MapReduce

MapReduce is a framework using which we can write applications to process huge amounts of data, in parallel, on large clusters of commodity hardware in a reliable manner.

What is MapReduce?

MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called mappers and reducers. Decomposing a data processing application into *mappers* and *reducers* is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model.

The Algorithm

- Generally MapReduce paradigm is based on sending the computer to where the data resides!
- MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.
    - **Map stage** − The map or mapper's job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.
    - **Reduce stage** − This stage is the combination of the **Shuffle**stage and the **Reduce** stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.
- During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster.

- The framework manages all the details of data-passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.
- Most of the computing takes place on nodes with data on local disks that reduces the network traffic.
- After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.

Inputs and Outputs (Java Perspective)

The MapReduce framework operates on <key, value> pairs, that is, the framework views the input to the job as a set of <key, value> pairs and produces a set of <key, value> pairs as the output of the job, conceivably of different types.

The key and the value classes should be in serialized manner by the framework and hence, need to implement the Writable interface. Additionally, the key classes have to implement the Writable-Comparable interface to facilitate sorting by the framework. Input and Output types of a **MapReduce job** − (Input) <k1, v1> → map → <k2, v2> → reduce → <k3, v3>(Output).

	Input	**Output**
Map	<k1, v1>	list (<k2, v2>)
Reduce	<k2, list(v2)>	list (<k3, v3>)

Terminology

- **PayLoad** − Applications implement the Map and the Reduce functions, and form the core of the job.
- **Mapper** − Mapper maps the input key/value pairs to a set of intermediate key/value pair.
- **NamedNode** − Node that manages the Hadoop Distributed File System (HDFS).
- **DataNode** − Node where data is presented in advance before any processing takes place.
- **MasterNode** − Node where JobTracker runs and which accepts job requests from clients.

- **SlaveNode** − Node where Map and Reduce program runs.
- **JobTracker** − Schedules jobs and tracks the assign jobs to Task tracker.
- **Task Tracker** − Tracks the task and reports status to JobTracker.
- **Job** − A program is an execution of a Mapper and Reducer across a dataset.
- **Task** − An execution of a Mapper or a Reducer on a slice of data.
- **Task Attempt** − A particular instance of an attempt to execute a task on a SlaveNode.

Example Scenario

Given below is the data regarding the electrical consumption of an organization. It contains the monthly electrical consumption and the annual average for various years.

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Avg
1979	23	23	2	43	24	25	26	26	26	26	25	26	25
1980	26	27	28	28	28	30	31	31	31	30	30	30	29
1981	31	32	32	32	33	34	35	36	36	34	34	34	34
1984	39	38	39	39	39	41	42	43	40	39	38	38	40
1985	38	39	39	39	39	41	41	41	00	40	39	39	45

If the above data is given as input, we have to write applications to process it and produce results such as finding the year of maximum usage, year of minimum usage, and so on. This is a walkover for the programmers with finite number of records. They will simply write the logic to produce the required output, and pass the data to the application written.

But, think of the data representing the electrical consumption of all the largescale industries of a particular state, since its formation.

When we write applications to process such bulk data,

- They will take a lot of time to execute.
- There will be a heavy network traffic when we move data from source to network server and so on.

To solve these problems, we have the MapReduce framework.

Input Data

The above data is saved as **sample.txt**and given as input. The input file looks as shown below.

```
1979   23   23   2   43   24   25   26   26   26   26   25   26  25
1980   26   27   28  28   28   30   31   31   31   30   30   30  29
1981   31   32   32  32   33   34   35   36   36   34   34   34  34
1984   39   38   39  39   39   41   42   43   40   39   38   38  40
1985   38   39   39  39   39   41   41   41   00   40   39   39  45
```

Example Program

Given below is the program to the sample data using MapReduce framework.

```
package hadoop;

import java.util.*;

import java.io.IOException;
import java.io.IOException;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

public class ProcessUnits {
   //Mapper class
   public static class E_EMapper extends MapReduceBase implements
   Mapper<LongWritable ,/*Input key Type */
   Text,                /*Input value Type*/
   Text,                /*Output key Type*/
   IntWritable>        /*Output value Type*/
   {
      //Map function
      public void map(LongWritable key, Text value,
      OutputCollector<Text, IntWritable> output,

      Reporter reporter) throws IOException {
         String line = value.toString();
         String lasttoken = null;
         StringTokenizer s = new StringTokenizer(line,"\t");
         String year = s.nextToken();

         while(s.hasMoreTokens()) {
```

```
            lasttoken = s.nextToken();
         }
         int avgprice = Integer.parseInt(lasttoken);
         output.collect(new Text(year), new IntWritable(avgprice));
      }
   }

   //Reducer class
   public static class E_EReduce extends MapReduceBase implements
Reducer< Text, IntWritable, Text, IntWritable > {

      //Reduce function
      public void reduce( Text key, Iterator <IntWritable> values,
      OutputCollector<Text, IntWritable> output, Reporter reporter)
throws IOException {
         int maxavg = 30;
         int val = Integer.MIN_VALUE;

         while (values.hasNext()) {
            if((val = values.next().get())>maxavg) {
               output.collect(key, new IntWritable(val));
            }
         }
      }
   }

   //Main function
   public static void main(String args[])throws Exception {
      JobConf conf = new JobConf(ProcessUnits.class);

      conf.setJobName("max_eletricityunits");
      conf.setOutputKeyClass(Text.class);
      conf.setOutputValueClass(IntWritable.class);
      conf.setMapperClass(E_EMapper.class);
      conf.setCombinerClass(E_EReduce.class);
      conf.setReducerClass(E_EReduce.class);
      conf.setInputFormat(TextInputFormat.class);
      conf.setOutputFormat(TextOutputFormat.class);

      FileInputFormat.setInputPaths(conf, new Path(args[0]));
      FileOutputFormat.setOutputPath(conf, new Path(args[1]));

      JobClient.runJob(conf);
   }
}
```

Save the above program as **ProcessUnits.java.** The compilation and execution of the program is explained below.

Compilation and Execution of Process Units Program

Let us assume we are in the home directory of a Hadoop user (e.g. /home/hadoop).

Follow the steps given below to compile and execute the above program.

Step 1

The following command is to create a directory to store the compiled java classes.

```
$ mkdir units
```

Step 2

Download **Hadoop-core-1.2.1.jar,** which is used to compile and execute the MapReduce program. Visit the following link mvnrepository.com to download the jar. Let us assume the downloaded folder is **/home/hadoop/.**

Step 3

The following commands are used for compiling the **ProcessUnits.java** program and creating a jar for the program.

```
$ javac -classpath hadoop-core-1.2.1.jar -d units ProcessUnits.java
$ jar -cvf units.jar -C units/ .
```

Step 4

The following command is used to create an input directory in HDFS.

```
$HADOOP_HOME/bin/hadoop fs -mkdir input_dir
```

Step 5

The following command is used to copy the input file named **sample.txt**in the input directory of HDFS.

```
$HADOOP_HOME/bin/hadoop fs -put /home/hadoop/sample.txt input_dir
```

Step 6

The following command is used to verify the files in the input directory.

```
$HADOOP_HOME/bin/hadoop fs -ls input_dir/
```

Step 7

The following command is used to run the Eleunit_max application by taking the input files from the input directory.

```
$HADOOP_HOME/bin/hadoop jar units.jar hadoop.ProcessUnits input_dir
output_dir
```

Wait for a while until the file is executed. After execution, as shown below, the output will contain the number of input splits, the number of Map tasks, the number of reducer tasks, etc.

```
INFO mapreduce.Job: Job job_1414748220717_0002
completed successfully
14/10/31 06:02:52
INFO mapreduce.Job: Counters: 49
   File System Counters

FILE: Number of bytes read = 61
FILE: Number of bytes written = 279400
FILE: Number of read operations = 0
FILE: Number of large read operations = 0
FILE: Number of write operations = 0
HDFS: Number of bytes read = 546
HDFS: Number of bytes written = 40
HDFS: Number of read operations = 9
HDFS: Number of large read operations = 0
HDFS: Number of write operations = 2 Job Counters


   Launched map tasks = 2
   Launched reduce tasks = 1
   Data-local map tasks = 2
   Total time spent by all maps in occupied slots (ms) = 146137
   Total time spent by all reduces in occupied slots (ms) = 441
   Total time spent by all map tasks (ms) = 14613
   Total time spent by all reduce tasks (ms) = 44120
   Total vcore-seconds taken by all map tasks = 146137
   Total vcore-seconds taken by all reduce tasks = 44120
   Total megabyte-seconds taken by all map tasks = 149644288
   Total megabyte-seconds taken by all reduce tasks = 45178880

Map-Reduce Framework

   Map input records = 5
   Map output records = 5
   Map output bytes = 45
   Map output materialized bytes = 67
   Input split bytes = 208
   Combine input records = 5
   Combine output records = 5
   Reduce input groups = 5
```

```
   Reduce shuffle bytes = 6
   Reduce input records = 5
   Reduce output records = 5
   Spilled Records = 10
   Shuffled Maps  = 2
   Failed Shuffles = 0
   Merged Map outputs = 2
   GC time elapsed (ms) = 948
   CPU time spent (ms) = 5160
   Physical memory (bytes) snapshot = 47749120
   Virtual memory (bytes) snapshot = 2899349504
   Total committed heap usage (bytes) = 277684224

File Output Format Counters

   Bytes Written = 40
```

Step 8

The following command is used to verify the resultant files in the output folder.

```
$HADOOP_HOME/bin/hadoop fs -ls output_dir/
```

Step 9

The following command is used to see the output in **Part-00000** file. This file is generated by HDFS.

```
$HADOOP_HOME/bin/hadoop fs -cat output_dir/part-00000
```

Below is the output generated by the MapReduce program.

```
1981    34
1984    40
1985    45
```

Step 10

The following command is used to copy the output folder from HDFS to the local file system for analyzing.

```
$HADOOP_HOME/bin/hadoop fs -cat output_dir/part-00000/bin/hadoop
dfs get output_dir /home/hadoop
```

Important Commands

All Hadoop commands are invoked by the **$HADOOP_HOME/bin/hadoop**command. Running the Hadoop script without any arguments prints the description for all commands.

Usage − hadoop [--config confdir] COMMAND

The following table lists the options available and their description.

Sr.No.	Option & Description
1	**namenode -format** Formats the DFS filesystem.
2	**secondarynamenode** Runs the DFS secondary namenode.
3	**namenode** Runs the DFS namenode.
4	**datanode** Runs a DFS datanode.
5	**dfsadmin** Runs a DFS admin client.
6	**mradmin** Runs a Map-Reduce admin client.
7	**fsck** Runs a DFS filesystem checking utility.
8	**fs** Runs a generic filesystem user client.
9	**balancer** Runs a cluster balancing utility.

10	**oiv** Applies the offline fsimage viewer to an fsimage.
11	**fetchdt** Fetches a delegation token from the NameNode.
12	**jobtracker** Runs the MapReduce job Tracker node.
13	**pipes** Runs a Pipes job.
14	**tasktracker** Runs a MapReduce task Tracker node.
15	**historyserver** Runs job history servers as a standalone daemon.
16	**job** Manipulates the MapReduce jobs.
17	**queue** Gets information regarding JobQueues.
18	**version** Prints the version.
19	**jar <jar>** Runs a jar file.
20	**distcp <srcurl> <desturl>**

	Copies file or directories recursively.
21	**distcp2 <srcurl> <desturl>** DistCp version 2.
22	**archive -archiveName NAME -p <parent path> <src>* <dest>** Creates a hadoop archive.
23	**classpath** Prints the class path needed to get the Hadoop jar and the required libraries.
24	**daemonlog** Get/Set the log level for each daemon

How to Interact with MapReduce Jobs

Usage − hadoop job [GENERIC_OPTIONS]

The following are the Generic Options available in a Hadoop job.

Sr.No.	GENERIC_OPTION & Description
1	**-submit <job-file>** Submits the job.
2	**-status <job-id>** Prints the map and reduce completion percentage and all job counters.
3	**-counter <job-id> <group-name> <countername>** Prints the counter value.
4	**-kill <job-id>**

	Kills the job.
5	**-events <job-id> <fromevent-#> <#-of-events>** Prints the events' details received by jobtracker for the given range.
6	**-history [all] <jobOutputDir> - history < jobOutputDir>** Prints job details, failed and killed tip details. More details about the job such as successful tasks and task attempts made for each task can be viewed by specifying the [all] option.
7	**-list[all]** Displays all jobs. -list displays only jobs which are yet to complete.
8	**-kill-task <task-id>** Kills the task. Killed tasks are NOT counted against failed attempts.
9	**-fail-task <task-id>** Fails the task. Failed tasks are counted against failed attempts.
10	**-set-priority <job-id> <priority>** Changes the priority of the job. Allowed priority values are VERY_HIGH, HIGH, NORMAL, LOW, VERY_LOW

To see the status of job

```
$ $HADOOP_HOME/bin/hadoop job -status <JOB-ID>
e.g.
$ $HADOOP_HOME/bin/hadoop job -status job_201310191043_0004
```

To see the history of job output-dir

```
$ $HADOOP_HOME/bin/hadoop job -history <DIR-NAME>
e.g.
$ $HADOOP_HOME/bin/hadoop job -history /user/expert/output
```

To kill the job

```
$ $HADOOP_HOME/bin/hadoop job -kill <JOB-ID>
e.g.
$ $HADOOP_HOME/bin/hadoop job -kill job_201310191043_0004
```

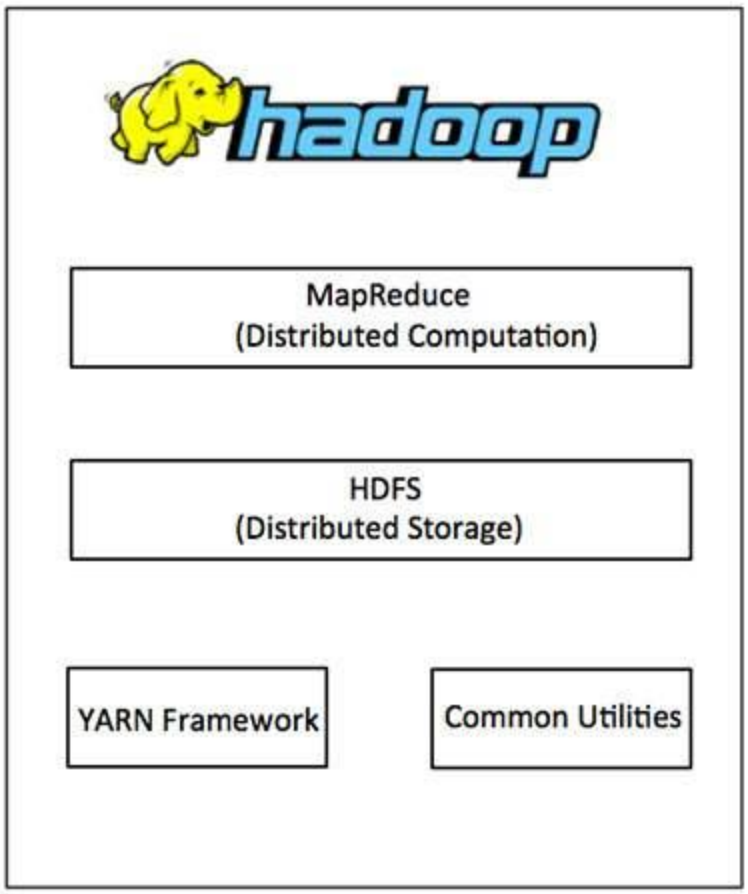
Hadoop - Introduction

Hadoop is an Apache open source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models. The Hadoop framework application works in an environment that provides distributed *storage* and *computation* across clusters of computers. Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.

Hadoop Architecture

At its core, Hadoop has two major layers namely −

- Processing/Computation layer (MapReduce), and
- Storage layer (Hadoop Distributed File System).



MapReduce

MapReduce is a parallel programming model for writing distributed applications devised at Google for efficient processing of large amounts of data (multi-terabyte data-sets), on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. The MapReduce program runs on Hadoop which is an Apache open-source framework.

Hadoop Distributed File System

The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. It is highly fault-tolerant and is designed to be deployed

on low-cost hardware. It provides high throughput access to application data and is suitable for applications having large datasets.

Apart from the above-mentioned two core components, Hadoop framework also includes the following two modules −

- **Hadoop Common** − These are Java libraries and utilities required by other Hadoop modules.
- **Hadoop YARN** − This is a framework for job scheduling and cluster resource management.

How Does Hadoop Work?

It is quite expensive to build bigger servers with heavy configurations that handle large scale processing, but as an alternative, you can tie together many commodity computers with single-CPU, as a single functional distributed system and practically, the clustered machines can read the dataset in parallel and provide a much higher throughput. Moreover, it is cheaper than one high-end server. So this is the first motivational factor behind using Hadoop that it runs across clustered and low-cost machines.

Hadoop runs code across a cluster of computers. This process includes the following core tasks that Hadoop performs −

- Data is initially divided into directories and files. Files are divided into uniform sized blocks of 128M and 64M (preferably 128M).
- These files are then distributed across various cluster nodes for further processing.
- HDFS, being on top of the local file system, supervises the processing.
- Blocks are replicated for handling hardware failure.
- Checking that the code was executed successfully.
- Performing the sort that takes place between the map and reduce stages.
- Sending the sorted data to a certain computer.
- Writing the debugging logs for each job.

Advantages of Hadoop

- Hadoop framework allows the user to quickly write and test distributed systems. It is efficient, and it automatic distributes the data and work across the machines and in turn, utilizes the underlying parallelism of the CPU cores.
- Hadoop does not rely on hardware to provide fault-tolerance and high availability (FTHA), rather Hadoop library itself has been designed to detect and handle failures at the application layer.
- Servers can be added or removed from the cluster dynamically and Hadoop continues to operate without interruption.
- Another big advantage of Hadoop is that apart from being open source, it is compatible on all the platforms since it is Java based.

Hadoop - HDFS Overview

Hadoop File System was developed using distributed file system design. It is run on commodity hardware. Unlike other distributed systems, HDFS is highly faulttolerant and designed using low-cost hardware.

HDFS holds very large amount of data and provides easier access. To store such huge data, the files are stored across multiple machines. These files are stored in redundant fashion to rescue the system from possible data losses in case of failure. HDFS also makes applications available to parallel processing.

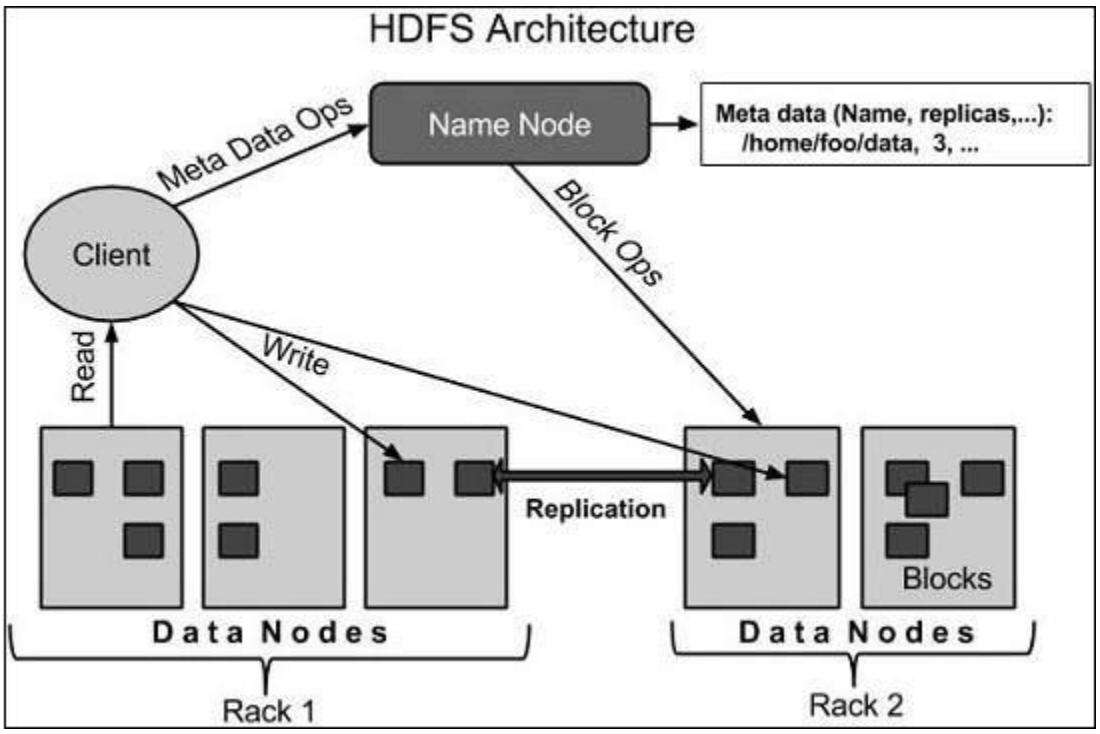
Features of HDFS

- It is suitable for the distributed storage and processing.
- Hadoop provides a command interface to interact with HDFS.
- The built-in servers of namenode and datanode help users to easily check the status of cluster.
- Streaming access to file system data.

- HDFS provides file permissions and authentication.

HDFS Architecture

Given below is the architecture of a Hadoop File System.



HDFS follows the master-slave architecture and it has the following elements.

Namenode

The namenode is the commodity hardware that contains the GNU/Linux operating system and the namenode software. It is a software that can be run on commodity hardware. The system having the namenode acts as the master server and it does the following tasks −

- Manages the file system namespace.
- Regulates client's access to files.
- It also executes file system operations such as renaming, closing, and opening files and directories.

Datanode

The datanode is a commodity hardware having the GNU/Linux operating system and datanode software. For every node (Commodity hardware/System) in a cluster, there will be a datanode. These nodes manage the data storage of their system.

- Datanodes perform read-write operations on the file systems, as per client request.
- They also perform operations such as block creation, deletion, and replication according to the instructions of the namenode.

Block

Generally the user data is stored in the files of HDFS. The file in a file system will be divided into one or more segments and/or stored in individual data nodes. These file segments are called as blocks. In other words, the minimum amount of data that HDFS can read or write is called a Block. The default block size is 64MB, but it can be increased as per the need to change in HDFS configuration.

Goals of HDFS

Fault detection and recovery − Since HDFS includes a large number of commodity hardware, failure of components is frequent. Therefore HDFS should have mechanisms for quick and automatic fault detection and recovery.

Huge datasets − HDFS should have hundreds of nodes per cluster to manage the applications having huge datasets.

Hardware at data − A requested task can be done efficiently, when the computation takes place near the data. Especially where huge datasets are involved, it reduces the network traffic and increases the throughput.

HBase - Overview

Since 1970, RDBMS is the solution for data storage and maintenance related problems. After the advent of big data, companies realized the benefit of processing big data and started opting for solutions like Hadoop.

Hadoop uses distributed file system for storing big data, and MapReduce to process it. Hadoop excels in storing and processing of huge data of various formats such as arbitrary, semi-, or even unstructured.

Limitations of Hadoop

Hadoop can perform only batch processing, and data will be accessed only in a sequential manner. That means one has to search the entire dataset even for the simplest of jobs.

A huge dataset when processed results in another huge data set, which should also be processed sequentially. At this point, a new solution is needed to access any point of data in a single unit of time (random access).

Hadoop Random Access Databases

Applications such as HBase, Cassandra, couchDB, Dynamo, and MongoDB are some of the databases that store huge amounts of data and access the data in a random manner.

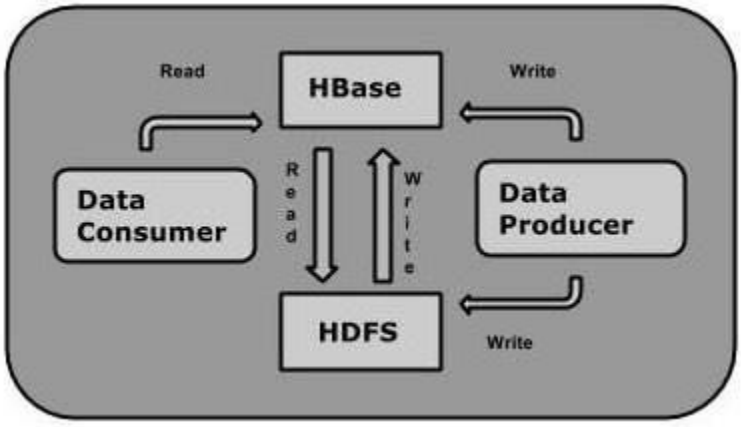
What is HBase?

HBase is a distributed column-oriented database built on top of the Hadoop file system. It is an open-source project and is horizontally scalable.

HBase is a data model that is similar to Google’s big table designed to provide quick random access to huge amounts of structured data. It leverages the fault tolerance provided by the Hadoop File System (HDFS).

It is a part of the Hadoop ecosystem that provides random real-time read/write access to data in the Hadoop File System.

One can store the data in HDFS either directly or through HBase. Data consumer reads/accesses the data in HDFS randomly using HBase. HBase sits on top of the Hadoop File System and provides read and write access.



HBase and HDFS

HDFS	HBase
HDFS is a distributed file system suitable for storing large files.	HBase is a database built on top of the HDFS.
HDFS does not support fast individual record lookups.	HBase provides fast lookups for larger tables.
It provides high latency batch processing; no concept of batch processing.	It provides low latency access to single rows from billions of records (Random access).
It provides only sequential access of data.	HBase internally uses Hash tables and provides random access, and it stores the data in indexed HDFS files for faster lookups.

Storage Mechanism in HBase

HBase is a **column-oriented database** and the tables in it are sorted by row. The table schema defines only column families, which are the key value pairs. A table have multiple column families and each column family can have any number of columns. Subsequent column values are stored contiguously on the disk. Each cell value of the table has a timestamp. In short, in an HBase:

- Table is a collection of rows.
- Row is a collection of column families.
- Column family is a collection of columns.
- Column is a collection of key value pairs.

Given below is an example schema of table in HBase.

Rowid	**Column Family**			**Column Family**			**Column Family**			**Column Family**		
	col1	**col2**	**col3**	**col1**	**col2**	**col3**	**col1**	**col2**	**col3**	**col1**	**col2**	**col3**
1												
2												
3												

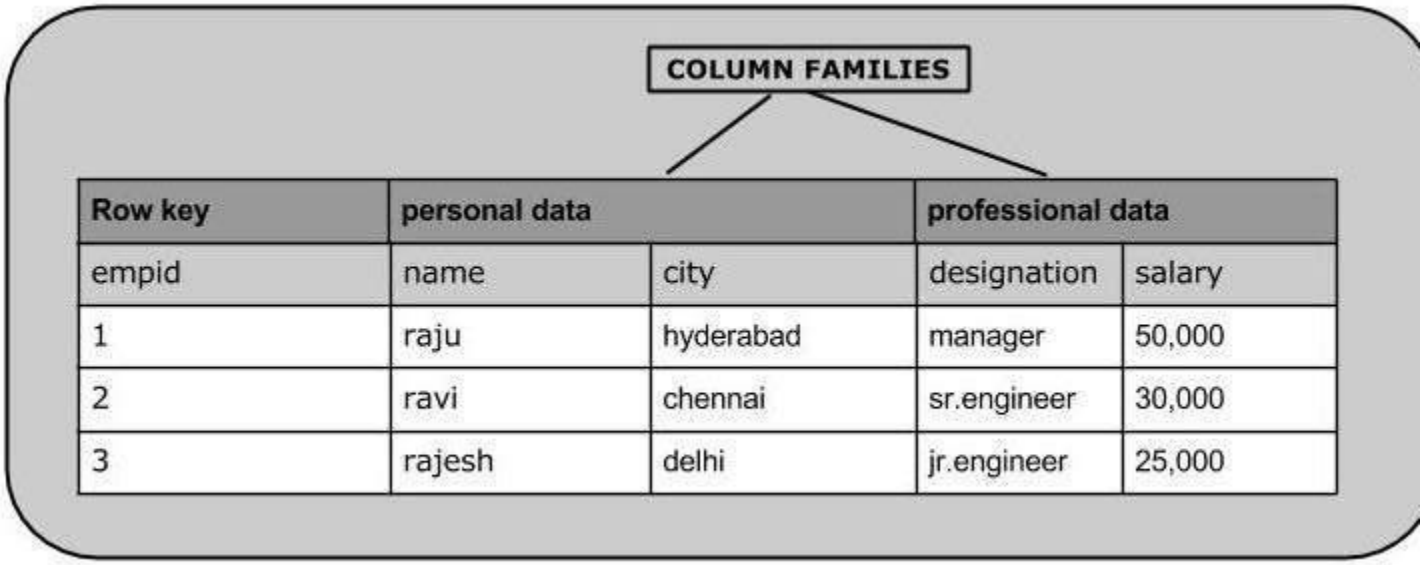
Column Oriented and Row Oriented

Column-oriented databases are those that store data tables as sections of columns of data, rather than as rows of data. Shortly, they will have column families.

Row-Oriented Database	**Column-Oriented Database**

It is suitable for Online Transaction Process (OLTP).	It is suitable for Online Analytical Processing (OLAP).
Such databases are designed for small number of rows and columns.	Column-oriented databases are designed for huge tables.

The following image shows column families in a column-oriented database:



HBase and RDBMS

HBase	RDBMS
HBase is schema-less, it doesn't have the concept of fixed columns schema; defines only column families.	An RDBMS is governed by its schema, which describes the whole structure of tables.
It is built for wide tables. HBase is horizontally	It is thin and built for small tables. Hard

scalable.	to scale.
No transactions are there in HBase.	RDBMS is transactional.
It has de-normalized data.	It will have normalized data.
It is good for semi-structured as well as structured data.	It is good for structured data.

Features of HBase

- HBase is linearly scalable.
- It has automatic failure support.
- It provides consistent read and writes.
- It integrates with Hadoop, both as a source and a destination.
- It has easy java API for client.
- It provides data replication across clusters.

Where to Use HBase

- Apache HBase is used to have random, real-time read/write access to Big Data.
- It hosts very large tables on top of clusters of commodity hardware.
- Apache HBase is a non-relational database modeled after Google's Bigtable. Bigtable acts up on Google File System, likewise Apache HBase works on top of Hadoop and HDFS.

Applications of HBase

- It is used whenever there is a need to write heavy applications.
- HBase is used whenever we need to provide fast random access to available data.
- Companies such as Facebook, Twitter, Yahoo, and Adobe use HBase internally.

HBase History

Year	Event
Nov 2006	Google released the paper on BigTable.
Feb 2007	Initial HBase prototype was created as a Hadoop contribution.
Oct 2007	The first usable HBase along with Hadoop 0.15.0 was released.
Jan 2008	HBase became the sub project of Hadoop.
Oct 2008	HBase 0.18.1 was released.
Jan 2009	HBase 0.19.0 was released.
Sept 2009	HBase 0.20.0 was released.
May 2010	HBase became Apache top-level project.

1. Bigtable

Google uses as a data storage a facility called Bigtable. Bigtable is a distributed, persistent, multidimensional sorted map. Bigtable is not a relational database. In Bigtable you can store strings under an index which consists out of a row key, a column key and a timestamp. This key points to a uninterpreted array of bytes (string) of size 64 KB.

(row key: type string, column key:type string, timestamp: type int64) → string

The key can get generated by the database or by the application.

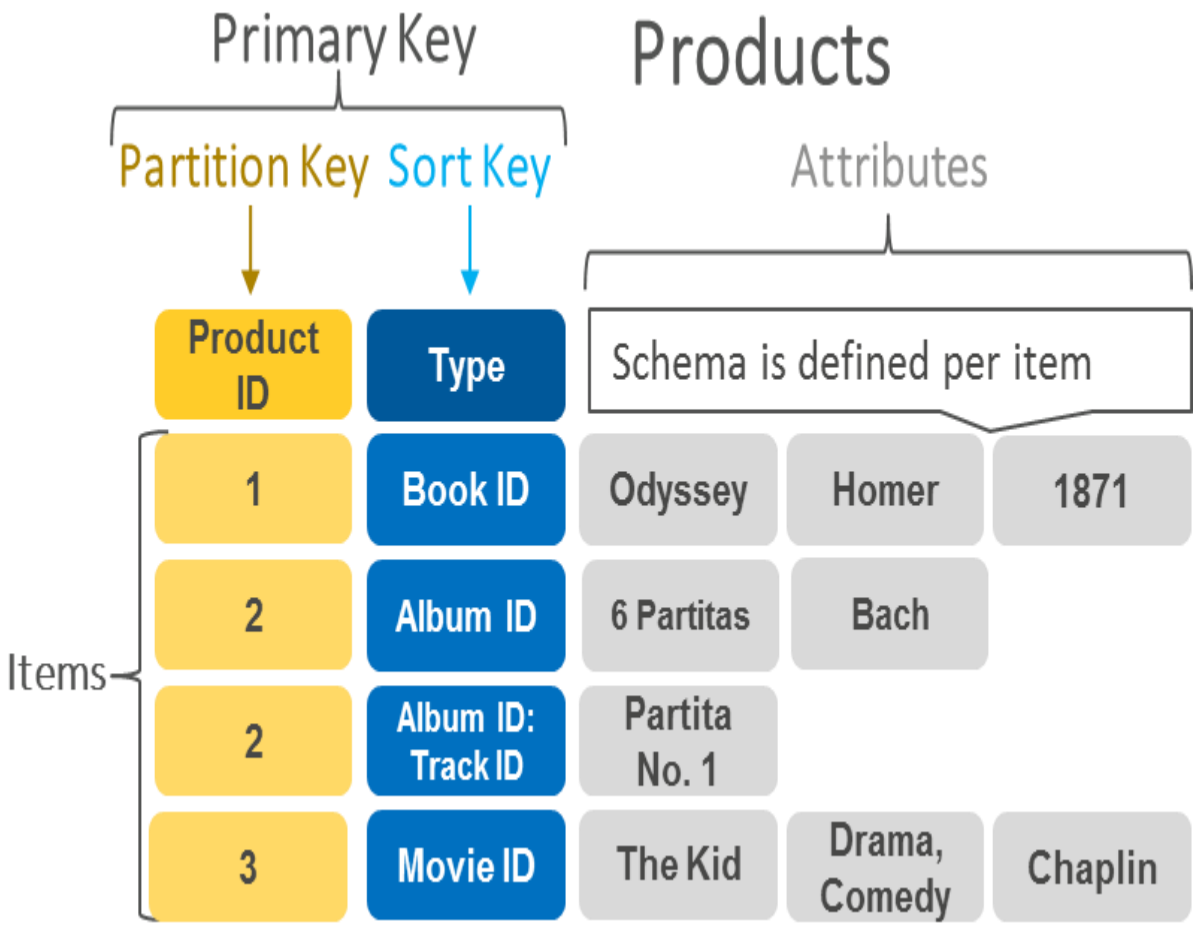
For example in the Google Webtable (for Google search) the reverse URL is used as the row key, the column used for different attributes of the webpage and the timestamp indicates from when the data is. The data this key points to is some content from the webpage.

Bigtable is build upon the Google File System and stored in an immutable datastructure called SSTable. The application can define how many entries based on the timestamp should be keep. Alternatively the application can also specify how long entries should be keep. Bigtable will clean-up the obsolete data by deleting the SSTables which only contains irrelevant data using a mark-and-sweep algorithm.

The AMAZON'S key-value database defined

A key-value database is a type of nonrelational database that uses a simple key-value method to store data. A key-value database stores data as a collection of key-value pairs in which a key serves as a unique identifier. Both keys and values can be anything, ranging from simple objects to complex compound objects. Key-value databases are highly partitionable and allow horizontal scaling at scales that other types of databases cannot achieve. For example, Amazon DynamoDB allocates additional partitions to a table if an existing partition fills to capacity and more storage space is required.

The following diagram shows an example of data stored as key-value pairs in DynamoDB.



Microsoft Azure - Windows

There are many cloud computing platforms offered by different organizations. Windows Azure is one of them, which is provided by Microsoft. Azure can be described as the managed data centers that are used to build, deploy, manage the applications and provide services through a global network. The services provided by Microsoft Azure are PaaS and IaaS. Many programming languages and frameworks are supported by it.

Azure as PaaS (Platform as a Service)

As the name suggests, a platform is provided to clients to develop and deploy software. The clients can focus on the application development rather than having to worry about hardware and infrastructure. It also takes care of most of the operating systems, servers and networking issues.

Pros

- The overall cost is low as the resources are allocated on demand and servers are automatically updated.
- It is less vulnerable as servers are automatically updated and being checked for all known security issues. The whole process is not visible to developer and thus does not pose a risk of data breach.
- Since new versions of development tools are tested by the Azure team, it becomes easy for developers to move on to new tools. This also helps the developers to meet the customer's demand by quickly adapting to new versions.

Cons

- There are portability issues with using PaaS. There can be a different environment at Azure, thus the application might have to be adapted accordingly.

Azure as IaaS (Infrastructure as a Service)

It is a managed compute service that gives complete control of the operating systems and the application platform stack to the application developers. It lets the user to access, manage and monitor the data centers by themselves.

Pros

- This is ideal for the application where complete control is required. The virtual machine can be completely adapted to the requirements of the organization or business.
- IaaS facilitates very efficient design time portability. This means application can be migrated to Windows Azure without rework. All the application dependencies such as database can also be migrated to Azure.
- IaaS allows quick transition of services to clouds, which helps the vendors to offer services to their clients easily. This also helps the vendors to expand their business by selling the existing software or services in new markets.

Cons

- Since users are given complete control they are tempted to stick to a particular version for the dependencies of applications. It might become difficult for them to migrate the application to future versions.
- There are many factors which increases the cost of its operation. For example, higher server maintenance for patching and upgrading software.
- There are lots of security risks from unpatched servers. Some companies have welldefined processes for testing and updating on-premise servers for security vulnerabilities. These processes need to be extended to the cloud-hosted IaaS VMs to mitigate hacking risks.
- The unpatched servers pose a great security risk. Unlike PaaS, there is no provision of automatic server patching in IaaS. An unpatched server with sensitive information can be very vulnerable affecting the entire business of an organization.
- It is difficult to maintain legacy apps in Iaas. It can be stuck with the older version of the operating systems and application stacks. Thus, resulting in applications that are difficult to maintain and add new functionality over the period of time.

It becomes necessary to understand the pros and cons of both services in order to choose the right one according your requirements. In conclusion it can be said that, PaaS has definite economic advantages for operations over IaaS for commodity applications. In PaaS, the cost of

operations breaks the business model. Whereas, IaaS gives complete control of the OS and application platform stack.

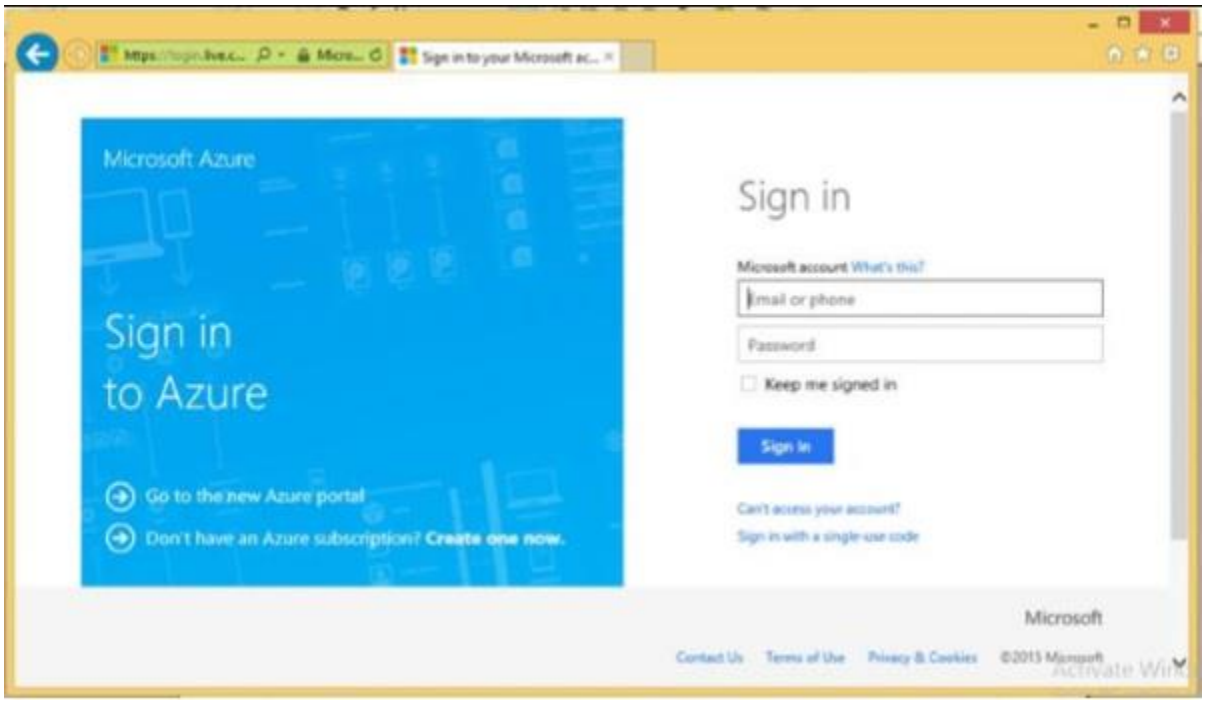
Azure Management Portal

Azure Management Portal is an interface to manage the services and infrastructure launched in 2012. All the services and applications are displayed in it and it lets the user manage them.

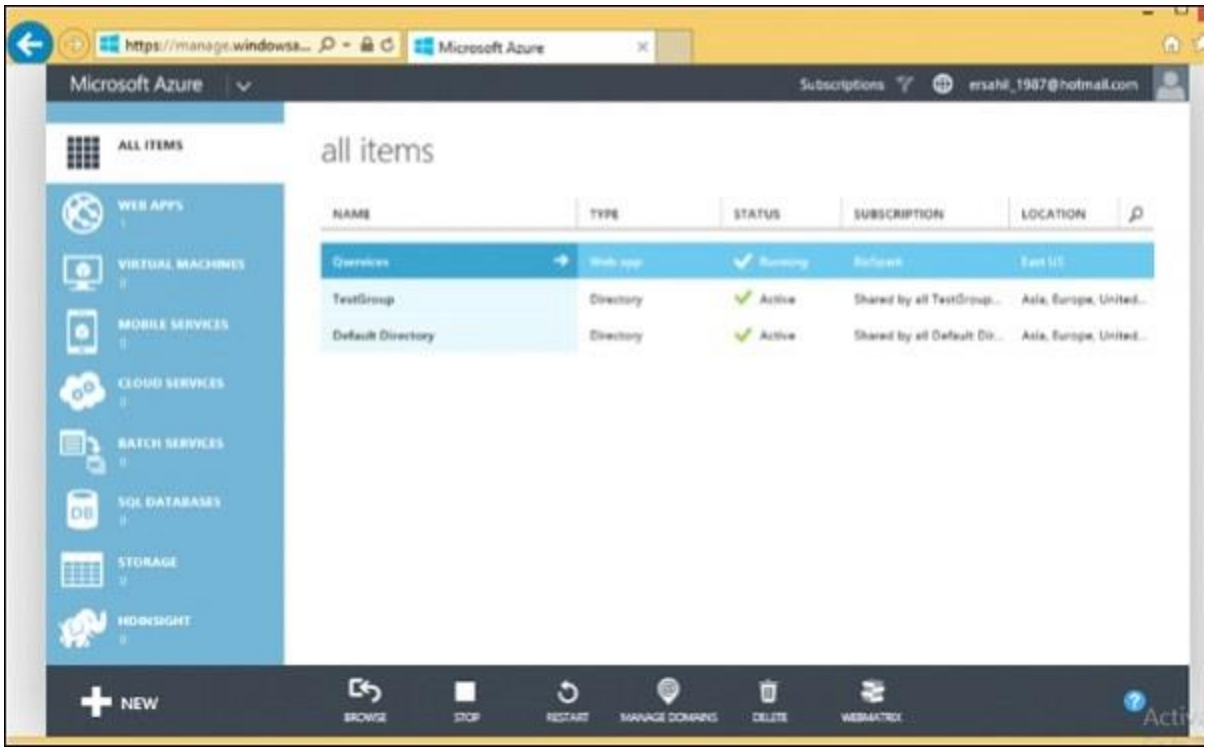
Getting started

A free trial account can be created on Azure management portal by visiting the following link - manage.windowsazure.com

The screen that pops up is as shown in the following image. The account can be created using our existing Gmail, Hotmail or Yahoo account.



Once logged in, you will be redirected to the following screen, where there is a list of services and applications on the left panel.



When you click on a category, its details are displayed on the screen. You can see the number of applications, virtual machine, mobile services and so on by clicking on the menu item.

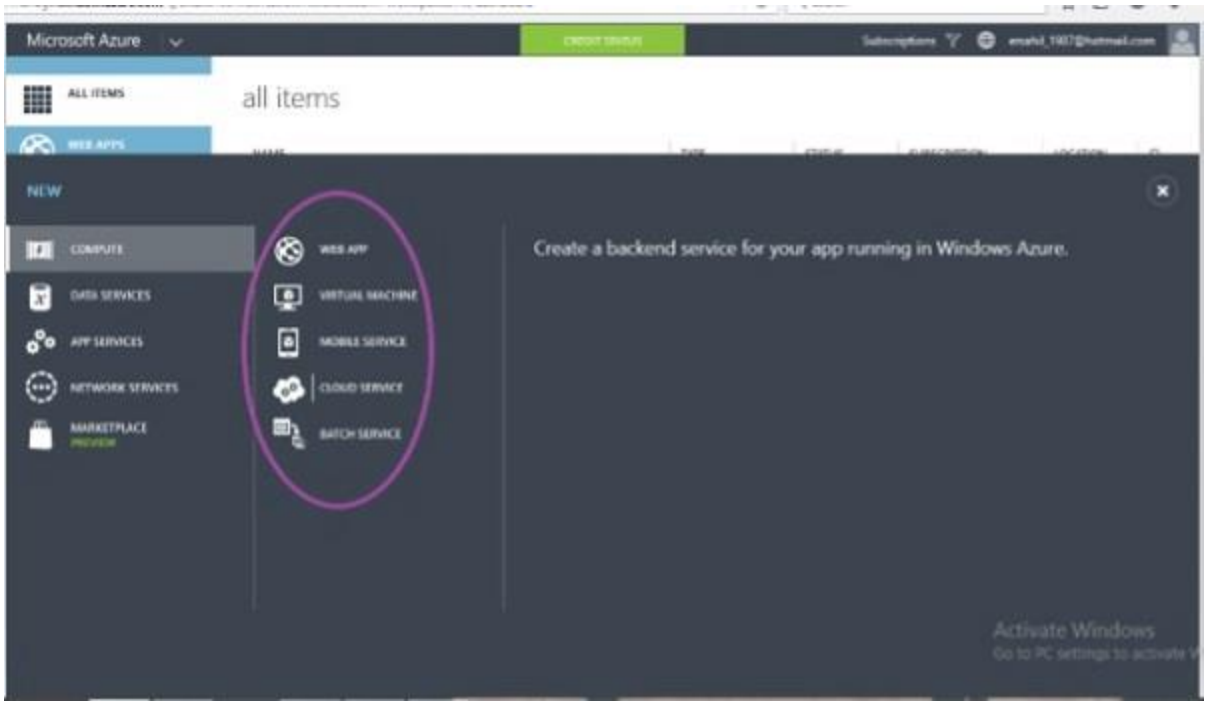
The next chapter contains a detailed explanation of how to use this portal to manage Azure services.

Microsoft Azure - Components

Categorizing the services would help you understand Azure better. These categories are termed as ‘Components’ in this tutorial. The Individual components are explained with detailed pictures in subsequent chapters.

Compute / Execution Models

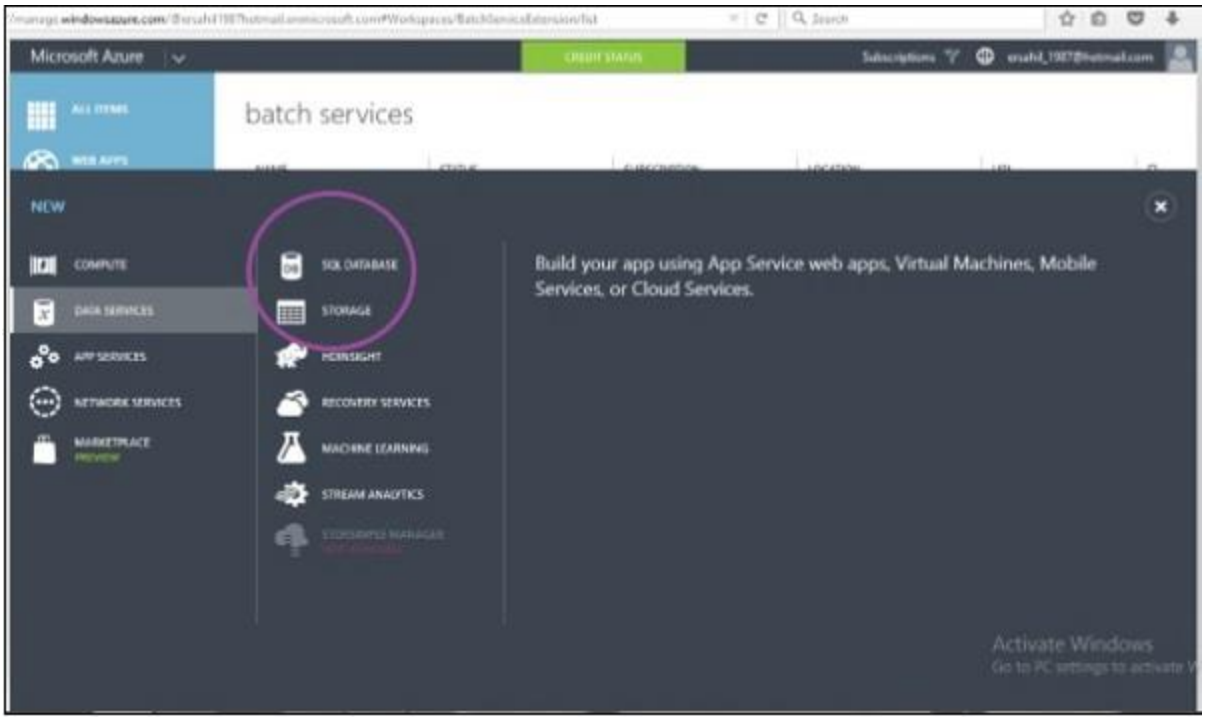
This is the interface for executing the application, which is one of the basic functions of Azure.



As seen in the above image, there are different models such as Web App, Virtual Machine, Mobile Service, Cloud Service, and Batch Service. These models can be used either separately or in combination as per the requirement.

Data Management

Data management can be done by using SQL server Database component or the simple data storage module offered by Windows Azure. SQL server database can be used for relational database. The storage module can store unrelated tables (without foreign key or any relation) and blobs. Blobs include binary data in the form of images, audio, video, and text files.

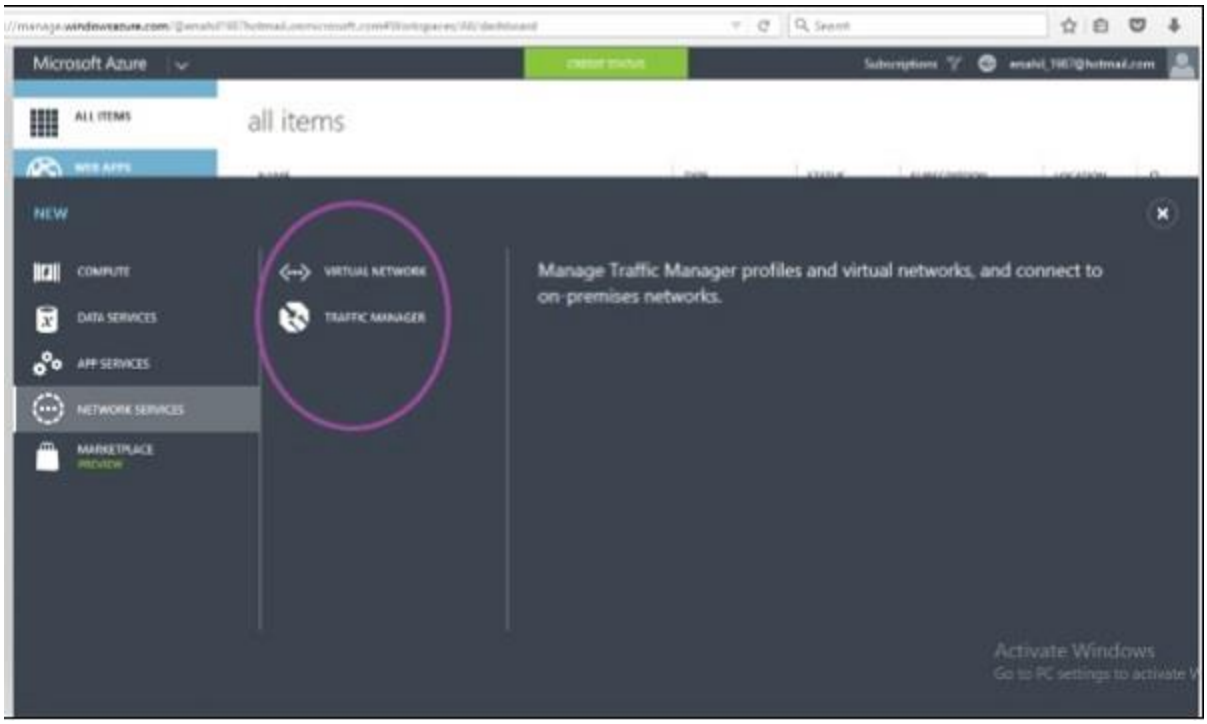


Networking

Azure traffic manager routes the requests of a user intelligently to an available datacenter. The process involves finding the nearest datacenter to the user who makes the request for web application, and if the nearest datacenter is not available due to various reasons, the traffic manager deviates the request to another datacenter. However, rules are set by the owner of the application as to how a traffic manager should behave.

The virtual network is another feature that is part of networking in services offered by Windows Azure. The virtual network allows a network between local machines at your premise and virtual machine in Azure Datacenter. IPs to virtual machines can be assigned in a way that makes them appear to be residing in your own premise. The virtual network is set up using a Virtual Private Network (VPN) device.

The following image shows how these two features actually look in Azure portal.



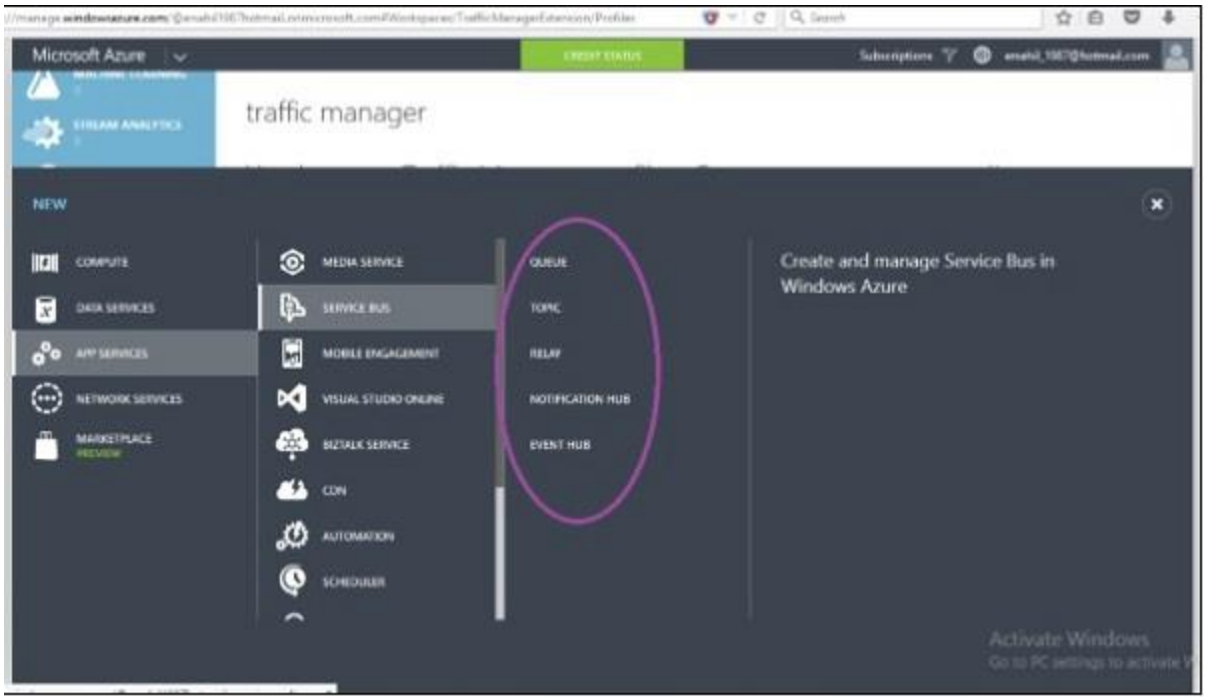
Big Data and Big Compute

The large amount of data can be stored and managed using Windows Azure. Azure offers HDInsight which is Hadoop-based service. Organizations often need to manage large amount of data which is necessarily not relational database management. Hadoop is a prominent technology used these days. Thus, Azure offers Hadoop service on their platform for clients.

The term ‘Big Compute’ refers to high performing computations. This is achieved by executing code on many machines at the same time.

Messaging

Windows Azure offers two options for handling the interactions between two apps. One falls under storage component of the service and is called **'Message Queues'**. The other one comes under the app service and is called **'Service Bus'**. The messages can be sent to initiate communication among different components of an application or among different applications using these two options.

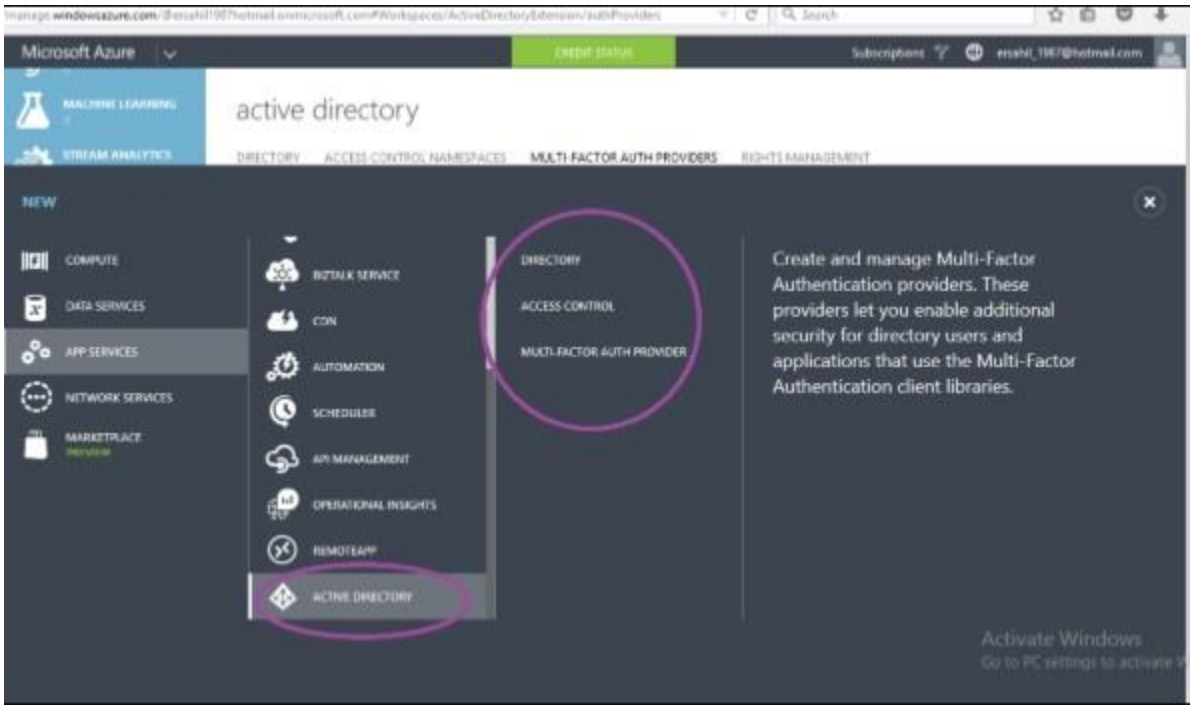


Caching

Microsoft Azure offers two kinds of caching which are in-memory Caching and Content Delivery Network (CDN) for caching frequently accessed data and improves the application performance. CDN is used to cache the blob data that will be accessed faster by users around the world.

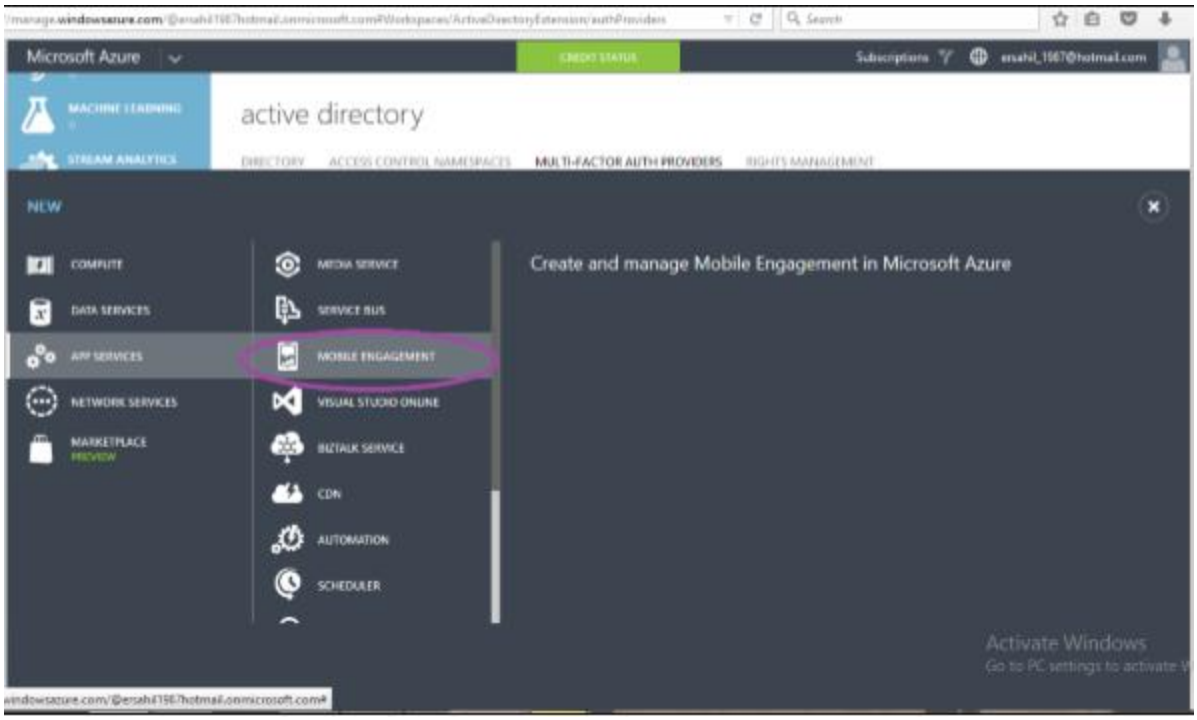
Identity and Access

This component is about management of users, authentication and authorization. Active directory stores the information of users accessing the application and also the organization's information. It can synchronize with the related information on local machines residing on premises. Multifactor Access (MFA) service is built to address the security concerns such as only the right user can access the application.



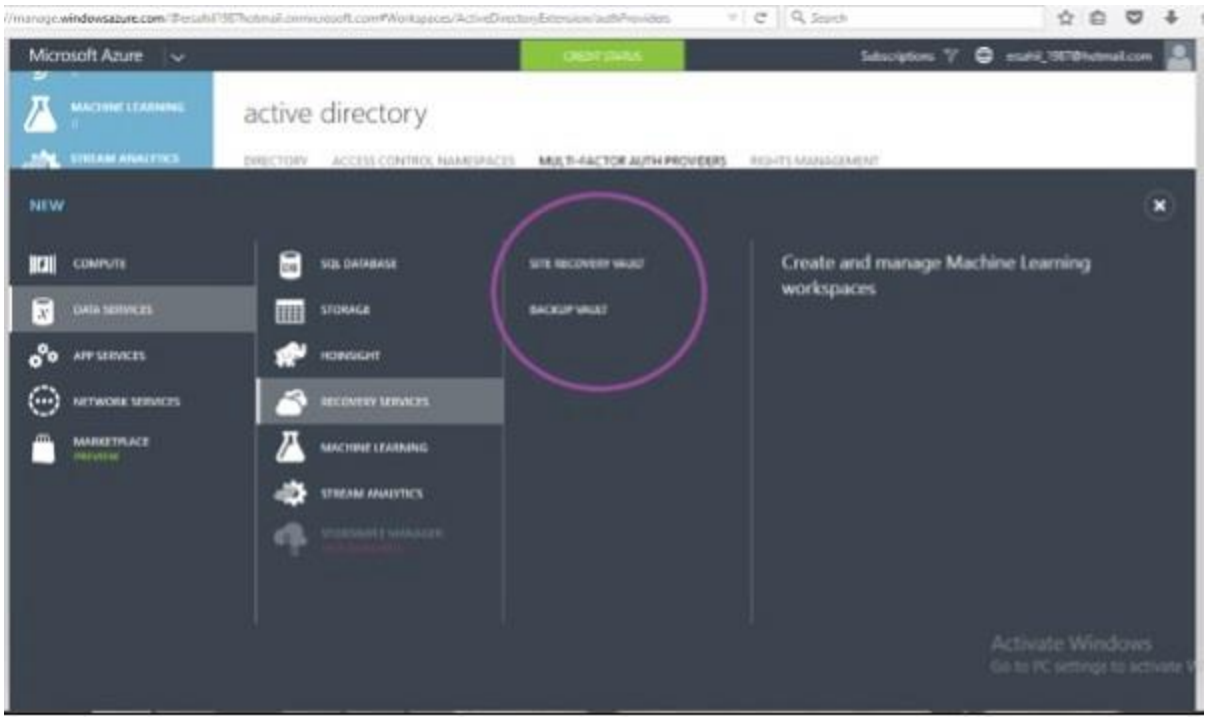
Mobile Service

Windows Azure offers a very easy platform to develop mobile application. You can simply start using mobile development tools after logging into your account. You don’t have to write big custom codes for the mobile application if you use this service. The push notifications can be sent, data can be stored and users can be authenticated in very less time.



Backup

The site recovery service replicates the data at secondary location as well as automates the process of recovery of data in case of data outage. Similarly Azure backup can be used to backing up the on premise data in clouds. Data is stored in encrypted mode in both the cases. Windows Azure offers a very effective and reliable backup service to clients and ensures they don’t face inconvenience in case of hardware failures.



Media

This service addresses multiple concerns related to uploading media and making it available to end users easily. Users can manage tasks related to the media like encoding, ad insertion, streaming, etc. easily.

Commerce

Windows Azure offers the opportunity to users to buy or sell applications and data through their platform. The applications are put in the marketplace or Azure store from where they can be accessed and bought by other users.

Software Development Kit (SDK)

Azure applications can be produced by the developers in various programming languages. Microsoft currently provides language-specific SDKs for Java, .NET, PHP, Node.js, Ruby, and Python. There is also a general Windows Azure SDK that supports language, such as C++.

MapReduce Tutorial: MapReduce Example Program

Before jumping into the details, let us have a glance at a MapReduce example program to have a basic idea about how things work in a MapReduce environment practically. I have taken the same word count example where I have to find out the number of occurrences of each word. And Don't worry guys, if you don't understand the code when you look at it for the first time, just bear with me while I walk you through each part of the MapReduce code.

MapReduce Tutorial: Explanation of MapReduce Program

The entire MapReduce program can be fundamentally divided into three parts:

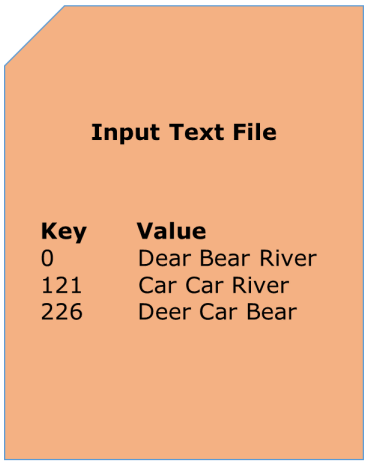
- Mapper Phase Code
- Reducer Phase Code
- Driver Code

We will understand the code for each of these three parts sequentially.

Mapper code:

```
1
2                   public static class Map extends Mapper&lt;LongWritable,Text,Text,I
3           public void map(LongWritable key, Text value, Context context) throws IOEx
4                                       String line = value.toString();
5                                StringTokenizer tokenizer = new StringTokenize
6                                       while (tokenizer.hasMoreTokens()) {
7                                          value.set(tokenizer.nextToken())
8                                      context.write(value, new IntWritable
                                                    }
```

- We have created a class Map that extends the class Mapper which is already defined in the MapReduce Framework.



- We define the data types of input and output key/value pair after the class declaration using angle brackets.
- Both the input and output of the Mapper is a key/value pair.
- Input:
  - The *key* is nothing but the offset of each line in the text file: *LongWritable*
  - The *value* is each individual line (as shown in the figure at the right): *Text*
- Output:
  - The *key* is the tokenized words: *Text*
  - We have the hardcoded *value* in our case which is 1: *IntWritable*
  - Example – Dear 1, Bear 1, etc.
- We have written a java code where we have tokenized each word and assigned them a hardcoded value equal to *1*.

Reducer Code:

```
1         public static class Reduce  extends Reducer&lt;Text,IntWritable,Text,IntWritable
2              public void reduce(Text  key,  Iterable&lt;IntWritable&gt;  values,Context  co
3                                       throws IOException,InterruptedException  {
4                                                  int sum=0;
5                                             for(IntWritable  x:  values)
6                                                        {
7                                                    sum+=x.get();
8                                                        }
9                                           context.write(key,  new IntWritable(sum));
10                                                     }
                                                     }
```

- We have created a class Reduce which extends class Reducer like that of Mapper.
- We define the data types of input and output key/value pair after the class declaration using angle brackets as done for Mapper.
- Both the input and the output of the Reducer is a key-value pair.
- Input:
    - The *key* nothing but those unique words which have been generated after the sorting and shuffling phase: *Text*
    - The *value* is a list of integers corresponding to each key: *IntWritable*
    - Example – Bear, [1, 1], etc.
- Output:
    - The *key* is all the unique words present in the input text file: *Text*
    - The *value* is the number of occurrences of each of the unique words: *IntWritable*
    - Example – Bear, 2; Car, 3, etc.
- We have aggregated the values present in each of the list corresponding to each key and produced the final answer.
- In general, a single reducer is created for each of the unique words, but, you can specify the number of reducer in mapred-site.xml.

Driver Code:

```
1
2                Configuration conf= new Configuration();
3             Job job = new Job(conf,"My Word Count Program");
4                   job.setJarByClass(WordCount.class);
5                      job.setMapperClass(Map.class);
6                   job.setReducerClass(Reduce.class);
7                   job.setOutputKeyClass(Text.class);
8                job.setOutputValueClass(IntWritable.class);
9              job.setInputFormatClass(TextInputFormat.class);
10            job.setOutputFormatClass(TextOutputFormat.class);
11                  Path outputPath = new Path(args[1]);
12      //Configuring the input/output path from the filesystem into the job
13             FileInputFormat.addInputPath(job, new Path(args[0]));
14           FileOutputFormat.setOutputPath(job, new Path(args[1]));
15
```

- In the driver class, we set the configuration of our MapReduce job to run in Hadoop.
- We specify the name of the job, the data type of input/output of the mapper and reducer.
- We also specify the names of the mapper and reducer classes.
- The path of the input and output folder is also specified.
- The method setInputFormatClass () is used for specifying how a Mapper will read the input data or what will be the unit of work. Here, we have chosen TextInputFormat so that a single line is read by the mapper at a time from the input text file.
- The main () method is the entry point for the driver. In this method, we instantiate a new Configuration object for the job.

Source code:

```
1  package co.edureka.mapreduce
2  import java.io.IOException;
3  import java.util.StringTokeniz
4  import org.apache.hadoop.io.IntWri
5  import org.apache.hadoop.io.LongWr
6  import org.apache.hadoop.io.Te
7  import org.apache.hadoop.mapreduce
8  import org.apache.hadoop.mapreduce.
9  import org.apache.hadoop.conf.Confi
10 import org.apache.hadoop.mapreduc
11 import org.apache.hadoop.mapreduce.lib.input
12 import org.apache.hadoop.mapreduce.lib.output
13 import org.apache.hadoop.mapreduce.lib.input
14 import org.apache.hadoop.mapreduce.lib.output
15 import org.apache.hadoop.fs.Pa
16
17
18 public class WordCount{
19 public static class Map extends Mapper&amp;lt;LongWritabl
20 public void map(LongWritable key, Text value,Context context)
21 String line = value.toS
22 StringTokenizer tokenizer = new St
23 while (tokenizer.hasMoreT
```

```
value.set(tokenizer.n
context.write(value, new
}
}
}
public static class Reduce extends Reducer&amp;lt;Text,IntW
```

```
24          public void reduce(Text key, Iterable&amp;lt;IntWritable&amp;gt; values,Co
25                                                                  int sum=0;
26                                                              for(IntWritable x: va
27                                                                  {
28                                                                sum+=x.get()
29                                                                  }
30                                                          context.write(key, new IntWr
31                                                                 }
32                                                              }
33                                                  public static void main(String[] args) t
34                                                          Configuration conf= new Confi
35                                                      Job job = new Job(conf,"My Word C
36                                                          job.setJarByClass(WordCount
37                                                            job.setMapperClass(Map.c
38                                                         job.setReducerClass(Reduce
39                                                         job.setOutputKeyClass(Text
40                                                       job.setOutputValueClass(IntWrit
41                                                      job.setInputFormatClass(TextInput
42                                                     job.setOutputFormatClass(TextOutpu
43                                                          Path outputPath = new Path(a
44                                                  //Configuring the input/output path from th
45                                                         FileInputFormat.addInputPath(job, ne
46                                                        FileOutputFormat.setOutputPath(job, n
47                                          //deleting the output path automatically from hdfs so tha
48                                                        outputPath.getFileSystem(conf).del
49                                                     //exiting the job only if the flag v
50                                                        System.exit(job.waitForCompletion
51                                                                 }
52                                                             }
53
54
55
56
57
```

Run the MapReduce code:

The command for running a MapReduce code is:

```
1       hadoop jar hadoop-mapreduce-example.jar WordCount /sample/input /sample/output
```

Now, we will look into a Use Case based on MapReduce Algorithm.

Use case: KMeans Clustering using Hadoop's MapReduce.

KMeans Algorithm is one of the simplest Unsupervised Machine Learning Algorithm. Typically, unsupervised algorithms make inferences from datasets using only input vectors without referring to known or labelled outcomes.

Executing the KMeans Algorithm using Python with a smaller **Dataset** or a **.csv** file is easy. But, when it comes to executing the Datasets at the level of Big Data, then the normal procedure cannot stay handy anymore.

That is exactly when you deal Big Data with Big Data tools. The Hadoop's **MapReduce.** The following code snippets are the Components of MapReduce performing the **Mapper, Reducer** and **Driver**Jobs

//Mapper Class

```
1
2       public void map(LongWritable key, Text value, OutputCollector&lt;DoubleWritable,
3                                                                   IOException {
4                                                               String line = value.toS
5                                                           double point = Double.parseD
6                                                   double min1, min2 = Double.MAX_VALUE, neare
7                                                               for (double c : mCent
8                                                                   min1 = c - poi
9                                                           if (Math.abs(min1) &lt; Ma
10                                                                  nearest_cente
11                                                                      min2 = mi
12                                                                  }
13                                                              }
14                                                  output.collect(new DoubleWritable
15                                                                  new DoubleWritabl
                                                                        }
                                                                    }
```

//Reducer Class

```
1                                                   public static class Reduce extends MapReduce
2                                                   Reducer&lt;DoubleWritable, DoubleWritable,
3                                                                       @Override
4               public void reduce(DoubleWritable key, Iterator&lt;DoubleWritable&gt; va
5                                                                   Reporter reporter)throws IOExcept
6                                                                   double newCenter
7                                                                   double sum = 0;
8                                                                 int no_elements =
9                                                                 String points = "
10                                                              while (values.hasNext
11                                                              double d = values.nex
12                                                      points = points + " " + Dou
13                                                                  sum = sum +
14                                                                  ++no_element
15                                                              }
                                                            newCenter = sum / no_e
                                                    output.collect(new DoubleWritable(newCe
                                                                }
                                                            }
```

```
16
17
18
```

//Driver Class

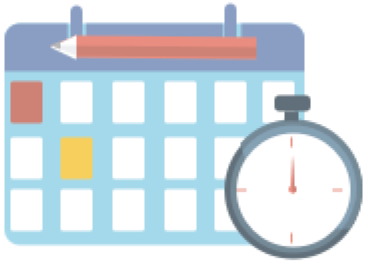
```
1  public static void run(String[] args) throws Exception {
2  IN = args[0];
3  OUT = args[1];
4  String input = IN;
5  String output = OUT + System.nanoTime();
6  String again_input = output;
7  int iteration = 0;
8  boolean isdone = false;
9  while (isdone == false) {
10 JobConf conf = new JobConf(KMeans.class);
11 if (iteration == 0) {
12 Path hdfsPath = new Path(input + CENTROID_FILE_NA
13 DistributedCache.addCacheFile(hdfsPath.toUri(), c
14 } else {
15 Path hdfsPath = new Path(again_input + OUTPUT_FIE_
16 DistributedCache.addCacheFile(hdfsPath.toUri(), c
17 }
18 conf.setJobName(JOB_NAME);
19 conf.setMapOutputKeyClass(DoubleWritable.class);
20 conf.setMapOutputValueClass(DoubleWritable.class)
21 conf.setOutputKeyClass(DoubleWritable.class);
22 conf.setOutputValueClass(Text.class);
23 conf.setMapperClass(Map.class);
24 conf.setReducerClass(Reduce.class);
25 conf.setInputFormat(TextInputFormat.class);
26 conf.setOutputFormat(TextOutputFormat.class);
27 FileInputFormat.setInputPaths(conf, new Path(input + DATA_FI
28 FileOutputFormat.setOutputPath(conf, new Path(output
29 JobClient.runJob(conf);
30 Path ofile = new Path(output + OUTPUT_FIE_NAME);
31 FileSystem fs = FileSystem.get(new Configuration()
32 BufferedReader br = new BufferedReader(new InputStreamReader(fs.
33 List<Double> centers_next = new ArrayList<Double>()
34 String line = br.readLine();
35 while (line != null) {
36 String[] sp = line.split("t| ");
37 double c = Double.parseDouble(sp[0]);
38 centers_next.add(c);
39 line = br.readLine();
40 }
41 br.close();
   String prev;
   if (iteration == 0) {
   prev = input + CENTROID_FILE_NAME;
   } else {
   prev = again_input + OUTPUT_FILE_NAME;
   }
   Path prevfile = new Path(prev);
   FileSystem fs1 = FileSystem.get(new Configuration()
   BufferedReader br1 = new BufferedReader(new InputStreamReader(fs1.
   List<Double> centers_prev = new ArrayList<Double>()
```

```
                String l = br1.readLine();
                while (l != null) {
                String[] sp1 = l.split(SPLITTER);
                double d = Double.parseDouble(sp1[0]);
                centers_prev.add(d);
                l = br1.readLine();
                }
                br1.close();
                Collections.sort(centers_next);
                Collections.sort(centers_prev);

                Iterator<Double> it = centers_prev.iterator();
                for (double d : centers_next) {
                double temp = it.next();
                if (Math.abs(temp - d) <= 0.1) {
                isdone = true;
                } else {
                isdone = false;
                break;
                }
                }
                ++iteration;
                again_input = output;
                output = OUT + System.nanoTime();
                }
                }
```

Now, we will go through the complete executable code

//Source Code



See Batch Details

```
import java.io.IOException
import java.util.*;
import java.io.*;
import org.apache.hadoop.conf.Conf
import org.apache.hadoop.filecache.Dis
import org.apache.hadoop.fs.File
import org.apache.hadoop.fs.P
import org.apache.hadoop.io
import org.apache.hadoop.mapr
import org.apache.hadoop.mapred.

@SuppressWarnings("deprecati
public class KMeans {
public static String OUT = "o
public static String IN = "inpu
public static String CENTROID_FILE_NAME
public static String OUTPUT_FILE_NAME
public static String DATA_FILE_NAME
public static String JOB_NAME =
public static String SPLITTER
public static List<Double> mCenters = new
public static class Map extends MapReduceBase implements Mapper<LongWr
@Override
public void configure(JobCo
try {
Path[] cacheFiles = DistributedCach
if (cacheFiles != null && cach
String l
mCenters.cl
BufferedReader cacheReader
new FileReader(cach
try {
while ((line = cacheReade
String[] temp =
mCenters.add(Double
}
} final
cacheReade
}
}
} catch (IOExceptio
System.err.println("Exception readi
}
```

```
36                                                                  }
37                                                                  @Override
38          public void map(LongWritable key, Text value, OutputCollector<DoubleWrit
39                                                                  IOException {
40                                                                  String line = value.to
41                                                                  double point = Double.pars
42                                                          double min1, min2 = Double.MAX_VALUE, near
43                                                                  for (double c : mCen
44                                                                  min1 = c - po
45                                                                  if (Math.abs(min1) < Ma
46                                                                  nearest_cent
47                                                                  min2 = m
48                                                                  }
49                                                                  }
50                                                          output.collect(new DoubleWritab
51                                                                  new DoubleWritab
52                                                                  }
53                                                                  }
54                                                  public static class Reduce extends MapRed
55                                                  Reducer<DoubleWritable, DoubleWritable
56                                                                  @Override
57          public void reduce(DoubleWritable key, Iterator<DoubleWritable> values
58                                                  reporter)throws IOException
59                                                                  double newCente
60                                                                  double sum = 0
61                                                                  int no_elements =
62                                                                  String points =
63                                                                  while (values.hasNe
64                                                                  double d = values.ne
65                                                                  points = points + " " + Do
66                                                                  sum = sum +
67                                                                  ++no_elemen
68                                                                  }
69                                                                  newCenter = sum / no_
70                                                          output.collect(new DoubleWritable(newC
71                                                                  }
72                                                                  }
73                                                  public static void main(String[] args)
74                                                                  run(args);
75                                                                  }
76                                                  public static void run(String[] args) t
77                                                                  IN = args[0];
78                                                                  OUT = args[1];
79                                                                  String input = IN
80                                                          String output = OUT + System
81                                                                  String again_input = o
82                                                                  int iteration = 0
83                                                                  boolean isdone = fal
84                                                                  while (isdone == fals
85                                                                  JobConf conf = new JobConf(
86                                                                  if (iteration ==
87                                                                  Path hdfsPath = new Path(input
88                                                                  DistributedCache.addCacheFile(h
89                                                                  } else {
90                                                                  Path hdfsPath = new Path(again_i
91                                                                  DistributedCache.addCacheFile(h
92                                                                  }
```

```
82                                                                  conf.setJobName(JOB
83                                                              conf.setMapOutputKeyClass(Doub
84                                                            conf.setMapOutputValueClass(Dou
85                                                              conf.setOutputKeyClass(Double
86                                                                  conf.setOutputValueClass
87                                                                conf.setMapperClass(Ma
88                                                               conf.setReducerClass(Red
89                                                             conf.setInputFormat(TextInpu
90                                                            conf.setOutputFormat(TextOutp
91                                                  FileInputFormat.setInputPaths(conf, new Pa
92                                                          FileOutputFormat.setOutputPath(co
93                                                                   JobClient.runJob(c
94                                                              Path ofile = new Path(output +
95                                                            FileSystem fs = FileSystem.get(n
96                                                  BufferedReader br = new BufferedReader(new Inp
97                                                  List<Double> centers_next = new A
98                                                               String line = br.rea
99                                                                  while (line != nu
100                                                               String[] sp = line.s
101                                                            double c = Double.parse
102                                                                    centers_next.a
103                                                                  line = br.readL
104                                                                         }
105                                                                   br.close();
106                                                                  String prev;
107                                                               if (iteration ==
108                                                             prev = input + CENTRO
109                                                                      } else {
110                                                           prev = again_input + OU
111                                                                         }
112                                                            Path prevfile = new Pa
113                                                        FileSystem fs1 = FileSystem.get(
114                                                BufferedReader br1 = new BufferedReader(new Inpu
115                                                  List<Double> centers_prev = new A
116                                                               String l = br1.read
117                                                                  while (l != null
118                                                               String[] sp1 = l.spli
119                                                            double d = Double.parse
120                                                                    centers_prev.a
121                                                                  l = br1.readLi
122                                                                         }
123                                                                  br1.close();
124                                                            Collections.sort(cente
125                                                            Collections.sort(cente
126
127                                                     Iterator<Double> it = centers
                                                                for (double d : center
                                                                    double temp = it
                                                                if (Math.abs(temp - c
                                                                        isdone =
                                                                      } else {
                                                                    isdone = f
                                                                         break
                                                                       }
                                                                    }
                                                                ++iteration;
```

```
                again_input = out
            output = OUT + System.n
                        }
                    }
                }
```

Now, you guys have a basic understanding of MapReduce framework. You would have realized how the MapReduce framework facilitates us to write code to process huge data present in the HDFS. There have been significant changes in the MapReduce framework in Hadoop 2.x as compared to Hadoop 1.x. These changes will be discussed in the next blog of this MapReduce tutorial series. I will share a downloadable comprehensive guide which explains each part of the MapReduce program in that very blog.