2 Hadoop

LEARNING OBJECTIVES

After reading this chapter, you will be able to:

- Understand the need for Hadoop.
- Learn the assumptions and goals of Hadoop.
- Learn about the Hadoop components.
- Understand and learn Hadoop Distributed File System.
- Learn how HDFS handles job processing.
- Learn about the MapReduce components.
- Learn the use of Hadoop ecosystem.
- Understand the limitation of Hadoop.

2.1 Introduction

In the late 1990s, search engines and indexes were created for helping people to find relevant information about the content searched. Open-source web search engine was invented to return results faster by distributing the data across different machines to process the tasks simultaneously. Web search engine used web crawlers to copy all the web pages visited; later the search engine processed and indexed the downloaded pages. Nutch was one such engine developed by Doug Cutting and Mike Cafarella. During the same period, Google also worked on the same concept – storing and processing data in a distributed environment so that more relevant search results could be fetched faster in an automated way.

Doug Cutting joined Yahoo in 2006 and he retained the name Nutch (a word for "meal" that Doug's son used as a toddler) for the web crawler portion of the project. He named the storage and distributed processing portion of the project as Hadoop (Hadoop was the name of Doug's son's toy elephant). In 2008, Yahoo released Hadoop as an open-source project. Today, Hadoop is developed as a framework by a non-profit organization Apache Software Foundation (ASF), a global community of software developers.

2.2 What is Hadoop?

Apache Hadoop is a framework that allows distributed processing of large datasets across clusters of commodity computers using a simple programming model. It is designed to scale-up from single servers to thousands of machines, each providing computation and storage. Rather than rely on hardware to deliver high-availability, the framework itself is designed to detect and handle failures at the application layer, thus delivering a highly available service on top of a cluster of computers, each of which may be prone to failures.

In short, Hadoop is an open-source software framework for storing and processing big data in a distributed way on large clusters of commodity hardware. Basically, it accomplishes the following two tasks:

1. Massive data storage.
2. Faster processing.

2.2.1 Why Hadoop?

Problems in data transfer made the organizations to think about an alternate way.

Example 1

1. The transfer speed is around 100 MB/s and a standard disk is 1 TB.
2. Time to read entire disk = 10,000 s or 3 h!
3. Increase in processing time may not be very helpful because of two reasons:
    - Network bandwidth is now more of a limiting factor.
    - Physical limits of processor chips are reached.

Example 2

If 100 TB of datasets are to be scanned on a 1000 node cluster, then in case of

1. remote storage with 10 Mbps bandwidth, it would take 165 min.
2. local storage with 50 Mbps, it will take 33 min.

So it is better to move computation rather than moving data.

Taking care of hardware failure cannot be made optional in Big Data Analytics but has to be made as a rule. In case of 1000 nodes, we need to consider say 4000 disks, 8000 core, 25 switches, 1000 NICs and 2000 RAMs (16 TB). Meantime between failures could be even less than a day since commodity hardware is used. There is a need for fault tolerant store to guarantee reasonable availability.

2.2.2 Hadoop Goals

The main goals of Hadoop are listed below:

1. **Scalable:** It can scale up from a single server to thousands of servers.
2. **Fault tolerance:** It is designed with very high degree of fault tolerance.

3. **Economical:** It uses commodity hardware instead of high-end hardware.
4. **Handle hardware failures:** The resiliency of these clusters comes from the software's ability to detect and handle failures at the application layer.

The Hadoop framework can store huge amounts of data by dividing the data into blocks and storing it across multiple computers, and computations can be run in parallel across multiple connected machines.

Hadoop gained its importance because of its ability to process huge amount of variety of data generated every day especially from automated sensors and social media using low-cost commodity hardware.

Since processing is done in batches, throughput is high but latency is low. Latency is the time (minutes/seconds or clock period) to perform some action or produce some result whereas throughput is the number of such actions executed or result produced per unit of time. The throughput of memory system is termed as memory bandwidth.

2.2.3 Hadoop Assumptions

Hadoop was developed with large clusters of computers in mind with the following assumptions:

1. Hardware will fail, since it considers a large cluster of computers.
2. Processing will be run in batches; so aims at high throughput as opposed to low latency.
3. Applications that run on Hadoop Distributed File System (HDFS) have large datasets typically from gigabytes to terabytes in size.
4. Portability is important.
5. Availability of high-aggregate data bandwidth and scale to hundreds of nodes in a single cluster.
6. Should support tens of millions of files in a single instance.
7. Applications need a write-once-read-many access model.

2.3 Core Hadoop Components

Hadoop consists of the following components:

1. **Hadoop Common:** This package provides file system and OS level abstractions. It contains libraries and utilities required by other Hadoop modules.
2. **Hadoop Distributed File System (HDFS):** HDFS is a distributed file system that provides a limited interface for managing the file system.
3. **Hadoop MapReduce:** MapReduce is the key algorithm that the Hadoop MapReduce engine uses to distribute work around a cluster.

4. **Hadoop *Yet Another Resource Negotiator* (YARN) (MapReduce 2.0):** It is a resource-management platform responsible for managing compute resources in clusters and using them for scheduling of users' applications.

2.3.1 Hadoop Common Package

This consists of necessary Java archive (JAR) files and scripts needed to start Hadoop. Hadoop requires Java Runtime Environment (JRE) 1.6 or higher version. The standard start-up and shut-down scripts need Secure Shell (SSH) to be setup between the nodes in the cluster.

HDFS (storage) and MapReduce (processing) are the two core components of Apache Hadoop. Both HDFS and MapReduce work in unison and they are co-deployed, such that there is a single cluster that provides the ability to move computation to the data. Thus, the storage system HDFS is not physically separate from a processing system MapReduce.

2.3.2 Hadoop Distributed File System (HDFS)

HDFS is a distributed file system that provides a limited interface for managing the file system to allow it to scale and provide high throughput. HDFS creates multiple replicas of each data block and distributes them on computers throughout a cluster to enable reliable and rapid access. When a file is loaded into HDFS, it is replicated and fragmented into "blocks" of data, which are stored across the cluster nodes; the cluster nodes are also called the DataNodes. The NameNode is responsible for storage and management of metadata, so that when MapReduce or another execution framework calls for the data, the NameNode informs it where the data that is needed resides. Figure 2.1 shows the NameNode and DataNode block replication in HDFS architecture.

1. HDFS creates multiple replicas of data blocks for reliability, placing them on the computer nodes around the cluster.
2. Hadoop's target is to run on clusters of the order of 10,000 nodes.
3. A file consists of many 64 MB blocks.

2.3.2.1 Main Components of HDFS

2.3.2.1.1 NameNode

NameNode is the master that contains the metadata. In general, it maintains the directories and files and manages the blocks which are present on the DataNode. The following are the functions of NameNode:

1. Manages namespace of the file system in memory.
2. Maintains "inode" information.
3. Maps *inode* to the list of blocks and locations.

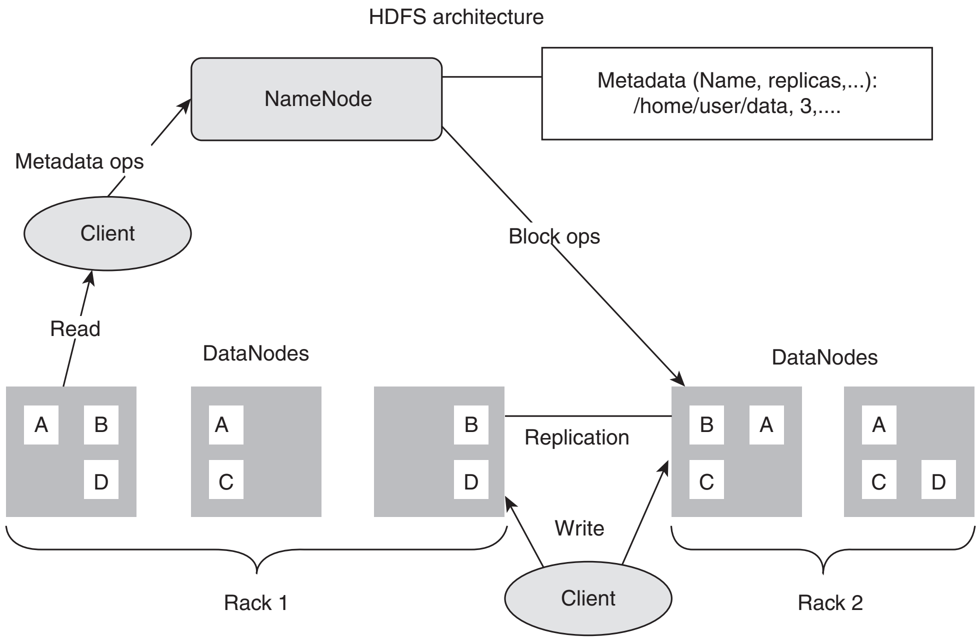

Figure 2.1 NameNode and DataNode block replication.

4. Takes care of authorization and authentication.
5. Creates checkpoints and logs the namespace changes.

So the NameNode maps DataNode to the list of blocks, monitors status (health) of DataNode and replicates the missing blocks.

2.3.2.1.2 DataNodes

DataNodes are the slaves which provide the actual storage and are deployed on each machine. They are responsible for processing read and write requests for the clients. The following are the other functions of DataNode:

1. Handles block storage on multiple volumes and also maintain block integrity.
2. Periodically sends heartbeats and also the block reports to NameNode.

Figure 2.2 shows how HDFS handles job processing requests from the user in the form of Sequence Diagram. User copies the input files into DFS and submits the job to the client. Client gets the input file information from DFS, creates splits and uploads the job information to DFS. JobTracker puts ready job into the internal queue. JobScheduler picks job from the queue and initializes the job by creating job object. JobTracker creates a list of tasks and assigns one map task for each input split.

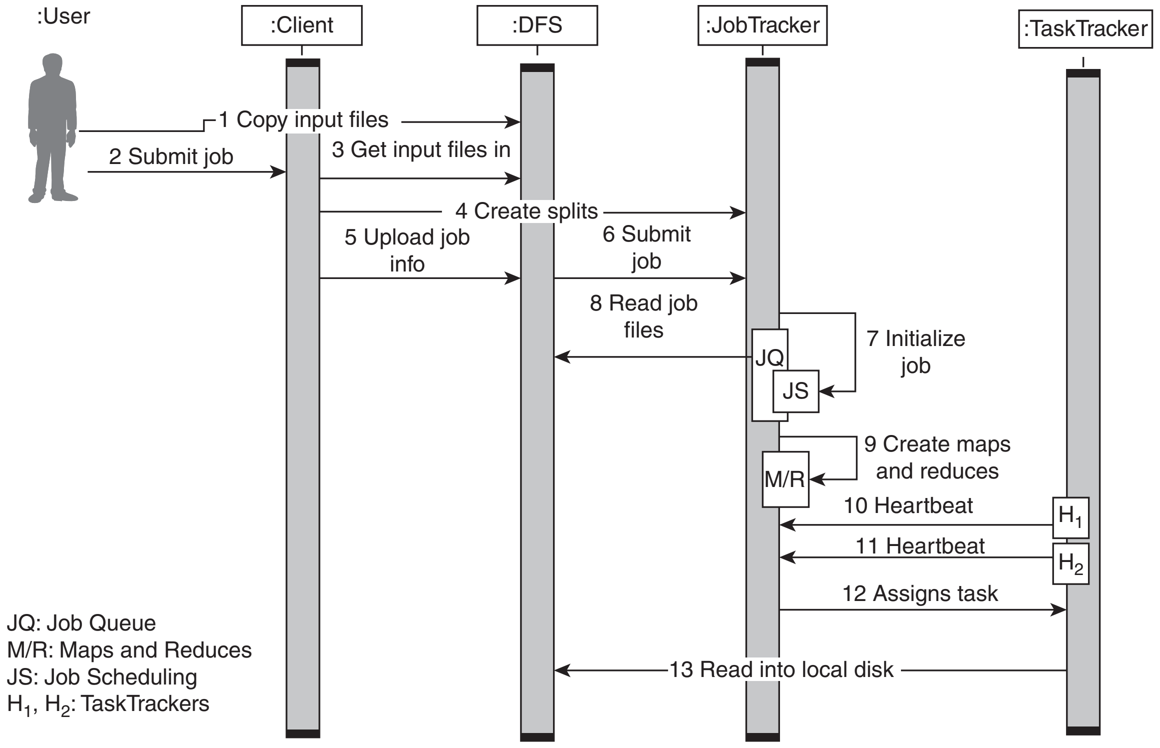

Figure 2.2 Sequence diagram depicting job processing requests from the user.

TaskTrackers send heartbeat to JobTracker to indicate if ready to run new tasks. JobTracker chooses task from first job in priority-queue and assigns it to the TaskTracker.

Secondary NameNode is responsible for performing periodic checkpoints. These are used to restart the NameNode in case of failure. MapReduce can then process the data where it is located.

2.3.3 MapReduce

The MapReduce algorithm aids in parallel processing and basically comprises two sequential phases: map and reduce.

1. In the map phase, a set of key–value pairs forms the input and over each key–value pair, the desired function is executed so as to generate a set of intermediate key–value pairs.
2. In the reduce phase, the intermediate key–value pairs are grouped by key and the values are combined together according to the reduce algorithm provided by the user. Sometimes no reduce phase is required, given the type of operation coded by the user.

MapReduce processes are divided between two applications, JobTracker and TaskTracker at the cluster level. JobTracker is responsible for scheduling job runs and managing computational resources across the cluster; hence it runs on only one node of the cluster. Each MapReduce job is split into a number of tasks which are assigned to the various TaskTrackers depending on which data is stored on that node. So TaskTracker runs on every slave node in the cluster. JobTracker oversees the progress of each TaskTracker as they complete their individual tasks.

The following points summarize the above discussion:

1. Hadoop implements Google's MapReduce, using HDFS.
2. MapReduce divides applications into many small blocks of work.
3. Performs Sort/merge-based distributed computing.
4. Follows functional style programming and so naturally is parallelizable across a large cluster of workstations or PCs.

In the MapReduce paradigm, each job has a user-defined map phase followed by a user-defined reduce phase as follows:

1. Map phase is a parallel, share-nothing processing of input.
2. In the reduce phase, the output of the map phase is aggregated.

HDFS is the storage system for both input and output of the MapReduce jobs.

2.3.3.1 Main Components of MapReduce

The main components of MapReduce are listed below:

1. **JobTrackers:** JobTracker is the master which manages the jobs and resources in the cluster. The JobTracker tries to schedule each map on the TaskTracker which is running on the same DataNode as the underlying block.
2. **TaskTrackers:** TaskTrackers are slaves which are deployed on each machine in the cluster. They are responsible for running the map and reduce tasks as instructed by the JobTracker.
3. **JobHistoryServer:** JobHistoryServer is a daemon that saves historical information about completed tasks/applications.

Note: If the map phase has M fragments and the reduce phase has R fragments, then M and R should be much larger than the number of worker machines. R is often decided by the users, because the output of each reduce task ends up in a separate output file. Typically (at Google), $M = 2{,}00{,}000$ and $R = 5000$, using 2000 worker machines.

2.3.4 Yet Another Resource Negotiator (YARN)

YARN addresses problems with MapReduce 1.0s architecture, specifically the one faced by JobTracker service.

Hadoop generally has up to tens of thousands of nodes in the cluster. Obviously, MapReduce 1.0 had issues with scalability, memory usage, synchronization, and also Single Point of Failure (SPOF) issues. In effect, YARN became another core component of Apache Hadoop.

2.3.4.1 What does YARN do?

It splits up the two major functionalities "resource management" and "job scheduling and monitoring" of the JobTracker into two separate daemons. One acts as a "global Resource Manager (RM)" and the other as a "ApplicationMaster (AM)" per application. Thus, instead of having a single node to handle both scheduling and resource management for the entire cluster, YARN distributes this responsibility across the cluster.

The RM and the NodeManager manage the applications in a distributed manner. The RM is the one that arbitrates resources among all the applications in the system. The per-application AM negotiates resources from the RM and works with the NodeManager(s) to execute and monitor the component tasks.

1. The RM has a scheduler that takes into account constraints such as queue capacities, user-limits, etc. before allocating resources to the various running applications.
2. The scheduler performs its scheduling function based on the resource requirements of the applications.
3. The NodeManager is responsible for launching the applications' containers. It monitors the application's resource usage (CPU, memory, disk, network) and reports the information to the RM.
4. Each AM runs as a normal container. It has the responsibility of negotiating appropriate resource containers from the scheduler, tracking their status and monitoring their progress.

2.4 Hadoop Ecosystem

Apart from HDFS and MapReduce, the other components of Hadoop ecosystem are shown in Fig. 2.3. The main ecosystems components of Hadoop architecture are as follows:

1. **Apache HBase:** Columnar (Non-relational) database.
2. **Apache Hive:** Data access and query.

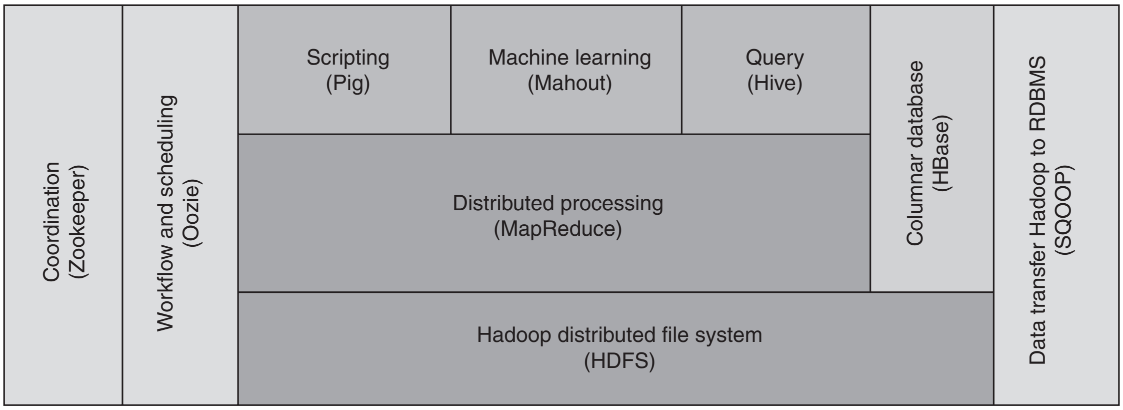

Figure 2.3 Hadoop ecosystem.

3. **Apache HCatalog:** Metadata services.
4. **Apache Pig:** Scripting platform.
5. **Apache Mahout:** Machine learning libraries for Data Mining.
6. **Apache Oozie:** Workflow and scheduling services.
7. **Apache ZooKeeper:** Cluster coordination.
8. **Apache Sqoop:** Data integration services.

These components are discussed in detail in the following subsections.

2.4.1 HBase

HBase "is an open-source, distributed, versioned, column-oriented store" that sits on top of HDFS. HBase is based on Google's Bigtable. HBase is based on columns rather than rows. This essentially increases the speed of execution of operations if they are need to be performed on similar values across massive datasets; for example, read/write operations that involve all rows but only a small subset of all columns. HBase does not provide its own query or scripting language, but is accessible through Java, Thrift and REST APIs.

2.4.2 Hive

Hive provides a warehouse structure for other Hadoop input sources and SQL-like access for data in HDFS. Hive's query language, HiveQL, compiles to MapReduce and also allows user-defined functions (UDFs). Hive's data model is based primarily on three related data structures: tables, partitions and buckets. Tables correspond to HDFS directories that are divided into partitions, which in turn can be divided into buckets.

2.4.3 HCatalog

HCatalog is a metadata and table storage management service for HDFS. HCatalog's goal is to simplify the user's interaction with HDFS data and enable data sharing between tools and execution platforms.

2.4.4 Pig

Pig is a run-time environment that allows users to execute MapReduce on a Hadoop cluster. Pig Latin is a high-level scripting language on Pig platform. Like HiveQL in Hive, Pig Latin is a higher-level language that compiles to MapReduce.

Pig is more flexible with respect to possible data format than Hive due to its data model. Pig's data model is similar to the relational data model, but here tuples can be nested. For example, a table of tuples can have a table in the third field of each tuple. In Pig, tables are called bags. Pig also has a "map" data type, which is useful in representing semi-structured data such as JSON or XML."

2.4.5 Sqoop

Sqoop ("SQL-to-Hadoop") is a tool which transfers data in both ways between relational systems and HDFS or other Hadoop data stores such as Hive or HBase. Sqoop can be used to import data from external structured databases into HDFS or any other related systems such as Hive and HBase. On the other hand, Sqoop can also be used to extract data from Hadoop and export it to external structured databases such as relational databases and enterprise data warehouses.

2.4.6 Oozie

Oozie is a job coordinator and workflow manager for jobs executed in Hadoop. It is integrated with the rest of the Apache Hadoop stack. It supports several types of Hadoop jobs, such as Java map-reduce, Streaming map-reduce, Pig, Hive and Sqoop as well as system-specific jobs such as Java programs and shell scripts. An Oozie workflow is a collection of actions and Hadoop jobs arranged in a Directed Acyclic Graph (DAG), since tasks are executed in a sequence and also are subject to certain constraints.

2.4.7 Mahout

Mahout is a scalable machine-learning and data-mining library. There are currently following four main groups of algorithms in Mahout:

1. Recommendations/Collective filtering.
2. Classification/Categorization.
3. Clustering.
4. Frequent item-set mining/Parallel frequent pattern mining.

Mahout is not simply a collection of pre-existing data mining algorithms. Many machine learning algorithms are non-scalable; that is, given the types of operations they perform, they cannot be executed as a set of parallel processes. But algorithms in the Mahout library can be executed in a distributed fashion, and have been written for MapReduce

2.4.8 ZooKeeper

ZooKeeper is a distributed service with master and slave nodes for storing and maintaining configuration information, naming, providing distributed synchronization and providing group services in memory on ZooKeeper servers. ZooKeeper allows distributed processes to coordinate with each other through a shared hierarchical name space of data registers called znodes. HBase depends on ZooKeeper and runs a ZooKeeper instance by default.

2.5 Physical Architecture

Organizations tend to store more and more data in cloud environments, since clouds offer business users scalable resources on demand. Combining processor-based servers and storage, along with networking resources used in cloud environments, with big data processing tools such as Apache Hadoop software, provides the high-performance computing power needed to analyze vast amounts of data efficiently and cost-effectively. The machine configuration for storage and computing servers typically are 32 GB memory, four core processors and 200–320 GB hard disk. Running Hadoop in virtualized environments continues to evolve and mature with initiatives from open-source software projects. Figure 2.4 shows cloud computing infrastructure required for Big Data Analytics.

Every Hadoop-compatible file system should provide location awareness for effective scheduling of work: the name of the rack or the network switch where a worker node is. Hadoop application uses this information to find the data node and run the task. HDFS replicates data to keep different copies of the data on different racks. The goal is to reduce the impact of a rack power or switch failure (see Fig. 2.5).

A small Hadoop cluster includes a single master and multiple worker nodes. The master node consists of a JobTracker, TaskTracker, NameNode and DataNode. A slave or *worker node* acts as both a DataNode and TaskTracker, though it is possible to have data-only worker nodes and compute-only worker nodes.

In case of a larger cluster, the HDFS is managed through a dedicated NameNode server to host the file system index, and a secondary NameNode that can generate snapshots of the NameNode's memory structures, thus reducing the impact of loss of data. Similarly, a standalone JobTracker server can manage job scheduling. In clusters where the Hadoop MapReduce engine is deployed against an alternate file system, the NameNode, secondary NameNode and DataNode architecture of HDFS are replaced by the file-system-specific equivalents.

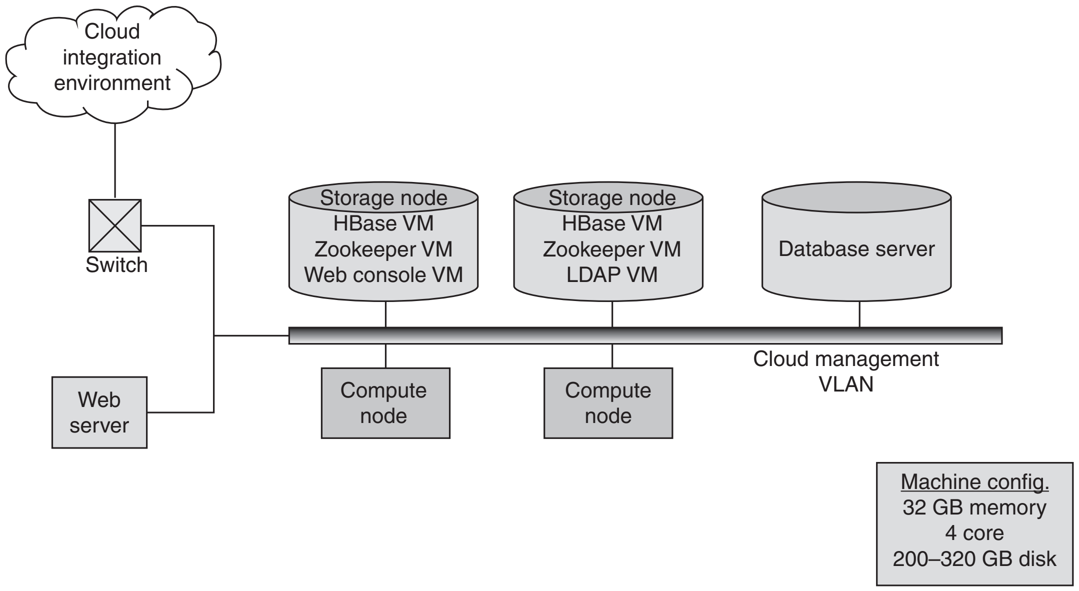

Figure 2.4 Cloud computing infrastructure to support Big Data Analytics.

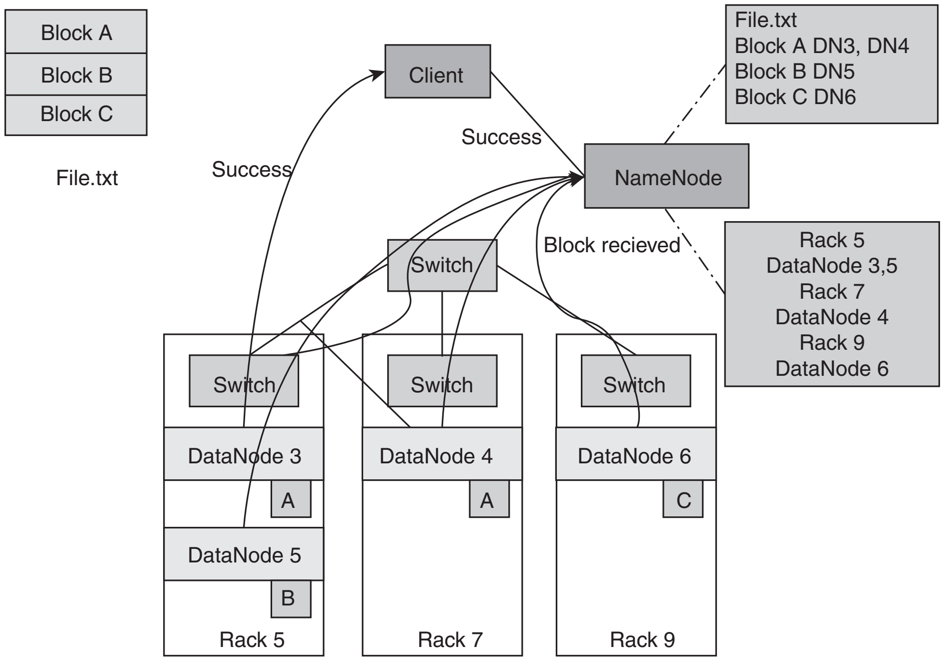

Figure 2.5 Hadoop-compatible file system provides location awareness.

HDFS stores large files in the range of gigabytes to terabytes across multiple machines. It achieves reliability by replicating the data across multiple hosts. Data is replicated on three nodes: two on the same rack and one on a different rack. Data nodes can communicate with each other to re-balance data and to move copies around. HDFS is not fully POSIX-compliant to achieve increased performance for data throughput and support for non-POSIX operations such as Append.

The HDFS file system includes a so-called secondary NameNode, which regularly connects with the primary NameNode and builds snapshots of the primary NameNode directory information, which the system then saves to local or remote directories. These check-pointed images can be used to restart a failed primary NameNode without having to replay the entire journal of file-system actions, then to edit the log to create an up-to-date directory structure.

An advantage of using HDFS is data awareness between the JobTracker and TaskTracker. The Job-Tracker schedules map or reduce jobs to TaskTrackers with an awareness of the data location. For example, if node *A* contains data (*x, y, z*) and node *B* contains data (*a, b, c*), the JobTracker schedules node *B* to perform map or reduce tasks on (*a,b,c*) and node *A* would be scheduled to perform map or reduce tasks on (*x,y,z*). This reduces the amount of traffic that goes over the network and prevents unnecessary data transfer.

When Hadoop is used with other file system, this advantage is not always available. This can have a significant impact on job-completion times, which has been demonstrated when running data-intensive jobs. HDFS was designed for mostly immutable files and may not be suitable for systems requiring concurrent write-operations.

2.6 Hadoop Limitations

HDFS cannot be mounted directly by an existing operating system. Getting data into and out of the HDFS file system can be inconvenient. In Linux and other Unix systems, a file system in Userspace (FUSE) virtual file system is developed to address this problem.

File access can be achieved through the native Java API, to generate a client in the language of the users' choice (C++, Java, Python, PHP, Ruby, etc.), in the command-line interface or browsed through the HDFS-UI web app over HTTP.

2.6.1 Security Concerns

Hadoop security model is disabled by default due to sheer complexity. Whoever's managing the platform should know how to enable it; else data could be at huge risk. Hadoop does not provide encryption at the storage and network levels, which is a major reason for the government agencies and others not to prefer to keep their data in Hadoop framework.

2.6.2 Vulnerable By Nature

Hadoop framework is written almost entirely in Java, one of the most widely used programming languages by cyber-criminals. For this reason, several experts have suggested dumping it in favor of safer, more efficient alternatives.

2.6.3 Not Fit for Small Data

While big data is not exclusively made for big businesses, not all big data platforms are suitable for handling small files. Due to its high capacity design, the HDFS lacks the ability to efficiently support the random reading of small files. As a result, it is not recommended for organizations with small quantities of data.

2.6.4 Potential Stability Issues

Hadoop is an open-source platform necessarily created by the contributions of many developers who continue to work on the project. While improvements are constantly being made, like all open-source software, Hadoop has stability issues. To avoid these issues, organizations are strongly recommended to make sure they are running the latest stable version or run it under a third-party vendor equipped to handle such problems.

2.6.5 General Limitations

Google mentions in its article that Hadoop may not be the only answer for big data. Google has its own Cloud Dataflow as a possible solution. The main point the article stresses is that companies could be missing out on many other benefits by using Hadoop alone.

Summary

- MapReduce brings compute to the data in contrast to traditional distributed system, which brings data to the compute resources.
- Hadoop stores data in a replicated and distributed way on HDFS. HDFS stores files in chunks which are physically stored on multiple compute nodes.
- MapReduce is ideal for operating on very large, unstructured datasets when aggregation across large datasets is required and this is accomplished by using the power of Reducers.
- Hadoop jobs go through a map stage and a reduce stage where
  - the mapper transforms the input data into key–value pairs where multiple values for the same key may occur.
  - the reducer transforms all of the key–value pairs sharing a common key into a single key–value.
- There are specialized services that form the Hadoop ecosystem to complement the Hadoop modules. These are HBase, Hive, Pig, Sqoop, Mahout, Oozie, Spark, Ambari to name a few.

Review Questions

1. What is Hadoop?
2. Why do we need Hadoop?
3. What are core components of Hadoop framework?
4. Give a brief overview of Hadoop.
5. What is the basic difference between traditional RDBMS and Hadoop?
6. What is structured, semi-structured and unstructured data? Give an example and explain.
7. What is HDFS and what are its features?
8. What is Fault Tolerance?
9. If replication causes data redundancy, then why is it pursued in HDFS?
10. What is a NameNode?
11. What is a DataNode?
12. Why is HDFS more suited for applications having large datasets and not when there are small files?
13. What is a JobTracker?
14. What is a TaskTracker?
15. What is a "block" in HDFS?
16. What are the benefits of block transfer?
17. What is a secondary NameNode? Does it substitute a NameNode?
18. What is MapReduce? Explain how do "map" and "reduce" work?
19. What sorts of actions does the JobTracker process perform?

Laboratory Exercise

A. Instructions to learners on how to run wordcount program on Cloudera

To start working with Hadoop for beginners, the best choice is to download ClouderaVM from their official website (and it is free).

Download Link:
http://www.cloudera.com/content/cloudera/en/downloads/quickstart_vms/cdh-5-4-x.html

Pre-requisite: Install VM Ware Player or Oracle Virtual Box.
For the above version of Cloudera we need virtual box.

INSTALLING AND OPENNING Cloudera in VIRTUAL BOX

STEP1: EXTRACT the downloaded zip file in the same folder or in home directory.

STEP2: Open virtualbox. Then

- Click New button which is at toolbar menu.
- A new window will open. Type name in the Name field, for example, "Cloudera". Next in Type field select the type as "Linux". In the version field select "Other Linux(64 bit)".

- Click on Next Button. A new window will open. Select the RAM size. Click on Next Button.
- Here you have three options, out of which select “use an existing virtual Hard drive file”. Browse your Cloudera folder for file with .vmdk extension. Select that file and press ENTER.

Now as we have successfully created vm we can start Cloudera. So start it by clicking on start button. It will take some time to open. Wait for 2 to 3 minutes. Here the operating system is CentOS.

Once the system gets loaded we will start with the simple program called “wordcount” using MapReduce function which is a simple “hello world” kind of program for Hadoop.

STEPS FOR RUNNING WORDCOUNT PROGRAM:

1. OPEN the Terminal. Install a package “wget” by typing the following command:

   ```
   $ sudo yum -y install wget
   ```
2. Make directory:

   ```
   $ mkdir temp
   ```
3. Goto temp:

   ```
   $cd temp
   ```
4. Create a file with some content in it:

   ```
   $ echo “This is SPIT and you can call me Sushil. I am good at statistical modeling and data analysis” > wordcount.txt
   ```
5. Make input directory in the HDFS system:

   ```
   $ hdfsdfs -mkdir /user/cloudera/input
   ```
6. Copy file from local directory to HDFS file system:

   ```
   $ hdfsdfs -put /home/cloudera/temp/wordcount.txt /user/cloudera/input/
   ```
7. To check if your file is successfully copied or not, use:

   ```
   $ hdfsdfs -ls /user/cloudera/input/
   ```
8. To check hadoop-mapreduce-examples, use:

   ```
   $ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar
   ```
9. Run the wordcount program by typing the following command:

   ```
   $ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar wordcount /user/cloudera/input/wordcount.txt /user/cloudera/output
   ```

   Note: The output will be generated in the output directory in HDFS file system and stored in part file “part-r-00000”.

10. Check output directory:

```
$hdfsdfs -ls /user/cloudera/output
```

11. Open the part file to check the output:

```
$ hdfsdfs -cat /user/cloudera/output/part-r-00000
```

Note: The above command will display the file content on the terminal. If you want to open it in the text editor then we need to copy it to our local file system. To do so, use the following command:

```
$ hdfsdfs -copyToLocal /user/cloudera/output/part-r-00000 /home/cloudera
```

B. Guidelines to Install Hadoop 2.5.2 on top of Ubuntu 14.04 and write WordCount Program in Java using MapReduce structure and test it over HDFS

Pre-requisite: Apache, JAVA, ssh packages must be installed. If not then follow the following steps.

B1. Steps for Installing Above Packages

1. Before installing above packages, Create a new user to run the Hadoop (hduser or huser) and give it sudo rights:

- Create group name hadoop:
  ```
  $ sudoaddgrouphadoop
  ```
- To create user and add it in group named Hadoop use
  ```
  $ sudoadduser --ingrouphadoophduser
  ```
- To give sudo rights to hduser use
  ```
  $ sudoadduserhdusersudo
  ```
- To switch user to hduser use
  ```
  $ suhduser
  ```

2. Install the following software:

```
# Update the source list

$ sudo apt-get update
```

2.1 Apache

```
$ sudo apt-get install apache2

# The OpenJDK project is the default version of Java.

# It is provided from a supported Ubuntu repository.
```

2.2 Java

```
$ sudo apt-get install default-jdk
$ java -version
```

2.3 Installing SSH: ssh has two main components, namely,

- ssh: The command we use to connect to remote machines – the client.
- sshd: The daemon that is running on the server and allows clients to connect to the server.

The ssh is pre-enabled on Linux, but in order to start sshd daemon, we need to install ssh first. Use the following command to do so:

```
$ sudo apt-get install ssh
```

This will install ssh on our machine. Verify if ssh is installed properly with which command:

```
$ whichssh
o/p:usr/bin/ssh

$ whichsshd
o/p:/usr/sbin/sshd
```

Create and Setup SSH Certificates: Hadoop requires SSH access to manage its nodes, that is, remote machines plus our local machine. For our single-node setup of Hadoop, we therefore need to configure SSH access to local host. So, we need to have SSH up and running on our machine and configured to allow SSH public key authentication. Hadoop uses SSH (to access its nodes) which would normally require the user to enter a password. However, this requirement can be eliminated by creating and setting up SSH certificates using the following commands. If asked for a filename just leave it blank and press the enter key to continue.

```
$ ssh-keygen -t rsa -P “”
```

Note: After typing the above command just press Enter two times.

```
$ cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
```

The second command adds the newly created key to the list of authorized keys so that Hadoop can use ssh without prompting for a password.

We can check if ssh works using the following command:

```
$ ssh localhost

o/p:
```

```
The authenticity of host ‘localhost (127.0.0.1)’ cannot be established.

ECDSA key fingerprint is e1:8b:a0:a5:75:ef:f4:b4:5e:a9:ed:be:64:be:5c:2f.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added ‘localhost’ (ECDSA) to the list of known hosts.

Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-40-generic x86_64)
```

B2. Installing Hadoop

1. Download and extract the hadoop-2.5.2.tar.gz to the Downloads directory from the link given below:
   https://archive.apache.org/dist/hadoop/core/hadoop-2.5.2/
2. To switch user to hduser use
   ```
   $ sudosuhduser
   ```
3. To move hadoop to/usr/local/Hadoop use
   ```
   $ sudo mv /home/admin/Downloads/* /usr/local/hadoop
   ```
4. To change the access rights use
   ```
   sudochown -R hduser:hadoop /usr/local/hadoop
   ```

B3. Setup Configuration Files

The following files will have to be modified to complete the Hadoop setup:

```
~/.bashrc
/usr/local/hadoop/etc/hadoop/hadoop-env.sh
/usr/local/hadoop/etc/hadoop/core-site.xml
/usr/local/hadoop/etc/hadoop/mapred-site.xml.template
/usr/local/hadoop/etc/hadoop/hdfs-site.xml
```

1. ~/.bashrc: Before editing the .bashrc file in our home directory, we need to find the path where Java has been installed to set the JAVA_HOME environment variable using the following command:

   ```
   $ update-alternatives --config java
   ```

   Now we can append the following to the end of ~/.bashrc:

   ```
   $ vi ~/.bashrc

   #HADOOP VARIABLES START
   export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
   export HADOOP_INSTALL=/usr/local/hadoop
   export PATH=$PATH:$HADOOP_INSTALL/bin
   export PATH=$PATH:$HADOOP_INSTALL/sbin
   export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
   export HADOOP_COMMON_HOME=$HADOOP_INSTALL
   export HADOOP_HDFS_HOME=$HADOOP_INSTALL
   export YARN_HOME=$HADOOP_INSTALL
   export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native
   export HADOOP_OPTS=”-Djava.library.path=$HADOOP_INSTALL/lib”
   #HADOOP VARIABLES END
   ```

```
$ source ~/.bashrc
```

Note that the JAVA_HOME should be set as the path just before the '.../bin/':

```
$ javac -version
$ whichjavac
$ readlink -f /usr/bin/javac
```

2. /usr/local/hadoop/etc/hadoop/hadoop-env.sh: We need to set JAVA_HOME by modifying hadoop-env.sh file.

   ```
   $ vi /usr/local/hadoop/etc/hadoop/hadoop-env.sh
   ```

   Add the following configuration:

   ```
   export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
   ```

3. /usr/local/hadoop/etc/hadoop/core-site.xml: This file contains configuration properties that Hadoop uses when starting up.This file can be used to override the default settings that Hadoop starts with.

   ```
   $ sudomkdir -p /app/hadoop/tmp
   ```

   ```
   $ sudochownhduser:hadoop /app/hadoop/tmp
   ```

   Open the file and enter the following in between the <configuration></configuration> tag:

   ```
   $ vi /usr/local/hadoop/etc/hadoop/core-site.xml
   ```

   ```
   <configuration>
   <property>
   <name>hadoop.tmp.dir</name>
   <value>/app/hadoop/tmp</value>
   <description>A base for other temporary directories.</description>
   </property>

   <property>
   <name>fs.default.name</name>
   <value>hdfs://localhost:54310</value>
   <description>The name of the default file system. A URI whose
   scheme and authority determine the FileSystem implementation. The
   uri's scheme determines the config property (fs.SCHEME.impl) naming
   the FileSystem implementation class. The uri's authority is used to
   determine the host, port, etc. for a filesystem.</description>
   </property>
   </configuration>
   ```

4. /usr/local/hadoop/etc/hadoop/mapred-site.xml: By default, the /usr/local/hadoop/etc/hadoop/ folder contains /usr/local/hadoop/etc/hadoop/mapred-site.xml.template file which has to be renamed/copied with the name mapred-site.xml.

   ```
   $ cp /usr/local/hadoop/etc/hadoop/mapred-site.xml.template /usr/local/hadoop/etc/hadoop/mapred-site.xml
   ```

   The mapred-site.xml file is used to specify which framework is being used for MapReduce.

   We need to enter the following content in between the <configuration></configuration> tag:

   ```
   <configuration>
   <property>
   <name>mapred.job.tracker</name>
   <value>localhost:54311</value>
   <description>The host and port that the MapReduce job tracker runs
   at. If "local", then jobs are run in-process as a single map
   and reduce task.
   </description>
   </property>
   </configuration>
   ```

5. /usr/local/hadoop/etc/hadoop/hdfs-site.xml: This file needs to be configured for each host in the cluster that is being used. It is used to specify the directories which will be used as the NameNode and the DataNode on that host. Before editing this file, we need to create two directories which will contain the NameNode and the DataNode for this Hadoop installation. This can be done using the following commands:

   ```
   $ sudomkdir -p /usr/local/hadoop_store/hdfs/namenode

   $ sudomkdir -p /usr/local/hadoop_store/hdfs/datanode

   $ sudochown -R hduser:hadoop /usr/local/hadoop_store
   ```

   Open the file and enter the following content in between the <configuration></configuration> tag:

   ```
   $ vi /usr/local/hadoop/etc/hadoop/hdfs-site.xml

   <configuration>
   <property>
   <name>dfs.replication</name>
   <value>1</value>
   <description>Default block replication.
   The actual number of replications can be specified when the file is created.
   The default is used if replication is not specified in create time.
   ```

```
</description>
</property>
<property>
<name>dfs.namenode.name.dir</name>
<value>file:/usr/local/hadoop_store/hdfs/namenode</value>
</property>
<property>
<name>dfs.datanode.data.dir</name>
<value>file:/usr/local/hadoop_store/hdfs/datanode</value>
</property>
</configuration>
```

B4. Format the New Hadoop File System: Now, the Hadoop file system needs to be formatted so that we can start to use it. The format command should be issued with write permission since it creates current directory:

under /usr/local/hadoop_store/hdfs/namenode folder:

```
$ hadoopnamenode -format
```

Note that hadoopnamenode -format command should be executed once before we start using Hadoop. If this command is executed again after Hadoop has been used, it will destroy all the data on the Hadoop file system.

Starting Hadoop: Now it is time to start the newly installed single node cluster. We can use start-all.sh or (start-dfs.sh and start-yarn.sh)

```
$ cd /usr/local/hadoop/sbin
```

```
$ start-all.sh
```

We can check if it is really up and running using the following command:

```
$ jps
```

o/p:

```
9026 NodeManager
7348 NameNode
9766 Jps
8887 ResourceManager
7507 DataNode
```

The output means that we now have a functional instance of Hadoop running on our VPS (Virtual private server).

```
$ netstat -plten | grep java
```

Stopping Hadoop

```
$ cd /usr/local/hadoop/sbin
```

```
$ stop-all.sh
```

B5. Running Wordcount on Hadoop 2.5.2: Wordcount is the hello_world program for MapReduce. Code for wordcount program is:

```
packageorg.myorg;
importjava.io.IOException;
importjava.util.*;
importorg.apache.hadoop.fs.Path;
importorg.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
importorg.apache.hadoop.util.*;
importorg.apache.hadoop.mapreduce.Mapper;
importorg.apache.hadoop.mapreduce.Reducer;
importorg.apache.hadoop.conf.Configuration;
importorg.apache.hadoop.conf.Configured;
importorg.apache.hadoop.mapreduce.Job;
importorg.apache.hadoop.mapreduce.lib.input.FileInputFormat;
importorg.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
importorg.apache.hadoop.mapred.JobConf;
public class myWordCount {
public static class Map extends Mapper
<LongWritable, Text, Text, IntWritable> {
private final static IntWritable one = new IntWritable(1);
private Text word = new Text();
public void map(LongWritable key, Text value, Context context) throws IOException, Inter-
ruptedException {
            String line = value.toString();
StringTokenizer tokenizer = new StringTokenizer(line);
while (tokenizer.hasMoreTokens()) {
word.set(tokenizer.nextToken());
context.write(word, one);
            }
        }
   }
public static class Reduce extends Reducer
<Text, IntWritable, Text, IntWritable> {
public void reduce(Text key, Iterator<IntWritable> values, Context context) throws IOExcep-
tion, InterruptedException {
```

```
int sum = 0;
while (values.hasNext()) {
sum += values.next().get();
            }
context.write(key, new IntWritable(sum));
        }
   }
public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
conf.set("mapreduce.job.queuename", "apg_p7");
System.out.println("This is a new version");
        Job job = new Job(conf);
job.setJarByClass(myWordCount.class);
job.setJobName("myWordCount");
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
job.setMapperClass(myWordCount.Map.class);
job.setCombinerClass(myWordCount.Reduce.class);
job.setReducerClass(myWordCount.Reduce.class);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
job.waitForCompletion(true);
   }
}
```

Note: Copy the above code and save it with .java extension. To run the program under MapReduce, the following steps needs to be done:

1. Put source code under this location

   /project/src/org/myorg/myWordCount.java

2. Compile java code

   ```
   $ mkdir /project/class;
   $ cd /project;
   $ javac -classpath `yarn classpath` -d ./class ./src/org/myorg/*.java
   ```

3. Create manifest.txt file

   ```
   $ cd project/class;
   $ sudo vim manifest.txt;
   ```
   The content of manifest.txt is
   ```
   Main-Class: org.myorg.myWordCount
   ```
   Leave an empty line at the end of manifest.txt

4. To Generate jar file

```
$ jar -cvmf manifest.txt myWordCount.jar org
```

5. Put input data on HDFS

```
$ mkdir input
$ echo "hadoop is fast and hadoop is amazing, hadoop is new Technology for Big Data Process-
ing" > input/file1
$ hadoop fs -put input /user/hadoop
```

6. Run the program

```
hadoop jar myWordCount.jar /user/hadoop/input /user/hadoop/output
```

3 What is NoSQL?

LEARNING OBJECTIVES

After reading this chapter, you will be able to:

- Understand NoSQL business drivers.
- Learn the desirable features of NoSQL that drive business.
- Learn the need for NoSQL through case studies.
- Learn NoSQL data architectural pattern.
- Learn the variations in NoSQL architectural pattern.
- Learn how NoSQL is used to manage big data.
- Learn how NoSQL system handles big data problems.

3.1 What is NoSQL?

NoSQL is database management system that provides mechanism for storage and retrieval of massive amount of unstructured data in a distributed environment on virtual servers with the focus to provide high scalability, performance, availability and agility.

In other words, NoSQL was developed in response to a large volume of data stored about users, objects and products that need to be frequently accessed and processed. Relational databases are not designed to scale and change easily to cope up with the needs of the modern industry. Also they do not take advantage of the cheap storage and processing power available today by using commodity hardware.

NoSQL database is also referred as **N**ot **o**nly **SQL**. Most NoSQL systems are entirely non-relational; they do not have fixed schemas or JOIN operations. Instead they use objects, key-value pairs, or tuples.

Some of the NoSQL implementations are SimpleDB, Google BigTable, Apache Hadoop, MapReduce and MemcacheDB. There are approximately 150 NoSQL databases available in the market. Companies that largely use NoSQL are NetFlix, LinkedIn and Twitter for analyzing their social network data.

In short:

1. NoSQL is next generation database which is completely different from the traditional database.
2. NoSQL stands for Not only SQL. SQL as well as other query languages can be used with NoSQL databases.

3. NoSQL is non-relational database, and it is schema-free.
4. NoSQL is free of JOINs.
5. NoSQL uses distributed architecture and works on multiple processors to give high performance.
6. NoSQL databases are horizontally scalable.
7. Many open-source NoSQL databases are available.
8. Data file can be easily replicated.
9. NoSQL uses simple API.
10. NoSQL can manage huge amount of data.
11. NoSQL can be implemented on commodity hardware which has separate RAM and disk (shared-nothing concept).

3.1.1 Why NoSQL?

The reality is that a traditional database model does not offer the best solution for all scenarios in applications. A relational database product can cater to a more predictable, structured data. NoSQL is required because today's industry needs a very agile system that can process unstructured and unpredictable data dynamically. NoSQL is known for its high performance with high availability, rich query language, and easy scalability which fits the need. NoSQL may not provide atomicity, consistency, isolation, durability (ACID) properties but guarantees eventual consistency, basically available, soft state (BASE), by having a distributed and fault-tolerant architecture.

3.1.2 CAP Theorem

Consistency, **A**vailability, **P**artition tolerance (**CAP) theorem**, also called as **Brewer's theorem**, states that it is not possible for a distributed system to provide all three of the following guarantees simultaneously:

1. Consistency guarantees all storage and their replicated nodes have the same data at the same time.
2. Availability means every request is guaranteed to receive a success or failure response.
3. Partition tolerance guarantees that the system continues to operate in spite of arbitrary partitioning due to network failures.

3.2 NoSQL Business Drivers

Enterprises today need highly reliable, scalable and available data storage across a configurable set of systems that act as storage nodes. The needs of organizations are changing rapidly. Many organizations operating with single CPU and Relational database management systems (RDBMS) were not able to cope up with the speed in which information needs to be extracted. Businesses have to capture and analyze a large amount of variable data, and make immediate changes in their business based on their findings.

Figure 3.1 shows RDBMS with the business drivers velocity, volume, variability and agility necessitates the emergence of NoSQL solutions. All of these drivers apply pressure to single CPU relational model and eventually make the system less stable.

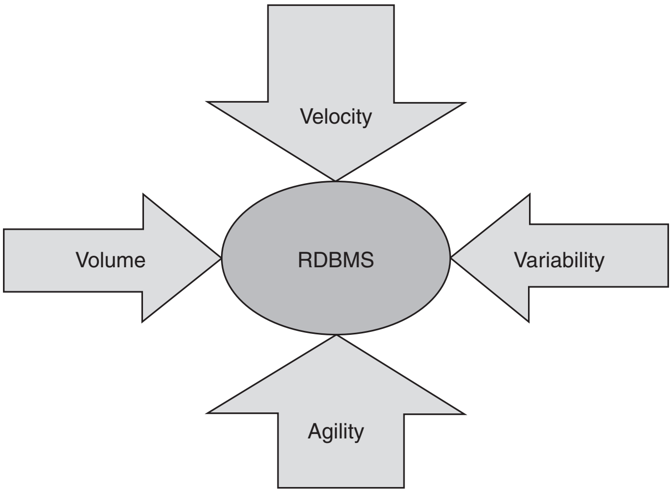

Figure 3.1 NoSQL business drivers.

3.2.1 Volume

There are two ways to look into data processing to improve performance. If the key factor is only speed, a faster processor could be used. If the processing involves complex (heavy) computation, Graphic Processing Unit (GPU) could be used along with the CPU. But the volume of data is limited to on-board GPU memory. The main reason for organizations to look at an alternative to their current RDBMSs is the need to query big data. The need to horizontal scaling made organizations to move from serial to distributed parallel processing where big data is fragmented and processed using clusters of commodity machines. This is made possible by the development of technologies like Apache Hadoop, HDFS, MapR, HBase, etc.

3.2.2 Velocity

Velocity becomes the key factor when frequency in which online queries to the database made by social networking and e-commerce web sites have to be read and written in real time. Many single CPU, RDBMS systems are unable to cope up with the demands of real-time inserts. RDBMS systems frequently index on many columns that decrease the system performance. For example, when online shopping sites introduce great discount schemes, the random bursts in web traffic will slow down the response for every user and tuning these systems as demand increases can be costly when both high read and write is required.

3.2.3 Variability

Organizations that need to capture and report on certain uncommon data, struggle when attempting to use RDBMS fixed schema. For example, if a business process wants to store a few special attributes for a few set of customers,then it needs to alter its schema definition. If a change is made to the schema, all customer rows within the database will also have this column. If there is no value related to this for most of the customers, then the row column representation will have sparse matrix. In addition to this, new columns to an RDBMS require to execute ALTER TABLE command. This cannot be done on the fly since the present executing transaction has to complete and database has to be closed, and then schema can be altered. This process affects system availability, which means losing business.

3.2.4 Agility

The process of storing and retrieving data for complex queries in RDBMS is quite cumbersome. If it is a nested query, data will have nested and repeated subgroups of data structures that are included in an object-relational mapping layer. This layer is responsible to generate the exact combination of SELECT, INSERT, DELETE and UPDATE SQL statements to move the object data from and to the backend RDBMS layer. This process is not simple and requires experienced developers with the knowledge of object-relational frameworks such as Java Hibernate. Even then, these change requests can cause slow-downs in implementation and testing.

Desirable features of NoSQL that drive business are listed below:

1. **24 × 7 Data availability:** In the highly competitive world today, downtime is equated to real dollars lost and is deadly to a company's reputation. Hardware failures are bound to occur. Care has to be taken that there is no single point of failure and system needs to show fault tolerance. For this, both function and data are to be replicated so that if database servers or "nodes" fail, the other nodes in the system are able to continue with operations without data loss. NoSQL database environments are able to provide this facility. System updates can be made dynamically without having to take the database offline.
2. **Location transparency:** The ability to read and write to a storage node regardless of where that I/O operation physically occurs is termed as "Location Transparency or Location Independence". Customers in many different geographies need to keep data local at those sites for fast access. Any write functionality that updates a node in one location, is propagated out from that location so that it is available to users and systems at other locations.
3. **Schema-less data model:** Most of the business data is unstructured and unpredictable which a RDBMS cannot cater to. NoSQL database system is a schema-free flexible data model that can easily accept all types of structured, semi-structured and unstructured data. Also relational model has scalability and performance problems when it has to manage large data volumes.

NoSQL data model can handle this easily to deliver very fast performance for both read and write operations.

4. **Modern day transaction analysis:** Most of the transaction details relate to customer profile, reviews on products, branding, reputation, building business strategy, trading decisions, etc. that do not require ACID transactions. The data consistency denoted by "C" in ACID property in RDBMSs is enforced via foreign keys/referential integrity constraints. This type of consistency is not required to be used in progressive data management systems such as NoSQL databases since there is no JOIN operation. Here, the "Consistency" is stated in the CAP theorem that signifies the immediate or eventual consistency of data across all nodes that participate in a distributed database.

5. **Architecture that suits big data:** NoSQL solutions provide modern architectures for applications that require high degrees of scale, data distribution and continuous availability. For this multiple data center support with which a NoSQL environment complies is one of the requirements. The solution should not only look into today's big data needs but also suit greater time horizons. Hence big data brings four major considerations in enterprise architecture which are as follows:
   - *Scale of data sources*: Many companies work in the multi-terabyte and even petabyte data.
   - *Speed is essential*: Overnight extract-transform-load (ETL) batches are insufficient and real-time streaming is required.
   - *Change in storage models*: Solutions like Hadoop Distributed File System (HDFS) and unstructured data stores like Apache Cassandra, MongoDb, Neo4j provide new options.
   - Multiple compute methods for Big Data Analytics must be supported.

Figure 3.2 shows the architecture that suits big data.

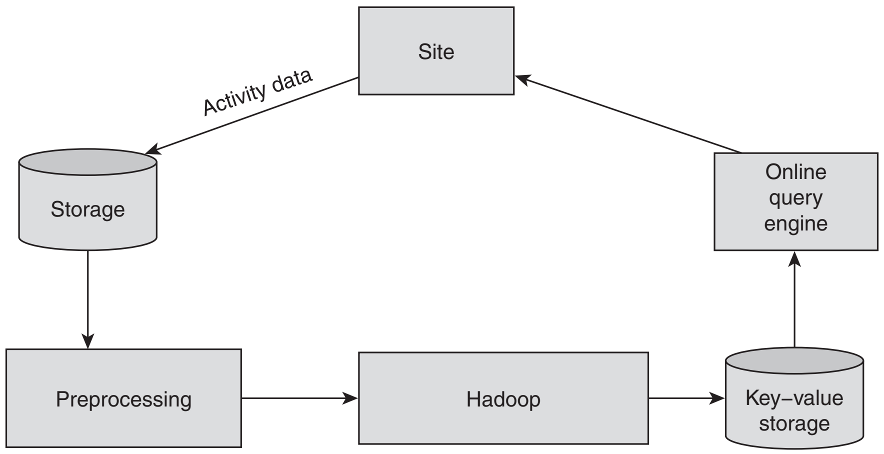

Figure 3.2 The architecture that suits big data.

6. **Analytics and business intelligence:** A key business strategic driver that suggests the implementation of a NoSQL database environment is the need to mine the data that is being collected in order to derive insights to gain competitive advantage. Traditional relational database system poses great difficulty in extracting meaningful business intelligence from very high volumes of data. NoSQL database systems not only provide storage and management of big data but also deliver integrated data analytics that provides instant understanding of complex datasets and facilitate various options for easy decision-making.

3.3 NoSQL Case Studies

Four case studies are discussed in the following subsections and each of them follow different architectural pattern, namely, key–value store, Column Family/BigTable, Document store and Graph store.

3.3.1 Amazon DynamoDB

Amazon.com has one of the largest e-commerce operations in the world. Customers from all around the world shop all hours of the day. So the site has to be up 24 × 7. Initially Amazon used RDBMS system for shopping cart and checkout system. Amazon DynamoDB, a NoSQL store brought a turning point.

DynamoDB addresses performance, scalability and reliability, the core problems of RDBMS when it comes to growing data. Developers can store unlimited amount of data by creating a database table and DynamoDB automatically saves it at multiple servers specified by the customer and also replicates them across multiple "Available" Zones. It can handle the data and traffic while maintaining consistent, fast performance. The cart data and session data are stored in the key–value store and the final (completed) order is saved in the RDBMS as shown in Fig. 3.3.

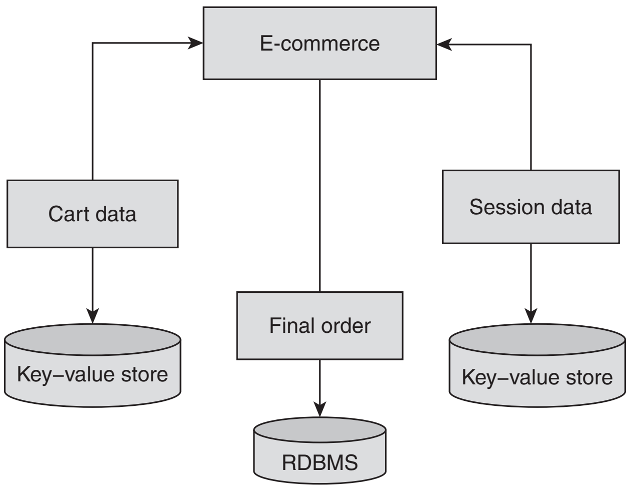

Figure 3.3 E-commerce shopping cart uses key–value store.

The salient features of key–value store are as follows:

1. **Scalable:** If application's requirements change, using the AWS Management Console or the DynamoDB APIs table throughput capacity can be updated.
2. **Automated storage scaling:** Using the DynamoDB write APIs, more storage can be obtained, whenever additional storage is required.
3. **Distributed horizontally shared nothing:** Seamlessly scales a single table over hundreds of commodity servers.
4. **Built-in fault tolerance:** DynamoDB automatically replicates data across multiple available zones in synchronous manner.
5. **Flexible:** DynamoDB has schema-free format. Multiple data types (strings, numbers, binary and sets) can be used and each data item can have different number of attributes.
6. **Efficient indexing:** Every item is identified by a primary key. It allows secondary indexes on non-key attributes, and query data using an alternate key.
7. **Strong consistency, Atomic counters:** DynamoDB ensures strong consistency on reads (only the latest values are read). DynamoDB service also supports atomic counters to atomically increment or decrement numerical attributes with a single API call.
8. **Secure:** DynamoDB uses cryptography to authenticate users. Access control for users within organization can be secured by integrating with AWS Identity and Access Management.
9. **Resource consumption monitoring:** AWS Management Console displays request throughput and latency for each DynamoDB table, since it is integrated with CloudWatch.
10. **Data warehouse – Business intelligence facility:** Amazon Redshift Integration – data from DynamoDB tables can be loaded into Amazon Redshift (data warehouse service).
11. **MapReduce integration:** DynamoDB is also integrated with Amazon Elastic MapReduce to perform complex analytics (hosted pay-as-you-go Hadoop framework on AWS).

3.3.2 Google's BigTable

Google's motivation for developing BigTable is driven by its need for massive scalability, better performance characteristics, and ability run on commodity hardware. Each time when a new service or increase in load happens, its solution BigTable would result in only a small incremental cost. Volume of Google's data generally is in petabytes and is distributed over 100,000 nodes.

BigTable is built on top of Google's other services that have been in active use since 2005, namely, Google File System, Scheduler, MapReduce and Chubby Lock Service. BigTable is a column-family database which is designed to store data tables (sparse matrix of row, column values) as section of column of data. It is distributed (high volume of data), persistent, multi-dimensional sorted map. The map is indexed by a row key, column key and a timestamp (int64).