CBCS SCHEME

USN 1CR22MC075

22MCA412

Fourth Semester MCA Degree Examination, June/July 2024

Big Data Analytics

Time: 3 hrs. Max. Marks: 100

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M : Marks , L: Bloom's level , C: Course outcomes.

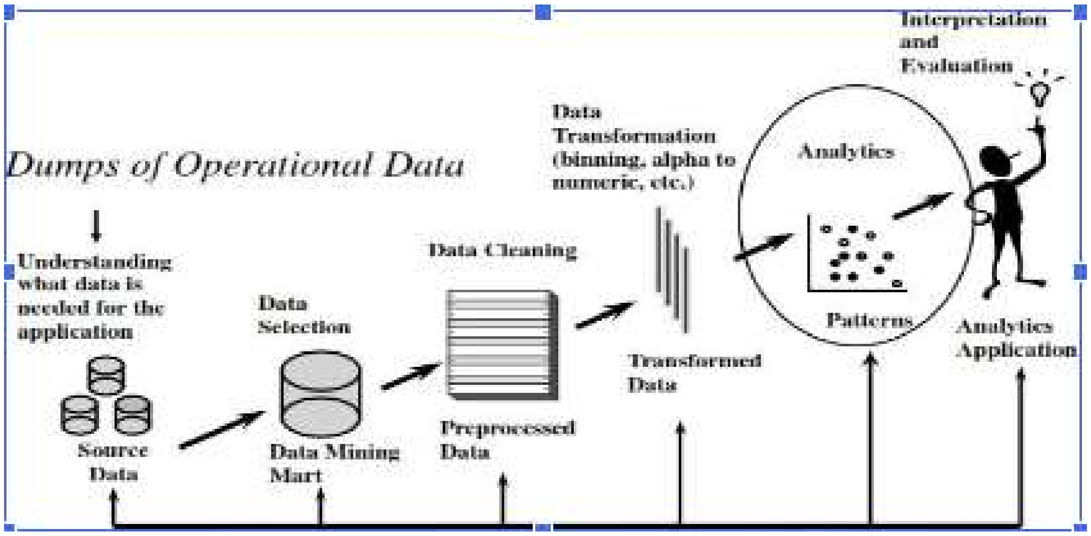
		Module – 1	M	L	C
Q.1	a.	With neat diagram, describe the working of analytical process model.	10	L2	CO1
	b.	Write a note on : i) Missing values ii) Data sampling.	10	L2	CO1
		OR			
Q.2	a.	What are the various types of data elements available for bigdata analytics? Explain.	10	L1	CO1
	b.	Calculate the z-scores and detect the outliner for the following data. Where mean $\mu = 40$ and standard deviation $\sigma = 20$, Data = 30, 50, 10, 40, 60, 80.	5	L3	CO1
	c.	Explain the different characteristics of Big Data Analytics.	5	L2	CO1
		Module – 2			
Q.3	a.	Define the terms : i) Data Discovery ii) Predictive Analytics.	10	L1	CO2
	b.	Define crowd sourcing? Explain the working of Kaggle's crowd sourcing.	10	L2	CO2
		OR			
Q.4	a.	With a neat diagram, explain athe inter and trans – firewall analytics.	10	L3	CO2
	b.	Explain mobile intelligence and Big Data.	10	L2	CO2
		Module – 3			
Q.5	a.	What is Data Flooding? Explain with an example. List out the source of Data flooding.	10	L3	CO3
	b.	Differentiate MapReduce with RDBMS and Grid Computing with MapReduce.	10	L2	CO3
		OR			
Q.6	a.	What is Volunteer Computing? Compare volunteer computing and MapReduce.	10	L3	CO3
	b.	Describe the Brief History of Hadoop? Discuss about Hadoop Releases in Detail.	10	L2	CO3
		Module – 4			
Q.7	a.	List the concept of HDFS. Explain any two concepts.	10	L1	CO4
	b.	Discuss the different file system operation.	10	L3	CO4
		OR			
Q.8	a.	Describe the Reading Data from file system API. Explain about anatomy of File Reading with a neat diagram.	10	L3	CO4
	b.	What is file system interface? Explain different file system interface.	10	L2	CO4
		Module – 5			
Q.9	a.	Explain the MapReduce function with a suitable diagram.	10	L3	CO5
	b.	What is Hadoop streaming? Discuss different streaming methods.	10	L3	CO5
		OR			
Q.10	a.	Demonstrate a Java program for MapReduce function to compute maximum temperature from the weather data set.	10	L3	CO5
	b.	Write a note on : i) Combiner function ii) Scaling out.	10	L2	CO5

* * * * *

1. a) **Analytics process model**

1. Define the business problems to be solved using analytics.
2. All source-data need to be identified that could be of potential interest. (Select of data will have a deterministic impact on the analytical model).
3. All data will be gathered in a staging area which could be data mart/ data warehouse. Basic exploratory analysis will be considered, for example: OLAP (Online Analytical Processing) facilities for multi-dimensional data analysis.
4. Data Cleaning steps to get rid of all inconsistencies – like missing data / values, outliers and duplicate data. Additional transformations may also be considered, such as binning, alphanumeric to numeric coding, geographical aggregation etc.
5. In the analytics steps, an analytical model will be estimated on the pre-processed and transformed data. Once the model is built, it will be interpreted and evaluated by the business experts. Many trivial patterns will be detected by the model.

Knowledge Pattern: Unexpected yet interesting and actionable patterns (referred to as knowledge pattern). Once analytical model has been appropriately validated and approved, it can be put into production as an analytical application.


Dumps of Operational Data
Understanding what data is needed for the application
Source Data
Data Selection
Data Mining Mart
Data Cleaning
Preprocessed Data
Data Transformation (binning, alpha to numeric, etc.)
Transformed Data
Analytics
Patterns
Interpretation and Evaluation
Analytics Application

b) **Missing Values**

Missing values can occur because of various reasons.

- Information can be non-applicable
- Information can also be undisclosed

Missing data can also originate an error during merging. Some analytical techniques (e.g., decision trees) can directly deal with missing values. They are:

- Replace (impute)
- Delete
- Keep

Replace: This implies replacing the missing value with a known value. Consider the table 1.2 one could calculate the missing credit bureau scores with the average or median of the known values. For marital status the mode can then be used.
Delete: This is the most straight forward option and consists of deleting observations or variables with lots of missing values. This, of course assumes that information is missing at random and has not meaningful interpretation and/or relationship to the target.

Keep: Missing values can be meaningful *(e.g a customer didn't disclose his/her income because if he/she is currently unemployed)*. Obviously, this is clearly related to the target *(e.g good/bad risk or churn)* and needs to be considered as a separate category.

SAMPLING

Sampling is to take a subset of past customer data and use that to build an analytical model.

- With the availability of high performance computing facilities (e.g grid computing and cloud computing) one could also directly analyze the full data set.
- Key requirement for a good sample is, it should be representative of the
- future customers on which the analytical model will be run.
- Timing aspect becomes important because customers of today are more similar to customers of tomorrow than customers of yesterday.
- Choosing the optimal time window for the sample involves the trade-off between the lost data and the recent data.
- The sample should also be taken from an average business period to get a picture of the target population that is as accurate as possible.

2. a) It is important to appropriately consider the different types of data elements at the start of the analysis. The different types of data elements can be considered:

- Continuous
- Categorical

Continuous: These are data elements that are defined on an interval that can be limited or unlimited. Examples include income, sales, RFM (recency, frequency, monetary).

Categorical: The categorical data elements are differentiated as follows:

Nominal: These are data elements that can only take on a limited let of values with no meaningful ordering in between. Examples: marital status, profession, purpose of loan.

Ordinal: These are data elements that can only take on a limited set of values with a meaningful ordering in between. Examples: credit rating; age coded as young, middle aged, and old.

Binary: These are data elements that can only take on two values. Example: gender, employment status.

Appropriately distinguishing between these different data elements is of key importance to start the analysis when importing the data into an analytics tool. For example, if marital status were to be incorrectly specified as a continuous data element, then the software would calculate its mean, standard deviation, and so on, this is obviously meaningless.

b) To calculate the Z-scores and detect the outlier for the given data, we will proceed step by step. The Z-score formula is:

\[

Z = \frac{{X - \mu}}{\sigma}

\]

Where:

- \(Z \) is the Z-score

- \(X \) is the value from the dataset

- \(\mu \) is the mean (which is 40)

- \(\sigma \) is the standard deviation (which is 20)

### Step 1: Calculate Z-scores

The dataset provided is: \(30, 50, 10, 40, 60, 80 \).

We will use the Z-score formula for each data point.

1. **For \(X = 30 \):**

\[

Z = \frac{{30 - 40}}{20} = \frac{{-10}}{20} = -0.5

\]

2. **For \(X = 50 \):**

\[

Z = \frac{{50 - 40}}{20} = \frac{{10}}{20} = 0.5

\]

3. **For \(X = 10 \):**

\[

Z = \frac{{10 - 40}}{20} = \frac{{-30}}{20} = -1.5

\]

4. **For \(X = 40 \):**

$$
Z = \frac{{40 - 40}}{20} = \frac{0}{20} = 0
$$

5. **For $X = 60$:**

$$
Z = \frac{{60 - 40}}{20} = \frac{{20}}{20} = 1
$$

6. **For $X = 80$:**

$$
Z = \frac{{80 - 40}}{20} = \frac{{40}}{20} = 2
$$

### Step 2: Z-Scores Summary

Data Value (X)	Z-Score
30	-0.5
50	0.5
10	-1.5
40	0

| 60 | 1 |

| 80 | 2 |

### Step 3: Detect the Outlier

An **outlier** is typically a data point that lies far from other points. One common method to detect outliers is by using Z-scores. Generally:

- A Z-score greater than **+3** or less than **-3** is considered an outlier in most cases.

For this dataset:

- The Z-scores range from **-1.5** to **2**, which is within the normal range.

- Therefore, **none of the data points are extreme outliers** according to the Z-score rule (i.e., no Z-scores exceed the ±3 threshold).

c) Characteristics of Big Data – 4 V's

(i) Volume – The name Big Data itself is related to a size which is enormous. Size of data plays a very crucial role in determining value out of data. Also, whether a particular data can actually be considered as a Big Data or not, is dependent upon the volume of data. Hence, **'Volume'** is one characteristic which needs to be considered while dealing with Big Data.
(ii) Variety – Variety refers to heterogeneous sources and the nature of data, both structured and unstructured. During earlier days, spreadsheets and databases were the only sources of data considered by most of the applications. Nowadays, data in the form of emails, photos, videos, monitoring devices, PDFs, audio, etc. are also being considered in the analysis applications. This variety of unstructured data poses certain issues for storage, mining and analyzing data.
(iii) Velocity – The term **'velocity'** refers to the speed of generation of data. How fast the data is generated and processed to meet the demands, determines real potential in the data.Big Data Velocity deals with the speed at which data flows in from sources like business processes, application logs, networks, and social media sites, sensors, Mobile devices, etc. The flow of data is massive and continuous.
(iv) Variability – This refers to the inconsistency which can be shown by the data at times, thus hampering the process of being able to handle and manage the data effectively.

3 a) **Data Discovery:**

Data discovery allows business users to interact with enterprise data visually to uncover hidden patterns and trends. Data discovery, in the context of IT, is the process of extracting actionable patterns from data. The extraction is general performed by humans or, in certain cases, by artificial intelligence systems.

Lot of buzz in the industry about data discovery, used to describe the new wave of business intelligence that enables users to explore data, make discoveries and uncover insights in a dynamic and intuitive way.

There are two software companies that stand out in the crowd by growing their business are: **Tableau Software and QlikTech International**. They grew through a model called as "land and expand".

It basically works by getting intuitive software in the hands of some business users to get in the door and grow upward. In order to succeed at the BI game of the "land and expand model" we need a product that is easy to use with lots of effective outputs.

The company's cofounder and chief scientist Pat Harahan: he invented the technology that helped to change the world of animation film. Even though Pat Harahan was not a software BI, he was able to bring the new creative lens in to BI software market. When we have a product that is "easy to use", it is also called as "self-service approach" named by Harahan and his colleagues. Analytics and reporting are produced by the people using the results. IT provides the infrastructure, but business people create their own reports and dashboards.

Predictive Analytics:

Predictive analytics is a branch of advanced analytics that uses historical data, statistical algorithms, and machine learning techniques to predict future outcomes. By analyzing patterns in existing data, predictive models can forecast future events, behaviors, or trends, providing valuable insights for decision-making.

Components of Predictive Analytics:

1. **Data Collection**:
   - Predictive analytics begins with data collection from various sources, including historical datasets, databases, and real-time data feeds. The more comprehensive and relevant the data, the more accurate the predictions.
2. **Data Processing**:
   - Collected data is cleaned, preprocessed, and transformed into a format suitable for analysis. This may involve handling missing values, removing duplicates, normalizing the data, and transforming categorical variables.
3. **Modeling**:
   - A variety of statistical models and machine learning algorithms are applied to the processed data. The choice of the model depends on the type of data, the problem being addressed, and the desired outcome. Some common models include:
     - **Linear regression**: For continuous outcome predictions.

        - **Logistic regression**: For binary outcomes (e.g., yes/no).
        - **Decision trees**: For classification and regression tasks.
        - **Neural networks**: For complex, non-linear patterns and deep learning.
        - **Time-series analysis**: For data with a time component, such as stock prices or sales forecasting.

4. **Evaluation and Tuning**:
    - Once the model is built, its accuracy and performance are evaluated using statistical metrics like Mean Squared Error (MSE), Root Mean Squared Error (RMSE), accuracy score, precision, recall, etc. The model is then fine-tuned by adjusting its parameters and hyperparameters to achieve the best results.
5. **Prediction**:
    - After tuning, the model is applied to new, unseen data to generate predictions. The predictions can range from numerical outcomes (e.g., sales revenue, temperature) to categorical outcomes (e.g., churn/no churn, fraud/no fraud).

b) **Crowdsourcing:** Crowdsourcing is a great way to capitalize on the resources that can build algorithms and predictive models

Kaggle: Kaggle describes itself as "an innovative solution for statistical/analytics outsourcing." That's a very formal way of saying that Kaggle manages competitions among the world's best data scientists. Here's how it works: Corporations, governments, and research laboratories are confronted with complex statistical challenges. They describe the problems to Kaggle and provide data sets. Kaggle converts the problems and the data into contests that are posted on its web site. The contests feature cash prizes ranging in value from $100 to $3 million. Kaggle's clients range in size from tiny start-ups to multinational corporations such as Ford Motor Company and government agencies such as NASA.

As per Anthony Goldbloom, Kaggle's founder and CEO: The idea is that someone comes to us with a problem, we put it up on our website, and then people from all over the world can compete to see who can produce the best solution."

Kaggle's approach is that it is truly a win-win scenario—contestants get access to real-world data (that has been carefully "anonymized" to eliminate privacy concerns) and prize sponsors reap the benefits of the contestants' creativity.

Crowdsourcing is a disruptive business model whose roots are in technology but is extending beyond technology to other areas.

There are various types of crowd sourcing, such as crowd voting, crowd purchasing, wisdom of crowds, crowd funding, and contests.

Take for example:

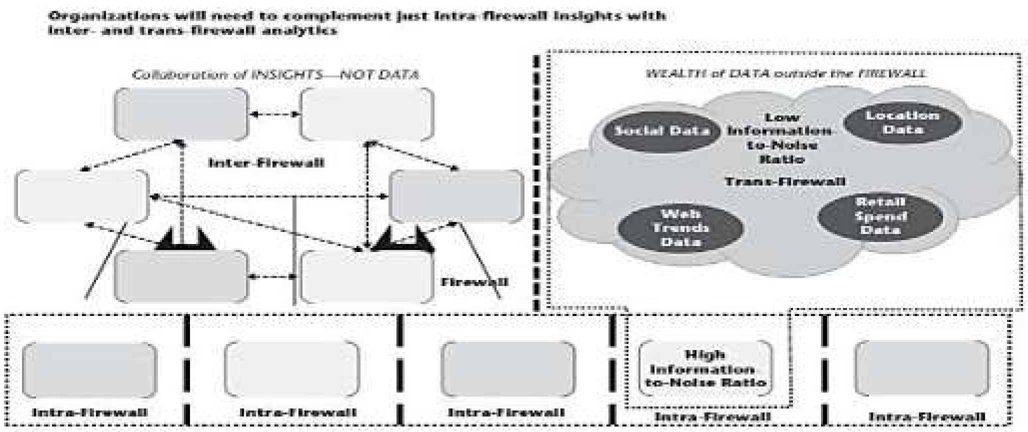
- 99designs.com/, which does crowdsourcing of graphic design
- agentanything.com/, which posts "missions" where agents vie for to run errands
- 33needs.com/, which allows people to contribute to charitable programs that make a social impact

▪

Inter- and Trans-Firewall Analytics

Over the last 100 years, supply chain has evolved to connect multiple companies and enable them to collaborate to create enormous value to the end-consumer through concepts like CPFR (collaborative planning, forecasting and replenishment—a collection of business practices that leverage the Internet and electronic data interchange to reduce inventories and expenses while improving customer service), VMI (vendor-managed inventory—a technique used by customers in which manufacturers receive sales data to forecast consumer demand more accurately), etc.

Decision sciences will witness a similar trend as enterprises begin to collaborate on insights across the value chain. For instance, in the healthcare industry, rich consumer insights can be generated by collaborating on data and insights from the health insurance provider, pharmacy delivering the drugs and the drug manufacturer. In fact, this is not necessarily limited to companies within the traditional demand-supply chain.


Organizations will need to complement just intra-firewall insights with inter- and trans-firewall analytics
Collaboration of INSIGHTS—NOT DATA
Inter-Firewall
Firewall
WEALTH of DATA outside the FIREWALL
Social Data
Low Information-to-Noise Ratio
Location Data
Trans-Firewall
Web Trends Data
Retail Spend Data
Intra-Firewall
Intra-Firewall
Intra-Firewall
High Information-to-Noise Ratio
Intra-Firewall
Intra-Firewall

There are instances where a retailer and a social media company can come together to share insights on consumer behaviour that will benefit both concerns. Some of the more progressive companies will take this a step further and work on leveraging the large volumes of data outside the firewall such as social data, location data, etc.

In other words, it will not be long before internal data and insights from within the enterprise firewall is no longer a differentiator. We call this trend the move from intra- to inter- and trans-firewall analytics. Yesterday, companies were doing functional silo-based analytics. Today they are doing intra- firewall analytics with data within the firewall. Tomorrow they will be collaborating on insights with other companies to do inter-firewall analytics as well as leveraging the public domain to do trans- firewall analytics.

Figure given below depicts, setting up inter- and trans-firewall analytics will add significant value. However, it does present some challenges.

They are:
As one moves outside the firewall, the information-to-noise ratio increases putting additional requirements on analytical methods and technology requirements

Organizations are often limited by a fear of collaboration and overreliance on proprietary information

b) Mobile Business Intelligence and Big Data

Analytics on mobile devices is what some refer to as putting BI in your pocket. Mobile drives straight to the heart of simplicity and ease of use that has been a major barrier to BI adoption since day one.

Kerzner explains his view on this topic:

We have been working on Mobile BI for a while but the iPad was the inflection point where I think it started to become mainstream. I have seen customers over the past decade who focused on the mobile space generally and mobile applications in particular. One client in particular told me that he felt like he was pushing a boulder up a hill until he introduced mobility to enhance productivity. Once the new smart phones and tablets arrived, his phone was ringing off the hook and he was trying to figure out which project to say yes to, because he couldn't say yes to everyone who suddenly wanted mobile analytics in the enterprise.

Ease of Mobile Application Deployment

Three elements that have impacted the viability of mobile BI:

1. Location—the GPS component and location . . . know where you are in time as well as the movement.
2. It's not just about pushing data; you can transact with your smart phone based on information you get.
3. Multimedia functionality allows the visualization pieces to really come into play.
4. Managing standards for rolling out these devices.
5. Managing security (always a big challenge).
6. Managing "bring your own device," where you have devices both owned by the company and devices owned by the individual, both contributing to productivity.

5. a)

We live in the data age. It' s not easy to measure the total volume of data stored electronically, but an International Data Corporation (IDC) estimate put the size of the "digital universe" at 1.48 zettabytes in 2016 and is forecasting a tenfold growth by 2020 to 4.8 zettabytes. A zettabyte is 1021 bytes, or equivalently one thousand exabytes, one million petabytes, or one billion terabytes. That is roughly the same order of magnitude as one disk drive for every person in the world.

Today's rapidly growing big data is called **data flooding**. It represents an immense opportunity for forward-thinking marketers. But to fully leverage the potential that exists within massive streams of structured and unstructured data, organizations must quickly optimize ad delivery, evaluate campaign results, improve site selection and retarget ads.

Some of the sources of data flood are:

- New York Stock Exchange generates 1 TB of new trade data per day
- Facebook hosts about 10 billion photos taking up 1 PB (=1,000 TB) of storage
- Internet Archive stores around 2 PB, and is growing at a rate of 20 TB per month
- Ancestry.com, the genealogy site, stores around 2.5 petabytes of data.
- The Large Hadron Collider near Geneva, Switzerland, will produce about 15 petabytes of data per year.

Most of the data is locked up in the largest web properties (like search engines), or scientific or financial institutions, isn't it? Does the advent of "Big Data," as it is being called, affect smaller organizations or individuals?

Sources of Data Flooding:

1. Social media platforms (e.g., Twitter during live events)
2. IoT devices (e.g., smart sensors in a smart city)
3. High-frequency trading platforms
4. Log files from servers
5. Satellite imagery or geospatial data

6 a) **Volunteer Computing projects** work by breaking the problem they are trying to solve into small chunks called work units, which are sent to computers around the world to analyse.
Example: SETI@home: It works as follows:

- It sends a work unit of 0.35 MB of radio telescope data and takes hours or days to analyse on a typical home computer. When the analysis is completed, results are sent back to the server.
- SETI, the Search for Extra-Terrestrial Intelligence, runs a project called SETI@home in which volunteers donate CPU time from their otherwise idle computers to analyze radio telescope data for signs of intelligent life outside earth.
- others include the Great Internet Mersenne Prime Search (to search for large prime numbers)
- and Folding@home (to understand protein folding and how it relates to disease).
- Volunteer computing projects work by breaking the problem they are trying to solve into chunks called *work units*, which are sent to computers around the world to be analyzed.
- For example, a SETI@home work unit is about 0.35 MB of radio telescope data, and takes hours or days to analyze on a typical home computer.
- When the analysis is completed, the results are sent back to the server, and the client gets another work unit.
- As a precaution to combat cheating, each work unit is sent to three different machines and needs at least two results to agree to be accepted.

Now the question is how MapReduce is different from Volunteer Computing.

- MapReduce also works in the similar way of breaking a problem into independent pieces that work in parallel.
- Volunteer computing problem is very CPU intensive, which makes it suitable for running on hundreds of thousands of computers across the world because the time to transfer the work unit is dwarfed by time to run the computation time.
- Volunteers are donating CPU cycle not the bandwidth.
- MapReduce is designed to run jobs that last minutes or hours on trusted, dedicated hardware running in a single data center with very high aggregate bandwidth interconnects.
- It is clear that volunteer computing runs a perpetual computation on untrusted machines on the Internet with highly variable connection speeds and no data locality.

b)

History of Hadoop

- Created by Doug Cutting, the creator of Apache Lucene, text search library
- Has its origin in Apache Nutch, an open source web search engine, a part of the Lucene project.
- 'Hadoop' was the name that Doug's kid gave to a stuffed yellow elephant toy

- In 2002, Nutch was started
  - A working crawler and search system emerged
  - Its architecture wouldn't scale to the billions of pages on the Web

  In 2003, Google published a paper describing the architecture of Google's distributed filesystem, GFS

  In 2004, Nutch project implemented the GFS idea into the Nutch Distributed Filesystem, NDFS

  In 2004, Google published the paper introducing MapReduce
- In 2005, Nutch had a working MapReduce implementation in Nutch

- By the middle of that year, all the major Nutch algorithms had been ported to run using MapReduce and NDFS

In Feb. 2006, Doug Cutting started an independent subproject of Lucene, called Hadoop

- In Jan. 2006, Doug Cutting joined Yahoo!
- Yahoo! Provided a dedicated team and the resources to turn Hadoop into a system at web scale

In Feb. 2008, Yahoo! announced its search index was being generated by a 10,000 core Hadoop cluster

In Apr. 2008, Hadoop broke a world record to sort a terabytes of data

In Nov. 2008, Google reported that its MapReduce implementation sorted one terabytes in 68 seconds.

In May 2009, Yahoo! used Hadoop to sort one terabytes in 62 seconds

Hadoop releases:

There are a few active release series. The 1.x release series is a continuation of the 0.20 release series, and contains the most stable versions of Hadoop currently available. This series includes secure Kerberos authentication, which prevents unauthorized access to Hadoop data. Almost all production clusters use these releases, or derived versions (such as commercial distributions).

The 0.22 and 0.23 release series are currently marked as alpha releases (as of early 2012), but this is likely to change by the time you read this as they get more real-world testing and become more stable (consult the Apache Hadoop releases page for the latest status). 0.23 includes several major new features:

- A new MapReduce runtime, called **MapReduce 2**, implemented on a new system called YARN (Yet Another Resource Negotiator), which is a general resource management system for running distributed applications. MapReduce 2 replaces the classic runtime in previous releases. It is described in more depth in "YARN".
- HDFS federation, which partitions the HDFS namespace across multiple namenodes to support clusters with very large numbers of files.
- HDFS high-availability, which removes the namenode as a single point of failure by supporting standby namenodes for failover.

The following figure shows the configuration of Hadoop 1.0 and Hadoop 2.0.

7 a) HDFS Concepts

The HDFS concepts covers:

- Blocks
- Namenode and Datanode
- HDFS Federation
- HDFS High Availability

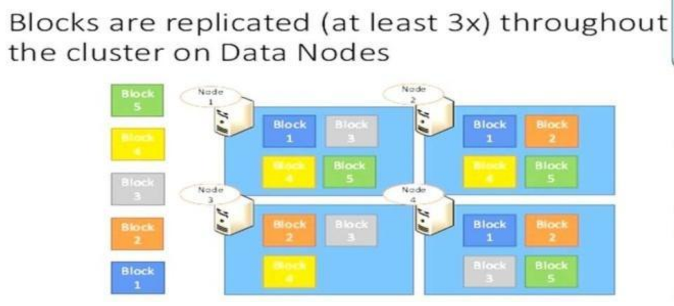
Blocks:

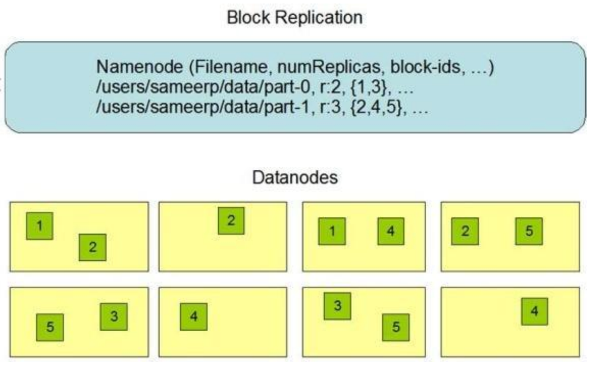
Definition: A disk has a block size, which is the minimum amount of data that it can read or write. Filesystems for a single disk build on this by dealing with data in blocks, which are an integral multiple of the disk block size. Filesystem blocks are typically a few kilobytes in size, while disk blocks are normally 512 bytes.

HDFS too has the concept of a block, but it is a much larger unit of 64 MB or 128MB by default. Like in a filesystem for a single disk, files in HDFS are broken into block-sized chunks, which are stored as independent units. Unlike a filesystem for a single disk, a file in HDFS that is smaller than a single block does not occupy a full blocks worth of underlying storage. There are tools to perform filesystem maintenance, such as *df* and *fsck*, that operate on the filesystem block level.

Benefits of using block abstraction

1. *A file can be larger than any single disk in the network*: There's nothing that requires the blocks from a file to be stored on the same disk, so they can take advantage of any of the disks in the cluster. In fact, it would be possible, if unusual, to store a single file on an HDFS cluster whose blocks filled all the disks in the cluster.
2. *Making the unit of abstraction a block rather than a file simplifies the storage subsystem:* Simplicity is something to strive for all in all systems, but is especially important for a distributed system in which the failure modes are so varied.
3. Blocks are just a chunk of data to be stored—file metadata such as permissions information does not need to be stored with the blocks, so another system can handle metadata separately. *Blocks fit well with replication for providing fault tolerance and availability.* To insure against corrupted blocks and disk and machine failure, each block is replicated to a small number of physically separate machines (**typically three)**. *If a block becomes unavailable, a copy can be read from another location in a way that is transparent to the client.*


Blocks are replicated (at least 3x) throughout the cluster on Data Nodes
Block 5
Block 4
Block 3
Block 2
Block 1
Node 1
Node 2
Node 3
Node 4


Block Replication
Namenode (Filename, numReplicas, block-ids, ...)
/users/sameerp/data/part-0, r:2, {1,3}, ...
/users/sameerp/data/part-1, r:3, {2,4,5}, ...
Datanodes

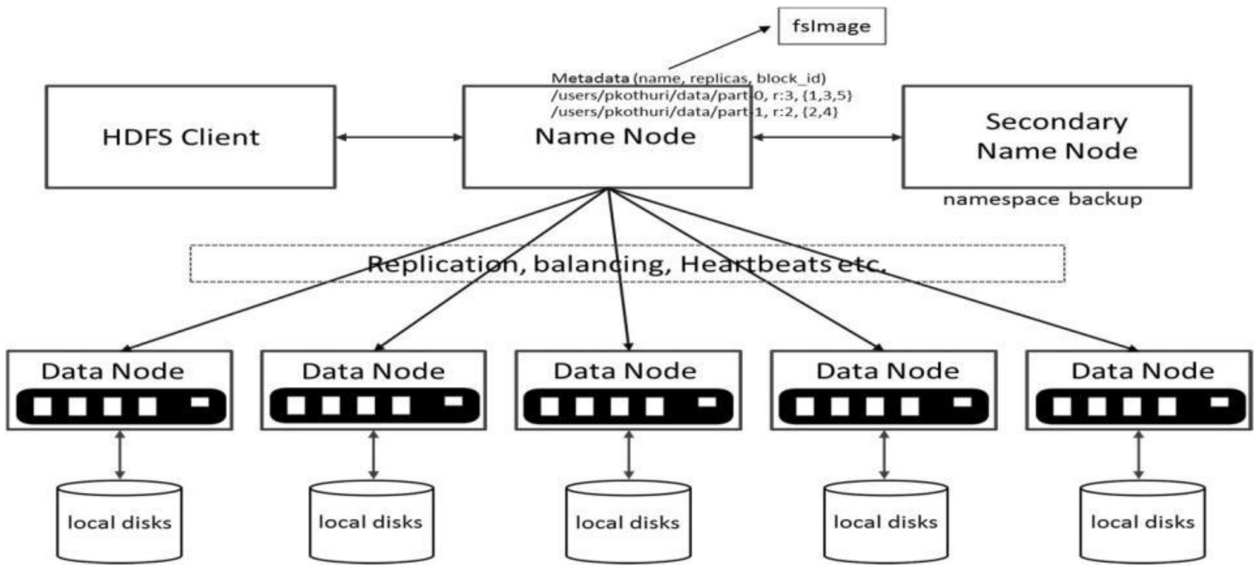
Name node and data node:

A HDFS cluster has two types of node operating in a master-worker pattern: a namenode (the master) and a number of datanodes (workers).

The **Namenode** manages the *filesystem namespace*. It maintains the *filesystem tree* and the *metadata* for all the files and directories in the tree. This information is stored persistently on the local disk in the form of two files:

- The namespace image (fsImage)
- The edit log

The namenode also knows the datanodes on which all the blocks for a given file are located, however, it does not store block locations persistently, since this information is reconstructed from datanodes when the system starts. A *client* accesses the filesystem on behalf of the user by communicating with the namenode and datanodes.


fsImage
Metadata (name, replicas, block_id)
/users/pkothuri/data/part 0, r:3, {1,3,5}
/users/pkothuri/data/part 1, r:2, {2,4}
HDFS Client
Name Node
Secondary Name Node
namespace backup
Replication, balancing, Heartbeats etc.
Data Node
local disks

Datanodes are the workhorses of the filesystem. They store and retrieve blocks when they are told to (by clients or the namenode), and they report back to the namenode periodically with lists of blocks that they are storing.

b) File system operations:

The filesystem is ready to be used, and we can do all of the usual filesystem operations

```
% hadoop fs -mkdir books
```

Creates a directory called books on the Hadoop File system. In the above command

hadoop indicates it is a Hadoop command

fs file system command

-mkdir command to execute on hdfs, ie creation of a directory Books

```
% hadoop fs -ls .
Found 2 items
drwxr-xr-x - tom supergroup 0 2009-04-02 22:41 /user/tom/books
-rw-r--r-- 1 tom supergroup 118 2009-04-02 22:29 /user/tom/quangle.txt
```

The information returned is very similar to the Unix command ls -l, with a few minor differences. The first column shows the file mode. The second column is the replication factor of the file (something a traditional Unix filesystem does not have). Remember we set the default replication factor in the site-wide configuration to be 1, which is why we see the same value here. The entry in this column is empty for directories since the concept of replication does not apply to them—directories are treated as metadata and stored by the namenode, not the datanodes. The third and fourth columns show the file owner and group. The fifth column is the size of the file in bytes, or zero for directories. The sixth and seventh columns are the last modified date and time. Finally, the eighth column is the absolute name of the file or directory.

```
%hadoop fs -help <<command>>
```

– to get detailed help on every command.

```
% hadoop fs -copyFromLocal input/docs/quangle.txt
                                        hdfs://localhost/user/tom/quangle.txt
% hadoop fs -copyFromLocal input/docs/quangle.txt
                                                        /user/tom/quangle.txt
  % hadoop fs -copyFromLocal input/docs/quangle.txt quangle.txt
```

- Start by copying a file from the local filesystem to HDFS

```
% Hadoop fs -copyToLocal hdfs://Buangle.txt c:/input/docs/Buangle.txt
```

- Start by copying a file from the HDFS to local filesystem

Hadoop FS changing the file permissions

Command: chmod

```
hadoop fs -chmod [-R] <MODE[,MODE]...| OCTALMODE> URI [URI …]
```

- Change the permissions of files. With -R, make the change recursively through the directory structure. The user must be the owner of the file, or else a super-user.

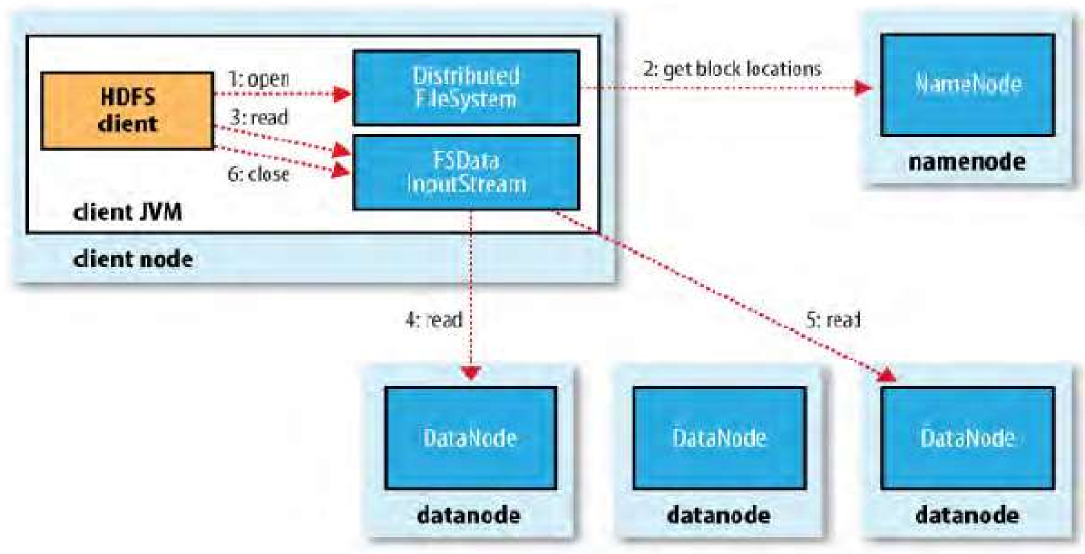
```
% hadoop fs -chmod 744 /user/training/samplezero.txt
```

– The above command changes the samplezero.txt permission

8. a) File read :

Anatomy of a File Read

To get an idea of how data flows between the client interacting with HDFS, the name-node and the datanodes, consider Figure 3-2, which shows the main sequence of events when reading a file.



The client opens the file it wishes to read by calling open() on the FileSystem object, which for HDFS is an instance of DistributedFileSystem (step 1 in Figure 3-2). DistributedFileSystem calls the namenode, using RPC, to determine the locations of the blocks for the first few blocks in the file (step 2). For each block, the namenode returns the addresses of the datanodes that have a copy of that block. Furthermore, the

datanodes are sorted according to their proximity to the client (according to the topology of the cluster's network; see "Network Topology and Hadoop" on page 71). If

the client is itself a datanode (in the case of a MapReduce task, for instance), then it will read from the local datanode, if it hosts a copy of the block (see also Figure 2-2).

The DistributedFileSystem returns an FSDataInputStream (an input stream that supports file seeks) to the client for it to read data from. FSDataInputStream in turn wraps

a DFSInputStream, which manages the datanode and namenode I/O.
The client then calls read() on the stream (step 3). DFSInputStream, which has stored the datanode addresses for the first few blocks in the file, then connects to the first (closest) datanode for the first block in the file. Data is streamed from the datanode back to the client, which calls read() repeatedly on the stream (step 4). When the end of the block is reached, DFSInputStream will close the connection to the datanode, then find the best datanode for the next block (step 5). This happens transparently to the client, which from its point of view is just reading a continuous stream.
Blocks are read in order with the DFSInputStream opening new connections to datanodes
as the client reads through the stream. It will also call the namenode to retrieve the

datanode locations for the next batch of blocks as needed. When the client has finished reading, it calls close() on the FSDataInputStream (step 6).

During reading, if the DFSInputStream encounters an error while communicating with a datanode, then it will try the next closest one for that block. It will also remember datanodes that have failed so that it doesn't needlessly retry them for later blocks. The DFSInputStream also verifies checksums for the data transferred to it from the datanode.
If a corrupted block is found, it is reported to the namenode before the DFSInput Stream attempts to read a replica of the block from another datanode.
One important aspect of this design is that the client contacts datanodes directly to retrieve data and is guided by the namenode to the best datanode for each block. This design allows HDFS to scale to a large number of concurrent clients, since the data traffic is spread across all the datanodes in the cluster. The namenode meanwhile merely
has to service block location requests (which it stores in memory, making them very

efficient) and does not, for example, serve data, which would quickly become a bot-tleneck as the number of clients grew.

b) **File System Interface in Big Data Analytics**

In the context of Big Data Analytics, a **file system interface** refers to the mechanism that provides access to the storage system, enabling users or applications to read, write, organize, and manage data files. This interface allows users to interact with large volumes of data stored across distributed systems. It abstracts the complexity of physical data storage, allowing operations like file creation, deletion, reading, and updating, while providing an organized way to store massive amounts of data.

Big Data environments often rely on specialized file systems that handle distributed and parallel storage across multiple machines. These file systems must efficiently manage high-throughput, scalable storage and ensure fault tolerance, making them different from traditional file systems.

### Types of File System Interfaces in Big Data Analytics

1. **Hadoop Distributed File System (HDFS)**:

   - **HDFS** is the core file system interface for the Hadoop ecosystem. It is designed for storing large data sets distributed across multiple nodes in a cluster. HDFS handles massive amounts of unstructured data by splitting files into blocks and distributing them across various nodes. It is optimized for large-scale data processing and fault tolerance.

  - **Key features**:

    - Data is stored in a distributed manner (blocks are replicated across nodes for redundancy).

    - Fault-tolerant, ensuring data availability even if some nodes fail.

    - Optimized for batch processing with high throughput.

    - Example operations: `hdfs dfs -put`, `hdfs dfs -get`, `hdfs dfs -ls`.

2. **Amazon S3 (Simple Storage Service)**:

  - **Amazon S3** is a scalable, object-based storage service commonly used in cloud environments. It provides a file system-like interface but is object storage-based. S3 is used in conjunction with various big data processing tools (like Amazon EMR or Hadoop) to store and retrieve data.

  - **Key features**:

    - Infinite scalability and durability.

    - Cost-effective cloud storage for large datasets.

    - Accessible via API for direct application integration.

    - Example operations: `s3 cp`, `s3 sync`.

3. **Google Cloud Storage (GCS)**:

  - **Google Cloud Storage** is another cloud-based object storage system with a file system interface. It is used in data-intensive applications like machine learning and big data analytics.

  - **Key features**:

    - High availability and durability.

    - Suitable for both structured and unstructured data.

    - Integration with Google's big data processing tools (e.g., BigQuery, Dataflow).

    - Example operations: `gsutil cp`, `gsutil ls`.

4. **Azure Data Lake Storage (ADLS)**:

- **Azure Data Lake Storage** is a file system interface designed for big data workloads, integrated into Microsoft's Azure cloud services. It supports hierarchical file structures, similar to HDFS, making it a preferred choice for Hadoop and Spark workloads.

- **Key features**:

  - Unlimited scalability with hierarchical namespace.

  - Built for distributed data processing applications.

  - Tight integration with Azure services.

  - Example operations: `az storage fs directory create`, `az storage blob upload`.

5. **MapR File System (MapR-FS)**:

- **MapR-FS** is a distributed file system that is part of the MapR Data Platform. It is optimized for big data processing and analytics, combining the benefits of HDFS with additional features like native support for containers and POSIX compliance.

- **Key features**:

  - Unified storage for files, tables, and streams.

  - High availability and low-latency operations.

  - Supports real-time applications and file-level data processing.

  - Example operations: `maprcli`, `hadoop fs`.

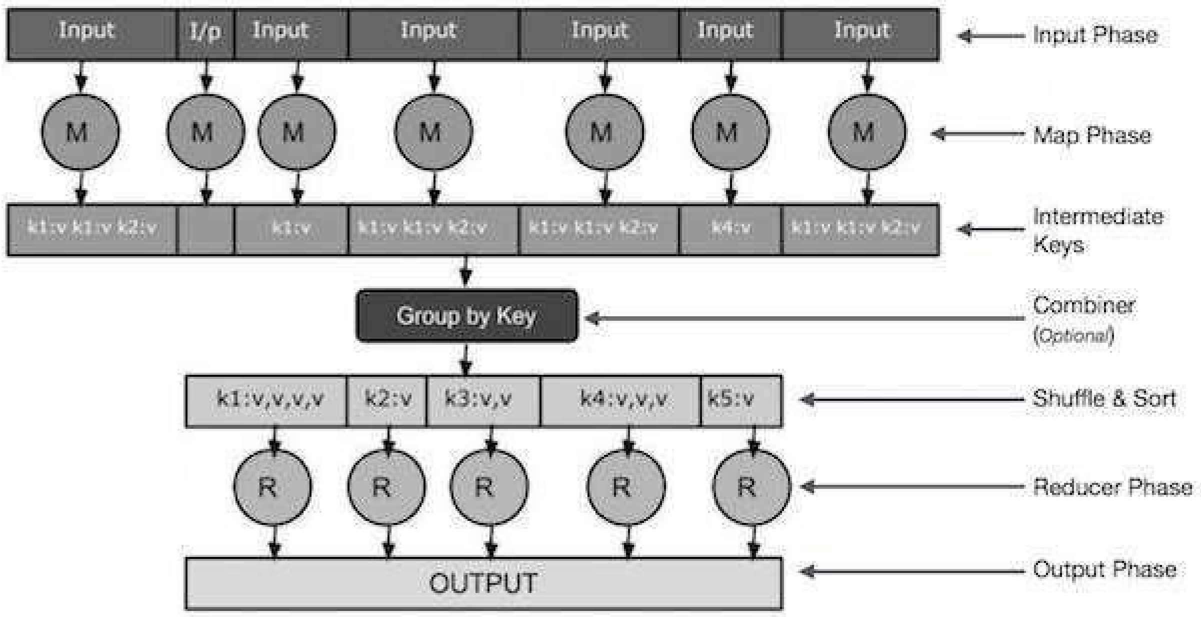
10 a) The MapReduce algorithm contains two important tasks, namely Map and Reduce.

- The Map task takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key-value pairs).
- The Reduce task takes the output from the Map as an input and combines those data

tuples (key- value pairs) into a smaller set of tuples.

The reduce task is always performed after the map job.

Let us now take a close look at each of the phases and try to understand their significance.

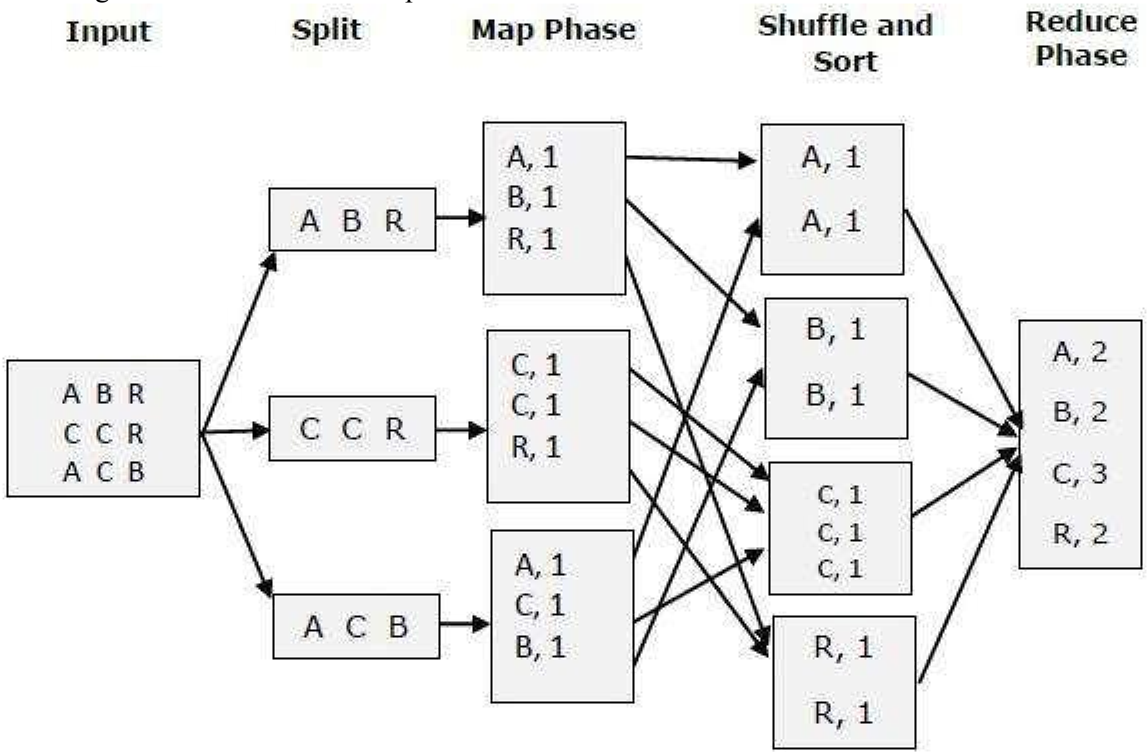


- **Input Phase** − Here we have a Record Reader that translates each record in an input file and sends the parsed data to the mapper in the form of key-value pairs.

- **Map** − Map is a user-defined function, which takes a series of key-value pairs and processes each one of them to generate zero or more key-value pairs.
- **Intermediate Keys** − They key-value pairs generated by the mapper are known as intermediate keys.
- **Combiner** − A combiner is a type of local Reducer that groups similar data from the map phase into identifiable sets. It takes the intermediate keys from the mapper as input and applies a user-defined code to aggregate the values in a small scope of one mapper. It is not a part of the main MapReduce algorithm; it is optional.
- **Shuffle and Sort** − The Reducer task starts with the Shuffle and Sort step. It downloads the grouped key-value pairs onto the local machine, where the Reducer is running. The individual key-value pairs are sorted by key into a larger data list. The data list groups the equivalent keys together so that their values can be iterated easily in the Reducer task.

- **Reducer** − The Reducer takes the grouped key-value paired data as input and runs a Reducer function on each one of them. Here, the data can be aggregated, filtered, and combined in a number of ways, and it requires a wide range of processing. Once the execution is over, it gives zero or more key-value pairs to the final step.
- **Output Phase** − In the output phase, we have an output formatter that translates the final key-value pairs from the Reducer function and writes them onto a file using a record writer.

Let us try to understand the two tasks Map &f Reduce with the help of a small diagram – Word Count Example.



b) Hadoop Streaming
Hadoop provides an API to MapReduce that allows you to write your map and reduce
functions in languages other than Java. Hadoop Streaming uses Unix standard streams
as the interface between Hadoop and your program, so you can use any language that
can read standard input and write to standard output to write your MapReduce program.
Streaming is naturally suited for text processing (although, as of version 0.21.0, it can
handle binary streams, too), and when used in text mode, it has a line-oriented view of

data. Map input data is passed over standard input to your map function, which pro-
cesses it line by line and writes lines to standard output. A map output key-value pair

is written as a single tab-delimited line. Input to the reduce function is in the same

format—a tab-separated key-value pair—passed over standard input. The reduce func-
tion reads lines from standard input, which the framework guarantees are sorted by

key, and writes its results to standard output.

Let's illustrate this by rewriting our MapReduce program for finding maximum tem-
peratures by year in Streaming.

Different Streaming Methods in Big Data

Streaming methods in Big Data refer to the real-time or near-real-time processing of continuous data streams. In contrast to batch processing, where data is processed in large, discrete chunks, streaming allows for ongoing, instantaneous data analysis as it is generated. Hadoop Streaming is just one approach, but several other frameworks and methods exist to handle streaming data in Big Data contexts. These include:

1. Apache Storm:

Apache Storm is a distributed real-time computation system for processing data streams. It is highly scalable and fault-tolerant, designed to process vast amounts of data in parallel across a cluster of nodes.

- **Key Features**:
    - **Real-time processing**: Unlike batch processing, Storm processes data in real-time as it arrives.
    - **Distributed architecture**: It distributes tasks across a cluster for parallel computation.
    - **Reliability**: Storm ensures data processing even in the event of node failures.
    - **Low Latency**: Storm processes events in milliseconds, providing near-instantaneous insights.
- **Use Case**: Apache Storm is used for real-time analytics, machine learning, and continuous monitoring tasks such as fraud detection or log analysis.

2. Apache Spark Streaming:

Apache Spark Streaming is an extension of the core Apache Spark framework that provides scalable and fault-tolerant stream processing. It allows real-time data to be processed as a series of micro-batches, thus bridging the gap between batch processing and stream processing.

- **Key Features**:
  - **Micro-batching**: Spark Streaming collects real-time data in small batches and processes them in parallel across the cluster.
  - **Integration with batch processing**: Spark Streaming allows for the use of the same code and APIs as batch jobs, making it easy to transition between batch and streaming modes.
  - **Fault tolerance**: Spark's resilient distributed datasets (RDDs) ensure that data is not lost in case of failures.
- **Use Case**: Spark Streaming is widely used for real-time data analytics, like monitoring server logs, real-time marketing campaigns, or real-time anomaly detection in data streams.

3. Apache Flink:

Apache Flink is a powerful stream processing framework that provides low-latency data processing for both bounded and unbounded datasets. It is designed to handle both batch and stream processing in a unified manner.

- **Key Features**:
  - **Event-driven processing**: Flink processes data based on events, allowing for real-time data analytics.
  - **Time-based windows**: Flink supports advanced time-based windows, such as sliding and tumbling windows, for stream processing.
  - **Exactly-once semantics**: Flink guarantees that data is processed exactly once, avoiding duplication or data loss.
- **Use Case**: Flink is used in applications like real-time fraud detection, streaming analytics, and IoT (Internet of Things) data processing.

4. Kafka Streams:

Kafka Streams is a client-side library that allows applications to process and transform data streams in real-time within an Apache Kafka cluster. Kafka Streams is designed to work natively with Kafka topics, offering scalability and reliability.

- **Key Features**:
  - **Stream Processing within Kafka**: Kafka Streams provides stream processing directly inside Kafka, reducing overhead and increasing efficiency.
  - **High throughput and scalability**: Kafka Streams is designed for high-performance and fault-tolerant stream processing.
  - **Windowing and aggregation**: It supports operations like windowing, joining, and aggregations on data streams.

- **Use Case**: Kafka Streams is widely used in financial services for real-time transaction analysis and in retail for processing real-time customer activity data.

5. Kinesis (AWS Kinesis Data Streams):

Amazon Kinesis is a fully managed service for real-time data streaming in the AWS ecosystem. It allows developers to build applications that can continuously collect and process large streams of data in real-time.

- **Key Features**:
  - **Seamless integration with AWS services**: Kinesis integrates with AWS Lambda, Redshift, and other services for real-time data analytics.
  - **Serverless architecture**: Kinesis abstracts the complexity of managing the underlying infrastructure, allowing developers to focus on their data pipelines.
  - **Horizontal scaling**: Kinesis can scale to process terabytes of data per hour.
- **Use Case**: Amazon Kinesis is used in scenarios like monitoring real-time sensor data, processing log files, and analyzing video streams.

10 a) 1. Mapper for maximum temperature example

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class MaxTemperatureMapper extends Mapper&lt;LongWritable, Text, Text,

IntWritable&gt; {

private static final int MISSING = 9999;

@Override

public void map(LongWritable key, Text value, Context context) throws

IOException,InterruptedException {

String line = value.toString();

String year = line.substring(15, 19);

int airTemperature;
if (line.charAt(87) == &#39;+&#39;) { // parseInt doesn&#39;t like leading plus signs

airTemperature = Integer.parseInt(line.substring(88, 92));

} else {

airTemperature = Integer.parseInt(line.substring(87, 92));

}

String quality = line.substring(92, 93);

if (airTemperature != MISSING &amp;&amp; quality.matches(&quot;[01459]&quot;)) {

context.write(new Text(year), new IntWritable(airTemperature));

}

}

}
```

2. Reducer for maximum temperature example

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class MaxTemperatureReducer extends Reducer&lt;Text, IntWritable, Text,

IntWritable&gt; {

@Override

public void reduce(Text key, Iterable&lt;IntWritable&gt; values, Context context)

throws IOException, InterruptedException {

int maxValue = Integer.MIN_VALUE;

for (IntWritable value : values) {

maxValue = Math.max(maxValue, value.get());

}
context.write(key, new IntWritable(maxValue));

}

}
```

b) **Combiner Functions**

Many MapReduce jobs are limited by the bandwidth available on the cluster, so it pays
to minimize the data transferred between map and reduce tasks. Hadoop allows the

user to specify a combiner function to be run on the map output—the combiner func-
tion's output forms the input to the reduce function. Since the combiner function is an

optimization, Hadoop does not provide a guarantee of how many times it will call it

for a particular map output record, if at all. In other words, calling the combiner func-
tion zero, one, or many times should produce the same output from the reducer.

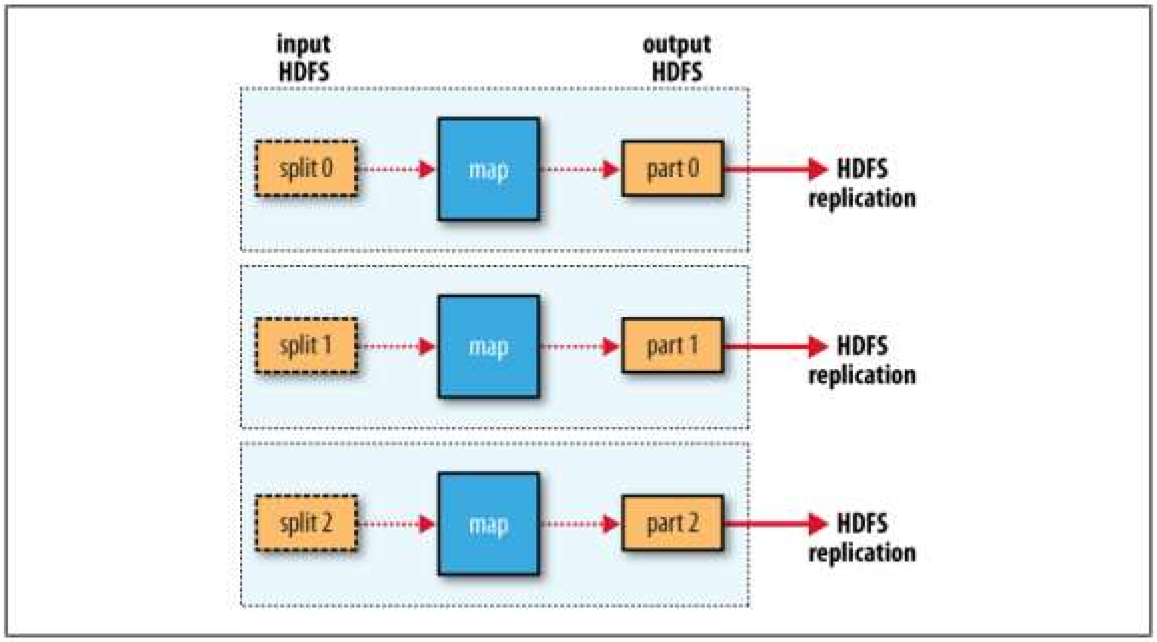


Figure 2-5. MapReduce data flow with no reduce tasks

Scaling out:

You’ve seen how MapReduce works for small inputs; now it’s time to take a bird’s-eye
view of the system and look at the data flow for large inputs. For simplicity, the
examples so far have used files on the local filesystem. However, to scale out, we need
to store the data in a distributed filesystem, typically HDFS (which you’ll learn about
in the next chapter), to allow Hadoop to move the MapReduce computation to each
machine hosting a part of the data. Let’s see how this works.