondaryNameNode is to perform periodic checkpoints that evaluate the status of the NameNode.

It also has two disk files that track changes to the metadata:

An image of the file system state when the NameNode was started. This file begins with fsimage_* and is used only at startup by the NameNode.

A series of modifications done to the file system after starting the NameNode. These files begin with edit_* and reflect the changes made after the fsimage_* file was read.

The location of these files is set by the dfs.namenode.name.dir property in the hdfs-site.xml file.

The SecondaryNameNode periodically downloads fsimage and edits files, joins them into a new fsimage, and uploads the new fsimage file to the NameNode.

The various roles in HDFS can be summarized as follows:

⇒ HDFS uses a master/slave model designed for large file reading/streaming.

⇒ The NameNode is a metadata server or "data traffic cop."
⇒ HDFS provides a single namespace that is managed by the NameNode.
⇒ Data is redundantly stored on DataNodes; there is no data on the NameNode.
⇒ The SecondaryNameNode performs checkpoints of NameNode file system's state but is not a failover node.

HDFS Block Replication

The amount of replication is based on the value of dfs.replication in the hdfs-site.xml file.

This default value can be overruled with the hdfs dfs-setrep command. For Hadoop clusters containing more than eight DataNodes, the replication value is usually set to 3.

The HDFS default block size is 64MB. In a typical operating system, the block size is 4KB or 8KB. The HDFS default block size is not the minimum block size. If a 20KB file is written to HDFS, it will create a block that is approximately 20KB in size. If a file of size 80MB is written to HDFS, a 64MB block and a 16MB block will be created.

HDFS blocks are not exactly the same as the data splits used by the MapReduce process.

The HDFS blocks are based on size, while the splits are based on a logical partitioning of the data. For instance, if a file contains discrete records, the logical split ensures that a record is not split physically across two separate servers during processing. Each HDFS block may consist of one or more splits.

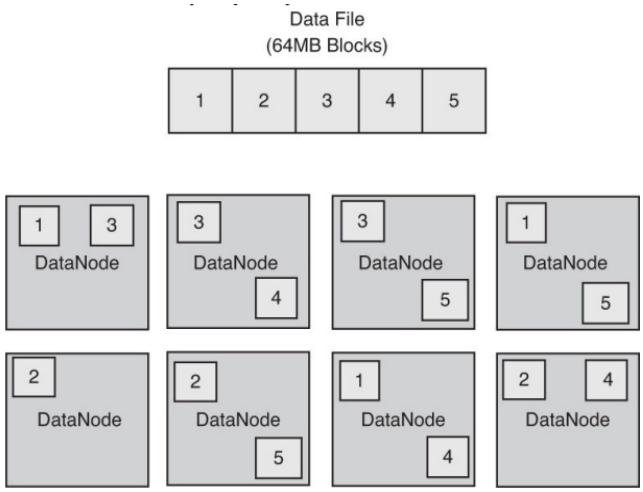


Figure: HDFS block replication example

HDFS User Commands

$ hdfs version - The version of HDFS
$ hdfs dfs -ls / - To list the files in the root HDFS directory
$ hdfs dfs -ls or $ hdfs dfs -ls /user/hdfs - To list files in your home directory
$ hdfs dfs -mkdir stuff - To make a directory in HDFS
$ hdfs dfs -put test stuff - To copy a file from your current local directory into HDFS
$ hdfs dfs -get stuff/test test-local – To copy Files from HDFS
$ hdfs dfs -cp stuff/test test.hdfs – To copy Files within HDFS
$ hdfs dfs -rm test.hdfs – To delete a File within HDFS

$ hdfs dfs -rm -r -skipTrash stuff – To delete a Directory in HDFS
$ hdfs dfsadmin -report – To get an HDFS Status Report

Essential Hadoop Tools

Using Apache Pig

Apache Pig is a high-level language that enables programmers to write complex MapReduce transformations using a simple scripting language.

Pig Latin defines a set of transformations on a data set such as aggregate, join, and sort.

Pig is used to extract, transform, and load (ETL) data pipelines, quick research on raw data, and iterative data processing.

Apache Pig has several usage modes. The first is a local mode in which all processing is done on the local machine.

The non-local (cluster) modes are MapReduce and Tez. These modes execute the job on the cluster using either the MapReduce engine or the optimized Tez engine.

	Local Mode	Tez Local Mode	MapReduce Mode	Tez Mode
Interactive Mode	Yes	Experimental	Yes	Yes
Batch Mode	Yes	Experimental	Yes	Yes

Figure: Apache Pig Usage Modes

Using Apache Hive

Apache Hive is a data warehouse infrastructure built on top of Hadoop for providing data summarization, ad hoc queries, and the analysis of large data sets using a SQL-like language called HiveQL.

Hive is considered the de facto standard for interactive SQL queries over petabytes of data using Hadoop and offers the following features:

Tools to enable easy data extraction, transformation, and loading (ETL)
A mechanism to impose structure on a variety of data formats
Access to files stored either directly in HDFS or in other data storage systems such as HBase
Query execution via MapReduce and Tez

Hive provides users to query the data on Hadoop clusters using SQL

Hive makes it possible for programmers who are familiar with the MapReduce framework to add their custom mappers and reducers to Hive queries.

Hive queries can be dramatically accelerated using the Apache Tez framework under YARN in Hadoop version 2.

Using Apache Sqoop to Acquire Relational Data

Sqoop is a tool designed to transfer data between Hadoop and relational databases.

Sqoop can be used with any Java Database Connectivity (JDBC)–compliant database and has been tested on Microsoft SQL Server, Postgres SQL, MySQL, and Oracle.

In version 1 of Sqoop, data were accessed using connectors written for specific databases.

Version 2 does not support connectors or version 1 data transfer from a RDBMS directly to Hive or HBase, or data transfer from Hive or HBase to your RDBMS. Instead, version 2 offers more generalized ways to accomplish these tasks.

Apache Sqoop Import and Export Methods

Sqoop data import (to HDFS) process is done in two steps:

1. In the first step, Sqoop examines the database to gather the necessary metadata for the data to be imported.

2. The second step is a map-only (no reduce step) Hadoop job that Sqoop submits to the cluster. This job does the actual data transfer using the metadata captured in the previous step. Each node doing the import must have access to the database.

The imported data are saved in an HDFS directory. Sqoop will use the database name for the directory, or the user can specify any alternative directory where the files should be populated.

By default, these files contain comma-delimited fields, with new lines separating different records.

We can override the format in which data are copied over by explicitly specifying the field separator and record terminator characters. Once placed in HDFS, the data are ready for processing.

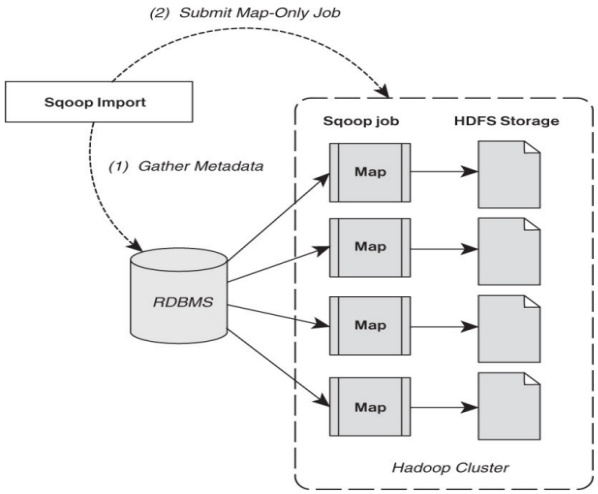


Figure: Two-step Apache Sqoop data import method

The export is done in two steps:

The first step is to examine the database for metadata. The export step again uses a map-only Hadoop job to write the data to the database.

Sqoop divides the input data set into splits, then uses individual map tasks to push the splits to the database. This process assumes the map tasks have access to the database.

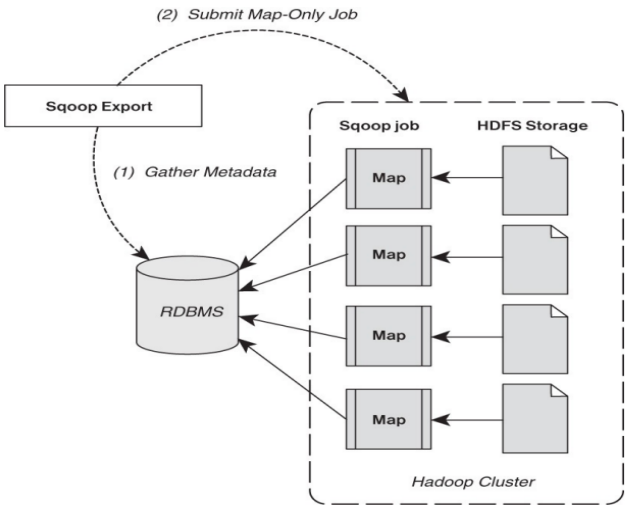


Figure: Two-step Sqoop data export method

Apache Sqoop Version Changes

Feature	Sqoop Version 1	Sqoop Version 2
Connectors for all major RDBMSs	Supported.	Not supported. Use the generic JDBC connector.
Kerberos security integration	Supported.	Not supported.
Data transfer from RDBMS to Hive or HBase	Supported.	Not supported. First import data from RDBMS into HDFS, then load data into Hive or HBase manually.
Data transfer from Hive or HBase to RDBMS	Not supported. First export data from Hive or HBase into HDFS, and then use Sqoop for export.	Not supported. First export data from Hive or HBase into HDFS, then use Sqoop for export.

Using Apache Flume to Acquire Data Streams

Apache Flume is an independent agent designed to collect, transport, and store data into HDFS.

Data transport involves a number of Flume agents that may traverse a series of machines and locations.

Flume is used for log files, social media-generated data, email messages, and any continuous data source.

Flume agent is composed of three components:

Source. The source component receives data and sends it to a channel. It can send the data to more than one channel. The input data can be from a real-time source (e.g., weblog) or another Flume agent.

Channel. A channel is a data queue that forwards the source data to the sink destination. It can be thought of as a buffer that manages input (source) and output (sink) flow rates.

Sink. The sink delivers data to destination such as HDFS, a local file, or another Flume agent.

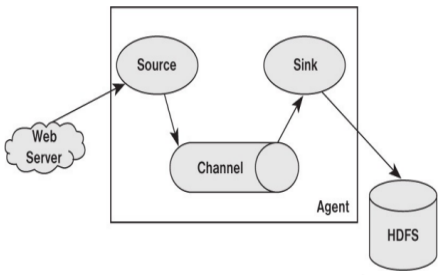


Figure: Flume agent with source, channel, and sink

A Flume agent can have several sources, channels, and sinks. Sources can write to multiple channels, but a sink can take data from only a single channel.

Data written to a channel remain in the channel until a sink removes the data.

By default, the data in a channel are kept in memory but may be optionally stored on disk to prevent data loss in the event of a network failure.

Sqoop agents may be placed in a pipeline, possibly to traverse several machines or domains.

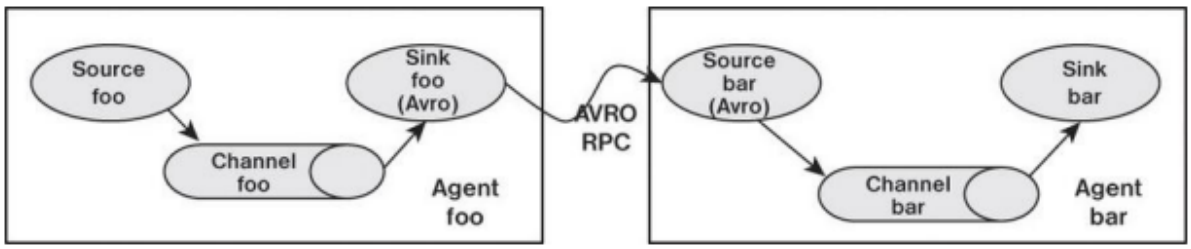


Figure: Pipeline created by connecting Flume agents

In a Flume pipeline, the sink from one agent is connected to the source of another. The data transfer format normally used by Flume, is called Apache Avro.

First, Avro is a data serialization/deserialization system that uses a compact binary format. The
schema is sent as part of the data exchange and is defined using JSON.

Avro also uses remote procedure calls (RPCs) to send data. Avro sink will contact an Avro source to send data.

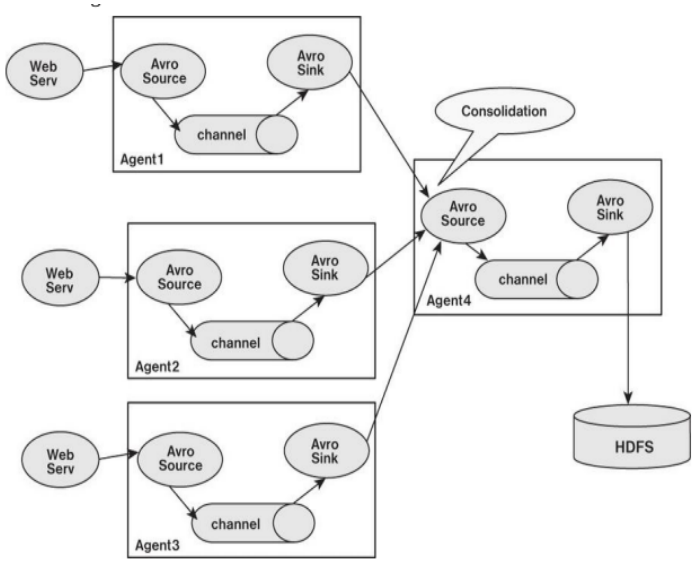


Figure: A Flume consolidation network

Manage Hadoop Workflows with Apache Oozie

Oozie is a workflow director system designed to run and manage multiple related Apache Hadoop jobs.

Oozie is not a substitute for the YARN scheduler. Oozie provides a way to connect and control Hadoop jobs on the cluster.

Oozie workflow jobs are represented as directed acyclic graphs (DAGs) of actions.

Three types of Oozie jobs are permitted:

Workflow—a specified sequence of Hadoop jobs with outcome-based decision points and control dependency. Progress from one action to another cannot happen until the first action is complete.
Coordinator—a scheduled workflow job that can run at various time intervals or when data become available.
Bundle—a higher-level Oozie abstraction that will batch a set of coordinator jobs.

Oozie is integrated with the rest of the Hadoop stack, supporting several types of Hadoop jobs out of the box as well as system-specific jobs

Oozie provides a CLI and a web UI for monitoring jobs.

Oozie runs a basic MapReduce operation. If the application was successful, the job ends; if an error occurred, the job is killed.

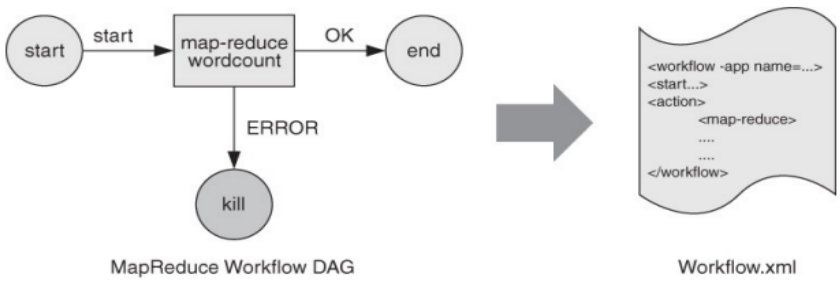


Figure: A simple Oozie DAG workflow

Oozie workflow definitions are written in hPDL. Such workflows contain several types of nodes:

Control flow nodes define the beginning and the end of a workflow. They include start, end, and optional fail nodes.

Action nodes are where the actual processing tasks are defined. When an action node finishes, the remote systems notify Oozie and the next node in the workflow is executed.
Action nodes can also include HDFS commands.

Fork/join nodes enable parallel execution of tasks in the workflow. The fork node enables two or more tasks to run at the same time. A join node represents a rendezvous point that must wait until all forked tasks complete.

Control flow nodes enable decisions to be made about the previous task. Control decisions are based on the results of the previous action. Decision nodes are essentially switch-case statements that use JSP EL (Java Server Pages—Expression Language)
that evaluate to either true or false.

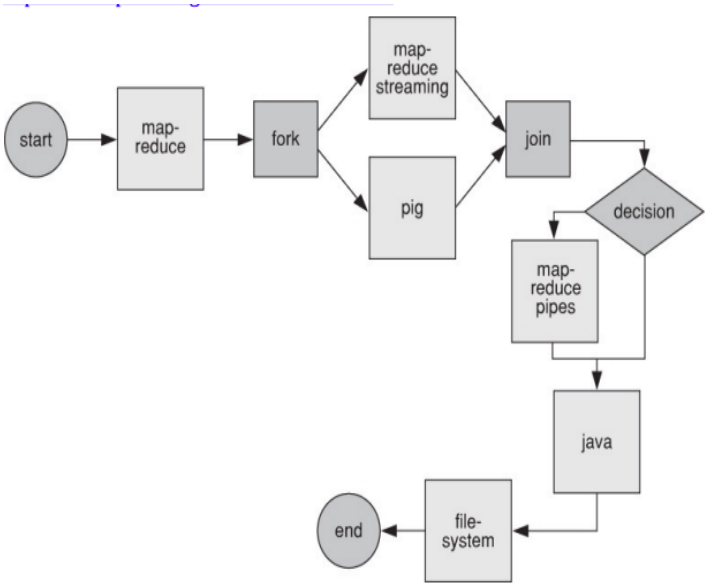


Figure: A more complex Oozie DAG workflow

Using Apache HBase

Apache HBase is an open source, distributed, versioned, nonrelational database modeled after Google's Bigtable.

Apache HBase provides Bigtable-like capabilities on top of Hadoop and HDFS.

Important features are:

Linear and modular scalability
Strictly consistent reads and writes
Automatic and configurable sharding of tables
Automatic failover support between RegionServers
Convenient base classes for backing Hadoop MapReduce jobs with Apache HBase tables
Easy-to-use Java API for client access