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. The computations are at an enterprise server or data warehouse integrated with the applications (Figure 1.6). An example is computations using Oracle application integration architecture (Section 1.6.1.7). The computing nodes need to connect to a central system through high-speed networks.

Assume that a centralized server does the function of collection, storing and analyzing. 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 (Example 1.6 (i)). 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.

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.

Example :2.1

Consider a jigsaw Puzzle Ravensburger Beneath the Sea (5000 pieces). Children above 14 years of age will assemble the pieces in order to solve the puzzle. 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 $[T/4 + T_I (4) + T_C (4)]$, where $T_I (4)$ is the time taken in seeking from others the pieces not available to a child during intermediate phases, and $T_C (4)$ 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 $[T/100 + T_I (100) + T_C (100)]$,
where $T_I (100)$ and $T_C (100)$ 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 $[T/200 + T_I (200) + T_C (200)]$, where $T_I (200)$ and $T_C (200)$ 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 $T_I (200) + T_C (200) < T/200$.

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.

(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 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.

Following are important key terms and their meaning.

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 accessibility 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. Partition means parts, which are active but may not cooperate as in distributed databases (DBs).

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 also includes managing the (i) availability for specified or scheduled periods, (ii) prevention of resource unavailability after a task finishes and (iii) 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 (BI), data visualization, knowledge discovery and machine learning (ML) capabilities which require additional ways to solve problems of greater complexities and greater processing, storage and interprocess 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. That is similar to a biological ecosystem, a complex system of living organisms, their physical environment and all their interrelationships in a particular unit of space.

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 (set of data records, key-value pairs, hash key-value pairs or applications data) 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 the requirements, i.e., bigger data store on greater number of servers when required and on smaller

data when smaller data used on limited number of servers. Big Data Analytics require deploying the clusters using the servers or cloud for computing as per the requirements.

Utility Cloud-based Services mean infrastructure, software and computing platform services similar to utility services, such as electricity, gas, water etc. Infrastructure refers to units for data-store, processing and network. The IaaS, SaaS and PaaS are the services at the cloud (Section 1.3.3).

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 enables Big Data storage and cluster computing. 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 (1 PB = 10*° B, nearly 2°° B) raw-storage. Each (commodity) node has 8 cores and 12 TB (1 TB = 10", nearly 2*°B = 1024 GB) of storage.

2.2.1 Hadoop Core Components

Figure 2.1 shows the core components of the Apache Software Foundation's Hadoop framework.

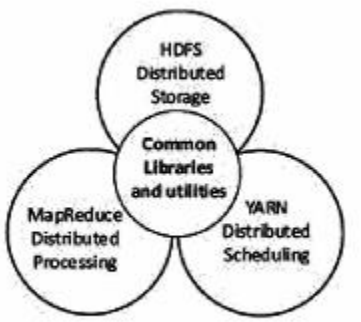


Figure 2.1 Core components of Hadoop

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 (Remote Procedure Call) 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 vl - Software programming model in Hadoop 1 using Mapper and Reducer. The v1 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.

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

Hadoop features are as follows:

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.

Hadoop 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 in Java.

HDFS is basically designed more for batch processing. Streaming uses standard input and output to communicate with the Mapper and Reduce codes. Stream analytics and real-time processing poses difficulties when streams have high throughput of data. The data access is required faster than the latency at DataNode at HDFS.

YARN provides a platform for many different modes of data processing, from traditional batch processing to processing of the applications such as interactive queries, text analytics and streaming analysis.

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.

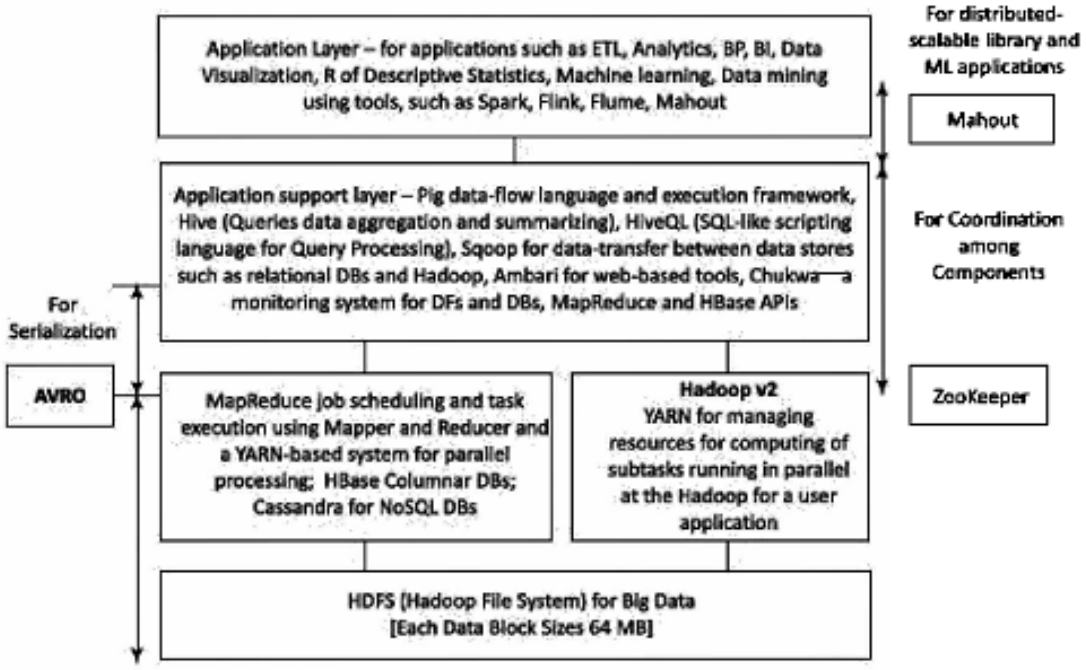


Figure 2.2 Hadoop main components and ecosystem components

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 (Figure 2.2).

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.

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. Simple and flexible architecture of Flink is suitable for streaming data.

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.

Self-Assessment Exercise linked to LO 2.1

1. How are core Hadoop components used for Big Data analytics?

2. Explain the meaning of distributed computing model with data locality?

3. Why are the Hadoop system and ecosystem components shown in the four layers? Explain by an example.

4. Differentiate between MapReduce v1 and MapReduce v2.

2.3 | HADOOP DISTRIBUTED FILE SYSTEM

Big Data analytics applications are software applications that leverage large-scale data. The applications analyze Big Data using massive parallel processing frameworks. 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° B) to PBs (1 PB = 10*° B, nearly the 2°° B). 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 (HDFS division of files concept is similar to Linux or virtual memory page in Intel x86 and Pentium processors where the block size is fixed and is of 4 KB).

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 = 2^{20}B). 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 = 2^{30}B) 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 = 2^{40}B) 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.

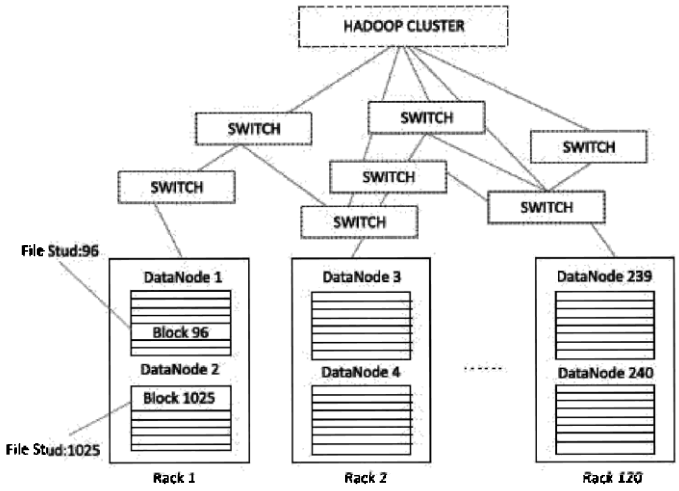
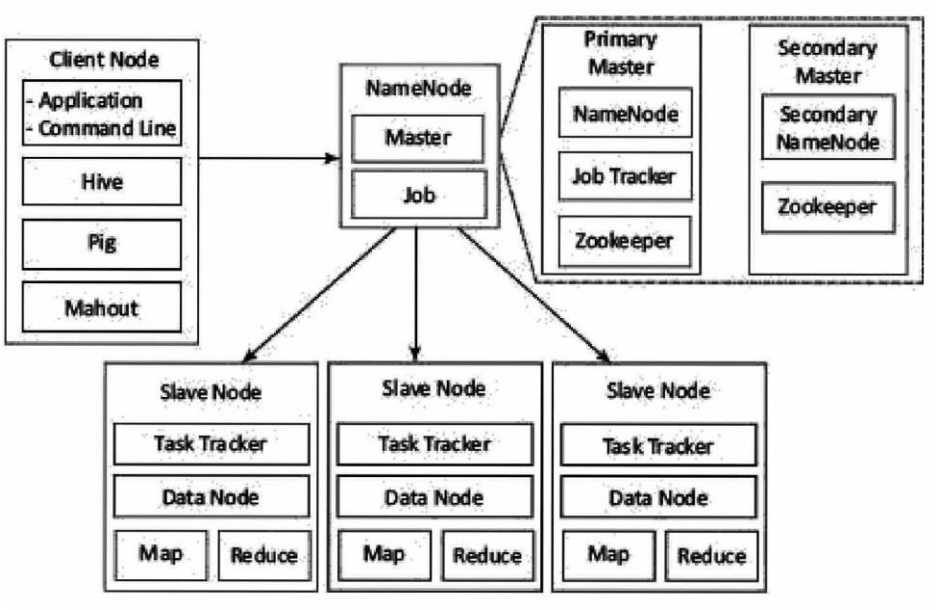


Figure 2.3 A Hadoop cluster example, and the replication of data blocks in racks for two students of IDs 96 and 1025

2.3.1.1 Hadoop Physical Organization

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



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 Master Node 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:

- Anassociated 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.

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.

Table 2.1 Examples of usages of commands

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 $Hadoop hdfs-put stuData_id96 /user/ stu_filesdir` copies file for student of id96 into stu_filesdir directory
-ls	Assume all files to be listed. Then `$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 directory

Self-Assessment Exercise linked to LO 2.2

1. (i) What does the create, append, delete, rename and attribute modification methods mean in the HDFS? (ii) Why is the content of an individual file not modified or replaced but appended at the end of the file? (iii) Why is the write once but read many times concept used in HDFS?

2. What are the functions of NameNode, DataNode, slave node and MasterNode?

3. What are the benefits of multiple MasterNodes?

4. What are the usages of meta data?

5. Make a data-store model using HDFS for SGPs, SGPAs and CGPAs of each student. Assume 50 UG and 10 PG courses offered at the university. Total intake capacity is 5000 each year. Each student information can extend up to 64 MB. How will the files of 5000 students be stored using HDFS? What shall be the minimum memory requirements in 20 years? (SGP means subject grade point awarded to a student, SGPA semester grade point average, and CGPA cumulative grade-point average.)

2.4 | MAPREDUCE FRAMEWORK AND PROGRAMMING MODEL

Figure 2.4 showed MapReduce functions as integral part of the Hadoop physical organization. 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:

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.

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 [Collins English language dictionary gives one of Daemon meaning as ‘a person who concentrates very hard or is very skilled at an activity and puts in lot of energy into it’].

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.

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. The MapReduce program performs two jobs on this input data, the Map job and the Reduce job. 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. 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.

EXAMPLE 2.3

Consider Example 1.6(i) of ACVMs selling KitKat, Milk, Fruit and Nuts, Nougat and Oreo chocolates. Assume 24 files are created every hour for each day. The files are at file_1, file_2, ..., file_24. Each file stores as key-value pairs as hourly sales log at the large number of machines.

(i) How will the large number of machines, say 5000 ACVMs hourly data for each flavor sales log store using HDFS? What will be the strategy to restrict the data size in HDFS?

(ii) How will the sample of data collected in a file for 0-1,1-2, ... 12-13,13-14, 15-16, up to 23-24 specific hour-sales log for sales at a large number of machines, say 5000?

(iii) What will be the output streams of map tasks for feeding the input streams to the Reducer?

(iv) What will be the Reducer outputs?

SOLUTION

5000 machines send sales data every hour for KitKat, Milk, Fruit and Nuts, Nougat and Oreo chocolates, i.e., a total of 5 flavors. Assume each sales data size = 64 B, then data bytes 64 × 5 × 5000 B = 1600000 B
will accumulate (append) each hour in a file.

Sales data are date-time stamped key-value pairs. Each of 24 hour hourly log files will use initially 24 data blocks at a DataNode and replicated at three DataNodes. A data file in one year will accumulate 1600000 × 24 × 365 B = 14016000000 B = nearly 16 GB. Each data block can store 64 MB. Therefore, 16 GB/64 MB = 250 data blocks in each file each year.

However, hourly and daily sales analytics is required only for managing supply chain for chocolate fill service and finding insight into sales during holidays and festival days compared to other days. Therefore, a strategy can be designed to replace the hourly sales data each month and create new files for monthly sales data.

A file sample-data of key-value pairs for hour-sales log in file_16 for sales during 15:00-16:00 will be as follows:

```
ACVM_id10KitKat, 23
ACVM_id2206Milk, 31
ACVM_id2Oreo, 36
```

```
ACVM_id10FruitNuts, 18
ACVM_id16Nougat, 8
.
.
.
ACVM_id1224KitKat, 48
ACVM_id4837Nougat, 28
.
```

Map tasks will map the input streams of key values at files, file_1, file_2, ... file_23, file_24 every hour. The resulting 5000 key value pairs maps each hour with keys for ACVM_idNKitKats (N = 1 to 5000). The output stream from Mapper will be as follows:

```
(ACVM_id10KitKat, 0), (ACVM_id1224KitKat, 3),.., ..,
...,... ,.., .., ..., ....,...,... ,.., .., ...
```

Hourly 5 output streams of mapped tasks for all chocolates of all 5000 machines will be input to the reduce task.

The Reducer processes each hour using 5 input streams, sums all machines sales and generates one output (ACVMs_KitKat, 109624), (ACVMs_Milk, 128324), (ACVMs_FruitNuts, 9835), (ACVMs_Nougat, 2074903), and (ACVMs_Oreo, 305163). The reduced output serializes and is input to the analytics applications each hour.

Chapter 4 describes MapReduce programming in detail.

Self-Assessment Exercise linked to LO 2.3

1. Why is mapping required when processing the stored data at HDFS?
2. How do Jobracker and TaskTracker function?
3. How does MapReducer along with the YARN resources manager enable faster processing of an application?
4. Consider HDFS DataNodes in a cluster. Draw a diagram depicting 10

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 subtasks and submits the resources for them at the Hadoop system.

2.5.1 Hadoop 2 Execution Model

Figure 2.5 shows the YARN-based execution model. The figure shows the YARN components—Client, Resource Manager (RM), Node Manager (NM), Application Master (AM) and Containers.

Figure 2.5 also illustrates YARN components namely, Client, Resource Manager (RM), Node Manager (RM), 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 (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 (AppIM), that estimates the resources requirement for running an application program or subtask. 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. AppIM 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.

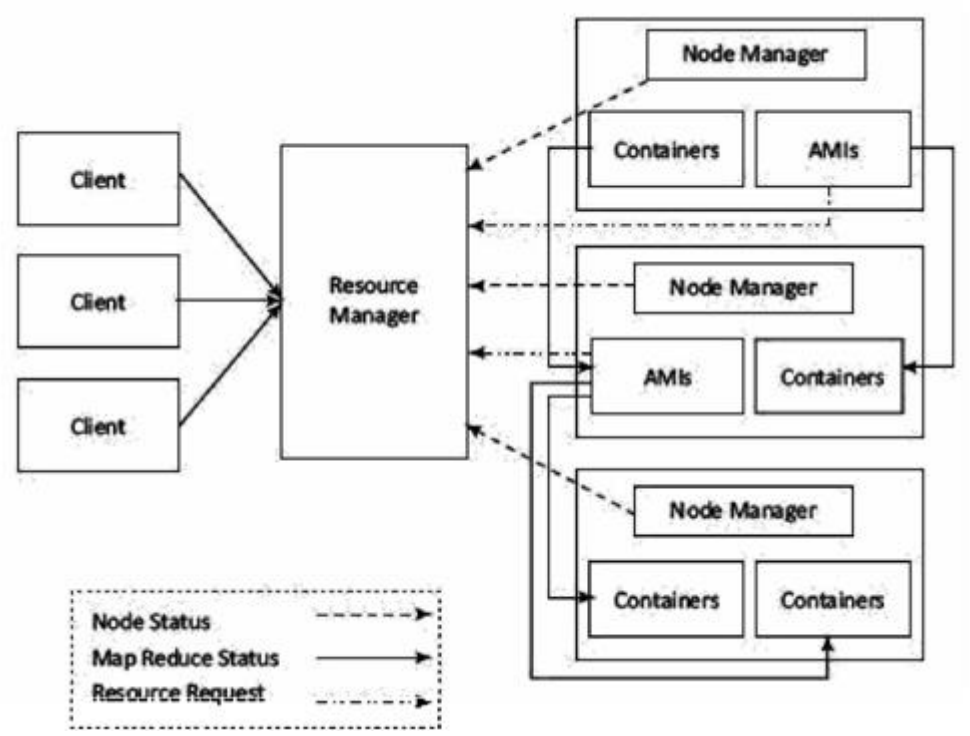


Figure 2.5 YARN-based execution model

Self-Assessment Exercise linked to LO 2.4

1. What are the resources required for running an application? How they are allocated?

2. List the functions of YARN.

3. Explain using Example 2.3, how the Application Master coordinates the execution of all tasks submitted for an application and requests for appropriate resource containers to executethe task.

4. List the functions of Client, Resource Manager, Node Manager, Application Master and Containers

2.6 | HADOOP ECOSYSTEM TOOLS

LO 2.4

Hadoop YARN for management and scheduling of resources for parallel running of application tasks

A simple framework of Hadoop enabled development of a number of open-source projects has quickly emerged (Figure 2.2). They solve very specific problems related to distributed storage and processing model. Table 2.2 gives the functionalities of the ecosystem tools and components.

Table 2.2 Functionalities of the ecosystem tools and components

ZooKeeper – Coordination service	Provisions high-performance coordination service for distributed running of applications and tasks (Sections 2.3.1.2 and 2.6.1.1)
Avro— Data serialization and transfer utility	Provisions data serialization during data transfer between application and processing layers (Figure 2.2 and Section 2.4.1)
Oozie	Provides a way to package and bundles multiple coordinator and workflow jobs and manage the lifecycle of those jobs (Section 2.6.1.2)
Sqoop (SQL-to-Hadoop) – A data-transfer software	Provisions for data-transfer between data stores such as relational DBs and Hadoop (Section 2.6.1.3)
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 (Section 2.6.1.4)
Ambari – A web-based tool	Provisions, monitors, manages, and viewing of functioning of the cluster, MapReduce, Hive and Pig APIs (Section 2.6.2)
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 (Section 2.6.3)
Cassandra – A database	Provisions scalable and fault-tolerant database for multiple masters (Section 3.7)
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 (Sections 2.6.4, 4.4 and 4.5)
Pig – A high-level dataflow language	Provisions dataflow (DF) functionality and the execution framework for parallel computations (Sections 2.6.5 and 4.6)
Mahout –A machine learning software	Provisions scalable machine learning and library functions for data mining and analytics (Sections 2.6.6 and 6.9)

The following subsections describe the Hadoop Ecosystem tools.

2.6.1 Hadoop Ecosystem Consider ZooKeeper, Oozie, Sqoop and Flume.

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

The loading of data into Hadoop clusters becomes an important task during data analytics. 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. Each map task will be assigned with part of data to be imported based on key defined 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:

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.

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:

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 of 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.

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. 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.

The console enables built-in tools for administering. Web console provides a link to server tools and open-source components associated with those.

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:

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 (Example 2.3).

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 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 (Example 2.3). The following example shows a structured table considering Examples 1.6 and 2.3.

EXAMPLE 2.4

Recapitulate Examples 1.6 and 2.3. Consider ACVMs selling KitKat, Milk, Fruit and Nuts, Nougat and Oreo chocolates. Following is the table for hourly sales of chocolates at multiple ACVMs.

ACVM_ID	Date (DT) mmddyy	Hour (hr)	KitKat Hourly Sale (KKHS)	Milk Hourly Sale(MHS)	Fruit and Nuts Hourly Sale (FNHS)	Nougat Hourly Sale (NHS)	Oreo Hourly Sale (OHS)
2206	121217	16	28	23	38	8	50
10	121217	16	29	14	44	15	38
1224	121217	16	40	41	23	37	8
16	121217	46	13	25	7	8	74
28	121217	16	52	22	16	28	7
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

(i) How does the HBase store the table?

(ii) How will the records created using shell command 'put'?

SOLUTION

Format of the HBase that stores rows line by line is:

```
Row-Key  Column-Family:  {Column-Qualifier:  Version:
Value}
```

HBase data model specifies the *column qualifiers.* For example, column qualifiers are DT, HR, KKHS, MHS, FNHS, NHS and OHS. Version corresponds to a number reflecting the time stamp which identifies the data of columns uniquely. Version is a number reflecting server time-stamp by default. Value is the *value* in the column field for the qualifier. The first row stores in the HBase as follows:

```
ACVM_id: '2206' {'DT':1600080000024: '121217', 'HR':
1600008007319: '16', 'KKHS': 1600081010821: '28',
'MHS': 1600082010582: '23', 'FNHS': 1600082018001:
'38', 'NHS': 1600080158868: '8', 'OHS':
1600038028229: '50'}
```

Write similarly for other rows of hourly sales table.

The records are put in rows and columns as follows:

```
hbase (main) 001:0> put 'ACVM_id', '2206', 'DT',
'121217', 'HR', '16', 'HourlySales: KKHS','28' 0
row(s) in 021120 seconds
hbase (main) 002:0> put 'ACVM_id', '2206',
'HourlySales: MHS', '23' 0 row(s) in 001120 seconds
hbase (main) 003:0> put 'ACVM_id', '2206',
'HourlySales: FNHS','38' 0 row(s) in 021120 seconds
hbase (main) 004:0> put 'ACVM_id', '2206',
'HourlySales: NHS', '8' 0 row(s) in 001120 seconds
hbase (main) 005:0> put 'ACVM_id', '2206',
'HourlySales: OHS', '50' 0 row(s) in 001120 seconds
```

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.

Self-Assessment Exercise linked to LO 2.5

1. Why is ZooKeeper required to behave as a centralized repository where the distributed applications can write the data?

2. What are the functions which Ambari perform? How does Ambari enable administering of clusters and Hadoop components?

3. Make a table of ecosystem tools and their functions which are required for analyzing performances from SGPs, SGPAs and CGPAs of each student. Assume that programmes are Master of Science in Computer Science, Master of Computer Appliations and Master of Technology in Computer Science.

4, What are the activities which Mahout supports in the Hadoop system?