Module 2
INTRODUCTION TO HADOOP, HDFS AND ESSENTIAL HADOOP TOOLS

Introduction To Hadoop

2.1 Introduction:

A programming model is **centralized computing** of data in which 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. An enterprise collects and analyzes data at the enterprise level.

For **example**, at 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 media. Applications running at the server does the following analysis: Suggests a strategy for filling the machines at minimum cost of logistics

- Finds locations of high sales such as gardens, playgrounds etc.
- Finds days or periods of high sales such as Xmas etc.
- Finds children's preferences for specific chocolate flavors
- Finds the potential region of future growth
- Identifies unprofitable machines
- Identifies need of replicating the number of machines at specific locations.

Another programming model is **distributed computing** that uses the databases at multiple computing nodes with data sharing between the nodes during computation.

Distributed computing in 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.

The following example shows why the simply scaling out and division of the computations on a large number of processors may not work well due to data sharing between distributed computing nodes.

Consider a jigsaw Puzzle Ravensburger Beneath the Sea (5000 pieces). What will be the effect on time intervals for solution in three situations, when 4, 100 and 200 children simultaneously attempt the solution?

Solution Let the time taken by a single child to solve the puzzle be T. Assume 4 children sit together and solve the puzzle by dividing the tasks. Each child assembles one-fourth part of the picture for which they pick the pieces from a common basket (Distributed computing and centralized data model). Alternatively, each child assembles one-fourth part of the picture for which the pieces are distributed in four baskets. The child in case does not find a piece in his/her basket, then searches for it in another basket (Distributed databases and distributed computing tasks with data sharing model). Partitioning of assembling jobs into four has an issue. A child may complete his/her part much later than the remaining children. Beneath-the-sea portion is too complex, while upper-depth-sea portion is just plain. The children combine all four parts and finally complete the puzzle. Each one has to look into the other three parts to find a match and complete the task. Time taken to solve the puzzle is [T/4 + TI (4) + TC (4)], where TI (4) is the time taken in seeking from others the pieces not available to a child during intermediate phases, and TC (4) in combining the results of the four children. Scaling factor is slightly less than 4. The proposed distributed model works well.

Assume a second situation in which 100 children assemble their parts of 50 pieces each, and finally combine all 100 parts and complete the puzzle. Each child must seek a piece, not available with her/him during the intermediate phase. Combining also becomes difficult and a time-consuming exercise compared to the four children case because each child now matches the results with the remaining 99 counterparts to arrive at the final solution. The time taken to solve the puzzle is [T/100 + TI (100) + TC (100)], where TI (100) and TC (100) are the time taken in seeking pieces not available with the child and combining results

of 100 children, respectively. Scaling is by factor less than 100. The distributed model has issues like sharing pieces, seeking pieces not available and combining issues. Issues are at the intermediate as well as at the end stages.

If 200 children attempt to solve the puzzle simultaneously at the same time then finally combining all 200 portions of the Beneath the Sea, the integration of 200 portions will be tedious and will be a far more time-consuming exercise than with 4 or 100. The time taken to solve the puzzle is [T/200 + TI (200)+ TC (200)], where TI (200) and TC (200) is the time taken in seeking the pieces not available and combining, respectively. Scaling up is by factor much less than 200 and may even be less than even 100. The distributed model with pieces sharing between the children is unsatisfactory because TI (200) + TC (200)< T/200.

Problem of inter-children interactions exponentially grows with the number of children in the proposed distributed model with seeking pieces in intermediate phases. Time TI becomes significantly high. Alternatively, the picture parts and corresponding pieces of each part distribute to each participating child distinctly (Distributed computing model with no data sharing). Time TI taken in seeking a piece not available with him/her is zero. The time taken in joining the assembled picture portions is only at the end. Problem of inter-children interactions during solving the puzzle does not exist.

Traditionally, a program when executes calls the data inputs. Centralized computing model requires few communication overheads. Distributed computing model requires **communication overheads** for seeking data from a remote source when not available locally, and arrive at the final result. The completion of computations will take more and more time when the number of distributed computing nodes increase**.**

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. Following are the reasons for this:

- 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.

Consider distributed computing model which requires no sharing between data nodes. The model is equivalent to distribution of the picture's parts and corresponding pieces of each part to each participating child distinctly. Multiple tasks of an application also distribute, run using machines associated with multiple data nodes and execute at the same time in parallel.

The application tasks and datasets needed for computations distribute at a number of geographic locations and remote servers. The enterprise uses MPPs or computing clusters when datasets are too large. Application is divided in number of tasks and sub-tasks. The sub-tasks get inputs from data nodes at the same cluster. The results of sub-tasks aggregate and communicate to the application. The aggregate results from each cluster collect using APIs at the application.

(i) Big Data Store Model :

A model for Big Data store is as follows: 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.

- **Data Store model of files in data nodes in racks in the clusters**. 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.

(ii) 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. Other example is instructions for sorting the examination performance data 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 period.
- Hadoop system uses the programming model, where jobs or tasks are assigned and scheduled on the same servers which hold the data.
- Hadoop is one of the widely used technologies. Google and Yahoo use Hadoop. Hadoop creators created a cost-effective method to build search indexes. Facebook, Twitter and LinkedIn use Hadoop. IBM implemented BigInsights and uses licensed Apache Hadoop. Oracle implements Hadoop system with Big Data Appliance, IBM with Infosphere and Microsoft with Big Data solutions.

2.2 Hadoop and its Ecosystem

- Apache initiated the project for developing storage and processing framework for Big Data storage and processing. Doug Cutting and Machael J. Cafarelle the creators named that framework as Hadoop.
- Cutting's son was fascinated by a stuffed toy elephant, named Hadoop, and this is how the name Hadoop was derived.
- The project consisted of two components, one of them is for data store in blocks in the clusters and the other is computations at each individual cluster in parallel with another.
- Hadoop components are written in Java with part of native code in C. The command line utilities are written in shell scripts. 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 (above 10 million bytes) 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 detect and handle failures at the task level.
- The hardware scales up from a single server to thousands of machines that store the clusters. Each cluster stores a large number of data blocks in racks. Default data block size is 64 MB. IBM BigInsights, built on Hadoop deploys default 128 MB block size.
- The Hadoop system manages both, large-sized structured and unstructured data in different formats, such as XML, JSON and text with efficiency and effectiveness. The Hadoop system performs better with clusters of many servers when the focus is on horizontal scalability. The system provides faster results from Big Data and from unstructured data as well.
- Yahoo has more than 100000 CPUs in over 40000 servers running Hadoop, with its biggest Hadoop cluster running 4500 nodes as of March 2017, according to the Apache Hadoop website. Facebook has 2 major clusters: a cluster has 1100-machines with 8800 cores and about 12 PB raw storage. A

300-machine cluster with 2400 cores and about 3 PB raw-storage. Each (commodity) node has 8 cores and 12 TB of storage.

2.2.1 Hadoop Core Components: The Hadoop core components of the framework are:

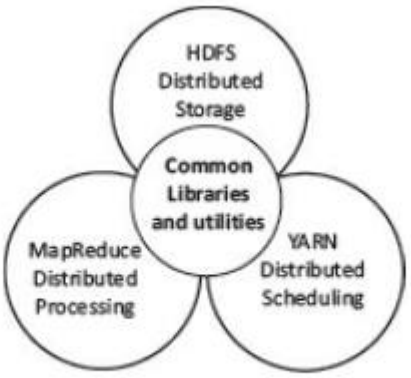


Figure 2.1 Core components of Hadoop

- **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 (Remote Procedure Call) and file-based data structures.
- **Hadoop Distributed File System (HDFS)** – A Java-based distributed file system which can store all kinds of data on the disks at the clusters.
- **MapReduce v1** – Software programming model in Hadoop 1 using Mapper and Reducer. The v1 processes large sets of data in parallel and in batches.
- **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.
- **MapReduce v2** – Hadoop 2 YARN-based system for parallel processing of large datasets and distributed processing of the application tasks.

2.2.1.1 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. Therefore, it also 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. Spark is now increasingly becoming popular.

2.2.2 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 (due to replications at least three times for each data block) 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 commands support.

2.2.3 Hadoop 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 enables the applications which run Big Data and deploy HDFS. The data store system consists of clusters, racks, DataNodes and blocks. Hadoop deploys application programming model, such as MapReduce and HBase. YARN manages resources and schedules sub-tasks of the application. HBase uses columnar databases and does OLAP.
- The system includes the application support layer and application layer components- AVRO, ZooKeeper, Pig, Hive, Sqoop, Ambari, Chukwa, Mahout, Spark, Flink and Flume.
- Figure also shows the components and their usages.

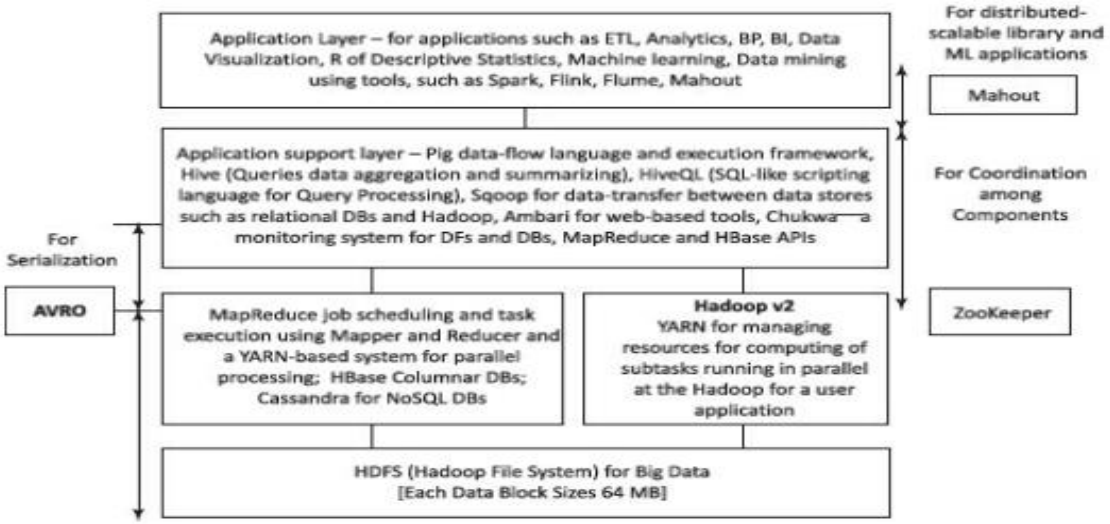


Figure 2.2 Hadoop main components and ecosystem components

- The four layers in Figure 2.2 are as follows:
  (i) Distributed storage layer
  (ii) Resource-manager layer for job or application sub-tasks scheduling and execution
  (iii) Processing-framework layer, consisting of Mapper and Reducer for the MapReduce process-flow
  (iv) APIs at application support layer (applications such as Hive and Pig). The codes communicate and run using MapReduce or YARN at processing framework layer. Reducer output communicate to APIs .
- AVRO enables data serialization between the layers. Zookeeper enables coordination among layer components. The holistic view of Hadoop architecture provides an idea of implementation of Hadoop components of the ecosystem. Client hosts run applications using Hadoop ecosystem projects, such as Pig, Hive and Mahout. Most commonly,
- 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.

2.2.4 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 processing 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.

2.2.5 Hadoop Pipes

Hadoop Pipes are the C++ Pipes which interface with MapReduce. Java native interfaces are not used in pipes. 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. Pipes do not use the standard I/O when communicating with Mapper and Reducer codes. Cloudera distribution including Hadoop (CDH) version CDH 5.0.2 runs the pipes. Distribution means software downloadable from the website distributing the codes. IBM PowerLinux systems enable working with Hadoop pipes and system libraries.

2.3 HADOOP DISTRIBUTED 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 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.

2.3.1 HDFS Data Storage

Hadoop data store concept implies storing the data at a number of 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. The following is an example how the files store at a Hadoop cluster.

Example 2.2 Consider a data storage for University students. Each student data, stuData which is in a file of size less than 64 MB (1 MB = 220B). A data block stores the full file data for a student of stuData_idN, where N = 1 to 500. (i) How the files of each student will be distributed at a Hadoop cluster? How many student data can be stored at one cluster? Assume that each rack has two DataNodes for processing each of 64 GB (1 GB = 230B) memory. Assume that cluster consists of 120 racks, and thus 240 DataNodes. (ii) What is the total memory capacity of the cluster in TB ((1 TB = 240B) and DataNodes in each rack? (iii) Show the distributed blocks for students with ID= 96 and 1025. Assume default replication in the DataNodes = 3. (iv) What shall be the changes when a stuData file size ≤ 128 MB?

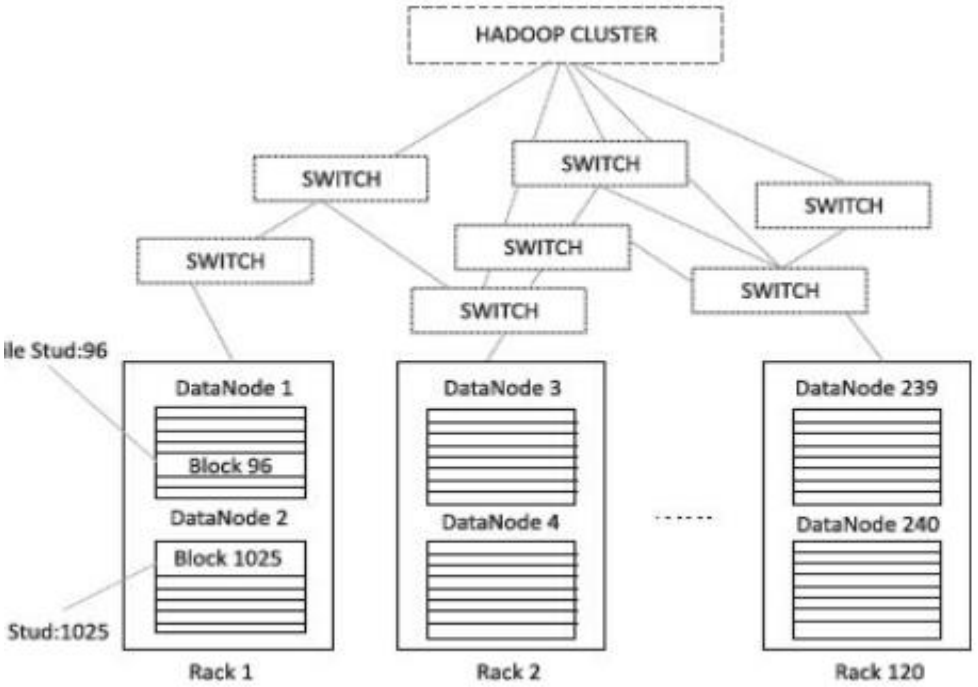
Solution (i) Data block default size is 64 MB. Each students file size is less than 64MB. Therefore, for each student file one data block suffices. A data block is in a DataNode. Assume, for simplicity, each rack has two nodes each of memory capacity = 64 GB. Each node can thus store 64 GB/64MB = 1024 data blocks = 1024 student files. Each rack can thus store 2 × 64 GB/64MB = 2048 data blocks = 2048

student files. Each data block default replicates three times in the DataNodes. Therefore, the number of students whose data can be stored in the cluster = number of racks multiplied by number of files divided by 3 = 120 × 2048/3 = 81920. Therefore, the maximum number of 81920 stuData_IDN files can be distributed per cluster, with N = 1 to 81920.

(ii) Total memory capacity of the cluster = 120 × 128 MB = 15360 GB = 15 TB. Total memory capacity of each DataNode in each rack = 1024 × 64 MB = 64 GB.

(iii) Figure 2.3 shows a Hadoop cluster example, and the replication of data blocks in racks for two students of IDs 96 and 1025. Each stuData file stores at two data blocks, of capacity 64 MB each. (iv) Changes will be that each node will have half the number of data blocks.

Figure 2.3 A Hadoop cluster example, and the replication of data blocks in racks for two students of IDs 96 and 1025



2.3.1.1 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.
- Similarly, 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 2.4 shows the client, master NameNode, primary and secondary MasterNodes and slave nodes in the Hadoop physical architecture.

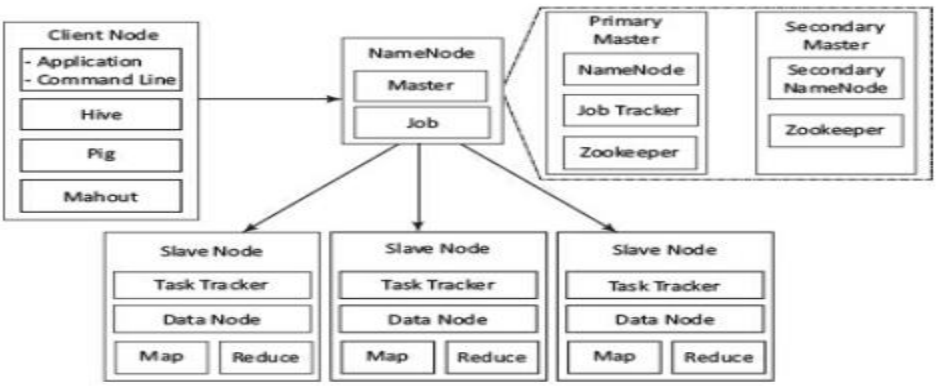


Figure 2.4 The client, master NameNode, MasterNodes and slave nodes

- Clients as the users run the application with the help of Hadoop ecosystem projects. For example, Hive, Mahout and Pig are the ecosystem's projects. They are not required to be present at the Hadoop cluster. 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:
  - ✓ The file section is stored in which part of the cluster
  - ✓ Last access time for the files
  - ✓ 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. Thus, 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.

2.3.1.2 Hadoop 2

Single NameNode failure in Hadoop 1 is an operational limitation. Scaling up was also restricted to scale beyond a few thousands of DataNodes and few number of clusters. 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.

The system takes care of failure issues as follows: 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. That 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 MN1 into

the JN2, which is at newly active MasterNode MN2. Therefore, the application runs uninterrupted and resources are available uninterrupted.

2.3.2 HDFS Commands

Figure 2.1 showed Hadoop common module, which contains the libraries and utilities. They are common to other modules of Hadoop. 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 HDFS require /bin/hdfs dfs <args>, where args stands for the command arguments.

–copyToLocal is the command for copying a file at HDFS to the local.
–cat is command for copying to standard output (stdout).
All Hadoop commands are invoked by the bin/Hadoop script. % Hadoop fsck / -files –blocks

Examples of usages of commands :

-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 hdfs-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

2.4 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.
- MapReduce allows writing applications to process reliably the huge amounts of data, in parallel, on large clusters of servers. The cluster size does not limit as such to process in parallel. The parallel programs of MapReduce are useful for performing large scale data analysis using multiple machines in the cluster.
- Features of MapReduce framework are as follows:
  - ✓ Provides automatic parallelization and distribution of computation based on several processors
  - ✓ Processes data stored on distributed clusters of DataNodes and racks
  - ✓ Allows processing large amount of data in parallel
  - ✓ Provides scalability for usages of large number of servers
  - ✓ Provides MapReduce batch-oriented programming model in Hadoop version 1
  - ✓ 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.

The following subsection describes Hadoop execution model using MapReduce Framework.

2.4.1 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 (Figure 2.4).
- 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:
  - ✓ 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.
  - ✓ 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.
- Figure 2.4 showed the job execution model of MapReduce. Here the JobTracker schedules jobs submitted by clients, keeps track of TaskTrackers and maintains the available Map and Reduce slots. The JobTracker also monitors the execution of jobs and tasks on the cluster. The TaskTracker executes the Map and Reduce tasks, and reports to the JobTracker.

2.4.2 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 (distributed in HDFS). 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.
- They are also termed as two phases— 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.

The MapReduce v2 uses YARN based resource scheduling which simplifies the software development. Here, the jobs can be split across almost any number of servers.

- For example, the ACVM Company can find the number of chocolates KitKat, Milk, Fruit and Nuts, Nougat and Oreo sold every hour at the number of ACVMs installed all over in the multiple cities on separate servers [Refer Example 1.6(i)]. A server maps the keys for KitKat and another for Oreo. It requires time to scan the hourly sales log sequentially. By contrast, MapReduce programmer can split the application task among multiple sub-tasks, say one hundred sub-tasks, where each sub-task

processes the data of the selected set of ACVMs. The results of all the sub-tasks then aggregate to get the final result, hourly sales figures of each chocolate flavor from all ACVMs of the company. Finally, the aggregated hourly results appear from the hourly log of transactions filed at Hadoop DataNodes. The company enterprise server runs analytics and applications consider the results as if from a single server application. The following example shows the usage of HDFS and the map and reduce functions.

2.5 Hadoop YARN

- YARN is a resource management platform. It 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.
- An application task has a number of sub-tasks. 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. YARN stands for Yet Another Resource Negotiator.
- 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.

The figure shows the YARN components—Client, Resource Manager (RM), Node Manager (NM), Application Master (AM) and Containers.

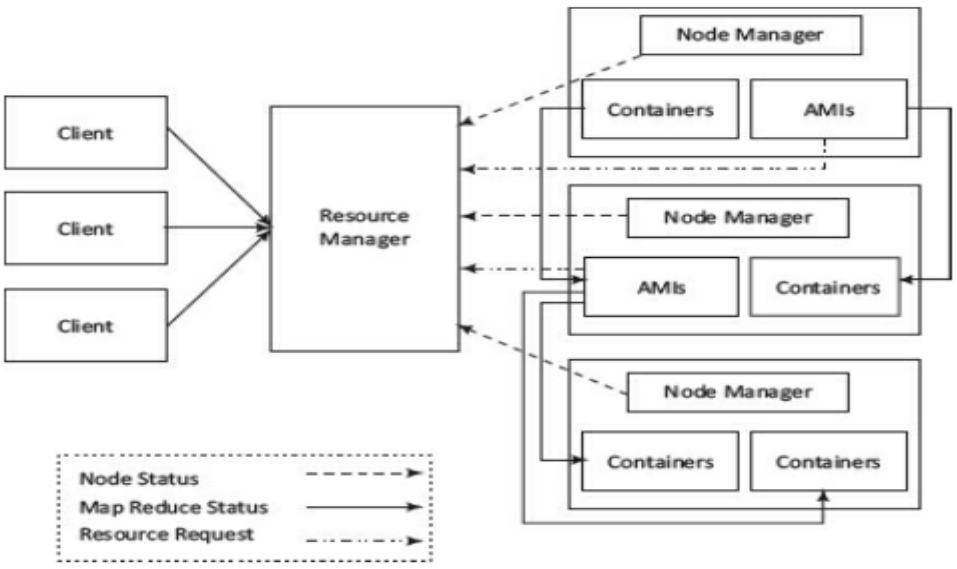


Figure 2.5 YARN-based execution model

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

- A MasterNode has two components: (i) Job History Server and (ii) 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 (Rack Awareness) 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.

2.6 HADOOP ECOSYSTEM TOOLS

Functionalities of the ecosystem tools and components Ecosystem Tool Functionalities

Tools	Functionalities
ZooKeeper	Coordination service Provisions high-performance coordination service for distributed running of applications and tasks
Avro	Data serialization and transfer utility Provisions data serialization during data transfer between application and processing layers
Oozie	Provides a way to package and bundles multiple coordinator and workflow jobs and manage the lifecycle of those jobs
Sqoop (SQL-to-Hadoop)	– A data-transfer software Provisions for data-transfer between data stores such as relational DBs and Hadoop
Flume – Large data transfer utility	Provisions for reliable data transfer and provides for recovery in case of failure. Transfers large amount of data in applications, such as related to social-media messages
Ambari – A web-based tool	Provisions, monitors, manages, and viewing of functioning of the cluster, MapReduce, Hive and Pig APIs
Chukwa – A data collection system	Provisions and manages data collection system for large and distributed systems
HBase – A structured data store using database	Provisions a scalable and structured database for large tables
Cassandra – A database	Provisions scalable and fault-tolerant database for multiple masters
Hive – A data warehouse system	Provisions data aggregation, data-summarization, data warehouse infrastructure, ad hoc (unstructured) querying and SQL-like scripting language for query processing using HiveQL
Pig – A high-level dataflow language	Provisions dataflow (DF) functionality and the execution framework for parallel computations
Mahout –A machine learning software	Provisions scalable machine learning and library functions for data mining and analytics

2.6.1 Hadoop Ecosystem

2.6.1.1 Zookeeper - Designing of a distributed system requires designing and developing the coordination services. Apache Zookeeper is a coordination service that enables synchronization across a cluster in distributed applications (Figure 2.2). The coordination service manages the jobs in the cluster. Since multiple machines are involved, the race condition and deadlock are common problems when running a distributed application.

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

ZooKeeper's main coordination services are:

1 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.

2 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.

3 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.

4 Failure – Distributed systems are susceptible to the problem of node failures. This requires implementing an automatic recovering strategy by selecting some alternate node for processing (Using two MasterNodes with a NameNode each).

2.6.1.2 Oozie

- Apache Oozie is an open-source project of Apache that schedules Hadoop jobs. An efficient process for job handling is required. Oozie design provisions the scalable processing of multiple jobs. Thus, 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:
- Oozie workflow jobs are represented as Directed Acrylic Graphs (DAGs), specifying a sequence of actions to execute.
- Oozie coordinator jobs are recurrent Oozie workflow jobs that are triggered by time and data availability.

Oozie provisions for the following:

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

2.6.1.3 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. Sqoop can also be accessed using Java APIs. 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. Parallel transfer of data results in parallel results and fast data transfer. 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.

2.6.1.4 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:

✓ **Sources** which accept data from a server or an application.
✓ **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.
✓ **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.
✓ **Agents** run the sinks and sources in Flume. The interceptors drop the data or transfer data as it flows into the system.

2.6.2 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.

2.6.2.1 Features

Features of Ambari and associated components are as follows:

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

2.6.2.2 Hadoop Administration

Hadoop large clusters pose a number of configuration and administration challenges. Administrator procedures enable managing and administering Hadoop clusters, resources and associated Hadoop ecosystem components (Figure 2.2). Administration includes installing and monitoring clusters. Ambari also provides a centralized setup for security.

This simplifies the administering complexities and configures security of clusters across the entire platform. Ambari helps automation of the setup and configuration of Hadoop using Web User Interface and REST APIs. IBM

BigInsights provides an administration console. 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.

2.6.3 HBase

Similar to database, 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:

✓ Uses a partial columnar data schema on top of Hadoop and HDFS.
✓ Supports a large table of billions of rows and millions of columns.
✓ 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 (Example 2.3).
✓ Supports data compression algorithms.
✓ Provisions in-memory column-based data transactions.
✓ Accesses rows serially and does not provision for random accesses and write into the rows.

- ✓ Provides random, real-time read/write access to Big Data.
- ✓ Fault tolerant storage due to automatic failure support between DataNodes servers.
- ✓ 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 the database table. Data accesses are performed using that key.
- HBASE applies a partial columnar scheme on top of the Hadoop and HDFS. An HBase column represents an attribute of an object, such as hourly sales of KitKat, Milk, Fruit and Nuts, Nougat and Oreo sold every hour at an ACVM

2.6.4 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 puts a partial SQL interface in front of Hadoop.
- Hive design provisions for batch processing of large sets of data. An application of Hive is for managing weblogs. Hive does not process real-time queries and does not update row-based data tables.
- Hive also enables data serialization/deserialization and increases flexibility in schema design by including a system catalog called Hive Metastore.
- HQL also supports custom MapReduce scripts to be plugged into queries. Hive supports different storage types, such as text files, sequence files (consisting of binary key/value pairs) and RCFiles (Record Columnar Files), ORC (optimized row columnar) and HBase.
- Three major functions of Hive are data summarization, query and analysis.
- Hive basically interacts with structured data stored in HDFS with a query language known as HQL (Hive Query Language)which is similar to SQL. HQL translates SQL-like queries into MapReduce jobs executed on Hadoop automatically.

2.6.5 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. The language used in Pig is known as Pig Latin. Pig Latin language is similar to SQL query language but applies on 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, irrespective of the individual Mapper and Reducer functions to produce the output results.

2.6.6 Mahout

- Mahout is a project of Apache with library of scalable machine learning algorithms. Apache implemented Mahout on top of Hadoop. Apache used the MapReduce paradigm.
- Machine learning is mostly required to enhance the future performance of a system based on the previous outcomes.
- Mahout provides the learning tools to automate the finding of meaningful patterns in the Big Data sets stored in the HDFS.
- Mahout supports four main areas:
- ✓ 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.

Hadoop Distributed File System

Design Features:

- The Hadoop Distributed File System (HDFS) was designed for Big Data processing.
- Assumes a large file write-once/read-many model that enables other optimizations and relaxes many of the concurrency and coherence overhead requirements of a true parallel file system. For instance, 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. Bytes are always appended to the end of a stream, and byte streams are guaranteed to be stored in the order written.
- 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. Thus, this approach is entirely unsuitable for standard POSIX file system use.
- In addition, due to the sequential nature of the data, 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.
- The most interesting aspect of HDFS—and the one that separates it from other file systems—is its data locality. A principal design aspect of Hadoop MapReduce is the emphasis on moving the computation to the data rather than moving the data to the computation. This distinction is reflected in how Hadoop clusters are implemented.
- In other high-performance systems, a parallel file system will exist on hardware separate from the compute hardware. Data is then moved to and from the computer components via high-speed interfaces to the parallel file system array. HDFS, in contrast, is designed to work on the same hardware as the compute portion of the cluster. That is, a single server node in the cluster is often both a computation engine and a storage engine for the application.
- Finally, Hadoop clusters assume node (and even rack) failure will occur at some point. To deal with this situation, HDFS has a redundant design that can tolerate system failure and still provide the data needed by the compute part of the program.
- The following points summarize 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.
    - A specialized file system is used, which is not designed for general use.

HDFS Components

- The design of HDFS is based on two types of nodes: a NameNode and multiple DataNodes. In a basic design, a single NameNode manages all the metadata needed to store and retrieve the actual data from the DataNodes. No data is actually stored on the NameNode, however.
- 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.
- An example of the client/NameNode/DataNode interaction is provided in Figure. 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 possible). 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 client gives a limited amount of time to complete the operation. If it does not complete in the time period, the operation is cancelled.

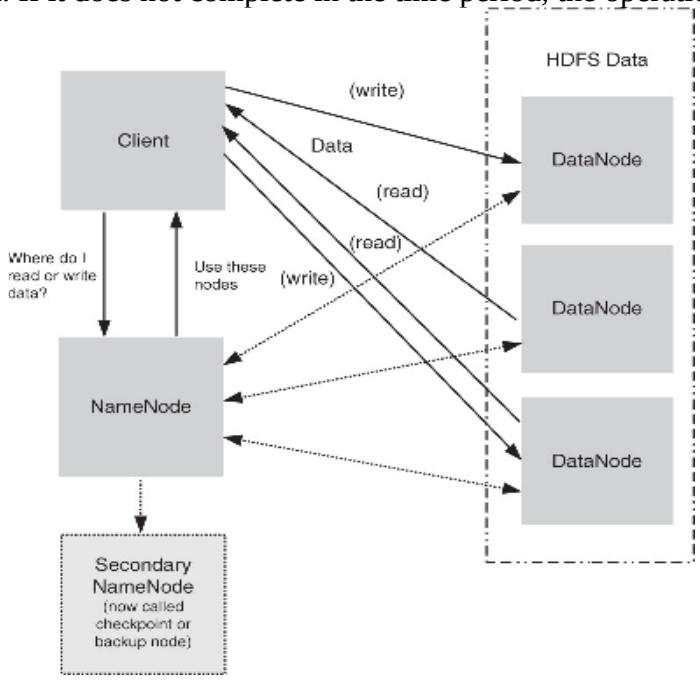


Figure : Various system roles in an HDFS deployment

- 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.
- Thus, once the metadata has been delivered to the client, the NameNode steps back and lets the conversation between the client and the DataNodes proceed.
- the NameNode also monitors the DataNodes by listening for heartbeats sent from DataNodes. The lack of a heartbeat signal indicates a potential node failure. In such a case, the NameNode will route around the failed DataNode and begin re-replicating the now-missing blocks. Because the file system is redundant, DataNodes can be taken offline (decommissioned) for maintenance by informing the NameNode of the DataNodes to exclude from the HDFS pool.
- The mappings between data blocks and the physical DataNodes are not kept in persistent storage on the NameNode. For performance reasons, the NameNode stores all metadata in memory. Upon startup, each DataNode provides a block report (which it keeps in persistent storage) to the NameNode. The block reports are sent every 10 heartbeats. (The interval between reports is a configurable property.) The reports enable the NameNode to keep an up-to-date account of all data blocks in the cluster.
- The purpose of the SecondaryNameNode(CheckPointNode) is is to perform periodic checkpoints that evaluate the status of the NameNode. Recall that the NameNode keeps all system metadata memory for fast access. It also has two disk files that track changes to the metadata:
  - ✓ An image of the file system state when the NameNode was started, named as fsimage_*
  - ✓ 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. Thus, when the NameNode restarts, the fsimage file is reasonably up-to-date and requires only the edit logs to be applied since the last checkpoint.

Thus, 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

- When HDFS writes a file, it is replicated across the cluster. 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. In a Hadoop cluster of eight or fewer DataNodes but more than one DataNode, a replication factor of 2 is adequate. For a single machine, the replication factor is set to 1.
- If several machines must be involved in the serving of a file, then a file could be rendered unavailable by the loss of any one of those machines. HDFS combats this problem by replicating each block across a number of machines (three is the default).
- In addition, the HDFS default block size is often 64MB. In a typical operating system, the block size is 4KB or 8KB. The HDFS default block size is not the minimum block size, however If a file of size 80MB is written to HDFS, a 64MB block and a 16MB block will be created.
- Figure below provides an example of how a file is broken into blocks and replicated across the cluster. In this case, a replication factor of 3 ensures that any one DataNode can fail and the replicated blocks will be available on other nodes—and then subsequently re-replicated on other DataNodes.

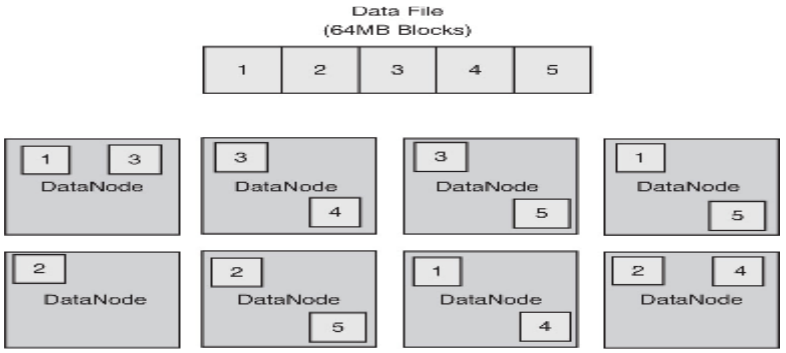


Figure: HDFS block replication example

HDFS Safe Mode

When the NameNode starts, it enters a read-only safe mode where blocks cannot be replicated or deleted. Safe Mode enables the NameNode to perform two important processes:

1. The previous file system state is reconstructed by loading the fsimage file into memory and replaying the edit log.
2. The mapping between blocks and data nodes is created by waiting for enough of the DataNodes to register so that at least one copy of the data is available. Not all DataNodes are required to register before HDFS exits from Safe Mode. The registration process may continue for some time.

HDFS may also enter Safe Mode for maintenance using the hdfs dfsadmin-safemode command or when there is a file system issue that must be addressed by the administrator.

Rack Awareness

Rack awareness deals with data locality. One of the main design goals of Hadoop MapReduce is to move the computation to the data. Assuming that most data center networks do not offer full bisection bandwidth, a typical Hadoop cluster will exhibit three levels of data locality:

1. Data resides on the local machine (best).
2. Data resides in the same rack (better).
3. Data resides in a different rack (good).

When the YARN scheduler is assigning MapReduce containers to work as mappers, it will try to place the container first on the local machine, then on the same rack, and finally on another rack.

In addition, the NameNode tries to place replicated data blocks on multiple racks for improved fault tolerance. In such a case, an entire rack failure will not cause data loss or stop HDFS from working.

HDFS can be made rack-aware by using a user-derived script that enables the master node to map the network topology of the cluster. A default Hadoop installation assumes all the nodes belong to the same (large) rack. In that case, there is no option 3.

NameNode High Availability

With early Hadoop installations, the NameNode was a single point of failure that could bring down the entire Hadoop cluster. NameNode hardware often employed redundant power supplies and storage to guard against such problems, but it was still susceptible to other failures. The solution was to implement NameNode High Availability (HA) as a means to provide true failover service.

As shown in Figure, an HA Hadoop cluster has two (or more) separate NameNode machines. Each machine is configured with exactly the same software. One of the NameNode machines is in the Active state, and the other is in the Standby state. The Active NameNode is responsible for all client HDFS operations in the cluster. The Standby NameNode maintains enough state to provide a fast failover (if required).

To guarantee the file system state is preserved, both the Active and Standby NameNodes receive block reports from the DataNodes. The Active node also sends all file system edits to a quorum of Journal nodes. At least three physically separate JournalNode daemons are required, because edit log modifications must be written to a majority of the JournalNodes. This design will enable the system to tolerate the failure of a single JournalNode machine. The Standby node continuously reads the edits from the JournalNodes to ensure its namespace is synchronized with that of the Active node. In the event of an Active NameNode failure, the Standby node reads all remaining edits from the JournalNodes before promoting itself to the Active state.

To prevent confusion between NameNodes, the JournalNodes allow only one NameNode to be a writer at a time. During failover, the NameNode that is chosen to become active takes over the role of writing to the JournalNodes.

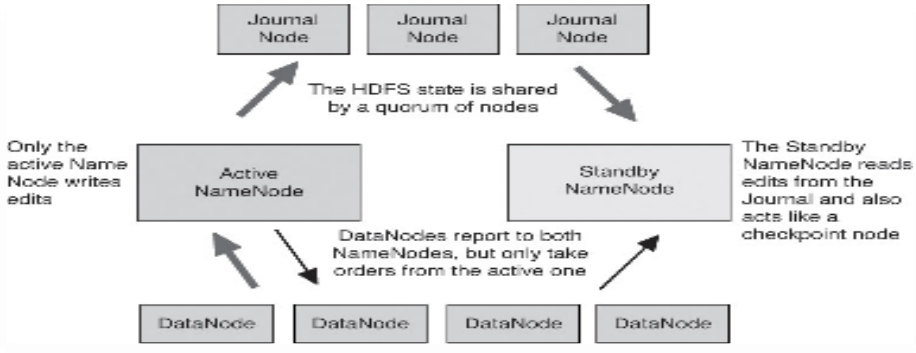


Figure : HDFS High Availability design

A SecondaryNameNode is not required in the HA configuration because the Standby node also performs the tasks of the Secondary NameNode.

Apache ZooKeeper is used to monitor the NameNode health. Zookeeper is a highly available service for maintaining small amounts of coordination data, notifying clients of changes in that data, and monitoring clients for failures. HDFS failover relies on ZooKeeper for failure detection and for Standby to Active NameNode election.

HDFS NameNode Federation

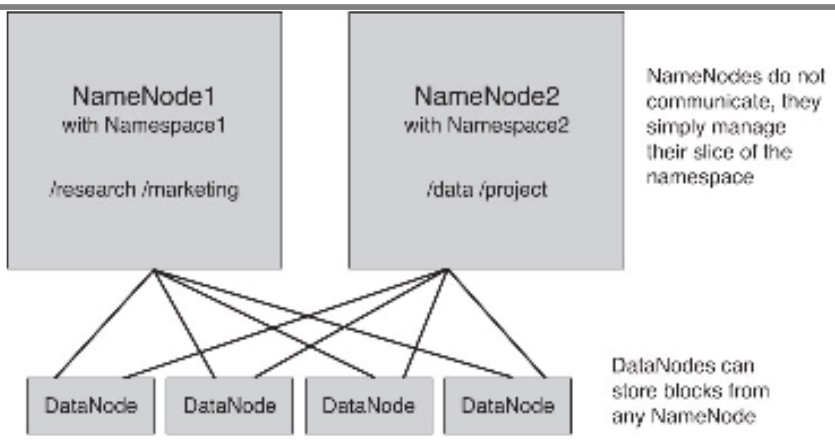
Older versions of HDFS provided a single namespace for the entire cluster managed by a single NameNode. Thus, the resources of a single NameNode determined the size of the namespace. Federation addresses this limitation by adding support for multiple NameNodes/namespaces to the HDFS file system. The key benefits are as follows:

• Namespace scalability. HDFS cluster storage scales horizontally without placing a burden on the NameNode.

• Better performance. Adding more NameNodes to the cluster scales the file system read/write operations throughput by separating the total namespace.

• System isolation. Multiple NameNodes enable different categories of applications to be distinguished, and users can be isolated to different namespaces.

Figure illustrates how HDFS NameNode Federation is accomplished. NameNode1 manages the /research and /marketing namespaces, and NameNode2 manages the /data and /project namespaces. The NameNodes do not communicate with each other and the DataNodes “just store data block” as directed by either NameNode.



HDFS Checkpoints and Backups

- File systems modifications are written to an edits log file, and at startup the NameNode merges the edits into a new fsimage. The SecondaryNameNode or CheckpointNode periodically fetches edits from the NameNode, merges them, and returns an updated fsimage to the NameNode.
- An HDFS BackupNode is similar, but also maintains an up-to-date copy of the file system namespace both in memory and on disk. Unlike a CheckpointNode, the BackupNode does not need to download the fsimage and edits files from the active NameNode because it already has an up-to-date namespace state in memory. A NameNode supports one BackupNode at a time. No CheckpointNodes may be registered if a Backup node is in use.

HDFS Snapshots

HDFS snapshots are similar to backups, but are created by administrators using the hdfs dfs snapshot command. HDFS snapshots are read-only point-in-time copies of the file system. They offer the following features:

• Snapshots can be taken of a sub-tree of the file system or the entire file system.

• Snapshots can be used for data backup, protection against user errors, and disaster recovery.

• Snapshot creation is instantaneous.

• Blocks on the DataNodes are not copied, because the snapshot files record the block list and the file size. There is no data copying, although it appears to the user that there are duplicate files.

• Snapshots do not adversely affect regular HDFS operations

HDFS NFS Gateway

The HDFS NFS Gateway supports NFSv3 and enables HDFS to be mounted as part of the client's local file system.

• Users can browse the HDFS file system through their local file systems that provide an NFSv3 client compatible operating system. This feature offers users the following capabilities: Users can easily download/upload files from/to the HDFS file system to/from their local file system.

• Users can stream data directly to HDFS through the mount point. Appending to a file is supported, but random write capability is not supported.

HDFS User Commands

The following is a brief command reference that will facilitate navigation within HDFS.

Brief HDFS Command Reference

The preferred way to interact with HDFS in Hadoop version 2 is through the **hdfs** command. Previously, in version , the hadoop **dfs** command was used to manage files in HDFS. (The hadoop dfs command will still work in version 2, but its use will cause a message to be displayed indicating that the use of hadoop dfs is deprecated.)

The following listing presents the full range of options that are available for the hdfs command.

```
Usage: hdfs [--config confdir] COMMAND
       where COMMAND is one of:
  dfs                  run a file system command on the file systems
                       supported in Hadoop.
  namenode -format     format the DFS file system
  secondarynamenode    run the DFS secondary namenode
  namenode             run the DFS namenode
  journalnode          run the DFS journalnode
  zkfc                 run the ZK Failover Controller daemon
  datanode             run a DFS datanode
  dfsadmin             run a DFS admin client
  haadmin              run a DFS HA admin client
  fsck                 run a DFS file system checking utility
  balancer             run a cluster balancing utility
  jmxget               get JMX exported values from NameNode or DataNode.
  mover                run a utility to move block replicas across
                       storage types
  oiv                  apply the offline fsimage viewer to an fsimage
  oiv_legacy           apply the offline fsimage viewer to an legacy
                       fsimage
  oev                  apply the offline edits viewer to an edits file
  fetchdt              fetch a delegation token from the NameNode
  getconf              get config values from configuration
  groups               get the groups which users belong to
  snapshotDiff         diff two snapshots of a directory or diff the
                       current directory contents with a snapshot
  lsSnapshottableDir   list all snapshottable dirs owned by the current
                       user Use -help to see options
  portmap              run a portmap service
  nfs3                 run an NFS version 3 gateway
  cacheadmin           configure the HDFS cache
  crypto               configure HDFS encryption zones
  storagepolicies      get all the existing block storage policies
  version              print the version

Most commands print help when invoked w/o parameters.
```

Some General Commands

```
$ hdfs version
Hadoop 2.6.0.2.2.4.2-2
Subversion git@github.com:hortonworks/hadoop.git -r
22a563ebe448969d07902aed869ac13c652b2872
Compiled by jenkins on 2015-03-31T19:49Z
Compiled with protoc 2.5.0
From source with checksum b3481c2cdbe2d181f2621331926e267
This command was run using /usr/hdp/2.2.4.2-2/hadoop/hadoop-
common-2.6.0.2.2.4.2-2.jar
```

HDFS provides a series of commands similar to those found in a standard POSIX file system. A list of those commands can be obtained by issuing the following command. Several of these commands will be highlighted here under the user account `hdfs`.

```
$ hdfs dfs
Usage: hadoop fs [generic options]
        [-appendToFile <localsrc> ... <dst>]
        [-cat [-ignoreCrc] <src> ...]
        [-checksum <src> ...]
        [-chgrp [-R] GROUP PATH...]
        [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
        [-chown [-R] [OWNER][:[GROUP]] PATH...]
        [-copyFromLocal [-f] [-p] [-l] <localsrc> ... <dst>]
        [-copyToLocal [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
        [-count [-q] [-h] <path> ...]
        [-cp [-f] [-p | -p[topax]] <src> ... <dst>]
        [-createSnapshot <snapshotDir> [<snapshotName>]]
        [-deleteSnapshot <snapshotDir> <snapshotName>]
        [-df [-h] [<path> ...]]
        [-du [-s] [-h] <path> ...]
        [-expunge]
        [-get [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
        [-getfacl [-R] <path>]
        [-getfattr [-R] {-n name | -d} [-e en] <path>]
        [-getmerge [-nl] <src> <localdst>]
        [-help [cmd ...]]
        [-ls [-d] [-h] [-R] [<path> ...]]
        [-mkdir [-p] <path> ...]
        [-moveFromLocal <localsrc> ... <dst>]
        [-moveToLocal <src> <localdst>]
        [-mv <src> ... <dst>]
        [-put [-f] [-p] [-l] <localsrc> ... <dst>]
        [-renameSnapshot <snapshotDir> <oldName> <newName>]
        [-rm [-f] [-r|-R] [-skipTrash] <src> ...]
        [-rmdir [--ignore-fail-on-non-empty] <dir> ...]
        [-setfacl [-R] [{-b|-k} {-m|-x <acl_spec>} <path>]|[--set
          <acl_spec> <path>]]
        [-setfattr {-n name [-v value] | -x name} <path>]
        [-setrep [-R] [-w] <rep> <path> ...]
        [-stat [format] <path> ...]
        [-tail [-f] <file>]
        [-test -[defsz] <path>]
        [-text [-ignoreCrc] <src> ...]
        [-touchz <path> ...]
        [-truncate [-w] <length> <path> ...]
        [-usage [cmd ...]]
```

Some general Commands used:

1. To list files : ls

```
$ hdfs dfs -ls /

Found 10 items
drwxrwxrwx   - yarn   hadoop          0 2015-04-29 16:52 /app-logs
drwxr-xr-x   - hdfs   hdfs            0 2015-04-21 14:28 /apps
drwxr-xr-x   - hdfs   hdfs            0 2015-05-14 10:53 /benchmarks
drwxr-xr-x   - hdfs   hdfs            0 2015-04-21 15:18 /hdp
drwxr-xr-x   - mapred hdfs            0 2015-04-21 14:26 /mapred
drwxr-xr-x   - hdfs   hdfs            0 2015-04-21 14:26 /mr-history
drwxr-xr-x   - hdfs   hdfs            0 2015-04-21 14:27 /system
drwxrwxrwx   - hdfs   hdfs            0 2015-05-07 13:29 /tmp
drwxr-xr-x   - hdfs   hdfs            0 2015-04-27 16:00 /user
drwx-wx-wx   - hdfs   hdfs            0 2015-05-27 09:01 /var
```

To list files in your home directory, enter the following command:

```
$ hdfs dfs -ls

Found 13 items
drwx------ - hdfs hdfs 0 2015-05-27 20:00 .Trash
drwx------ - hdfs hdfs 0 2015-05-26 15:43 .staging
drwxr-xr-x - hdfs hdfs 0 2015-05-28 13:03 DistributedShell
drwxr-xr-x - hdfs hdfs 0 2015-05-14 09:19 TeraGen-50GB
drwxr-xr-x - hdfs hdfs 0 2015-05-14 10:11 TeraSort-50GB
drwxr-xr-x - hdfs hdfs 0 2015-05-24 20:06 bin
drwxr-xr-x - hdfs hdfs 0 2015-04-29 16:52 examples
drwxr-xr-x - hdfs hdfs 0 2015-04-27 16:00 flume-channel
drwxr-xr-x - hdfs hdfs 0 2015-04-29 14:33 oozie-4.1.0
drwxr-xr-x - hdfs hdfs 0 2015-04-30 10:35 oozie-examples
drwxr-xr-x - hdfs hdfs 0 2015-04-29 20:35 oozie-oozi
drwxr-xr-x - hdfs hdfs 0 2015-05-24 18:11 war-and-peace-input
drwxr-xr-x - hdfs hdfs 0 2015-05-25 15:22 war-and-peace-output
```

The same result can be obtained by issuing the following command:

```
$ hdfs dfs -ls /user/hdfs
```

2. **Make a Directory in HDFS:** To make a directory in HDFS, use the following command.
   $ hdfs dfs –mkdir stuff
3. **Copy Files to HDFS:** To copy file from your current local directory into HDFS, use the following command. In this case, the file test is placed in the directory stuff.
   $ hdfs dfs –put test stuff
   The file transfer can be confirmed by using the –ls command.

   $ hdfs dfs –ls stuff
   Found 1 items
   -rw-r--r-- 2 hdfs hdfs 12857 2015-05-29 13:12 stuff/test
4. **Copy Files from HDFS:** Files can be copied back to your local file system using the following command. . In this case, the file that we copied into HDFS, test , will be copied back into the current local directory test-local.
   $ hdfs dfs –get stuff/test test-local
5. **Copy Files within HDFS:** The following command will copy a file in HDFS
   $ hdfs dfs –cp stuff/test test.hdfs
6. **Delete a File Within HDFS:** The following command will delete the HDFS file test.dhfs that was created previously.
   $ hdfs dfs –rm test.hdfs
7. **Delete a Directory in HDFS:** The following command will delete the HDFS directory stuff and all its contents.
   $ hdfs dfs –rm –r –skipTrash stuff

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 (the actual language) defines a set of transformations on a data set such as aggregate, join, and sort.
- Pig is often 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. (optimizes multistep Hadoop jobs such as those found in many Pig queries.)
- There are also interactive and batch modes available; they enable Pig applications to be developed locally in interactive modes, using small amounts of data, and then run at scale on the cluster in a production mode. The modes are summarized in Table 7.1

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

Table 7.1 **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 (optimized MapReduce)

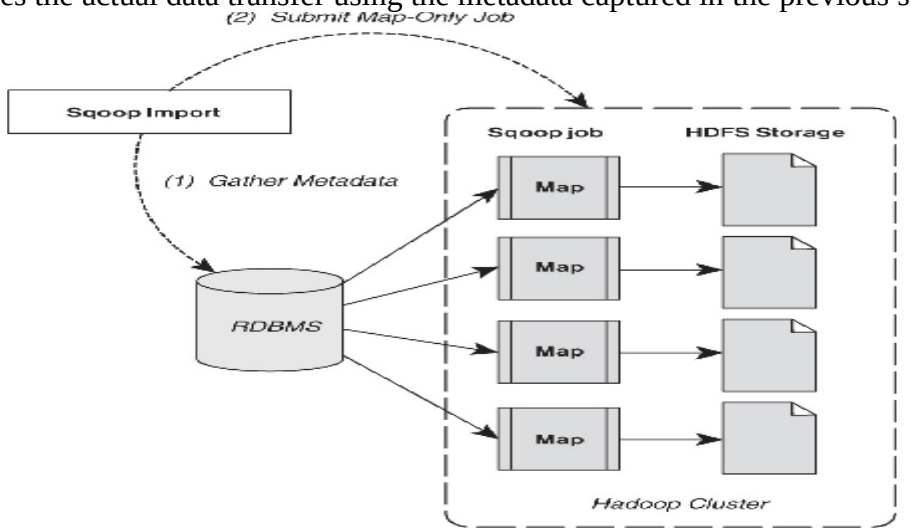
Hive provides users who are already familiar with SQL the capability to query the data on Hadoop clusters. At the same time, Hive makes it possible for programmers who are familiar with the MapReduce framework to add their custom mappers and reducers to Hive queries.

Using Apache Sqoop to Acquire Relational Data

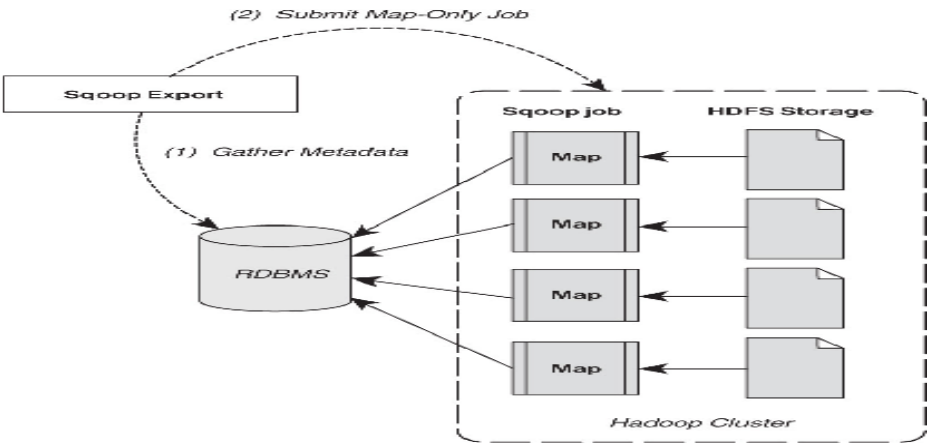
- Sqoop is a tool designed to transfer data between Hadoop and relational databases. Sqoop is used to import data from a relational database management system (RDBMS) into the Hadoop Distributed File System (HDFS), transform the data in Hadoop, and then export the data back into an RDBMS.
- Sqoop can be used with any Java Database Connectivity (JDBC)–compliant database and has been tested on Microsoft SQL Server, PostgresSQL, MySQL, and Oracle.

Apache Sqoop Import and Export Methods

The figure below describes the Sqoop data import (to HDFS) process. The data import is done in two steps. In the first step, shown in the figure, Sqoop examines the database to gather the necessary metadata for the data to be imported. 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.



- 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.
- Data export from the cluster works in a similar fashion. The export is done in two steps, as shown in Figure below. As in the import process, 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. Again, this process assumes the map tasks have access to the database.



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. Often data transport involves a number of Flume agents that may traverse a series of machines and locations. Flume is often used for log files, social media-generated data, email messages, and just about any continuous data source. As shown in Figure 7.3, a Flume agent is composed of three components.

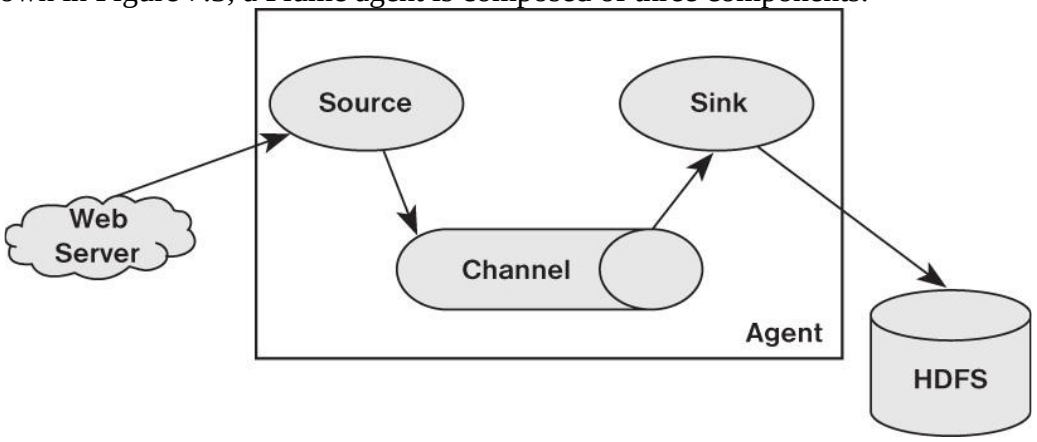


Figure 7.3 Flume agent with source, channel, and sink

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.

A Flume agent must have all three of these components defined.

- 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.
- As shown in Figure 7.4, Sqoop agents may be placed in a pipeline, possibly to traverse several machines or domains. This configuration is normally used when data are collected on one machine (e.g., a web server) and sent to another machine that has access to HDFS.
- In a Flume pipeline, the data transfer format normally used by Flume, which is called Apache Avro, provides several useful features. 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 (JavaScript Object Notation). Avro also uses remote procedure calls (RPCs) to send data. That is, an Avro sink will contact an Avro source to send data.

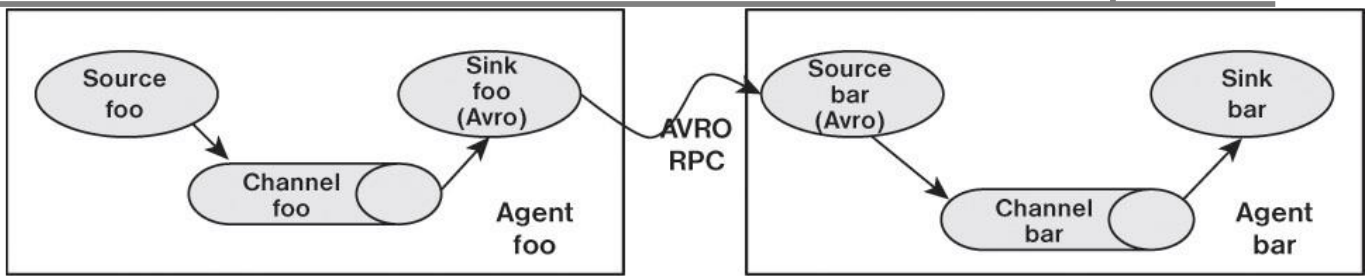


Figure 7.4:

- Another useful Flume configuration is shown in Figure 7.5. In this configuration, Flume is used to consolidate several data sources before committing them to HDFS. There are many possible ways to construct Flume transport networks.

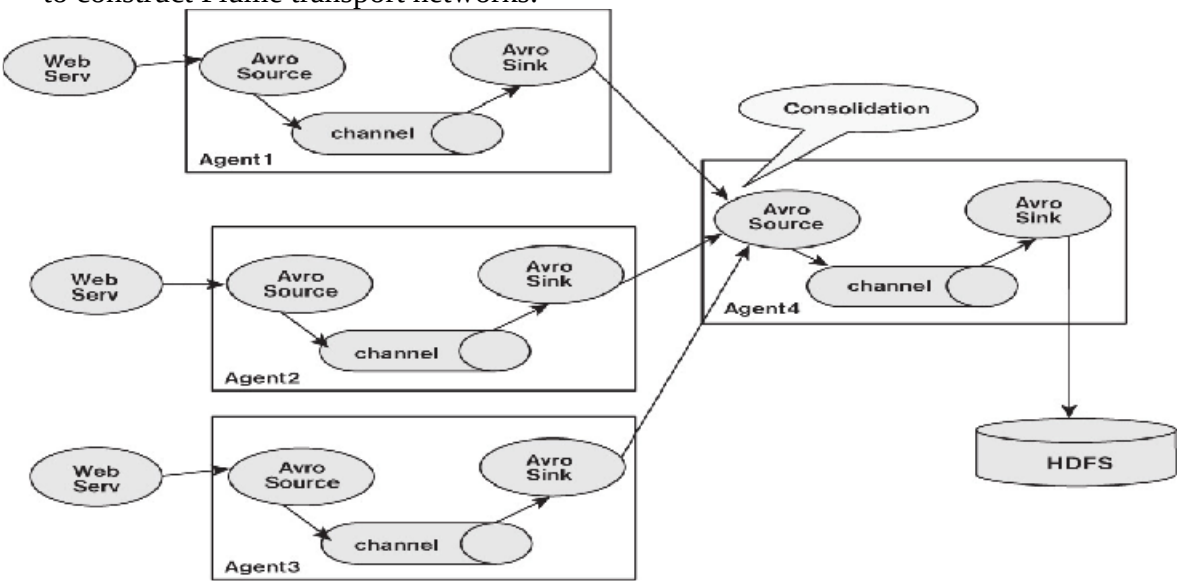


Figure 7.5 : A Flume consolidation Network

Manage Hadoop Workflows with Apache Oozie

- Oozie is a workflow director system designed to run and manage multiple related Apache Hadoop jobs. For instance, complete data input and analysis may require several discrete Hadoop jobs to be run as a workflow in which the output of one job serves as the input for a successive job.
- Oozie is designed to construct and manage these workflows. Oozie is not a substitute for the YARN scheduler. That is, YARN manages resources for individual Hadoop jobs, and 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. (DAGs are basically graphs that cannot have directed loops.) Three types of Oozie jobs are permitted:

Workflow—a specified sequence of Hadoop jobs with outcome-based decision points and control dependency.

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.

Figure 7.6 depicts a simple Oozie workflow. In this case, Oozie runs a basic MapReduce operation. If the application was successful, the job ends; if an error occurred, the job is killed.

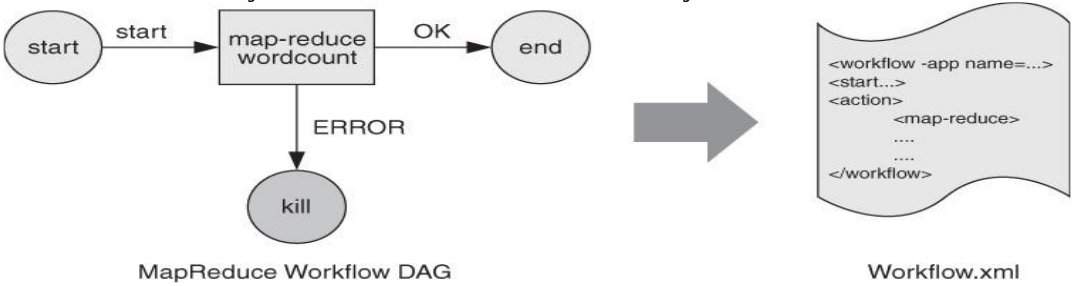


Figure 7.6: A simple Oozie workflow

Oozie workflow definitions are written in hPDL (an XML Process Definition Language). 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 (e.g., file size or file existence).

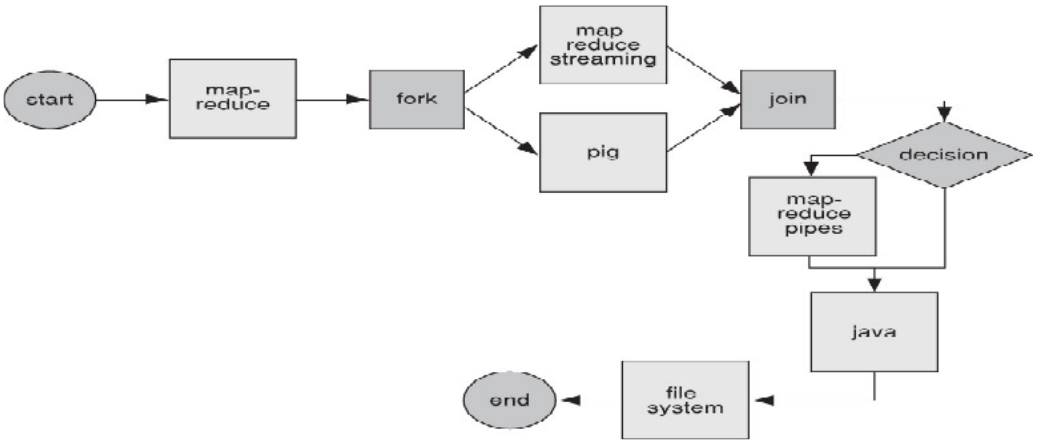


Figure 7.7 : A more complex Oozie DAG workflow

Using Apache HBase

Apache HBase is an open source, distributed, versioned, nonrelational database modeled after Google's Bigtable. Like Bigtable, HBase leverages the distributed data storage provided by the underlying distributed file systems spread across commodity servers.

Some of the more important features include the following capabilities:

- 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

HBase Data Model Overview

- A table in HBase, having rows and columns. Columns in HBase are grouped into column families, all with the same prefix. For example, consider a table of daily stock prices. There may be a column family called "price" that has four members—price:open, price:close, price:low, and price:high.
- A column does not need to be a family. For instance, the stock table may have a column named "volume" indicating how many shares were traded. All column family members are stored together in the physical file system.
- Specific HBase cell values are identified by a row key, column (column family and column), and version (timestamp).
- It is possible to have many versions of data within an HBase cell. A version is specified as a timestamp and is created each time data are written to a cell.
- Rows are lexicographically sorted with the lowest order appearing first in a table. The empty byte array denotes both the start and the end of a table's namespace.
- All table accesses are via the table row key, which is considered its primary key.