Introduction to Map Reduce Programming and *Big Data Analytics*

Syllabus:

Introduction to Map Reduce Programming: *Introduction, Mapper, Reducer, Combiner, Partitioner, Searching, sorting, compression.*

Big Data Analytics: *Big Data Analytics. Classification of Analytics, Greatest challenges on Big Data, Big Data Analytics importance, Data Science, Terminologies in Big Data.*

Resources: *https://data-flair.training/blogs/hadoop-mapreduce-tutorial/ , Textbook-1*

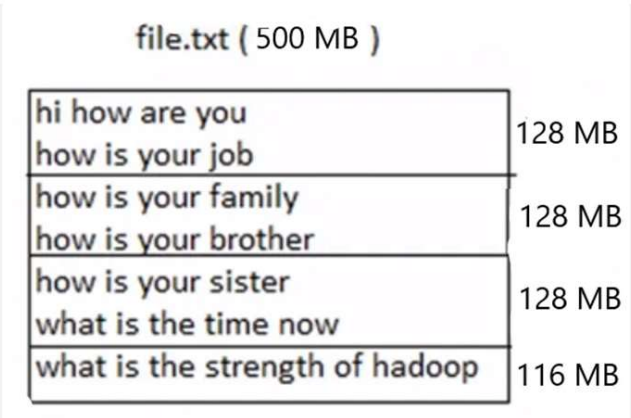
- MapReduce is a programming model for data processing. Hadoop can run MapReduce programs written in Java, Ruby and Python.
- MapReduce programs are inherently parallel, thus very large scale data analysis can be done fastly.
- In MapReduce programming, Jobs(applications) are split into a set of **map tasks and reduce tasks**.
- Map task takes care of **loading, parsing, transforming and filtering.**
- The responsibility of reduce task is **grouping and aggregating** data that is produced by map tasks to generate final output.
- Hadoop assigns **map tasks to the DataNode** where the actual data to be processed resides. This way, Hadoop ensures data locality.
- Data locality means that data is not moved over network; only **computational code is moved** to process data which saves network bandwidth.

MAPREDUCE ANATOMY

- In a typical Anatomy of MapReduce job, input files are read from the Hadoop Distributed File System (HDFS).
- The process begins with input data stored on HDFS (Hadoop Distributed File System), a storage system used in Hadoop for managing large datasets. The input data is first processed in an "Input Format" phase, where it is divided into *Input Splits*. These splits divide the data into manageable chunks for parallel processing.

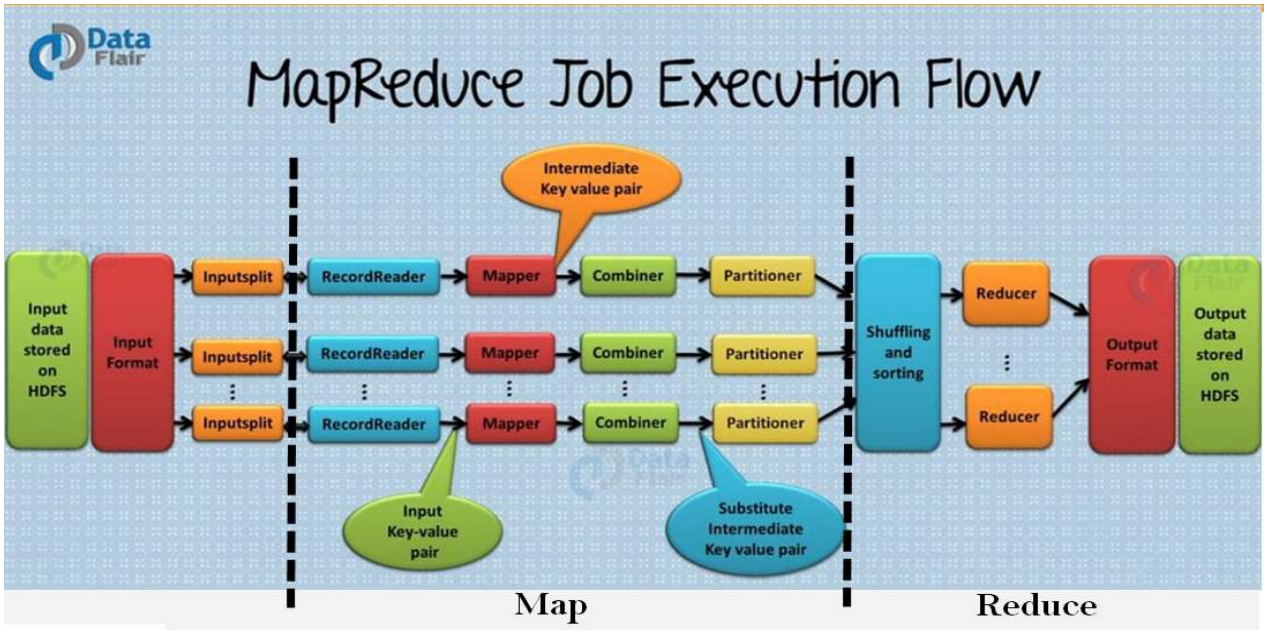
Example:

- Lets take a file — file.txt of 500 MB size. Default block size is 128 MB in HDFS. file.txt is divide into four input splits of (128 MB, 128 MB, 128 MB, 116 MB)



- Four input splits each assigned with a mapper task (i.e., No of input splits = No of map task). We cannot change the number of map tasks to a file. It takes based on the number of input splits.

- In MapReduce programming, Jobs(applications) are split into a set of **map tasks(Mapper Phases) and reduce tasks**.



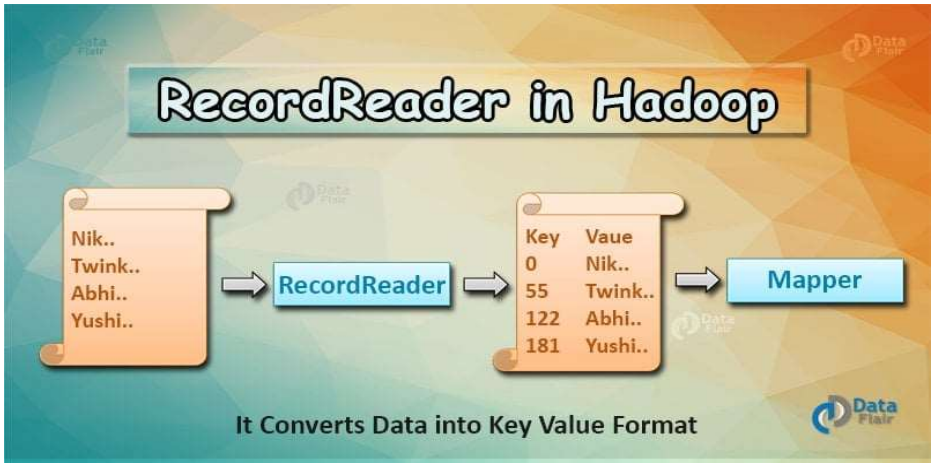
- Each **map** task(**Mapper Phases**) is broken down into the following phases:
  1. Record Reader
  2. Mapper
  3. Combiner
  4. Partitioner.
- The output produced by the map task is known as **intermediate <keys, value> pairs**. These intermediate <keys, value> pairs are sent to reducer.
- The **reduce** tasks are broken down into the following phases:
  1. Shuffle
  2. Sort
  3. Reducer
  4. Output format.

Mapper Phases

The *Mapper* processes each key-value pair independently, performing tasks such as filtering or transforming the data. The output of this phase is also in the form of key-value pairs, called intermediate key-value pairs.

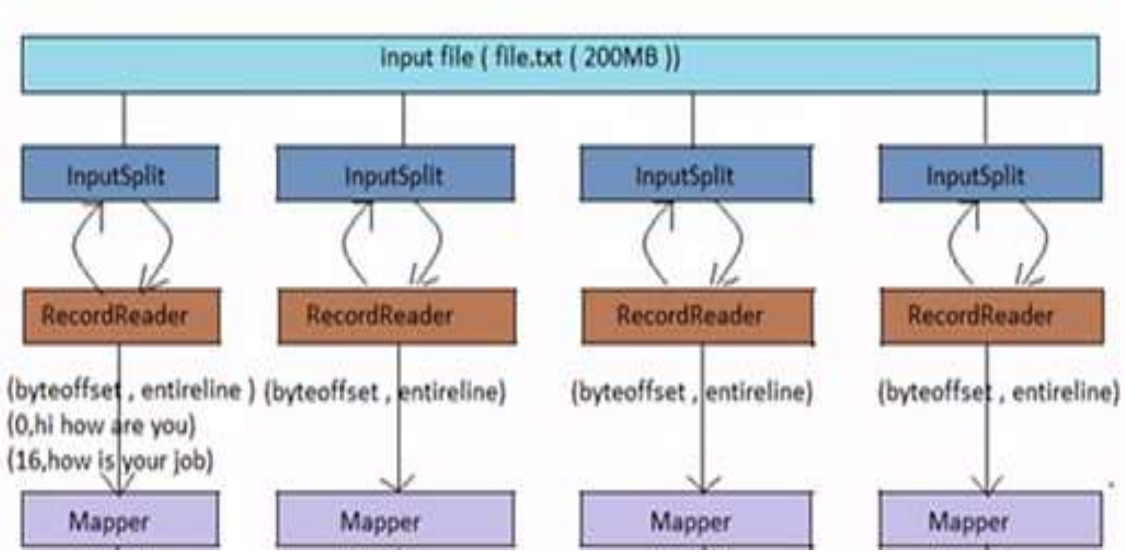
Mapper Phases- 1. RecordReader

- Each input split is processed by a *RecordReader*.
- **RecordReader** converts **byte oriented view** of input in to **Record oriented view** and presents it to the Mapper tasks.
- It converts it into key-value pairs (the basic data unit for MapReduce) that can be processed by the *Mapper*.
- Generally the key is the positional information and value is a chunk of data that constitutes the record.



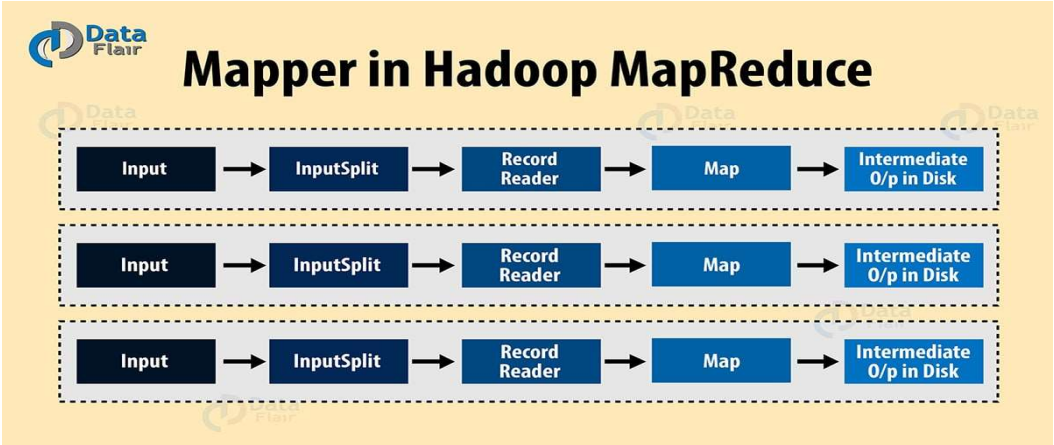
Example:

- Here RecordReader reads one line at a time and it will input the mapper class with key, value pair (byteoffset, entire line). Byteoffset value is the number of character in a line for example "hi how are you\n" has 15 chars so the next line will start from 16. and mapper class will produce a key value pair in TextInputFormat (defalut).

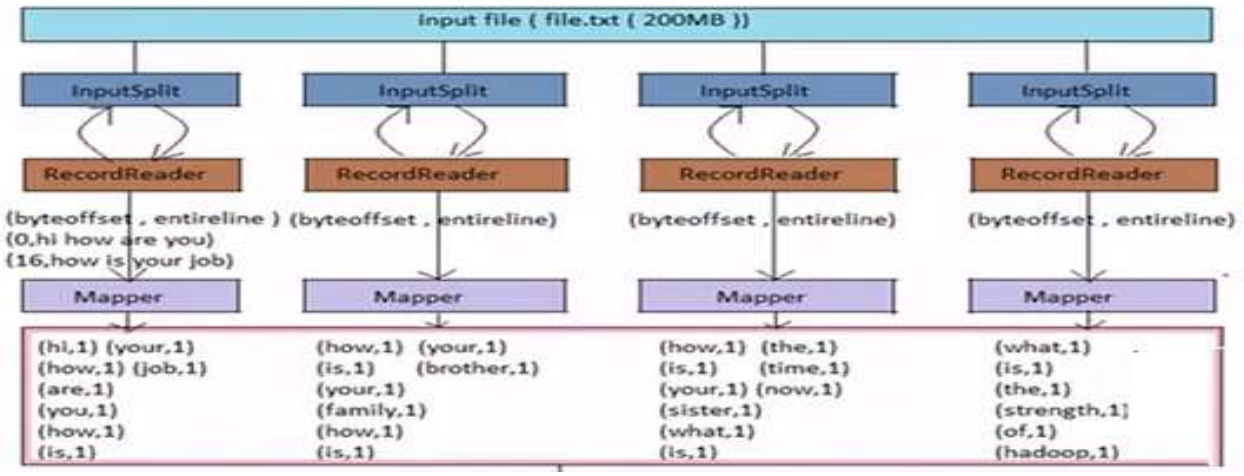


Mapper Phases- 2. Mapper

- Map function works on the <keys, value> pairs produced by RecordReader and generates zero or more intermediate (key, value) pairs.
- The MapReduce decides the key-value pair based on the context.



Example:



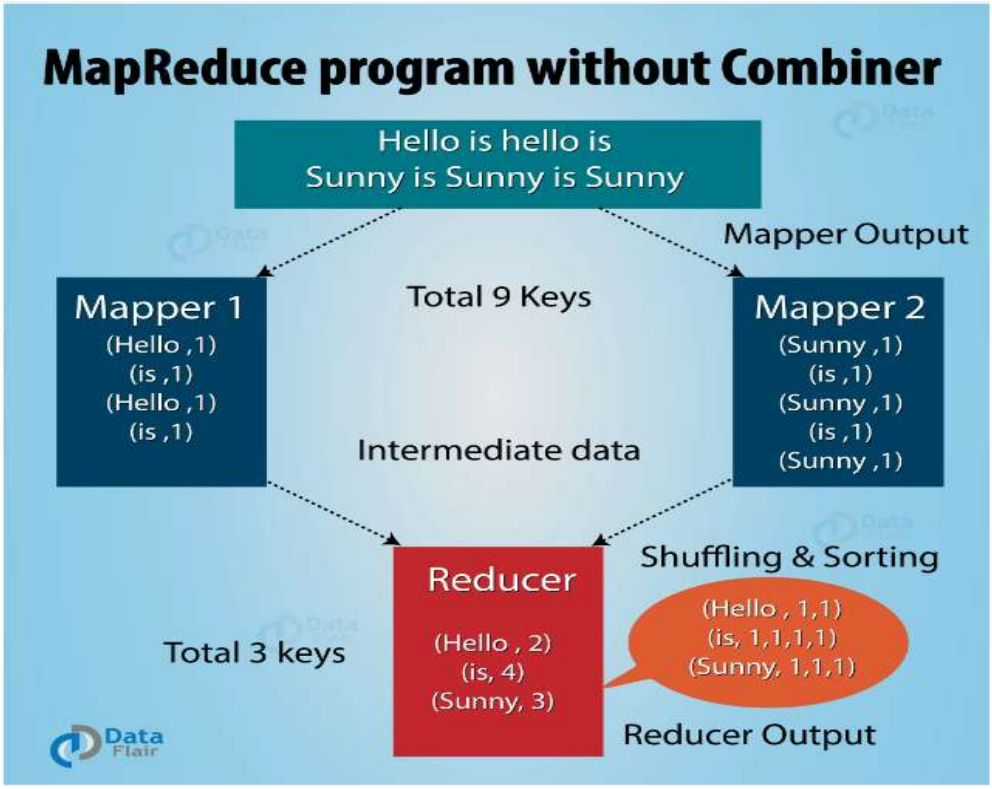
Mappcr Phascs- 3. Combiner

- **Hadoop Combiner** is also known as "**Mini-Reducer**" that summarizes the Mapper output record with the same Key before passing to the Reducer.
- It takes intermediate <keys, value> pairs provided by mapper and applies user specific aggregate function to only one mapper.
- We can optionally specify a combiner to perform local aggregation on intermediate outputs.
- It is an optional function but provides high performance in terms of network bandwidth and disk space.
- It takes intermediate key-value pair provided by mapper and applies user-specific aggregate function to only that mapper.
- It is also known as **local reducer**.
- It is an optimization technique for MapReduce Job.
- Generally, the reducer class is set to be the combiner class.
- The difference between combiner class and reducer class is as follows:
    - Output generated by combiner is intermediate data and it is passed to the reducer.
    - Output of the reducer is passed to the output file on disk.
- We can optionally specify a combiner using **Job.setCombinerClass(ReducerClass) in driver class** to perform local aggregation on intermediate outputs.

Example:

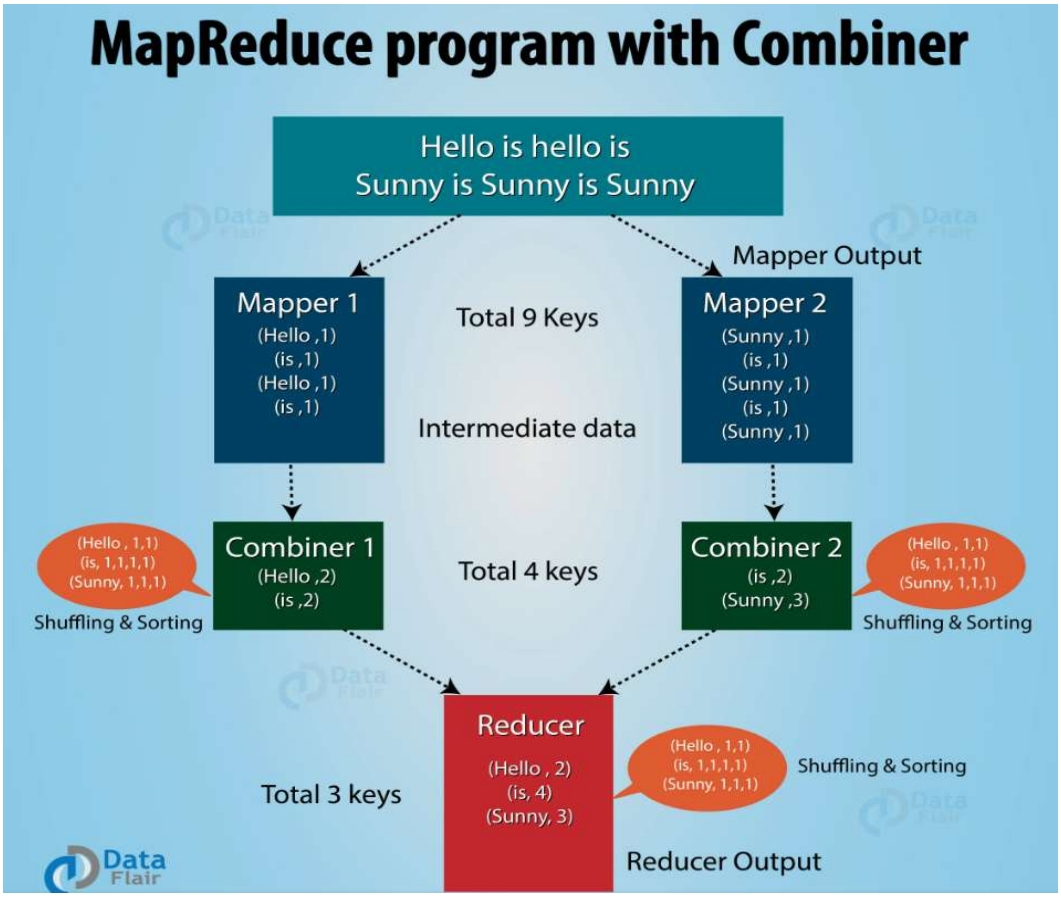
MapReduce program without Combiner:

In the below diagram, no combiner is used. Input is split into two mappers and 9 keys are generated from the mappers. Now we have (9 key/value) intermediate data, the further mapper will send directly this data to reducer and while sending data to the reducer, it consumes some network bandwidth (bandwidth means time taken to transfer data between 2 machines). It will take more time to transfer data to reducer if the size of data is big.



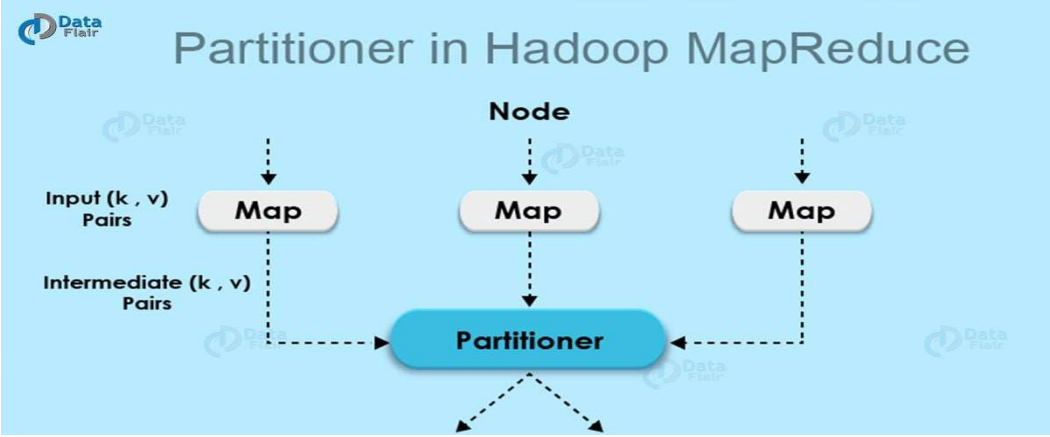
MapReduce program with Combiner in between Mapper and Reducer

Reducer now needs to process only 4 key/value pair data which is generated from 2 combiners. Thus reducer gets executed only 4 times to produce final output, which increases the overall performance.



Mapper Phases- 4. Partitioner

- The **Partitioner** in MapReduce controls the partitioning of the key of the intermediate mapper output.
- Take intermediate **<keys, value>** pairs produced by the mapper, splits them into partitions and sends the partitioned data to the particular reducer as per user-defined condition.

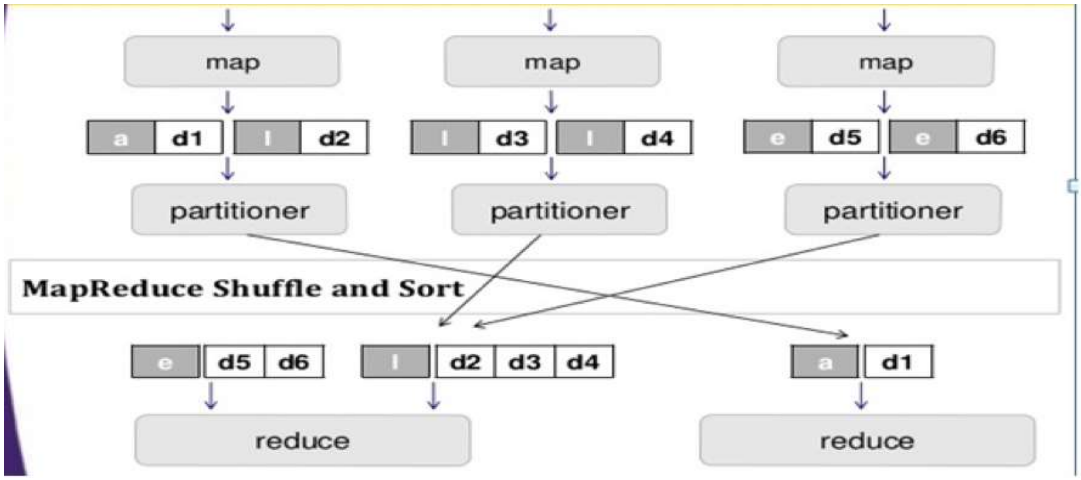


- According to the key-value each mapper output is partitioned and records having the same key value go into the same partition (within each mapper), and then each partition is sent to a reducer.
- Partition class determines which partition a given (key, value) pair will go. Partition phase takes place after map phase and before reduce phase.
- The partitioning phase happens after map phase and before reduce phase.
- Usually the number of partitions are equal to the number of reducers.
- The default partitioner is ***hash*** partitioner.
- User can control by using the method:

  int getPartition(KEY key, VALUE value, int numPartitions)

Example:

- The partitioner assigns each key-value pair to a specific reducer based on the key. This ensures that all values with the same key end up on the same reducer.
- In the below example:
  - (a, d1) is assigned to one reducer.
  - (l, d2), (l, d3), and (l, d4) are assigned to another reducer.
  - (e, d5) and (e, d6) go to a third reducer.



Reducer Phases

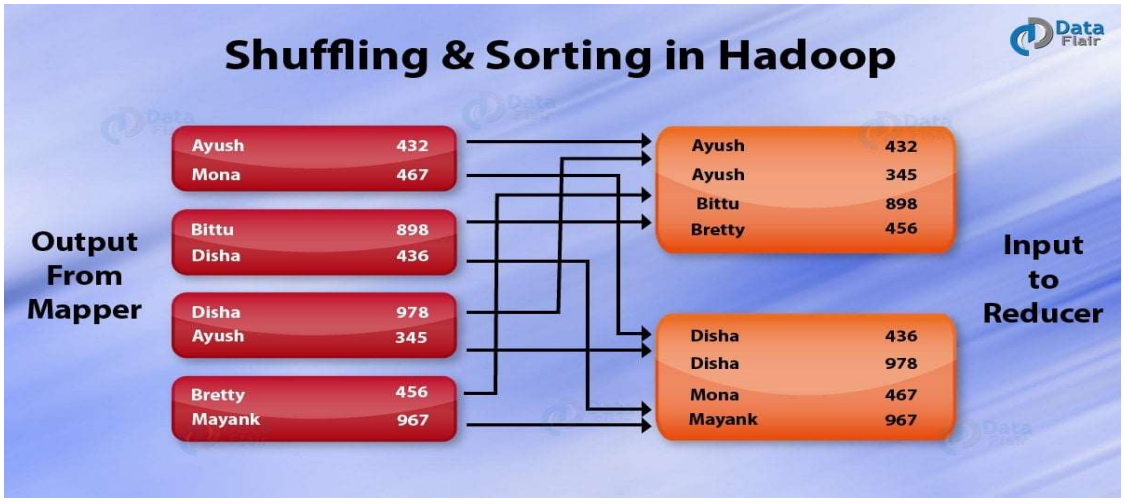
- The primary chore of the Reducer is to reduce a set of intermediate values (the ones that share a common key) to a smaller set of values.
- Hadoop Reducer takes a set of an intermediate **key-value pair** produced by the mapper as the input and runs a Reducer function on each of them. One can aggregate, filter, and combine this data (key, value) in a number of ways for a wide range of processing. Reducer first processes the intermediate values for particular key generated by the map function and then generates the output (zero or more key-value pair).
- One-one mapping takes place between keys and reducers. Reducers run in parallel since they are independent of one another. The user decides the number of reducers. By default number of reducers is 1.

Reducer Phases- 1. Shuffle & Sorting

- **Shuffle phase** in Hadoop transfers the map output from Mapper to a Reducer in MapReduce.
- **Sort phase** in MapReduce covers the merging and sorting of map outputs. Data from the mapper are grouped by the key, split among reducers and sorted by the key. Every reducer obtains all values associated with the same key.

Example: In the example below,

- All pairs with the key **Ayush (432 and 345)** are grouped together.
- Pairs with the key **Disha (436 and 978)** are grouped together. And so on.
- The shuffled and sorted pairs are then sent as input to the reducers.
- Each reducer processes a unique set of keys:
  - **Reducer 1 receives (Ayush, 432), (Ayush, 345), (Bittu, 898), and (Bretty, 456).**
  - **Reducer 2 receives (Disha, 436), (Disha, 978), (Mona, 467), and (Mayank, 967).**



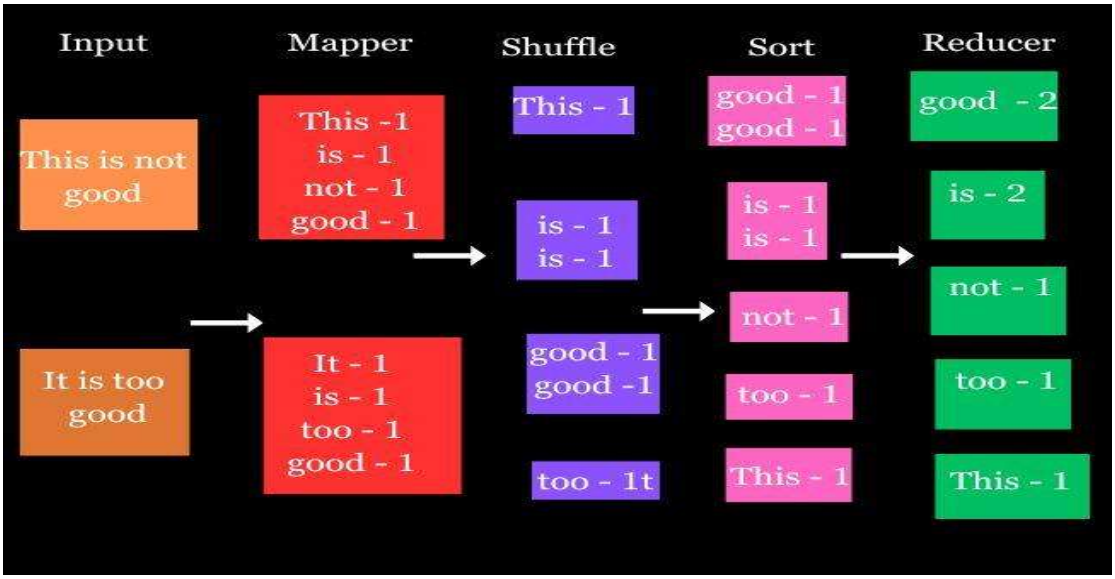
Reducer Phases- 2. Reduce

- Once shuffling and sorting will be done the Reducer combines the obtained result and perform the computation operation as per the requirement.
- The core operation in this phase is the "Reduce function" which takes a key and all the values associated with that key from the Map phase as input, and then performs the

necessary aggregation logic to produce a single output value for that key.

- Data is still handled as key-value pairs in the Reduce phase, allowing for efficient grouping and aggregation based on the keys.

Example:



Reducer Phases- 3. Output Format

- 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.
- The output format separates key-value pair with tab (default).

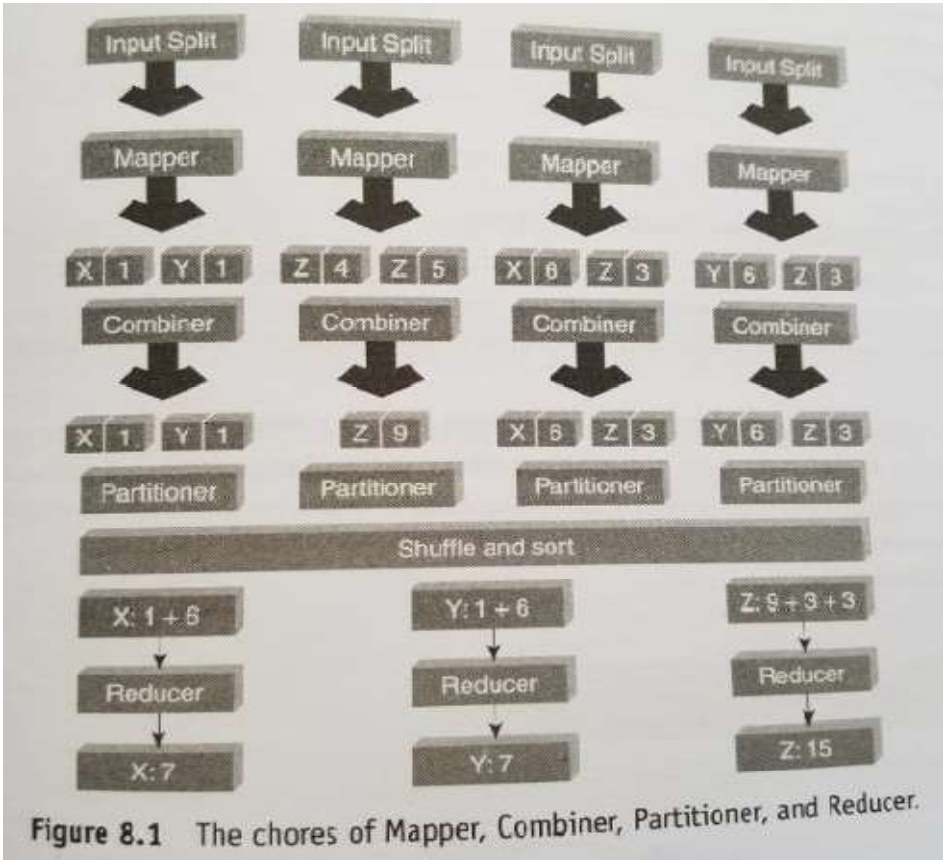


Figure 8.1 The chores of Mapper, Combiner, Partitioner, and Reducer.

Partitioner

- The partitioning phase happens after map phase and before reduce phase.
- Usually the number of partitions are equal to the number of reducers.
- The default partitioner is ***hash*** partitioner.
- User can control by using the method:
  int getPartition(KEY key, VALUE value, int numPartitions)

Write a user define partitioner class for WordCount problem.

WordCountPartitioner.java

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Partitioner;
public class WordCountPartitioner extends Partitioner<Text, IntWritable> {
        @Override
        public int getPartition(Text key, IntWritable value, int numPartitions) {
                String word = key.toString();
                char alphabet = word.toUpperCase().charAt(0);
                int partitionNumber = 0;
                  switch(alphabet) {
                  case 'A': partitionNumber = 1; break;
                  case 'B': partitionNumber = 2; break;
                  case 'C': partitionNumber = 3; break;
                  case 'D': partitionNumber = 4; break;
                  case 'E': partitionNumber = 5; break;
                  case 'F': partitionNumber = 6; break;
                  case 'G': partitionNumber = 7; break;
                  case 'H': partitionNumber = 8; break;
                  case 'I': partitionNumber = 9; break;
                  case 'J': partitionNumber = 10; break;
                  case 'K': partitionNumber = 11; break;
                  case 'L': partitionNumber = 12; break;
                  case 'M': partitionNumber = 13; break;
                  case 'N': partitionNumber = 14; break;
                  case 'O': partitionNumber = 15; break;
                  case 'P': partitionNumber = 16; break;
                  case 'Q': partitionNumber = 17; break;
                  case 'R': partitionNumber = 18; break;
                  case 'S': partitionNumber = 19; break;
                  case 'T': partitionNumber = 20; break;
                  case 'U': partitionNumber = 21; break;
                  case 'V': partitionNumber = 22; break;
                  case 'W': partitionNumber = 23; break;
                  case 'X': partitionNumber = 24; break;
                  case 'Y': partitionNumber = 25; break;
                  case 'Z': partitionNumber = 26; break;
                  default: partitionNumber = 0; break;
                  }
                  return partitionNumber;
        }
}
```

Explanation:

- The **WordCountPartitioner** extends the Hadoop Partitioner class to define custom partitioning behavior for Text keys and IntWritable values.
- The method **getPartition** determines the partition number for each key-value pair. It overrides the getPartition method from the Partitioner class.
  - **Text key:** The key for partitioning, which in this case is a Text object.
  - **IntWritable value:** The value associated with the key.
  - **int numPartitions:** The total number of partitions.
- **Partitioning Logic**:

```
String word = key.toString();
char alphabet = word.toUpperCase().charAt(0);
int partitionNumber = 0;
```

  - The key (a word) is converted to a string and capitalized.
  - The first character of the word is extracted (alphabet).
  - An integer variable partitionNumber is initialized to 0 to store the partition result.
- **Switch Statement:**

```
switch(alphabet) {
   case 'A': partitionNumber = 1; break;
   case 'B': partitionNumber = 2; break;
   ...
   case 'Z': partitionNumber = 26; break;
   default: partitionNumber = 0; break;
}
```

  - The code uses a switch statement to assign a partition number based on the first letter of the word.
  - For example, words starting with 'A' are assigned to partition 1, 'B' to partition 2, and so on up to 'Z', which is assigned to partition 26.
  - If the first character is not an alphabet letter (default case), it assigns partitionNumber = 0.
- In the drive program set the partitioner class as shown below:

```
job.setNumReduceTasks(27);
job.setPartitionerClass(WordCountPartitioner.class);
```

Searching

Objective: To write a MapReduce program to search for a specific keyword in a file.

Input Data:

```
1001,John,45
1002,Jack,39
1003,Alex,44
1004,Smith,38
1005,Bob,33
```

WordSearcher.java [Driver class]

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordSearcher {
        public static void main(String[] args) throws IOException,
                        InterruptedException, ClassNotFoundException {
                Configuration conf = new Configuration();
                Job job = new Job(conf);
                job.setJarByClass(WordSearcher.class);
                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(Text.class);
                job.setMapperClass(WordSearchMapper.class);
                job.setReducerClass(WordSearchReducer.class);
                job.setInputFormatClass(TextInputFormat.class);
                job.setOutputFormatClass(TextOutputFormat.class);
                job.setNumReduceTasks(1);
                job.getConfiguration().set("keyword", "Jack");
                FileInputFormat.setInputPaths(job, new Path("/mapreduce/student.csv"));
                FileOutputFormat.setOutputPath(job, new Path("/mapreduce/output/search"));
                System.exit(job.waitForCompletion(true) ? 0 : 1);
        }
}
```

Explanation:

- The main method of the driver class sets up and configures a job to search for a specific keyword within an input file and output the results.
- **setJarByClass:** Sets the class containing the main method, which is WordSearcher.
- **setOutputKeyClass and setOutputValueClass:** Define the output key and value types for the MapReduce job, both as Text.
- **setMapperClass and setReducerClass**: Specify the Mapper and Reducer classes (**WordSearchMapper and WordSearchReducer)** that contain the logic for this job.
- **setInputFormatClass and setOutputFormatClass:** Specify that input and output will be in text format.
- **job.getConfiguration().set("keyword", "Jack");**
  - This sets a configuration parameter with the key "keyword" and the value "Jack".
  - The Mapper or Reducer can retrieve this keyword and use it to filter or search for matching records.

WordSearchMapper.java [Mapper class]

```
public class WordSearchMapper extends Mapper<LongWritable, Text, Text, Text> {
        static String keyword;
        static int pos = 0;
        protected void setup(Context context) throws IOException,
                        InterruptedException {
                Configuration configuration = context.getConfiguration();
                keyword = configuration.get("keyword");
        }
        protected void map(LongWritable key, Text value, Context context)
                        throws IOException, InterruptedException {
                InputSplit i = context.getInputSplit(); // Get the input split for this map.
                FileSplit f = (FileSplit) i;
                String fileName = f.getPath().getName();
                Integer wordPos;
                pos++;
                if (value.toString().contains(keyword)) {
                        wordPos = value.find(keyword);
                        context.write(value, new Text(fileName + ","+ new IntWritable(pos).
toString() + ", "+ wordPos.toString()));
                }
        }
}
```

Explanation:

- This class extends the Mapper class. It processes lines of text (the key is the line number, represented by LongWritable, and the value is the line content, represented by Text). The output key and value are both of type Text.
- **setup Method:**
  - This method is called once before the map function begins.
  - The setup method retrieves the keyword from the job configuration (set in the main class) and assigns it to the keyword variable.
- **map Method:**
  - The map method is called for each line in the input file.
  - InputSplit and FileSplit are used to get the file name of the current split being processed.
  - pos is incremented to represent the line position in the file.
- **Searching for the Keyword**:

```
if (value.toString().contains(keyword)) {
    wordPos = value.find(keyword);
    context.write(value, new Text(fileName + "," + new
IntWritable(pos).toString() + "," + wordPos.toString()));
}
```

  - The value (line content) is checked to see if it contains the keyword.
  - If the keyword is found, wordPos stores the index position of the keyword in the line.
  - The context.write method outputs:
    - **Key**: The line content (value).
    - **Value**: A Text object containing:
      - The file name (fileName).
      - The line position (pos).
      - The index position of the keyword within the line (wordPos).

WordSearchReducer.java [Reducer class]

WordSearchReducer.java

```
import java.io.IOException;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class WordSearchReducer extends Reducer<Text, Text, Text, Text> {

        protected void reduce(Text key, Text value, Context context)
                        throws IOException, InterruptedException {

                context.write(key, value);
        }
}
```

Explanation:

- The WordSearchReducer will receive each unique key with the corresponding values.
- Since it doesn't perform any processing and simply writes the key-value pairs as-is, the final output of the reducer would look the same as the mapper output.

Sorting

Objective: To write a MapReduce program to sort data by student name (value).

Input Data:

```
1001,John,45
1002,Jack,39
1003,Alex,44
1004,Smith,38
1005,Bob,33
```

```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class SortStudNames {

        public static class SortMapper extends
                        Mapper<LongWritable, Text, Text, Text> {

                protected void map(LongWritable key, Text value, Context context)
                                throws IOException, InterruptedException {

                        String[ ] token = value.toString().split(",");
                        context.write(new Text(token[1]), new Text(token[0]+" - "+token[1]));
                }
        }
    // Here, value is sorted...
        public static class SortReducer extends
                        Reducer<Text, Text, NullWritable, Text> {

                public void reduce(Text key, Iterable<Text> values, Context context)
                                throws IOException, InterruptedException {

                        for (Text details : values) {
                                context.write(NullWritable.get(), details);
                        }
                }
        }
```

```
    public static void main(String[ ] args) throws IOException,
                InterruptedException, ClassNotFoundException {
            Configuration conf = new Configuration();
            Job job = new Job(conf);
            job.setJarByClass(SortEmpNames.class);
            job.setMapperClass(SortMapper.class);
            job.setReducerClass(SortReducer.class);
            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(Text.class);
            FileInputFormat.setInputPaths(job, new Path("/mapreduce/student.csv"));
            FileOutputFormat.setOutputPath(job, new
Path("/mapreduce/output/sorted/"));
            System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Explanation:

- In the mapper stage,
  - It splits each line of input data by commas, storing the tokens in an array.
  - The student name (token[1]) is set as the output key.
  - The value (a combination of the student ID and name) is set as the output value.
  - This effectively organizes the data by student name, which is the key for sorting.
  - So, the complete mapper output will be:

    (Alex, 1003-Alex)
    (Bob, 1005-Bob)
    (Jack, 1002-Jack)
    (John, 1001-John)
    (Smith, 1004-Smith)
- After the mapper output is shuffled and sorted by the MapReduce framework, it will be organized alphabetically by name, producing the following input for the reducer:

  (Alex, [1003-Alex])
  (Bob, [1005-Bob])
  (Jack, [1002-Jack])
  (John, [1001-John])
  (Smith, [1004-Smith])
- In the reducer, for each key, it iterates over the values and writes them out (since there's only one value per key in this example). Here's the output:

  (Alex, [1003-Alex])
  (Bob, [1005-Bob])
  (Jack, [1002-Jack])
  (John, [1001-John])
  (Smith, [1004-Smith])

Compression

- In MapReduce programming we can compress the output file. Compression provides two benefits as follows:
  1. Reduces the space to store files.
  2. Speeds up data transfer across the network.
- We can specify compression format in the Driver program as below:

```
conf.setBoolean("mapred.output.compress",true);
conf.setClass("mapred.output.compression.codec",GzipCodec.class,CompressionCodec.class);
```

- Here, codec is the implementation of a compression and decompression algorithm, GzipCodec is the compression algorithm for gzip.

Big Data Analytics

- **Big Data Analytics** is the process of examining big data to uncover patterns, unearth trends, and find unknown correlations and other useful information to make faster and better decisions.
- **Few Top Analytics tools are:** MS Excel, SAS, IBM SPSS Modeler, R analytics, Statistica, World Programming Systems (WPS), and Weka.
- The **open source analytics tools are**: R analytics and Weka.

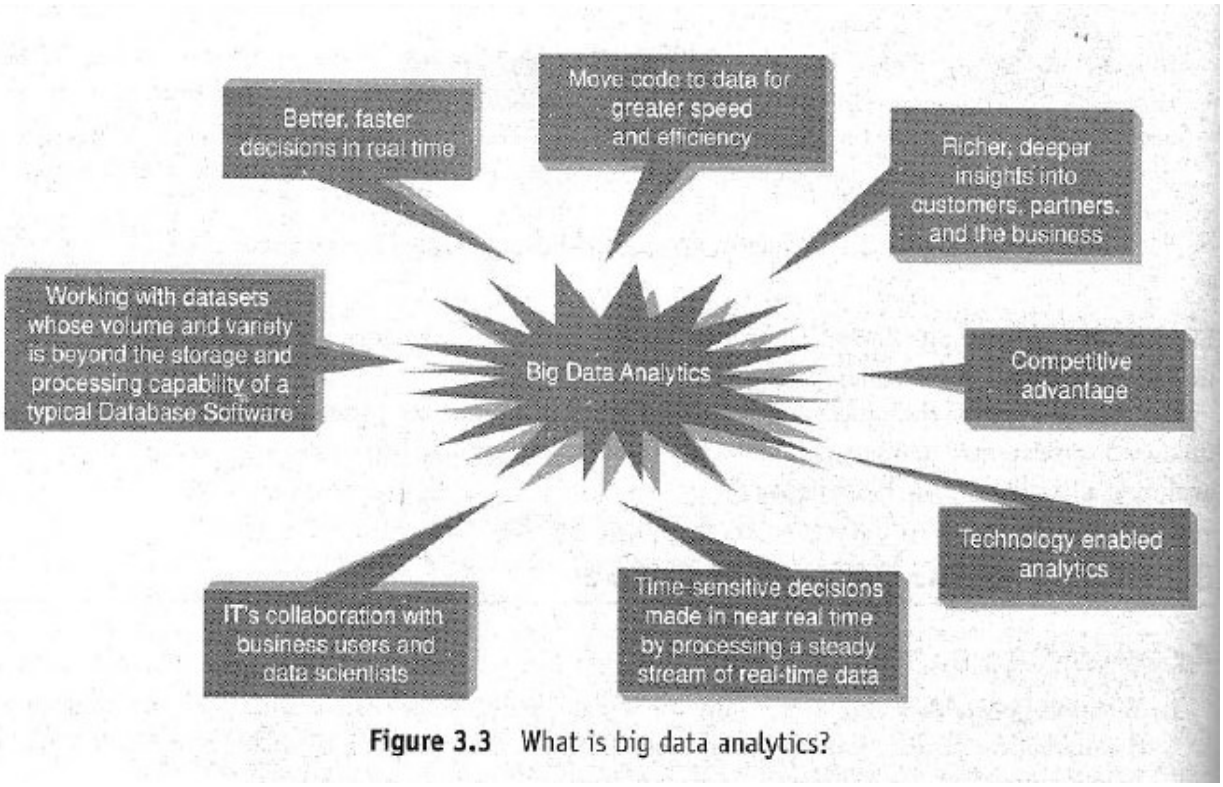


Figure 3.3 What is big data analytics?

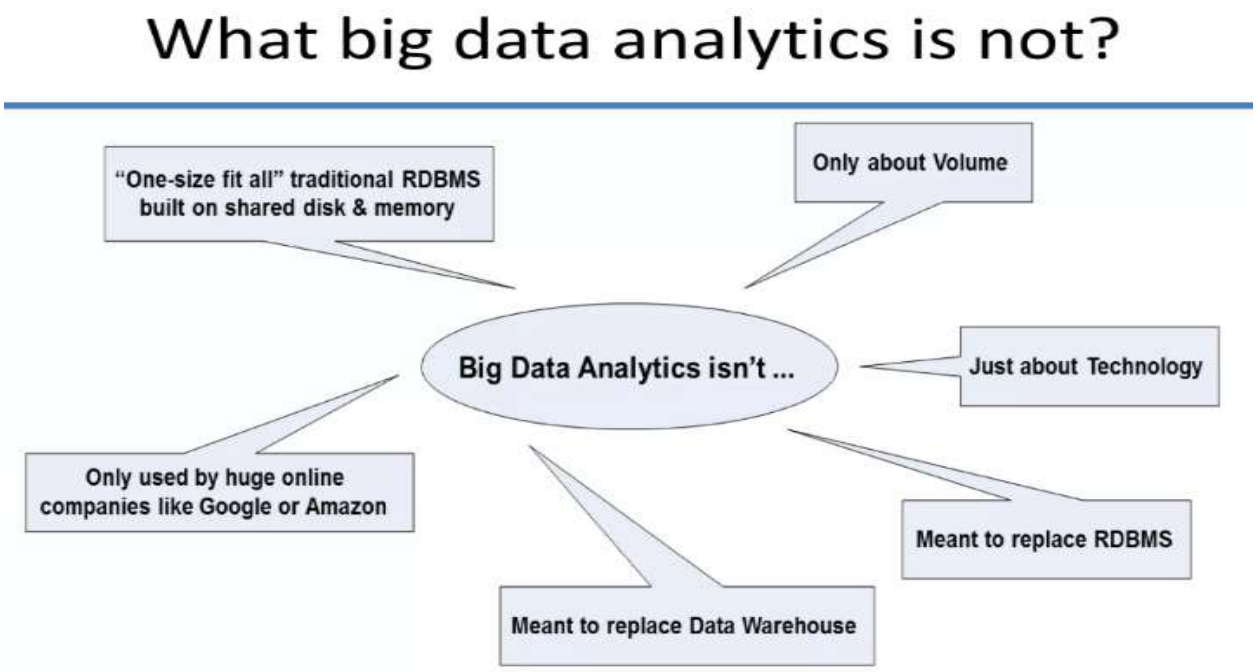
Big Data Analytics is:

1. **Technology enabled analytics:** The analytical tools help to process and analyze big data.

2. About **gaining** a meaningful, deeper, and richer **insights** into **business to drive in right direction**, understanding the customer's demographics, better leveraging the services of vendors and suppliers etc.

3. About a competitive edge over the competitors by enabling with finding that allow **quicker and better decision making.**

4. A tight **handshake between 3 communities**: **IT, Business users and Data Scientists.**
5. Working with datasets whose **volume and variety** exceed the current storage and processing capabilities and infrastructure of the enterprise.
6. About **moving code to data**. This makes perfect sense as the program for distributed processing is tiny compared to the data.



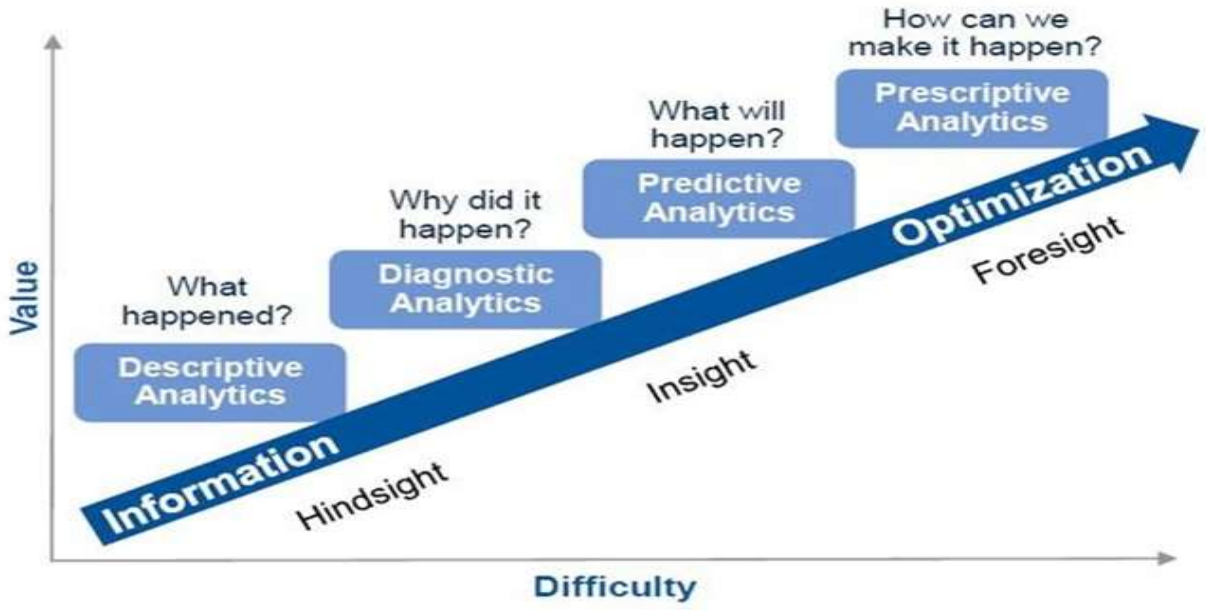
- Big Data Analtics coexist with both RDBMS and Data Warehouse, leveraging the power of each to yield business value.
- Big Data Analtics isn't:
  - Only about volume
  - Just about technology
  - Meant to replace RDBMS
  - Meant to replace data warehouse
  - Only used by huge online companies like Google or Amazon
  - "One-size fit all" traditionaly RDBMS built on shared disk and memory.

Why Sudden Hype around Big Data Analytics

1. Data is growing at a 40% compound annual rate, reaching nearly 74 ZB by 2021. The volume of business data worldwide is expected to double every 1.2 years.
   - Examples:
   - Wal-Mart: Process one million customer transactions per hour.
   - Twitter: 500 million tweets per day.
   - 2.7 billion "Likes" and comments are posted by Facebook users in a day.
2. Cost per gigabyte of storage has hugely dropped.
3. An overwhelming number of user friendly analytics tools are available in the market today.
4. Three Digital Accelerators: bandwidth, digital storage, and processing power.

Types of Analytics

Types of Analytics focus on stages of analysis (e.g., descriptive to prescriptive) that progressively offer deeper insights. In essence, **types of analytics** are about the ***depth of insight.*** Analytics encompasses a range of techniques for examining data and drawing insights to aid decision-making. They are:



1. Descriptive Analytics

- Descriptive analytics can show ***"what happened"*** and is the foundation of data insights.
- It is the interpretation of historical data to better understand changes that have occurred in a business.
- This type of analytics can be used to gain an overall picture of how a business is performing and is often used alongside predictive and prescriptive analytics.
- Common insights include year-over-year comparisons, the number of users, and revenue per subscriber.

2. Diagnostic Analytics

- Diagnostic analytics addresses ***"why things happened."***
- Common diagnostic analytic techniques/insights include drill-down, data discovery, data mining, and correlations.
- Companies use this data to identify patterns of behavior and make deep connections within the data they have collected.
- In order to be effective, diagnostic data must be detailed and accurate.

3. Predictive Analytics

- Businesses use predictive analytics to ***"see the future"*** and predict ***"what is likely to happen."***
- Uses statistical models and machine learning to forecast future events.
- Predictive models are especially useful for marketing and insurance companies which need to make decisions based on what could be coming up**.**
- Common processes in predictive analytics include decision trees, neural networks, and regression models.
- **Examples:** Sales forecasting, risk assessment, demand prediction.

4. Prescriptive Analytics

- Prescriptive analytics, analytics driven by AI (Artificial Intelligence) systems, helps companies make decisions and determine ***"what they should do next."***
- This is the most in-demand type of analytics today, however, it is talent and resource-expensive: Few companies have the skilled employees and resources to conduct it.
- It requires tremendous data and continuously updated data to help it learn, refine its decisions, and then communicate and act on these decisions in a business setting.
- **Examples:** Optimization, decision-support systems, recommendation engines.

Classification of Analytics

Classification of Analytics organizes analytics into categories based on context, data handling, industry, or specific analytical methods, giving a more detailed view of when and how analytics is applied across different use cases. In essence, **classification of analytics** is about the *context and approach* to analysis.

There are basically **two schools of thought**:

1. Those that classify analytics into **basic, operational, advanced and monetized**.

2. Those that classify analytics into **analytics 1.0, analytics 2.0 and analytics 3.0.**

First school of thought:

1. Basic analytics:

- It deals with slicing and dicing of data to help with basic business insights.
- Reporting on historical data, basic visualization, etc.

2. Operationalized Analytics:

- Focusing on the integration of analytics into business units in order to take specific action on insights.
- Analytics are integrated into both production technology systems and business processes.

3. Advanced Analytics: This largely is about forecasting for the future by predictive and prescriptive modeling.

4. Monetized analytics:

- Used to derive direct business revenue.
- The act of generating measurable economic benefits from available data sources.

Second school of thought:

Analytics 1.0	Analytics 2.0	Analytics 3.0
Era: mid 1950s to 2009	2005 to 2012	2012 to present
Descriptive statistics (report on events, occurrences, etc. of the past)	Descriptive statistics + predictive statistics (use data from the past to make predictions for the future)	Descriptive + predictive + prescriptive statistics (use data from the past to make prophecies for the future and at the same time make recommendations to leverage the situation to one's advantage)

Analytics 1.0	Analytics 2.0	Analytics 3.0
Key questions asked: What happened? Why did it happen?	Key questions asked: What will happen? Why will it happen?	Key questions asked: What will happen? When will it happen? Why will it happen? What should be the action taken to take advantage of what will happen?
Data from legacy systems, ERP, CRM, and 3rd party applications.	Big data	A blend of big data and data from legacy systems, ERP, CRM, and 3rd party applications.
Small and structured data sources. Data stored in enterprise data warehouses or data marts.	Big data is being taken up seriously. Data is mainly unstructured, arriving at a much higher pace. This fast flow of data entailed that the influx of big volume data had to be stored and processed rapidly, often on massive parallel servers running Hadoop.	A blend of big data and traditional analytics to yield insights and offerings with speed and impact.
Data was internally sourced.	Data was often externally sourced.	Data is both being internally and externally sourced.
Relational databases	Database appliances, Hadoop clusters, SQL to Hadoop environments, etc.	In memory analytics, in database processing, agile analytical methods, machine learning techniques, etc.

Challenges that prevent business from capitalizing on big data

1. Getting the business units to share information across organizational silos.
2. Finding the right skills (business analysts and data scientists) that can manage large amounts of SD, SSD, and USD and create insight from it.
3. The need to address the storage and processing of large volume, velocity and variety of big data.
4. Deciding whether to use SD or USD, internal or external data to make business decisions.
5. Choosing the optimal way to report finding and analysis of big data for the presentation to make the more sense.
6. Determining what to do with insights created from big data.

Top challenges facing big data

1. The practical issues of storing all the data (Scale)
2. Security: lack of authentication and authorization
3. Data schema (No fixed and rigid schema) leads to dynamic schema.
4. Consistency/Eventual consistency
5. Availability (24 x 7) (Failure transparency) 6. Partition tolerance (Both H/w and S/w)
6. Validating big data (Data quality) accuracy, completeness and timeliness.

Advantages of Big Data Analytics

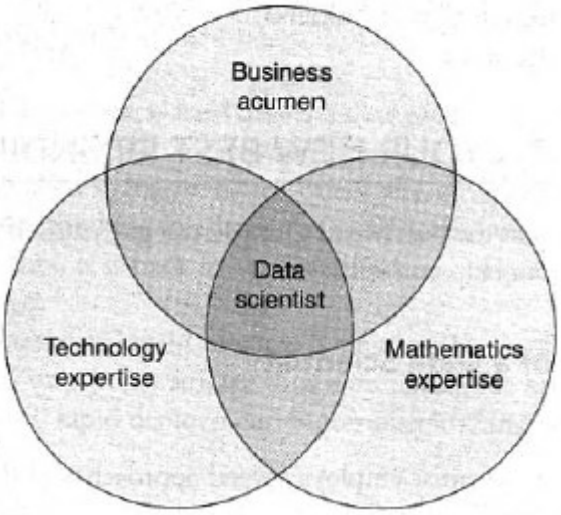
1. **Business Transformation** In general, executives believe that big data analytics offers tremendous potential to revolution their organizations.
2. **Competitive Advantage** According survey 57 percent of enterprises said their use of analytics was helping them achieve competitive advantage, up from 51 percent who said the same thing in 2015.
3. **Innovation Big data analytics** can help companies develop products and services that appeal to their customers, as well as helping them identify new opportunities for revenue generation.
4. **Lower Costs** In the New Vantage Partners Big Data Executive Survey 2017, 49.2 percent of companies surveyed said that they had successfully decreased expenses as a result of a big data project.
5. **Improved Customer Service Organizations** often use big data analytics to examine social media, customer service, sales and marketing data. This can help them better gauge customer sentiment and respond to customers in real time.
6. **Increased Security** Another key area for big data analytics is IT security. Security software creates an enormous amount of log data.

Different Big Data Analytics Approaches.

1. **Reactive – Business Intelligence:** It is about analysis of the pas or historical data and then displaying the finding of the analysis or reports in the form of enterprise dash boards, alerts, notifications etc.
2. **Reactive – BigData Analytics:** Here the analysis is done on huge datasets but the approach is still reactive as it is still base don static data.
3. **Proactive – Analytics:** This is to support futuristic decision making by the use of data mining, predictive modeling, text mining and statistical analysis. This analysis is not on bigdata as it still used traditional data base management practices.
4. **Proactive – Big Data Analytics:** This is sieving through terabytes of information to filter out the relevant data to analyze. This also includes high performance analytics to gain rapid insights from big data and the ability to sovle complex problems using more data.

Data Science

- **Data Science** is a multidisciplinary field that focuses on extracting insights, patterns, and knowledge from both structured and unstructured data using techniques from statistics, computer science, and machine learning.
- It combines various methods, processes, and systems to analyze data in meaningful ways, helping organizations make informed decisions.
- **Data scientists** use programming, analytical tools, data engineering, and domain expertise to derive valuable insights and build predictive models.



The Venn diagram illustrates the key skill sets required for a data scientist, intersecting three main areas:

- **Business Acumen**: This involves understanding the business context, knowing how to interpret data in terms of organizational goals, and having the ability to communicate findings to stakeholders. Business acumen helps data scientists ensure that the insights they derive are relevant and actionable for business decision-making.
- **Technology Expertise**: This includes proficiency in various technologies and tools for data analysis, such as programming languages (e.g., Python, R), databases (SQL, NoSQL), and big data frameworks (e.g., Hadoop, Spark). Data scientists need technical skills to manage, process, and analyze large datasets effectively.
- **Mathematics Expertise**: This includes knowledge in statistics, probability, and other areas of mathematical modeling. Mathematics expertise is essential for analyzing data rigorously, interpreting statistical models, and building accurate predictive and descriptive models.

Relationship between Data Science and Big Data Analytics

- **Big Data Analytics** is a subset of data science specifically concerned with analyzing and processing large volumes of data (big data). Big data typically involves high **volume**, **velocity**, and **variety** of data—attributes that require specialized tools, frameworks, and infrastructure to handle effectively. The relationship between data science and big data analytics can be summarized as follows:
- **Data Science Uses Big Data**: Big data provides data scientists with the raw material to extract insights. As the amount of data grows, data scientists use big data analytics techniques to handle, process, and analyze vast datasets.
- **Analytical Tools and Techniques**: Big data analytics leverages advanced tools like Hadoop, Spark, and NoSQL databases to handle the massive datasets, while data science integrates these tools with statistical, machine learning, and visualization methods to analyze data.
- **Building Models and Insights**: Big data analytics helps organize and clean large datasets, which is critical for data science projects. Data scientists can then use this processed data to create predictive models, uncover trends, and identify patterns.
-

Big Data Terminologies

In order to get a good handle on the big data environment, one should be familiar with a few key terminologies in this arena. They are:

a. In-Memory Analytics
b. In-Database processing
c. Symmetric Mulit-processor system
d. Massively parallel processing
e. Shared nothing architecture
f. CAP Theorem

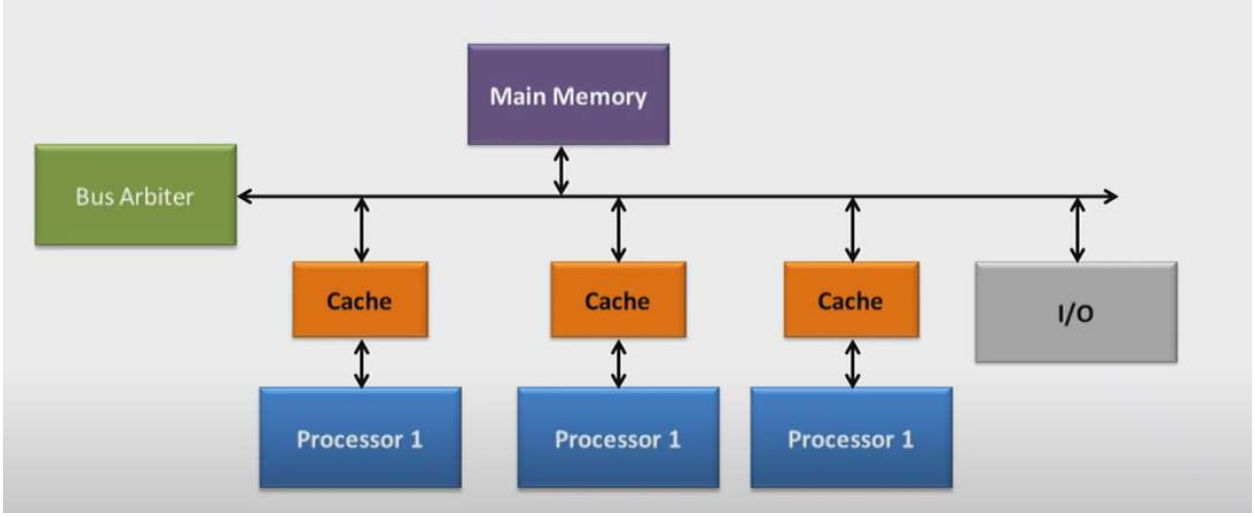
a. In-Memory Analytics

- Data access from non-volatile storage such as hard disk is a slow process. The data is required to be fetched from hard disk or secondary storage. One way combat this challenge is to pre-processor and store data. Example: cubes, aggregate tables, query sets, etc. The initial process of pre-computing and storing data or fetching it from secondary storage.
- This problem has been addressed using In-memory Analytics. Here all the relevant data is stored in Random Access memory (RAM) or primary storage thus eliminating the need to access the data from hard disk.
- The advantage is faster access, rapid deployment, better insights, and minimal IT involvement.

b. In-Database processing

- **In-database analytics** is a technology that allows data processing to be conducted within the database by building analytic logic into the database itself.
- **In-database processing**, sometimes referred to as **in- database analytics**, refers to the integration of data analytics into data warehousing functionality.
- The data from various enterprise Online Transaction Processing (OLTP) system after cleaning up are stored in Enterprise Data Warehouse and are exported to analytical programs for processing.
- The advantages are:
  - Provides parallel processing, partitioning, scalability and optimization features.
  - Data retrieval and analysis are much faster.
  - Information is more secure.

c. Symmetric Mulit-processor system



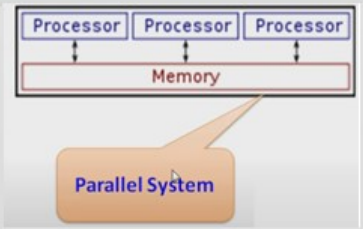
- In this there is single common main memory that is shared by two or more identical processors. The processors have full access to all I/O devices and are controlled by single operating system instance.
- SMP are **tightly coupled** multiprocessor systems.
- Each processor has its own high speed memory called cache memory and are connected using a system bus.

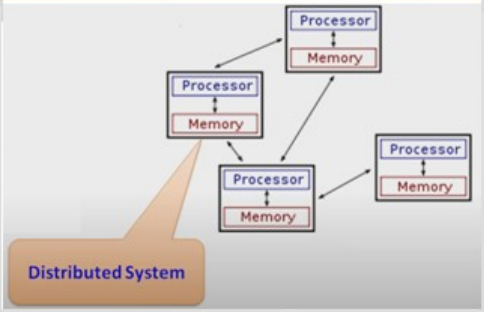
d. Massively Parallel Processing

- Massively parallel Processing (MPP) refers to the coordinated processing of programs by a number of processors working parallel. The processors each have their own OS and dedicated memory. They work on different parts of the same program. The MPP processors communicate using some sort of messaging interface.
- MPP is different from symmetric multiprocessing in that SMP works with processors sharing the same OS and same memory. SMP also referred as tightly coupled Multiprocessing.

Difference Between Parallel and Distributed Systems

	Parallel system	Distributed system
Memory	Tightly coupled system shared memory	Weakly coupled system Distributed memory
Control	Global clock control	No global clock control
Processor Interconnection	Order of Tbps	Order of Gbps
Main focus	Performance Scientific Computing	Performance(cost and scalability) Reliability/ availability information/resource sharing

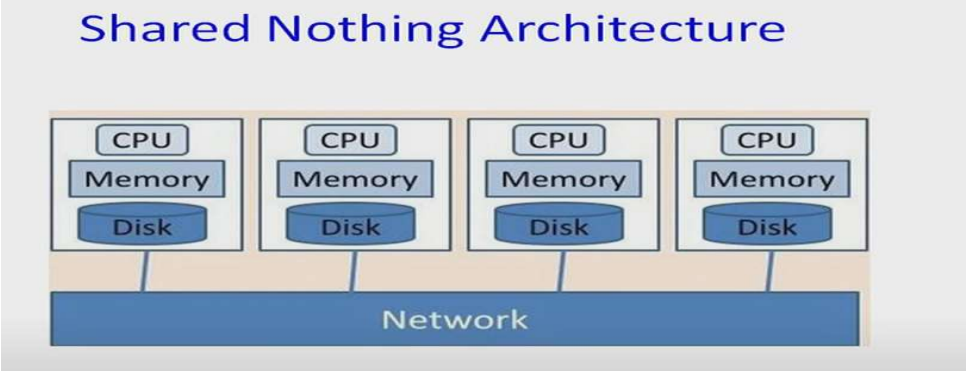




e. Shared nothing Architecture

The three most common types of architecture for multiprocessor systems:
1. Shared memory
2. Shared disk
3. Shared nothing.

In **shared memory** architecture, a common central memory is shared by multiple processors.
In **shared disk** architecture, multiple processors share a common collection of disks while having their own private memory.
In **shared nothing** architecture, neither memory nor disk is shared among multiple processors.



Advantages of shared nothing architecture:

Fault Isolation:

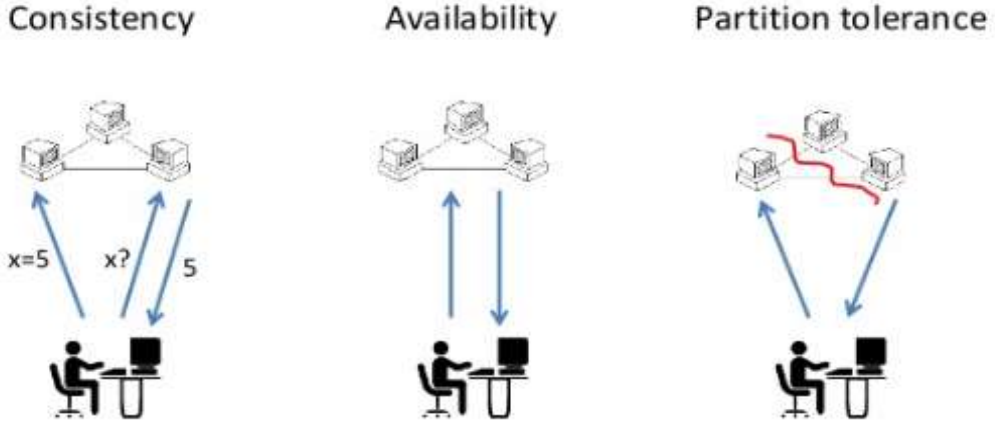
- A "shared nothing architecture" provides the benefit of isolating fault.
- A fault in a single node is contained and confined to that node exclusively and exposed only through messages or lack of it.

Scalability:

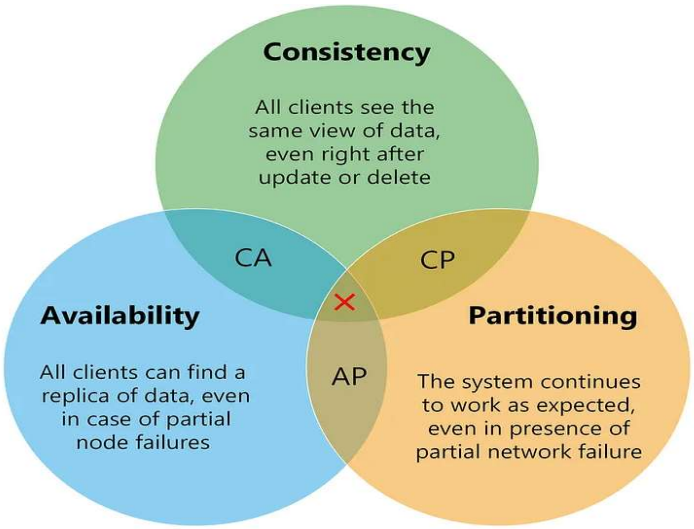
- The disk is a shared resource it implies that the controller and the disk band-width are also shared.
- Synchronization will have to be implemented to maintain a consistent shared state.
- This would mean that different nodes will have to take turns to access the critical data.
- This imposes a limit on how many nodes can be added to the distributed shared disk system, thus compromising on the scalability.
-

f. CAP Theorem

- The **CAP theorem** is also called the **Brewer's theorem**.
- **CAP** stands for **C**onsistency, **A**vailability, **P**artition tolerance
  - **Consistency** Implies that every read fetches the last write. Consistency means that all nodes see the same data at the same time. If there are multiple replicas and there is an update being processed, all users see the update go live at the same time even if they are reading from different replicas.
  - **Availability** implies that reads and writes always succeed. Availability is a guarantee that every request receives a response about whether it was successful or failed.
  - **Partition tolerance** implies that the system will continue to function when network partition occurs. It means that the system continues to operate despite arbitrary message loss or failure of part of the system.



- **CAP theorem** states that in a distributed computing environment, it is impossible to provide the following guarantees. At best you can have two of the three**(C, A, P)** and one must be sacrificed.



In the context of Big Data, the CAP Theorem means that distributed databases and data processing systems must make a trade-off among these properties(shown in figure which is a triangular view of CAP theorem):

1. Availability and Partition Tolerance **(AP)**
2. Consistency and Partition Tolerance **(CP)**
3. Consistency and Availability **(CA)**

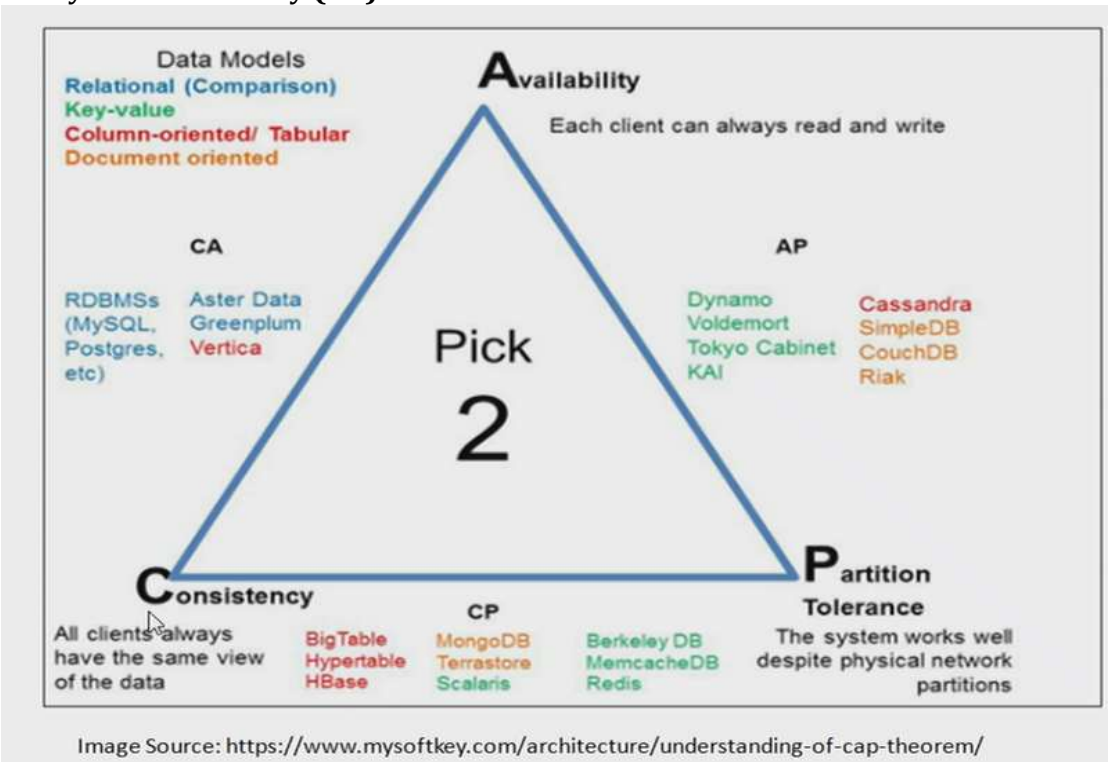


Image Source: https://www.mysoftkey.com/architecture/understanding-of-cap-theorem/

1. AP (Availability and Partition Tolerance):

- The distributed system is highly available and the system is partition-tolerant, so every read request will not guarantee the most recent information but will be processed. This means:
  - AP systems allow different nodes to hold slightly different versions of the data, especially during network partitions, leading to **eventual consistency**.
  - In an AP system, if a network partition occurs, each part of the network can continue to serve requests independently. Once the network partition is resolved, the nodes will

reconcile any data discrepancies to eventually reach a consistent state.

- Examples: Voldemort, SimpleDB, CouchDB.

2. CP (Consistency and Partition Tolerance):

- Data is consistent among all the nodes and the nodes maintain partition tolerance. The de-sync node will not accept any request. Some requests will be dropped instead of returning inconsistent (the most recent) information.
- Let's say we have two nodes and one of the nodes (N2) stops servicing read/write requests upon detecting that the network connecting the system got partitioned. This means that system treats node N2 as no longer available and any request on this node will be rejected, as it will end up returning the old/stale copy of the data. Hence, this kind of system provides consistency and partition tolerance.
- Examples: Google Bigtable, Hbase, MongoDB, MemcacheDB, Redis.

3. CA (Consistency and Availability):

- These systems prioritize data consistency and availability but may sacrifice partition tolerance. This means:
  - CA systems work best in environments where network partitions (i.e., communication failures between nodes) are unlikely or rare.
  - When a network partition occurs, a CA system might either temporarily sacrifice availability (rejecting requests) or consistency (allowing potentially conflicting data), or even fail altogether until the partition is resolved.
- Examples: traditional RDBMS like **MySQL** and **PostgreSQL**.

Questions on Unit-3

1. Explain CAP Theorem in Big Data Systems.
2. Define Data Science and explain its relationship with Big Data analytics.
3. Classify the different types of analytics used in Big Data. Provide examples for each type.
4. Write a MapReuduce program for sorting following data according to name.

   Input:
   001,chp
   002,vr
   003,pnr
   004,prp
5. Describe the chores of Mapper, Combiner, Partitioner, and Reducer for the word count problem.
6. Define Big Data Analytics. What are the various types of analytics?
7. Define Data Science and explain its relationship with Big Data analytics.
8. Describe the role of the Mapper in the MapReduce framework. Explain its different phases with example.
9. Write a MapReduce program to search an employee name in the following data:

   Input:
   001,chp
   002,vr
   003,pnr
   004,prp
10. Explain the following terminology of Big Data
    a. In-Memory Analytics
    b. In-Database processing
    c. Symmetric Mulit-processor system
    d. Massively parallel processing
    e. Shared nothing architecture
    f. CAP Theorem

11. Identify and discuss the greatest challenges faced in Big Data analytics.
12. Write a user define partitioner class for WordCount problem.
13. Explain different Big Data Analytics Approaches.
14. Discuss parallel and distributed systems in big data environment.
15. Explain MapReduce anatomy.
16. Write a MapReuduce program for WordCount problem.
17. Differentiate Analytics 1.0, 2.0, 3.0.