# Module 2
# Chapter 2 : Introduction to Hadoop

**Syllabus**: Introduction to Hadoop (T1): Introduction, Hadoop and its Ecosystem, Hadoop Distributed File System, MapReduce Framework and Programming Model, Hadoop Yarn, Hadoop Ecosystem Tools. Hadoop Distributed File System Basics (T2): HDFS Design Features, Components, HDFS User Commands. Essential Hadoop Tools (T2): Using Apache Pig, Hive, Sqoop, Flume, Oozie, HBase. Text book 1: Chapter 2 :2.1-2.6 Text Book 2: Chapter 3 Text Book 2: Chapter 7 (except walk throughs)

**Course Outcome 2: Determine Hadoop framework components and Hadoop Distributed File system.**

## 2.1 INTRODUCTION

- A programming model is **centralised** 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

- Applications running at the server does the following analysis:
    - 1.Suggests a strategy for filling the machines at minimum cost of logistics
    - 2.Finds locations of high sales such a gardens, playgrounds etc.
    - 3. Finds days or periods of high sales such as Xmas etc.
    - 4 Finds children's preferences for specific chocolate flavors
    - 5 Finds the potential region of future growth
    - 6 Identifies unprofitable machines
    - 7. Identifies need of replicating the number of machines at specific locations.
- 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
    - ***Cooperation (sharing) between the DBs in a transparent manner*** *- Each user within the system may access all the data within all databases as if they were a single database.*
    - ***Location independence-*** *Results should be independent of geographical locations.*

**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 1:**

- 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.**

**Solution 2:**

- 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 of time.

- A child may **complete his/her part much late** than the remaining children. Beneath-the-sea portion is too complex, while upper-depth-sea portion 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.***
- **Solution for 100 children**
- Assume a second situation in which 100 children assemble their parts of 50 pieces each, and finally combine and complete the puzzle.
- Each child must seek a piece, not available with her/him daring 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)]
- 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 aw well as at the end stages.
- **Solution for 200 Children**
- 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, (200) + Tc (200)],**
- 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, (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
- **Final Solution**

- 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 from 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.

**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:

1.Distributed data storage systems do not use the concept of joins.

2.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.

3. 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.

Recall Table 1.2.

- Traditionally, a program when executes calls the data inputs.

**1.Centralized computing** model requires few communication overheads.

**2.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.*

- The application tasks and datasets needed for computations distribute at a number of geographic locations and remote servers.

3. **Massive Parallel Processing MPP**s 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 same cluster**. 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.

**(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 programming model, where jobs or tasks are assigned scheduled on the same servers which hold the data. Hadoop is one of widely technologies. Google and Yahoo use Hadoop. Hadoop creators created cost-effective method to search indexes. Facebook, Twitter, and Linkedln use Hadoop. IBM implemented BigInsights and licensed Apache Hadoop. Oracle implement Hadoop system with Big Appliance, IBM Infosphere Microsoft with Big solutions

**Following important terms and their meaning**

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

- Each cluster forms by a set loosely or tightly connected computing nodes that work together and many of operations can be scheduled and can be realized as if from single computing system.
- Clusters improve the performance, provide cost-effective and improved node accessibility compared to single computing node.
- Each node of computational cluster is set to perform the same task sub-tacks, such MapReduce, which software control and schedule.

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

**Data Consistency** means copies of blocks have the values.

**Data Availability** means at least one copy is available in case a partition becomes inactive or fails. example, in web applications, copy in other partition is available. Partition means parts, which active but may not cooperate as distributed databases (DB).

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

**Resource management** refers managing resources such their creation, deletion and controlled usages. The manager functions also include managing

(i) Availability for specified or scheduled periods,
(ii) Prevention of resource unavailability after task finishes and
(iii) Resources allocation when multiple tasks attempt to use same set resources,

**Horizontal scalability** means increasing the number systems working in coherence. For example, using MPP's number servers as per the size of dataset. Processing different datasets of large store running similar application deploys horizontal scalability.

**Vertical scalability** means scaling up using giving system resources and increasing number of tasks in system. For example, extending analytics processing including reporting, business processing (BP). business intelligence (BI), data visualization, knowledge discovery and machine learning (ML) capabilities which require additional ways to solve problems greater complexities and greater processing, storage and inter-process communication among resources. Processing different datasets of large data sure running multiple application tasks deploys vertical scalability.

**Ecosystem** refers to system made up of multiple computing components, which work together. That similar to biological ecosystem, complex system living organisms, their physical environment and all their inter-relationships in particular unit space.

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

**Hadoop Distributed File System** means system of storing files (set 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** means 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.

## 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.
- **High Availability Distributed Object Oriented Platform**
- The project consisted of **two components**,

- ***Data store*** *in blocks in the clusters*
- ***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
- 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**. Software enable the service or task execution even in case of communication or node failure.
- 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**.
- Hadoop framework provides the computing features of a system of distributed, flexible, scalable, fault tolerant computing with high computing power.
- Hadoop system is an efficient platform for the distributed storage and processing of a large amount of data.
- 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 **1,00,000 CPUs** in over **40,000 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 = $10^{15}$ B. nearly $2^{50}$ B) raw-storage.
- Each (commodity) node has 8 cores and 12 TB.

### 2.2.1 Hadoop Core Components

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

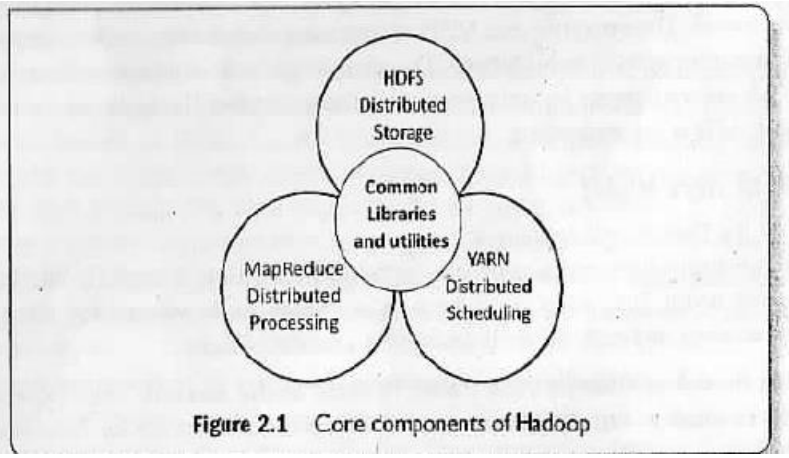



**Figure 2.1** Core components of Hadoop

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 v1** - Software programming model in Hadoop 1 using Mapper and Reducer. The vl processes large sets of data in parallel and in batches.

4 -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 dist**. YARN** ributed running of the tasks.

5 **MapReduce v2-**Hadoop 2 YARN-based sym for parallel pocessing of large dates 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 cables 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

**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 analysing 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 classer fails. This is because of Hadoop provisions for backup (due to replications atleast 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 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.

These different types of data can be moved in HDFS for analysis using interactive query processing tools of Hadoop ecosystem components, such as Hive or can be provided to online business processes with the help of Apache HBase.

### 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 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

Figure 2.2 shows Hadoop core components HDFS, MapReduce and YARN along with the ecosystem. Figure 2.2 also shows Hadoop ecosystem. The system includes the application support layer and application layer components- AVRO, ZooKeeper, Pig, Hive, Sqoop, Ambari, Chukwa, Mahout, Spark, Flink and Flume. The figure also shows the components and their usages.

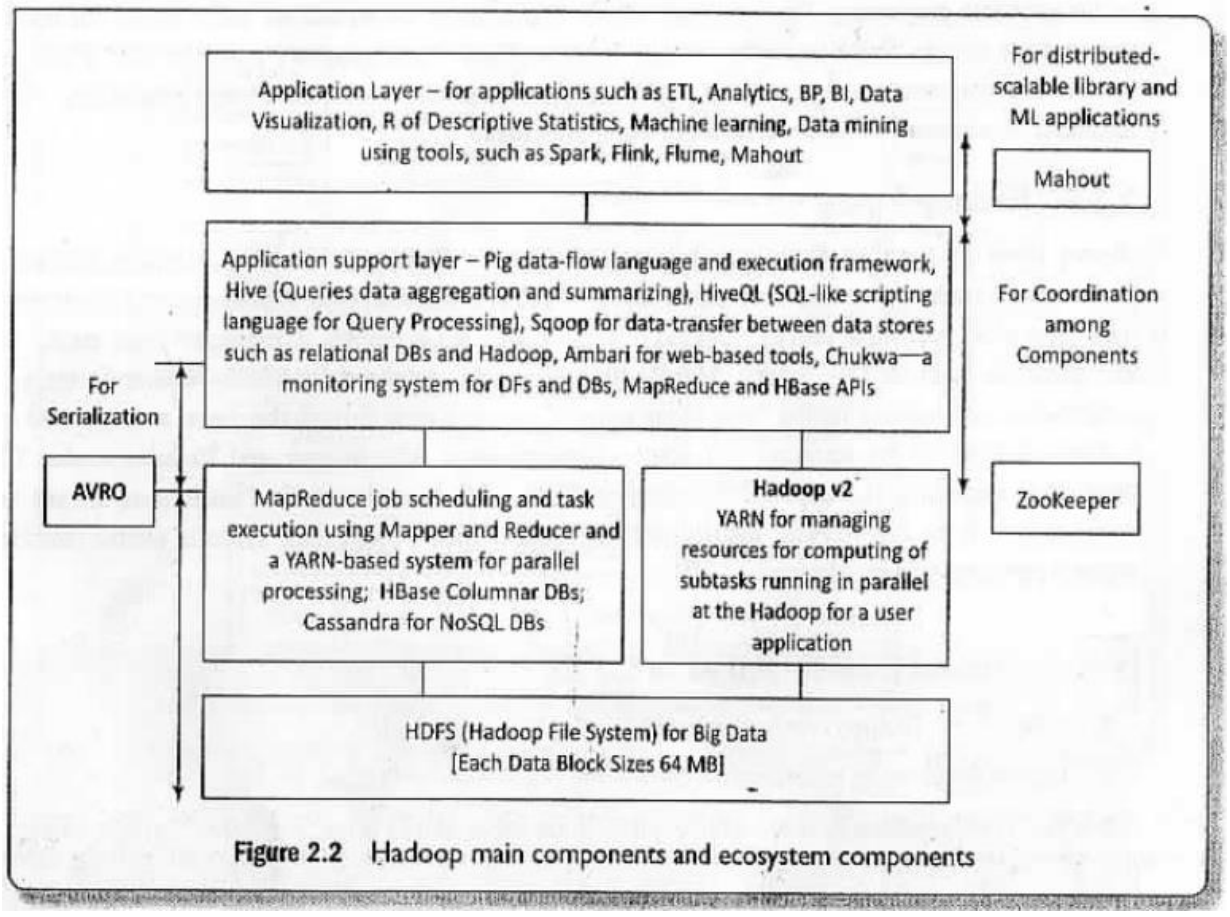



**Figure 2.2** Hadoop main components and ecosystem components

The four layers in Figure 2.2 are as follows:

(i) Distributed storage layer
(ii) (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.
- 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**.
- 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 a 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.
- 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 codes.
- IBM Power Linux systems enable working with Hadoop pipes and system libraries.

## 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 Big Data which may range from GBs (1 GB = $2^{30}$ B) to PBs (1 PB = $10^{15}$ B. nearly the $2^{50}$ 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 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:**

(1) Create, append, delete, rename and attribute modification functions
(2) Content of individual file cannot be modified or replaced but appended with new data at the end of the file.
(3) Write once but read many times during usages and processing.
(4) 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 = 2B). 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 student's 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 x 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 x 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 x 128 MB = 15360 GB = 15 TB. Total memory capacity of each DataNode in each rack = 1024 x 64 MB = 64 GB.

(iv) 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.

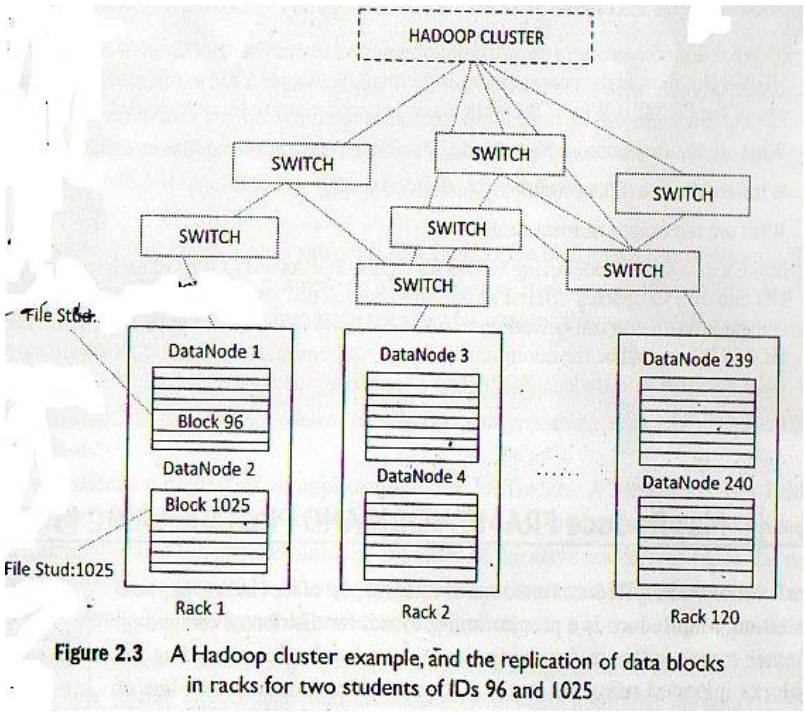


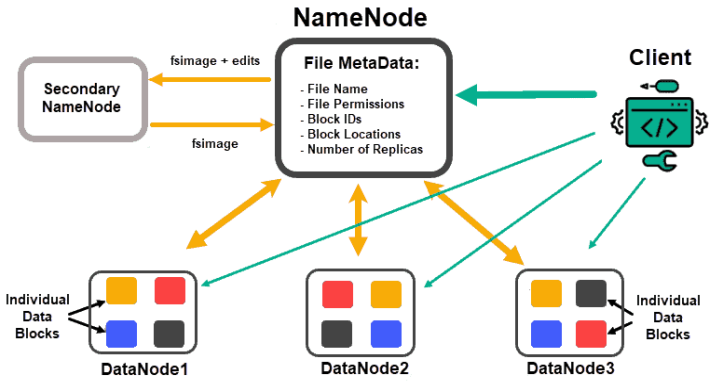

**Figure 2.3** A Hadoop cluster example, and the replication of data blocks in racks for two students of IDs 96 and 1025

(v) Changes will be that each node will have half the number of data blocks.

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

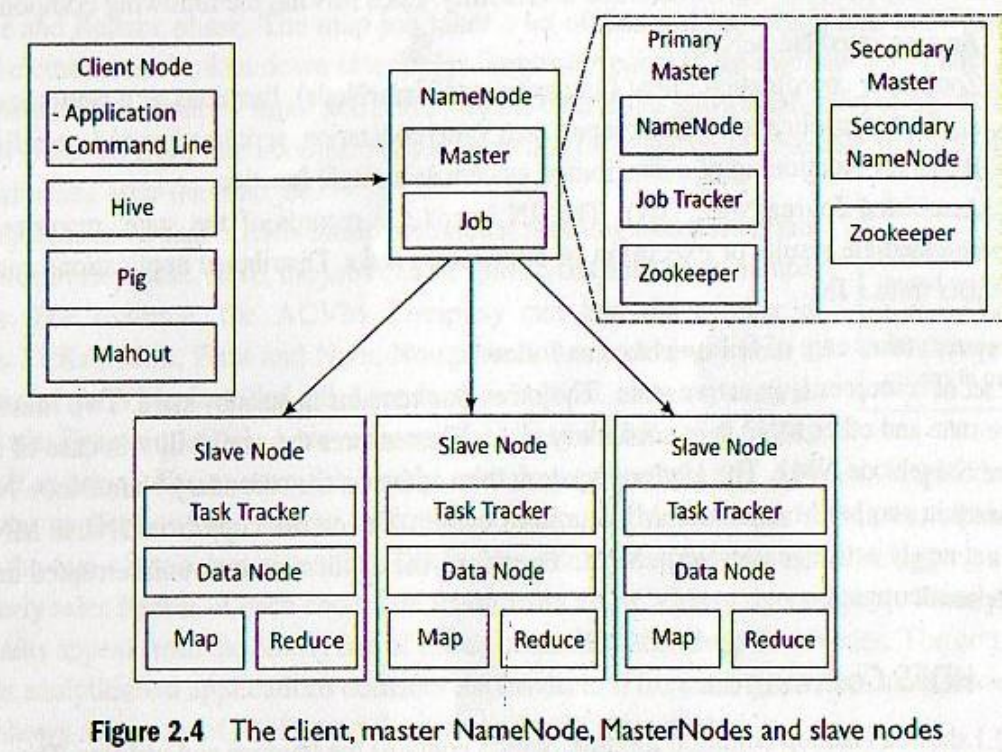



**Figure 2.4** The client, master NameNode, MasterNodes and slave nodes

- 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.

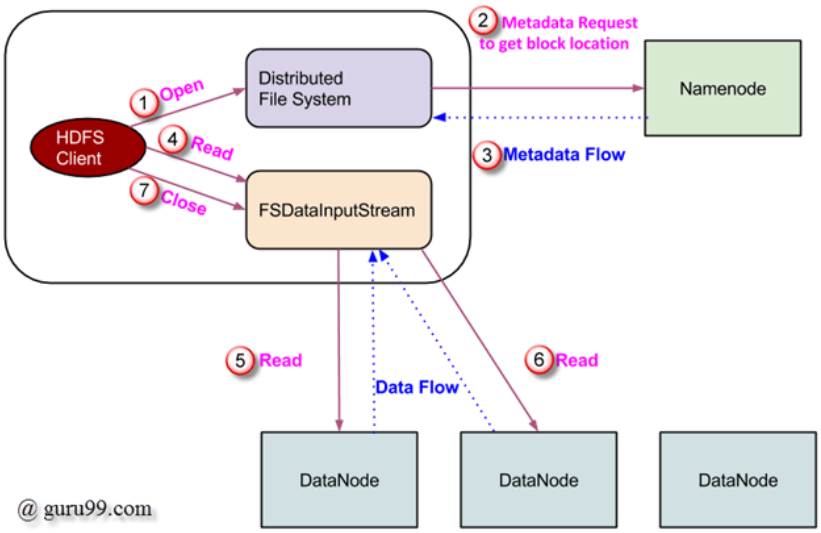



**Fig. Hadoop Read Request**

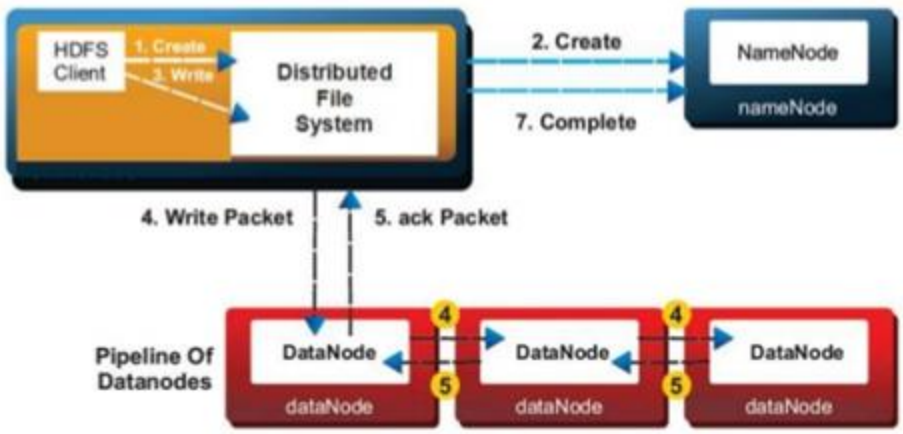



**Fig. Hadoop Write Request**

### 2.3.1.2 Hadoop 2

Single NameNode failure in Hadoop 1 is an oerational 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.
- Full set of the Hadoop shell commands can be at Apache Software Foundation website.
- -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
- Table 2.1 gives the examples of command usages.

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 **$Hadoop hdfs-mkdir/user/stu_filesdir** creates the directory named stu_filesdir |
| -put | Assume file stuData_id96 to be copied at stu_filesdir directory in Example 2.2. Then **$Hadoop hdfs-put stuData_id96 /user/ stu_filesdir** copies file for student of id96 into stu_filesdir directory |
| -ls | Assure 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. |

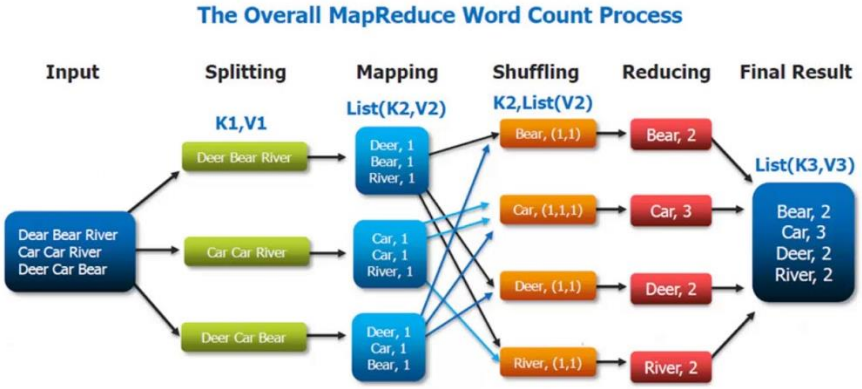

## 2.4 MapReduce FRAMEWORK AND PROGRAMMING MODEL

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 processing very large data Mapper.
- The command maps the key to the data, which an application uses.

**Reducer**

- 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 I

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"].
- MapRedace 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 MapRedoce:**

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.

(i) 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.

(ii) The second is a number of subordinate processes called Task Trackers. These processes to assigned tasks and periodically report the progress to the JobTracker

- Figure 2.4 showed the job execution model of MapReduce.
- Here the Job Tracker schedules jobs submitted by clients, keeps track of Task Trackers 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 Job Tracker.

### 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 datasets (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, ie, structured or unstructured stored in HDFS.

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

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

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 server multiple cities on separate.

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 sales appear from the hourly log of transactions filed at Hadoop DataNodes. The company enterprise runs analytics and applications consider the results as if from a single server application.

The following angle shows the usage of HDPS and the map and reduce functions.

EXAMPLE 2.3

Consider Example 1.60) of ACVMs selling Kitkat, Milk, Fruit and Not Nougat and Oreo chocolates Am 24 tiles are created every hour for each day. The files are at ile 1, file 2. file 24. Each file stores a key-value pairs as hourly sales log at the large number of machines.

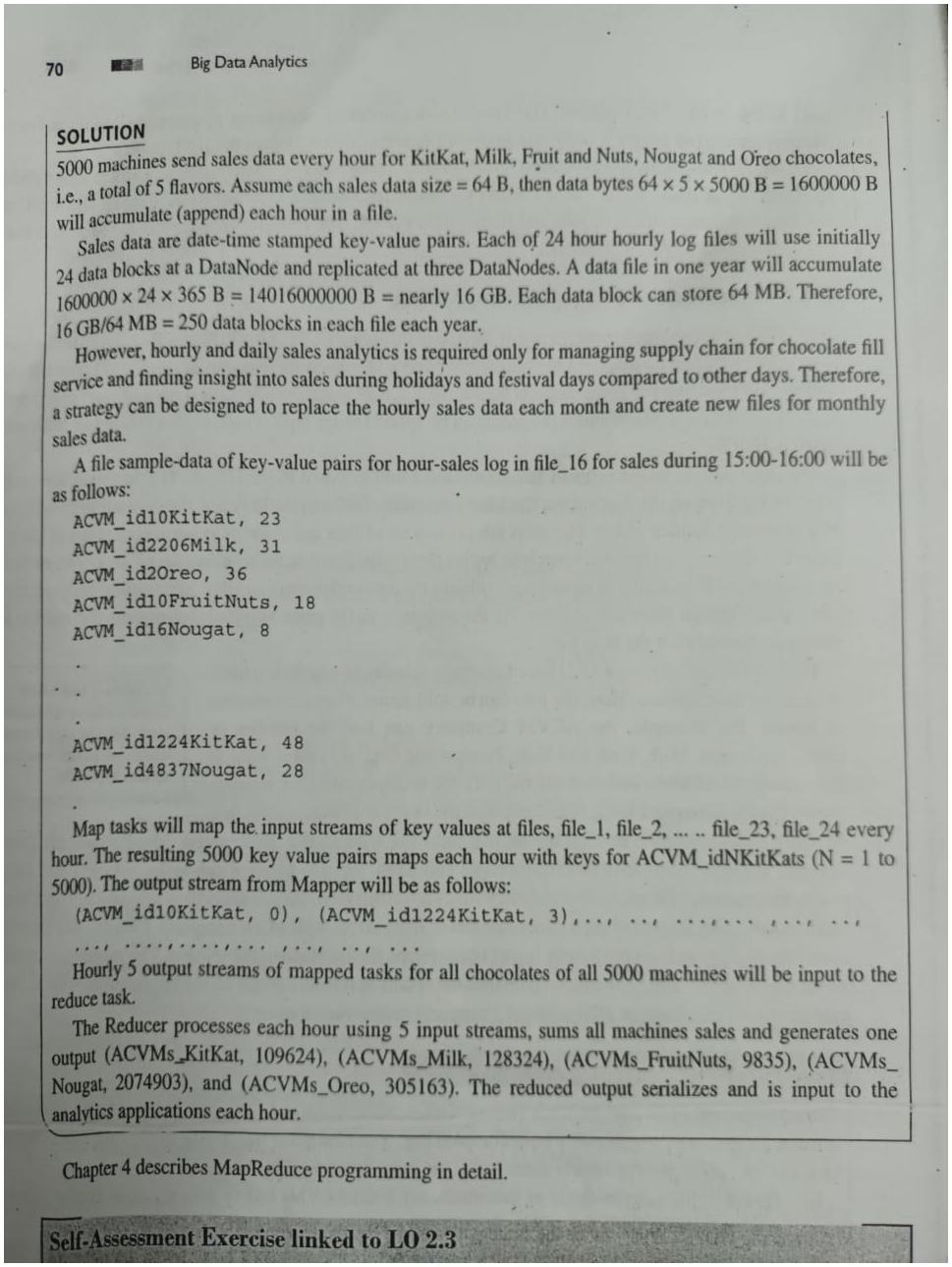
1. How will the large number of machines, say 5000 ACVMs hourly data for each flavor sales log store using HDFS? What will be the strategy to restrict the data size in HDFS?
2. How will the sample of data collected in a file for 0-1.1-2, 12-13,13-14, 15-16, up to 23-24 specific hour sales at a large number of machines, say 5000.
3. What will be the output scams of map tasks for feeding the input streams to the Reducer?
4. What will the Reducer output?

SOLUTION

5000 machines send sales data every hour for KitKat, Milk, Fruit and Nut, Nougat and Oreo chocolates, Le, a total of 5 flavor. Assume each sales data size=64 B, the data bytes 64 x 5 x 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 34 data blocks at a DataNode and replicated at three DataNodes. A data file in one year will accumulate 1600000 x 24 x 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:



70 Big Data Analytics

**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_id20reo, 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**

**Sell-Assessment Exercise linked to**

1. Why a mapping required when processing the start data HDFS?
2. How JobTracker and TaskTracker function?
3. How does MapRreducer along with the YARN resource manager enable faster processing of an application?
4. Consider HDFS DataNodes in a cluster. Draw a diagram depicting 10 data nodes storing the data of 4 groups of students. Using the diagrams, show the execution of MapReduce de tasks for each grow in parallel on the DataNodes in a cluster.

## 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 submit 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 Maser (AM) and Containers.
- List of actions of YARN resource allocation and scheduling functions is as follows:
- A MasterNode has two components: () 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 (AMD) and starts aps. 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 (AppM), that estimates the resources requirement for running an application program or sub-task The App Ms 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. Il signals whenever it initializes. All active NMs send the cooling 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.

**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 execute the task

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

## 2.6 HADOOP ECOSYSTEM TOOLS

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

| Ecosystem Tool | Functionalities |
|---|---|
| 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 | Provisions dataflow (DF) functionality and the execution framework for |

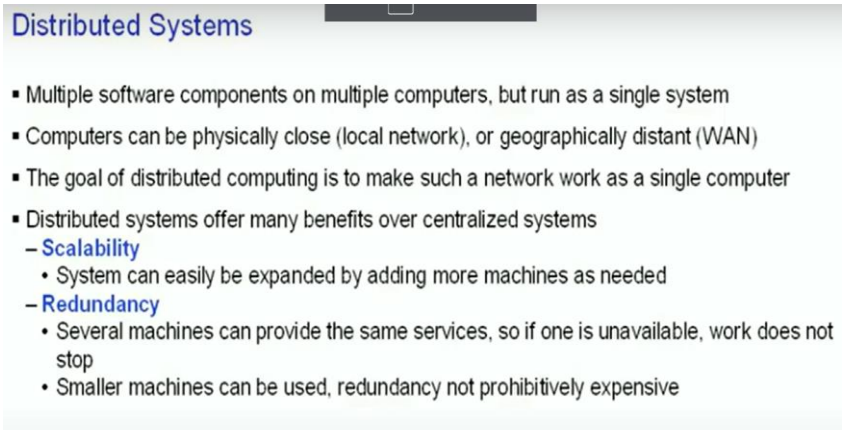
| | |
|---|---|
| language | parallel computations (Sections 2.6.5 and 4.6) |
| Mahout –A machine learning software | Provisions scalable machine learning and library functions for data mining and analytics (Sections 2.6.6 and 6.9) |

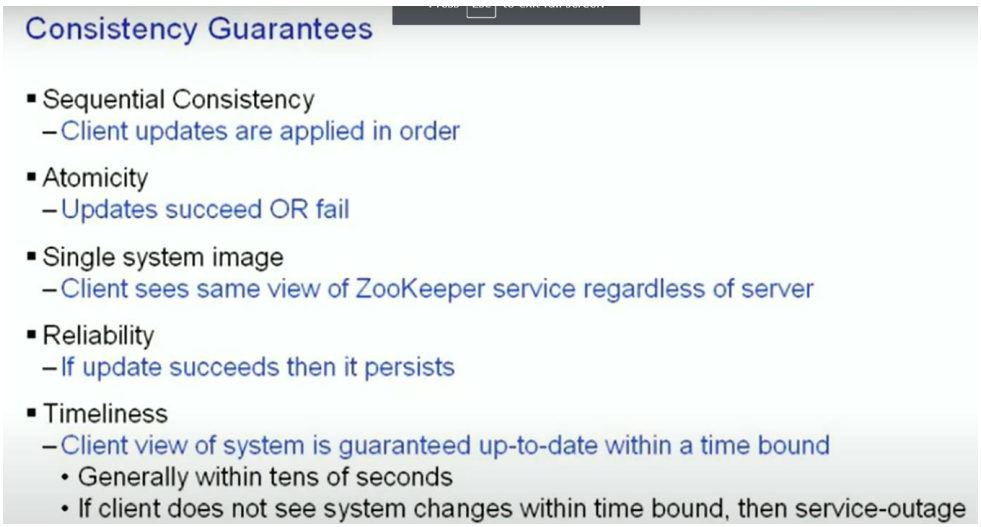
### 2.6.1 Hadoop Ecosystem

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's 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 these are up and running, and looks up their states by name in name service.

2 Concurrency control Concurrent access to a shared resource may cause inconsistency of these 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 jobs 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 handle 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 (DAG), specifying a sequence of actions to execute. Oozie coordinator jobs are current Oozie workflow jobs that are triggered time and data availability.

Oozie provisions for the following

1. Integrates multiple jobs in a sequential manner

2 Stores and supports Hadoop jobs for MapReduce, Hive, Pig, and Sqoop

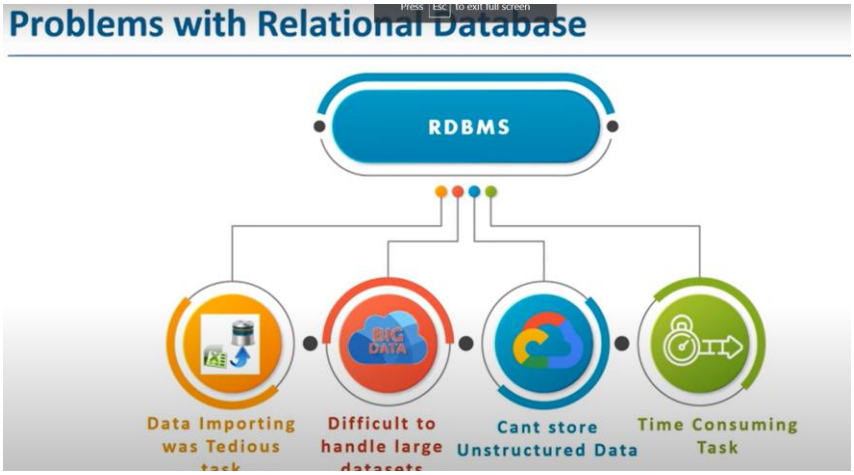
3. Runs workflow jobs based on time and data triggers

4. Manages batch coordinator for the applications

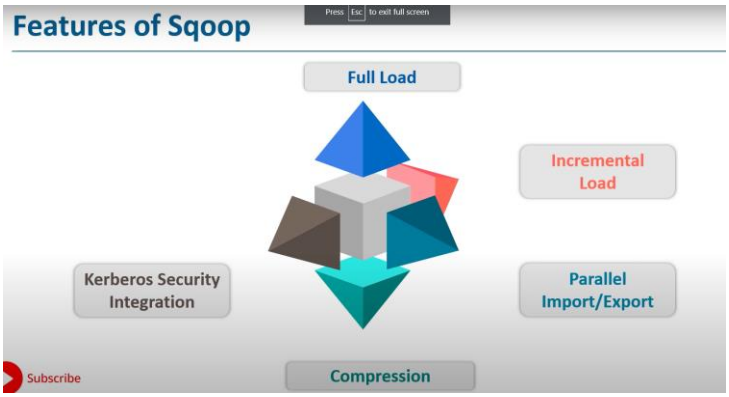
5. Manages the timely execution of tens of elementary jobs lying in thousands of workflows in Hadoop cluster.

#### 2.6.1.3 Sqoop

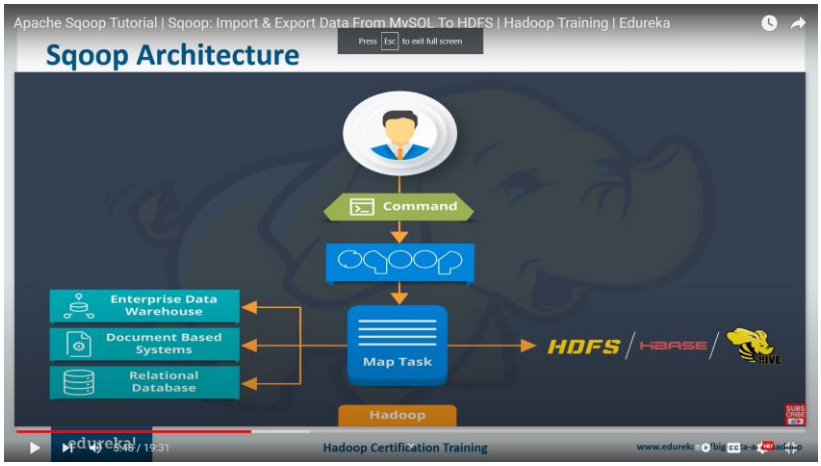
- The loading of data into Hadoop clusters becomes an important task during data analytics.

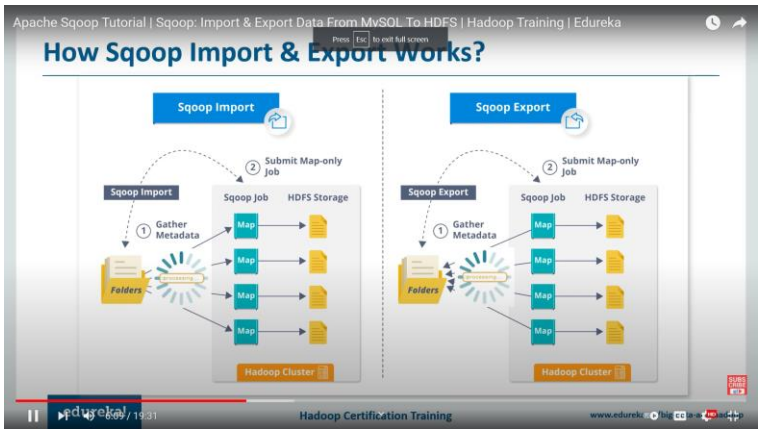
- Apache Sqoop in a tool that is built for loading efficiently the voluminous amount of data between Hadoop and external data repositories that reside on enterprise application servers or relational databases.
- Spoop works with relational databases such as Oracle, MySQL. PostgreSQL and DB2.
- 

- 

- 

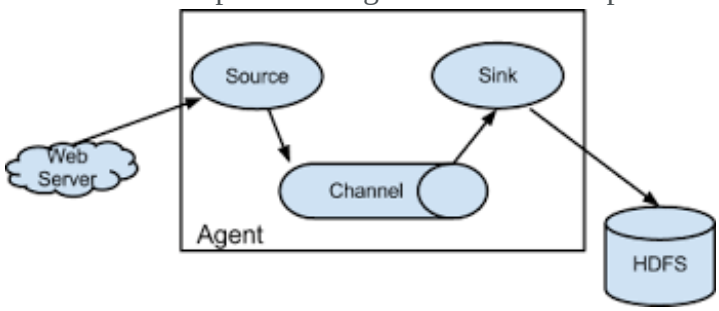
- Spoop provides command line interface to its users. Sqoop can also be accessed using Java APIs.
- The tool allows defining the schema of the data for import, Sqoop exploits Mapreduce framework to import and expect the data, and transfers for parallel processing of sub-tasks. Sqoop provisions for fault tolerance. Parallel transfer of data results in parallel results and fast data transfer
- Spoop 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 HDPS/Hive/HBase as per the choice provided in the command line.

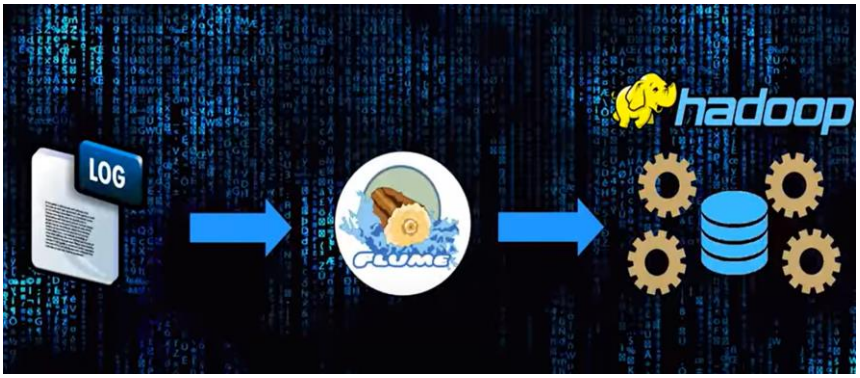


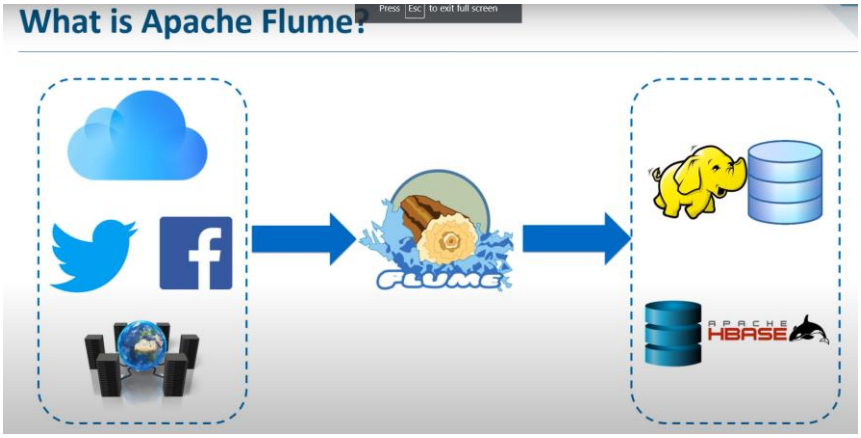




### 2.6.1.4 Flume

- Apache Flume provides a distributed, reliable and available service.
- Flume efficiently collects, aggregates and transfers a large amount of unstructured data into HDFS.
- Flume enables upload of large files into Hadoop clusters.



- 

- 


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 HDPF 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 queueing 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 and remove the event from the queue to a target
4. Agents runs the sinks and sources in Flume The interceptors drop the data or transfer data as it flows into the system.

Sqoop is **used for bulk transfer of data between Hadoop and relational databases** and supports both import and export of data. Flume is used for collecting and transferring large quantities of data to a centralized data store.

### 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 Amburi 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. tools and open

The console enables built-in tools for administering. Web console provides a link to server 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 HDPS.

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 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 Big Table

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 bour at an ACVM (Example 2.3).

### 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 pair) 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. Sections 4.4 and 45 will describe the Hive and HiveQL in detail.

### 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 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.
(iii) 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

Mabout in 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 groups, such that items belonging to the same group are similar to each other.
- Classification that means leaming 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

# Chapter 3 Hadoop Distributed File Systems Basic (Text Book 2)

The Hadoop Distributed File System is the backbone of Hadoop MapReduce processing. New users and administrators often find HDFS different than most other UNIX/Linux file systems.

## Hadoop Distributed File System Design Features

- The Hadoop Distributed File System (HDFS) was designed for Big Data processing. Although capable of supporting many users simultaneously, HDFS is not designed as a true parallel file system.
- Rather, the design 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 HDFS block size is typically 64MB or 128MB Thus, this a 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.
- Perhaps 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 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 occur at some
  1. The write-once/read-many design is intended to facilitate streaming reads.
  2. Files may be appended, but random seeks are not permitted. There is no caching of data.
  3. Converged data storage and processing happen on the same server nodes.
  4. "Moving computation is cheaper than moving data."
  5. A reliable file system maintains multiple copies of data across the cluster. Consequently, failure of a single node (or even a rack in a large cluster) will bring down the file system.
  6. 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**.
- 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 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.
- 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.**
- 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.
- Note that the NameNode does not write any data directly to the DataNodes.
- It gives the client a limited **amount of time** to complete the operation.
- If it does not complete in the time period, the operation is **cancelled**.
- **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 access 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.
- While data transfer is progressing, the **NameNode also monitors the DataNodes** by listening for **heartbeats** sent from DataNodes.
- The **lack of a heartbeat** signal indicates a potential node **failure**.
- 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 reports enable the NameNode to keep an **up-to-date account** of all data blocks in the cluster.
- In almost all Hadoop deployments, there is a **Secondary NameNode.** While not explicitly required by a NameNode, it is highly recommended.
- The term "Secondary NameNode" (now called **CheckPointNode**) is somewhat **misleading**. It is not an active **failover node and cannot replace the primary NameNode** in case of its failure.

- The purpose of the Secondary NameNode 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. This file begins with **fsimage_*** and is used only at startup by the NameNode.
- A **series of modifications done** to the file system after starting the NameNode. These files begin with **edit_*** and reflect the changes made after the fsimage_* file was read.
- The **location** of these files is set by the **dfs.namenode.name.dir** property in the **hdfs-site.xml** file.
- The Secondary NameNode 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.
-

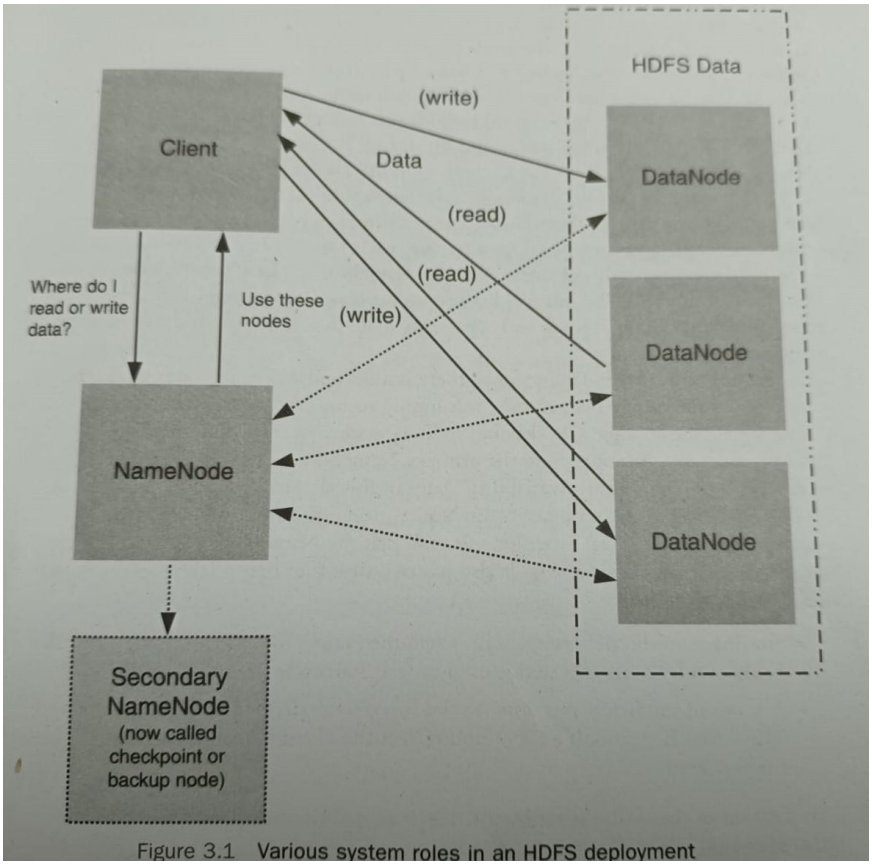



Figure 3.1 Various system roles in an HDFS deployment

**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 Secondary NameNode performs checkpoints of NameNode file system's State but is not a failover node.

**HDFS Block Replication**

- As mentioned, 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 Data Nodes but more than one DataNode, a replication factor of 2 is adequate.
- For a single machine, like the pseudo-distributed, 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).

Figure 3.2 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 subse quently re-replicated on other DataNodes.

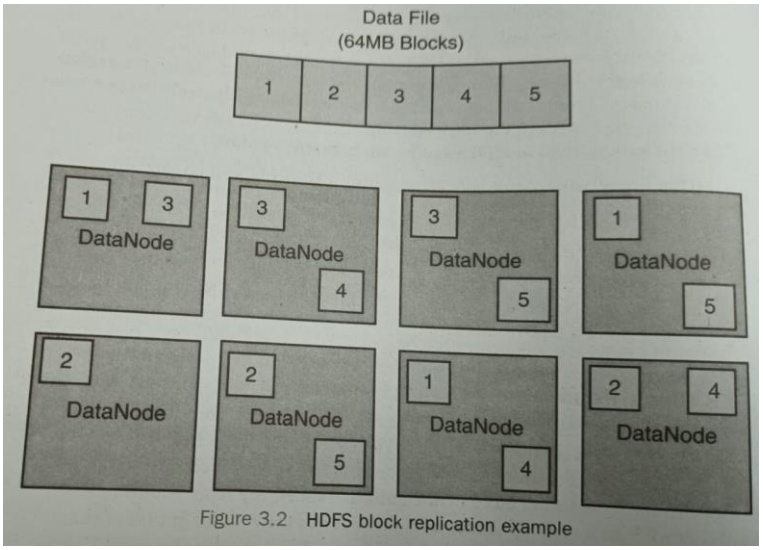


Figure 3.2 HDFS block replication example

**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***
- 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. Recall that 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 chaster will exhibit three levels of data locality:

    - Data resides on the local machine (best).
    - Data resides in the same rack (better).
    - 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. Performance may be degraded, however.

- 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 3.3, 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 Like a single NameNode cluster, 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 Name Node 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 Journal Nodes.A Secondary NameNode 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. The Zookeeper components are not depicted in Figure 3.3.

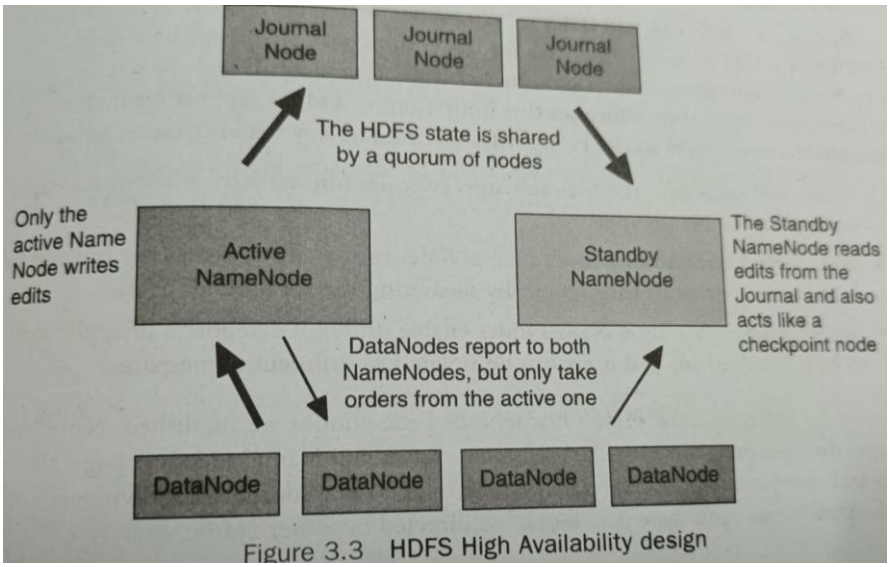
- 

Figure 3.3 HDFS High Availability design

## HDFS NameNode Federation

Another important feature of HDFS is 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:**

1. Namespace scalability, HDFS cluster storage scales horizontally without placing a burden on the NameNode.
2. Better performance. Adding more NameNodes to the cluster scales the file system read/write operations throughput by separating the total namespace.
3. System isolation. Multiple NameNodes enable different categories of applications to be distinguished, and users can be isolated to different namespaces.

Figure 3.4 illustrates how HDFS NameNode Federation is accomplished. NameNodel 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.

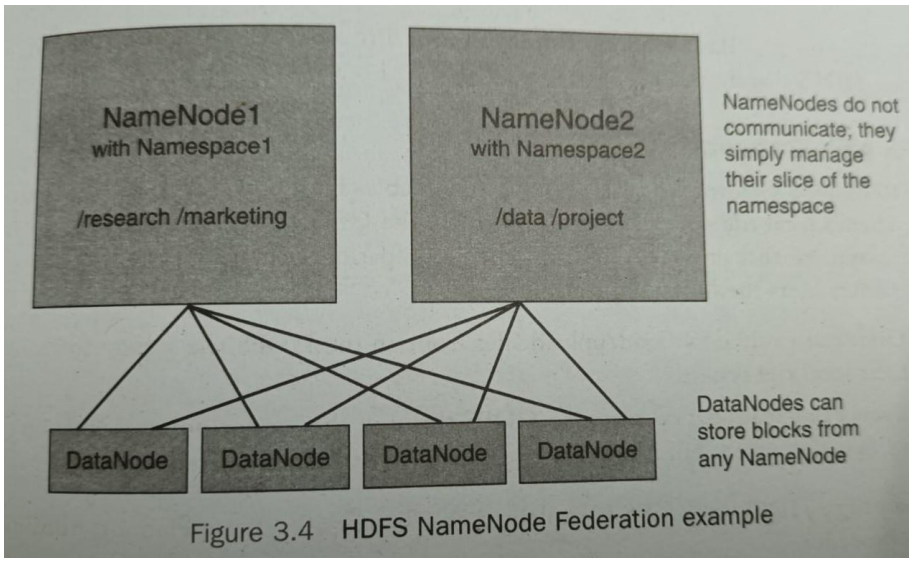


Figure 3.4 HDFS NameNode Federation example

## HDFS Checkpoints and Backups

- HDFS Backup node maintains an up-to-date copy of file system namespace both in memory and disk.
- HDFS Snapshots
- Similar to backup but created by administrators using hdfs dfs –snapshots command
- They are readonly point-in-time copies of the file system.
- They offer the following featrures:
- Snapshot of sub-tree of file system
- Used for data backup, protection against user errors and disaster recovery.
- Snapshot creation is instantaneous.

## HDFS NFS Gateway

- Supports Enables HDFS to be mounted as part of the client local file 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 mount point.

# Hadoop General Commands

## Version of HDFS can be found from the version option.

```
$ 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 b3481c2cdbe2d181f2621331926
This command was run using /usr/hdp/2.2.4.2-2/hadoop/
common-2.6.0.2.2.4.2-2.jar
```

HDFS provides a series of commands similar to those found in standard POSIX file system.
A list of commands can be obtained by issuing the following command:

**$hdfs dfs**

```
[-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> ... <local
[-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]
```

```
[-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]
```

The list of files in the root HDFS directory , enter the following command:

```
$ 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
drwxr-xr-x   - hdfs hdfs
```

- **Make a directory in HDFS**
  - *$hdfs dfs –mkdir stuff*
- **Copy files to HDFS**
  - *$hdfs dfs –put test stuff*
- **Copy files from HDFS**
  - *$hdfs dfs –get stuff/test test-local*
- **Copy files within HDFS**
  - *$hdfs dfs –cp stuff/test test.hdfs*
- **Delets files within HDFS**
  - *$hdfs dfs –rm test.hdfs*
- Delete a directory in HDFS
- $hdfs dfs –rm –r –skiptrash test.hdfs

```
$ hdfs dfsadmin -report

Configured Capacity: 1503409881088 (1.37 TB)
Present Capacity: 1407945981952 (1.28 TB)
DFS Remaining: 1255510564864 (1.14 TB)
DFS Used: 152435417088 (141.97 GB)
DFS Used%: 10.83%
Under replicated blocks: 54
Blocks with corrupt replicas: 0
Missing blocks: 0
```

# Chapter 7 Essential Hadoop Tools (Text Book 2)

The Hadoop ecosystem offers many tools to help with data input, high-level processing. workflow management, and creation of huge databases. Each tool is managed as a separate Apache Software foundation project, but is designed to operate with the core Hadoop services including HDFS, YARN, and MapReduce. Background on each tool provided in this chapter, along with a start to finish example.

## Using Apache Pig

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

Pig Latin (the actual guage) 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 MapReduce engine or optimized Tez engine.

Table 7.1 Apache Pig Usage Modes

| | Local Mode | Tez Local Mode | MapReduce Mode | Tez Mod |
|---|---|---|---|---|
| Interactive Mode | Yes | Experimental | Yes | Yes |
| Batch Mode | Yes | Experimental | Yes | Yes |

### Pig Example Walk-Through

For this example, the following software environment is assumed. Other environments should work in a similar fashion.

- OS: Linux
- Platform: RHEL 6.6
- Hortonworks HDP 2.2 with Hadoop version: 2.6
- Pig version: 0.14.0

In this simple example, Pig is used to extract user names from the /etc/passwd. A full description of the Pig Latin language is beyond the scope of this introduction but more information about Pig can be found at http://pig.apache.org/docs/r0.14.C start.html. The following example assumes the user is hdfs, but any valid user with access to HDFS can run the example.

To begin the example, copy the passwd file to a working directory for local Pig operation:

**$ cp /etc/passed.**

Next, copy the data file into HDFS for Hadoop MapReduce operation:

**$ hdfs dfs -put passwd passwd**

You can confirm the file is in HDFS by entering the following command:

**hdfs dfs -ls passwd**

-rw-r--r-- 2 hdfs hdfs 2526 2015-03-17 11:08 passwd

In the following example of local Pig operation, all processing is done on the local machine (Hadoop is not used). First, the interactive command line is started:

$pig -x local

If Pig starts correctly, you will see a grunt> prompt.Next, enter the following commands to load passwd file and then grab the user name and dump it to the terminal Note that Pig commands must end with a semicolon (:).

**grunt> A= load 'passwd' using PigStorage (':');**

**grunt> B foreach A generate $0 as id;**

**grunt> dump B;**

The processing will start and a list of user names will be printed to the screen. To exit the interactive session, enter the command quit.

**$ grunt> quit**

To use Hadoop MapReduce, start Pig as follows (or just enter pig):

**$ pig -x mapreduce**

The same sequence of commands can be entered at the grunt> prompt. You may wish to change the $0 argument to pull out other items in the passwd file. In the case of this simple script, you will notice that the MapReduce version takes much longer. Also, because we are running this application under Hadoop, make sure the file is placed in HDFS.

If you are using the Hortonworks HDP distribution with tez installed, the tez engine can be used as follows:

**$ pig -x tez**

Pig can also be run from a script. An example script (id.pig) is available from the example code download (see Appendix A, "Book Webpage and Code Download"). This script, which is repeated here, is designed to do the same things as the interactive version:

```
/*id.pig */
A= load 'passwd using PigStorage (':');          -- load the passwd file
B= foreach A generate $0 as id;                  -- extract the user IDs
dump B:
store B into 'id.out';                           -- write the results to a directory name id.out
```

Comments are delineated by /* */ and -- at the end of a line. The script will create a directory called id. out for the results. First, ensure that the id. out directory not in your local directory, and then start Pig with the script on the command line:

**$/bin/rm -r id.out/**

**$pig -x local id.pig**

**#pid -x local**

If the script worked correctly, you should get at least one data file with the results and a zero-length file with the name _SUCCESS. To run the MapReduce version, use the same procedure; the on;y difference is that now all reading and writing takes place in HDFS.

**$ hdfs dfs -rm -r id.out**

**$ pig id.pig**

## 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 programmer who are familiar with the MapReduce framework to add their custom mappers an reducers to Hive queries. Hive queries can also be dramatically accelerated using th Apache Tez framework under YARN in Hadoop version 2.

### Hive Example Walk-Through

For this example, the following software environment is assumed. Other environment should work in a similar fashion.

- OS: Linux
- Platform: RHEL 6.6
- Hortonworks HDP 2.2 with Hadoop version: 2.6
- Hive version: 0.14.0

To start Hive, simply enter the hive command. If Hive starts correctly, you should get a hive prompt

**$ hive**

As a simple test, create and drop a table. Note that Hive commands must ends with a semicolon (;).

**hive>CREATE TABLE pokes (foo INT, bat STRING)**

**hive> SHOW TABLES;**

OK

pokes

Tine taken: 0.174 seconds, Fetched: 1 cow(s)

**Hive> DROP TABLE pokes;**

OK

Time taken: 4.938 seconds

A more detailed example can be developed using a web server log file to summarize message types. First, create a table using the following command

**hive> CREATE TABLE logs (t1 string, t2 string, t3 string, t4 string, t5 string, t6 string, t7 string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ' ';**

Next, load the data in this case, from the sample.log fille. This file is available from the example code download (see Appendix A). Note that the file is found in the local directory and not in HDFS.

**Hive> LOAD DATA LOCAL INPATH sample.log" OVERWRITE INTO TABLE Logs:**

Finally, apply the select step to the file. Note that this invokes a Hadoop MapReduce operation. The results appear at the end of the output (eg, tocals for the message types DEBUG, ERROR, and so on).

**hive> SELECT t4 AS sev. COUNT(*) AS cnt FROM logs WHERE t4 LIKE '[ %'GROUP BY t4;**

[DEBUG] 434

[ERROR] 3

[FATAL] 1

[INFO] 96

[TRACE] 816

[WARN] 4

Time taken: 32.624 seconds, Fetched: 6 row (s)

To exit Hive, simply type exit;:

**hive> exit;**

**A More Advanced Hive Example**

A more advanced usage case from the Hive documentation can be developed using the movie rating data files obtained from the GroupLens Research (http://group lens.org/datasets/movielens) webpage. The data files are collected from Movie Lens website (http://movielens.org). The files contain various numbers of movie reviews, starting at 100,000 and going up to 20 million entries. The data file and queries used in the following example are available from the book website (see Appendix A).

In this example, 100,000 records will be transformed from userid, movieid, rating, unixtime to userid, movieid, rating, and weekday using Apache Hive and a

Python program (.e., the UNIX time notation will be transformed to the day of the week). The first step is to download and extract the data:

**s wget http://files.grouplens.org/datasets/movielens/ml-100k.zip**

$ unzip ml-100k.zip $ cd 1-100%

Before we use Hive, we will create a short Python program called weekday_mapper.py with following contents:

Import sys

import datetime

for line in sys.stdin:

line line.strip()

userid, novieid, rating, unixtine = line.split("\t")

weekday = datetime.datetime.frontimestamp (float (unixtime)).isoweekday()

print \t.join(userid, movieid, rating, str(weekday)])LOAD DATA LOCAL INPATH ./.data' OVERWRITE INTO TABLE u_data;

Next, start Hive and create the data table (u_data) by entering the following at the hive> prompt:

CREATE TABLE u_data ( userid INT,

movieid INT.

rating INT,

unixtine STRING) ROW FORMAT DELIMITED

FIELDS TERMINATED BY \t'

STORED AS TEXTFILE;

Load the movie data into the table with the following command:

**hive> LOAD DATA LOCAL INPATH ‘./u.data' OVERWRITE INTO TABLE u_data;**

The number of rows in the table can be reported by entering the following command:

**hive > SELECT COUNT(*) FROM u_data;**

This command will start a single MapReduce job and should finish with the following lines:

MapReduce Jobs Launched:

Stage-Stage-1: Map: 1 Reduce: 1 HDPS Write: 7 SUCCESS Cumulative CPU: 2.26 sec Total MapReduce CPU Time Spent: 2 seconds 260 msec HDFS Read: 1979380

100000

Time taken: 28.366 seconds, Fetched: 1 row (s)

Now that the table data are loaded, use the following command to make the new table (u_data_new):

**hive> CREATE TABLE u_data_new (**

**userid INT. movieid INT.**

**rating INT. weekday INT)**

**ROW FORMAT DELIMITED.**

**FIELDS TERMINATED BY \t':**

The next command adds the weekday_mapper.py to Hive resources:

hive> add FILE weekday_mapper.py;

Once weekday_mapper.py is successfully loaded, we can enter the transformation query:

hive> INSERT OVERWRITE TABLE u_data_new.

SELECT

TRANSFORM (userid, movieid, rating, unixtime)

USING ‘python weekday_mapper.py'

AS (userid, movieid, rating, weekday)

FROM u_data;

If the transformation was successful, the following final portion of the output should be displayed:

Table default.u_data_new stats: [numFiles-1, numRows=100000, totalSize=1179171, rawDataSize-1079173]

MapReduce Jobs Launched:

Stage-Stage-1: Map: 1 Cumulative CPU: 3.44 sec HDFS Read: 1979380 HDFS Write:

1179256 SUCCESS

Total MapReduce CPU Time Spent: 3 seconds 440 msec

OK

Time taken: 24.06 seconds

The final query will sort and group the reviews by weekday: hive> SELECT weekday, COUNT(*) FROM u_data_new GROUP BY weekday:

Final output for the review counts by weekday should look like the following:

MapReduce Jobs Launched:

Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 2.39 sec

HDFS Write: 56 SUCCESS

OK

Total MapReduce CPU Time Spent: 2 seconds 390 msec

1 13278

2 14816

3 15426

4 13774

5 17964

6 12318

7 12424

Time taken: 22.645 seconds, Fetched: 7 cow(s)

As shown previously, you can remove the tables used in this example with the DROP TABLE command.

hive> -e 'drop table u_data_new!

$ hive -e 'drop table u_date"

**Using Apache Sqoop to Acquire Relational Data**

Sqoop is a tool designed to transfer data between Hadoop and relational databases You can use Sqoop to import data from a relational database management system (RDBMS) into the Hadoop Distributed File System (HDPS), 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, Postgres, M. and Oracle. In version 1 of Sqoop, data were accessed using connectors written for specific databases. Version 2 (in beta) does not support connectors or version 1 data transfer from a RDBMS directly to Hive or HBase, or data transfer from Hive or HBase to your RDBMS. Instead, version 2 offers more generalized ways to accomplish these tasks.

The remainder of this section provides a brief overview of how Sgoop works with Hadoop. In addition, a basic Sqoop example walk-through is demonstrated. To fully explore Sqoop, more information can found by consulting the Sqoop project website: http://sqoop.apache.org

**Apache Sqoop Import and Export Methods**

Figure 7.1 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. Note that each node doing the import must have access to the database.

The imported data are saved in an HDFS directory. Sqoop will use the database name for the directory, or the user can specify any alternative directory where the files should be populated. By default, these files contain comma-delimited fields, with new lines separating different records. You can easily override the format in which data are copied over by explicitly specifying the field separator and record terminator characters. Once placed in HDFS, the data are ready for processing

Figure 7.1

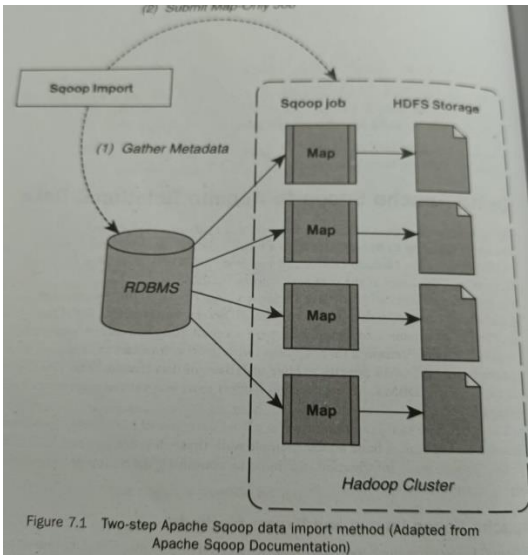



Figure 7.1 Two-step Apache Sqoop data import method (Adapted from Apache Sqoop Documentation)

Data export from the cluster works in a similar fashion. The export is done in two steps, as shown in Figure 7.2. 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 have access to the database.

**Apache Sqoop Version Changes**

Sqoop Version 1 uses specialized connectors to access external systems. These connectors are often optimized for various RDBMSS or for systems that do not support up data export method (Adapted from Apache Sqoop Documentation)

JDBC. Connectors are plug-in components based on Sqoop's extension framework and can be added to any existing Sqoop installation. Once a connector is installed, Sqoop can use it to efficiently transfer data between Hadoop and the external store supported by the connector. By default, Sqoop version 1 includes connectors for popular databases such as MySQL, PostgreSQL, Oracle, SQL Server, and DB2. It also supports direct transfer to and from the RDBMS to HBase or Hive.

In contrast, to streamline the Sqoop input methods, Sqoop version 2 no longer supports specialized connectors or direct import into HBase or Hive. All imports and exports are done through the JDBC interface. Table 7.2 summarizes the changes from version 1 to version 2. Due to these changes, any new development should be done with Sqoop version 2.

Table 7.2 Apache Sqoop Version Comparison

| 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. |

**Sqoop Example Walk-Through**

The following simple example illustrates use of Sqoop. It can be used as a foundation from which to explore the other capabilities offered by Apache Sqoop. The following steps will be performed:

Download Sqoop

2. Download and load sample MySQL data.

3. Add Sqoop user permissions for the local machine and cluster.

4. Import data from MySQL to HDFS.

5. Export data from HDFS to MySQL.

For this example, the following software environment is assumed. Other environments should work in a similar fashion.

OS: Linux

Platform : RHEL 6.6

• Hortonworks HDP 2.2 with Hadoop version: 2.6

A working installation of MySQL on the host

If you are using the pseudo-distributed installation from Chapter 2 or want to install Sqoop by hand, see the installation instructions on the Sqoop website: http://sqoop.apache.org. Sqoop is also installed as part of the Hortonworks HDP Sandbox.

Step 1: Download Sqoop and Load Sample MySQL Database

If you have not done so already, make sure Sqoop is installed on your cluster. Sqoop is needed on only a single node in your cluster. This Sqoop node will serve as an entry point for all connecting Sqoop clients.

Because the Sqoop node is a Hadoop MapReduce client, it requires both Hadoop installation and access to HDFS.

**$ yum install sqoop sqoop-metastore**

Using, Apache Sqoop to Acquire Relational Data an entry point for all connecting Sqoop clients. Because the Sqoop node is a Hadoop MapReduce client, it requires both a Hadoop installation and access to HDFS. To install Sqoop using the HDP distribution RPM files, simply enter:

For this example, we will use the world example database from the MySQL (http://dev.mysql.com/doc/world-setup/en/index.html). This database has three tables: .

- Country: information about countries of the world
- City: information about some of the cities in those countries .
- Country Language: languages spoken in each country

To get the database, use wget to download and then extract the file:

**$ wget http://downloads.mysql.com/docs/world_innodb.sql.gz**

**$ gunzip world_innodb.sql.gz**

Next, log into MySQL (assumes you have privileges to create a database) and import the desired database by following these steps:

**$ mysql -u root -p**

**mysql> CREATE DATABASE world;**

**mysql> USE world;**

**mysql> SOURCE world_innodb.sql;**

**mysql> SHOW TABLES;**

**| Tables_in_world |**

**| City |**

**| Country |**

**| Country Language |**

**3 rows in set (0.01 sec)**

The following MySQL command will let you see the table details (output omitted for clarity):

**mysql> SHOW CREATE TABLE Country;**

**mysql> SHOW CREATE TABLE City;**

**mysql> SHOW CREATE TABLE Country Language;**

**Step 2: Add Sqoop User Permissions for the Local Machine and Cluster**

In MySQL, add the following privileges for user sqoop to MySQL. Note that you must use both the local host name and the cluster subnet for Sqoop to work properly. Also, for the purposes of this example, the sqoop password is scoop.

**mysql> GRANT ALL PRIVILEGES ON world. To 'sqoop'@'limulus IDENTIFIED BY "sqoop'**

**mysql> GRANT ALL PRIVILEGES ON world. To 'sqoop'@'10.0.0.8 IDENTIFIED BY 'sqoop'**

**mysql> quit**

**Step 3: Import Data Using Sqoop**

As a test, we can use Sqoop to list databases in MySQL. The results appear after the warnings at the end of the output. Note the use of local host name (limulus) in the JDBC statement.

**$sqoop list-databases --connect jdbc:mysql://limulus/world --username sqoop**

**---password sqoop**

**Resultset.**

**information_schema**

**test**

**world**

In a similar fashion, you can use Sqoop to connect to MySQL and list the tables in the world database:

**$sqoop list-tables --connect jdbc: mysql://limulus/world --username sqoop:**

**--password sgoop**

resultset.

City

Country

Country Language

**To import data, we need to make a directory in HDFS**

**$hdfs dfs -mkdir sqoop-mysql-import**

The following command imports the Country table into HDFS. The option -table signifies the table to import,--target-dir is the directory created previously. and-m 1 tells Sqoop to use one map task to import the data.

$sqoop import --connect jdbeimysql://1imulus/world –username sqoop

--password sqoop-table Country -m 1--target-dir

/user/hdfs/sqoop-mysql-import/country