Chapter-1

Introduction to Big Data

Big Data Overview

- Big data is a large and complex collection of data sets which may not be correlated and which is impossible to process using traditional data processing applications.
- Big Data implies the method of extracting value from data, where the size of data is unpredictable.
- Big Data comes into existence because it is really a smart idea to use simple algorithm in large data to achieve higher efficiency and accuracy in analysis.
- The world has very huge amount of data but they are not properly analyzed into the required information. To generate appropriate information's from the large size of data we have, Big data acts as an important tool in today's era.
- The amount of data sets arises from various sources such as web, e-commerce, social network, bank transactions, sensor technology, mobile computing and so on.

Characteristics of Big Data

1. Volume:
- Volume indicates the quantity of generated and stored data.
- The size of data determines the potential value and insight.
- It also determines whether it is considered to be big data or not.
- The volume of data in the world is increasing exponentially.

2. Variety:
- Variety indicates the different types and nature of data.
- All the data present in big data analysis may not be of same type.
- Even a single application may be generating variety of data.
- This increases complexity in big data analysis and knowledge extraction.

- The data may be web data, relational data, XML, structured data, streaming data, graph data and so on.
- For efficient extraction of information or patterns, all these variety of data must be linked together and analyzed together.

3. Velocity:
- Velocity indicates the speed at which data is generated and processed to meet the demands.
- The data obtained is of dynamic nature, so they must be analyzed very fast to provide efficient and effective knowledge.

Challenges in Big Data

1. Big data consists of huge amount of data sets. The main challenge evolves in identifying the appropriate data from such mass of data and determining how to make best use of the relevant data.

2. Even though the data and analysis method are determined, there is struggle in finding the appropriate and skilled manpower capable of working with both new technology and data analysis for relevant business insight.

3. The variety of data types and formats may generate hindrance in the data analysis as it is very difficult to connect variety of data points for a single insight. Data integration is the important aspect of effective big data analysis, but it is also one of the major challenges that prevails.

4. The technology landscape in the data world is evolving very fast. So, efficient handling of the technology along with adaptation to cope with technology is must for big data analysis.

5. The organizational structure for big data project management should be apart from another project management task because this field is very much different and needs a strong and motivational project management team.

6. During the big data analysis, the data analysts do not get full benefits from the data they have due to the security concerns about data protection.

7. The technology infrastructure necessary to work with big data is very expensive.

Background of Data Analytics

Big Data Analytics

- Big Data analytics is the process of analyzing the huge volume of variety of data so as to discover the hidden patterns and other essential information that can be used for decision making process.
- The traditional data analytics tools cannot be used for big data due to its unstructured data format.
- The most commonly used technologies in big data analytics are NoSQL databases, Hadoop and Map Reduce.
- It focuses on efficient use of a simple model applied to large volume of data.

Big Data Analyst

- Data analyst is the person who is responsible to collect the relevant data, analyze it using big data analytics tools and use the outcome for gaining competitive advantages for the organizational decision making.

- The various skills required by the data analyst are as follows:
1. Analytical skill
2. Communication skill
3. Critical thinking
4. Attention to detail
5. Mathematical skill

Objective of big data analytics

1. Avoid aggregation
2. Reduce data movement and replication

3. Bring analytics as close as possible to the data
4. Optimize computation speed.

Process of Data Analytics

Phase 1 - Discovery Phase

- Discovery phase is the phase in data analytics process in which knowledge are gathered from the available data by analyzing the data to discover the hidden pattern.
- This phase consists of following processes:

a) Acquisition
b) Pre-processing
c) Integration
d) Analysis
e) Interpretation
f) Algorithmic model

- Acquisition is the process in which the data necessary for the concerned analytical problems are collected or gathered. The data can be gathered through internal or external sources.

- Pre-processing is the process that converts the collected data into the standard format that can be easily used in further processes.

- Integration is the process in which the relevant data are integrated from different sources and the redundant data are eliminated.

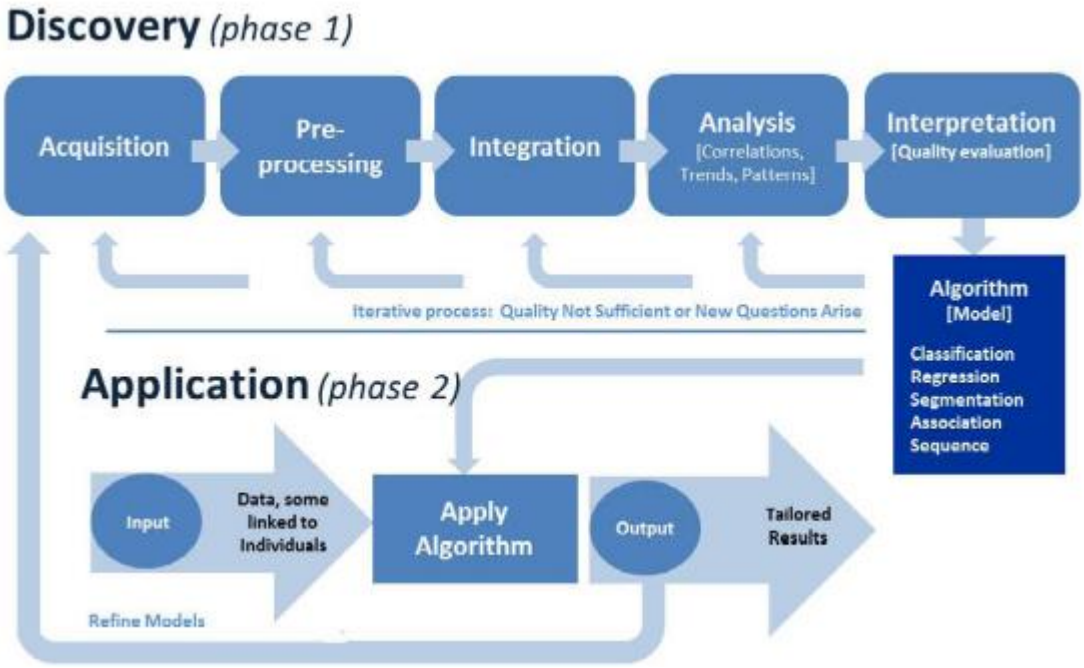
- Analysis is the process in which the relationships among the data items are searched so as to yield some useful information. This process is used to discover hidden patterns from the data.

- Interpretation is the process in which the results of the analysis are examined and the quality of the results are determined. It provides the information whether or not the analysis results can be used for decision making.

- After all these steps, if the result is trustworthy, then an algorithmic model is generated that incorporates all the knowledge discovered in analysis phase, which can be used further with new set of data to check for the outcome.

Phase 2 - Application

- Application phase is the phase in which the algorithmic model generated as a result of data analytics is used in the real domain.
- With the help of the model, the organization acts based on the outcome of the model.
- The algorithm is applied to some sort of input from the organization so as to gain the outcome based on the discovered knowledge or pattern. This helps in proper formulation of strategy by an organization.



Role of Distributed System in Big Data

Distributed System

- Distributed system is the system that is composed of autonomous computers, connected through a network and middleware for coordinating and sharing the resources, in such a way that the whole system looks as a single integrating computer facility for the user.

Use of Distributed System to solve Big Data problems

- Big data consists of massive amount of data which cannot be stored in a single computer or node.
- So, there is necessity for big data to be distributed across multiple nodes.
- Distributed system helps to solve big data problems without the requirement of a single resource capable to handle it. This makes the big data analytics cost efficient and performance improvement.

Role of Data Scientist

Data Scientist

- Data science is the field that deals with scientific methods, processes and systems so as to extract knowledge from the data.
- It requires techniques from different fields like mathematics, statistics, computer science and so on.
- Data scientist is the person who apply powerful tools and advanced statistical modeling techniques to make discoveries about business processes and problems through the curious problem solving mechanism applied on the available data sets.
- Data scientist has the ability to handle the raw data using the latest technologies, can perform the analysis and can present the acquired knowledge in understandable manner.

Roles of Data Scientist

1. Use the big data technologies to make new discoveries from the data.
2. Develop structure from the large volume of unstructured data and generate possibility for analysis.
3. Identify the data sets from multiple sources, integrate them and eliminate the unnecessary data present.
4. Communicate the results and suggest implications for new business

direction.
5. Display information in clear and understandable way.

Skills Needed for Data Scientist

1. Knowledge of basic tools like statistical programming languages (R and Python) and database.
2. Basic Statistics
3. Machine Learning
4. Calculus and Linear Algebra
5. Data munging skills (Conversion of data to necessary format)
6. Data Visualization and Communication Skills
7. Software Engineering
8. Problem Solving
9. Business Skills

Data Scientist vs Data Analyst

- Data scientist is responsible to formulate the questions for business progress and proceed in solving them. Data analysts is responsible to pursue a solution for the provided questions.
- Data scientist is expected to convert the data and analysis into a business scenario. Data analyst is expecting just to analyze the data to find patterns.

Current Trend in Big Data Analytics

1. Big data analytics services are available on the cloud. This makes possibility for faster and easier analysis and processing.

2. Hadoop is being general purpose and enterprise wide solutions for big data.

3. Emergence of big data lakes. It removes the necessity for designing the data set before entering the data. It allows all the data to be collected in the repository or data lake.

4. The value of data is being more. All the business organizations are enrolled to get valuable insights from their data for their progress and growth.

5. Use of NoSQL database helps to store the unstructured data that is the most prevalent data in the world today.

6. Use of in memory analytics allows speeding of the analytic processes.

7. Variety is becoming the single driving factor for big data investments.

8. The rise of metadata catalogs helps the users to discover and understand relevant data which are worth analyzing.

Chapter-2

Google File System

Architecture of Google File System

Google File System

- Google file system is a scalable distributed file system developed by Google to provide efficient and reliable access to data using large clusters of commodity hardware.
- It is designed to meet the rapidly growing demand of Google's data processing need.
- It provides performance, scalability, reliability and availability of data across distributed system for handling and processing big data.

Why built GFS?

1. Node failures happen frequently
2. Files are huge
3. Most files are modified by appending at the end
4. High sustained bandwidth is important than low latency

Characteristics of GFS

1. Files are organized hierarchically in directories and identified by path name.
2. It supports all the general operations on files like read, write, open, delete and so on.
3. It provides atomic append operation known as record append.
4. The concurrent write to the same region is not serializable.
5. It performs two operations: snapshot and record append.

Common goals of GFS

1. Performance
2. Reliability
3. Automation
4. Fault Tolerance
5. Scalability
6. Availability

Limitations of GFS

1. It lacks support for POSIX features.
2. It has high storage overheads.
3. It needs specialization for record append operation.

GFS Architecture

The basic architecture of Google File System is shown in given figure:

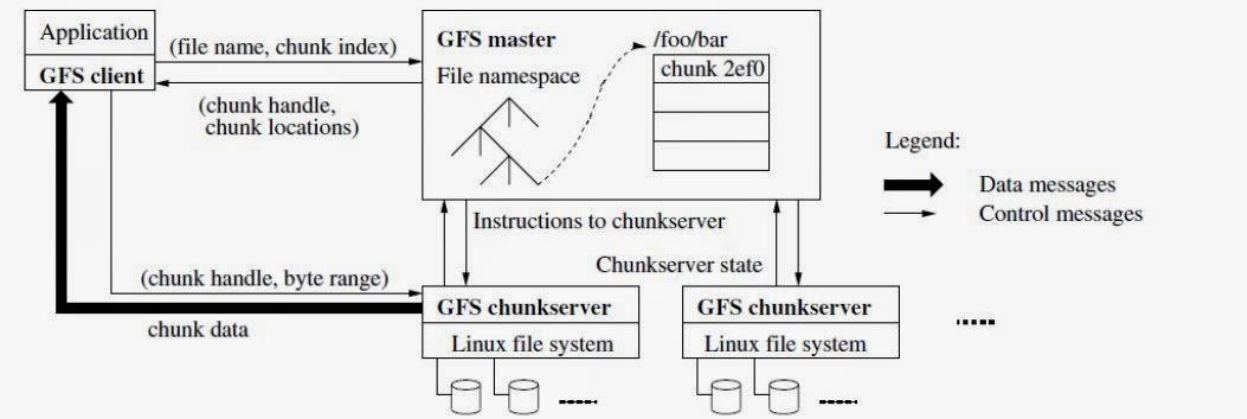


Figure 1 GFS Architecture

---

1. Master Node:
- It is responsible for the activities of the system such as managing chunk leases, load balancing and so on.
- It maintains all the file system metadata.
- It contains operation log that stores namespaces and file to chunk mappings.
- It periodically communicates with chunk server to determine chunk locations and assesses state of the overall system.
- Each node on the namespace tree has its own read-write lock to manage concurrency.

2. Chunk and Chunk Server
- The files are divided into fixed sized chunks.
- Each chunk has immutable and globally unique 64-bit chunk handle.
- Chunk server is responsible to store chunks on local disk as a Linux files.
- By default, each chunk is replicated 3 times across multiple chunk servers.
- The size of chunk is 64 MB.
- Due to such large chunk, it results in space wastage because of internal fragmentation.
- The advantages of large chunk size are as follows:
a) It reduces client's need to interact with master. It means read or write in a single chunk requires only one request to master.
b) It reduces network overhead by keeping a persistent TCP connection to the chunk server for multiple operations performed by client.
c) It reduces the size of metadata stored in the master. It enables storage of metadata in memory.

3. Client Node
- Client node is linked with the application that implements GFS API.
- It communicates with the master and the chunk server to read or write data.
- Client communicates with master to get the metadata.
- For read and write, client directly communicates with the chunk server.

Operation Log and Meta Data

- Operation log is the persistent records of metadata.
- It defines the logical timeline about serialized order of concurrent operations.
- The state is recovered by master by replaying the operation log.

- The metadata stored in GFS master are as follows:
1. Namespace (directory hierarchy)
2. Access control information per file
3. Mapping from file to chunk
4. Current location of chunks (Chunk servers)

Reasons for Single Master in GFS

- Single master design simplifies the GFS design.
- It enables the master to make sophisticated chunk placement and replication decisions.
- Since, the master interacts with client for sharing metadata only, there is no necessity to have multiple master.
- All the heavy processes of read and write is handled by the chunk servers. So, lot of chunk servers are necessary for performance of the system.

Manage overloading of the Master

- The read and write operation is completely separated from master. Instead, these operations are performed by the chunk servers.
- For reliability, master state is replicated on multiple machines, using the operation logs and checkpoints.
- If master fails, GFS starts a new master process at any of these replica.
- Such replica of master state is known as shadow master.
- Shadow master is also able to perform read only access to the file system even when the primary master is down, but it lags the primary master by few fractions of second.

Consistency Model of GFS

- A file region is said to be consistent if all the clients will see the same data each time they load data from any replica of that file region.
- A file region is said to be defined after a file data mutation if it is consistent and the client is able to see what the mutation writes in it.

- The general consistency model is shown in given figure:

<table>
<tr><th></th><th>Write</th><th>Record Append</th></tr>
<tr><td>serial success</td><td>defined</td><td rowspan="2">defined interspersed with inconsistent</td></tr>
<tr><td>concurrent success</td><td>consistent but undefined</td></tr>
<tr><td>failure</td><td colspan="2">inconsistent</td></tr>
</table>

Interactions in GFS

1. Leases and Mutation Order
- A mutation is an operation that changes the metadata of the chunks.
- It is resulted due to the write or append operations to the file.
- Leases are used to maintain consistent mutation order across replicas.
- The master grants chunk lease to one of the replica.
- The replica is known as primary replica.
- Primary replica is responsible to design the order for all the mutations to the chunk.

2. Record Append
- It is the atomic append operation provided by GFS.
- The client specifies the data only. The GFS automatically appends it to the file at least once at any offset chosen by the GFS itself and returns the offset to the client.
- The client then pushes the data to all the replicas of the last chunk of that file and then sends request to the primary replica.
- The primary replica checks whether appending process will exceed the chunk size. If it exceeds, then it pads the chunk to the maximum size, tells all other replicas to do so and replies client indication re operation on next chunk. Otherwise, the data is appended on the chunk. And replies the client with success.
- For operation to be success, the data should be written at the same offset on all the replicas of same chunk.

3. Snapshot:
- The snapshot operation is responsible to makes a copy of a file or a directory tree.
- The client uses snapshot to create branch copies of huge data sets or to checkpoint the current state.
- When the master receives snapshot request, it revokes any leases on the chunks. After the leases have been revoked or expired, the master logs the operation to disk.
- The master then applies log record to in-memory state by duplicating the metadata for the source file or directory tree.
- The created snapshot points at the same chunk as the main files.

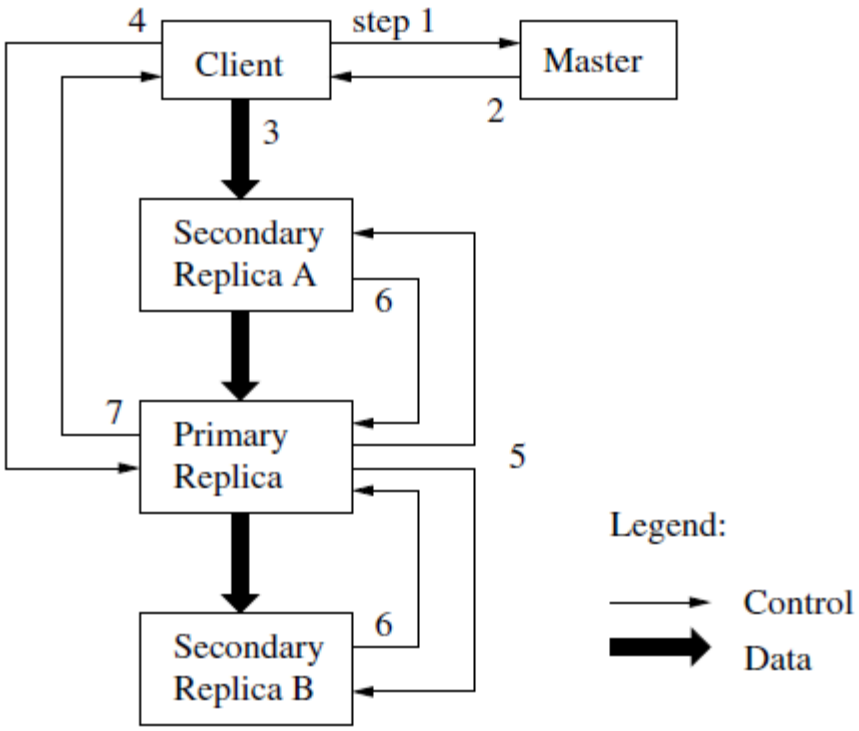


Figure 2: Write Control and Data Flow

Write Control and Data Flow in GFS

1. Client asks the master which chunk server holds the current lease for the chunk and the locations of the other replicas. If no one has a lease, the master grants one to a replica it chooses.

2. Master replies with the identity of the primary and the locations of the other (secondary) replicas. The client caches this data for future mutations. It needs to contact the master again only when the primary becomes unreachable or replies that it no longer holds a lease.

3. The client pushes the data to all the replicas. A client can do so in any order. Each chunk server will store the data in an internal LRU buffer cache until the data is used or aged out.

4. Once all the replicas have acknowledged receiving the data, the client sends a write request to the primary. The request identifies the data pushed earlier to all of the replicas. The primary assigns consecutive serial numbers to all the mutations it receives, possibly

from multiple clients, which provides the necessary serialization. It applies the mutation to its own local state in serial number order.

5. The primary forwards the write request to all secondary replicas. Each secondary replica applies mutations in the same serial number order assigned by the primary.

6. The secondaries all reply to the primary indicating that they have completed the operation.

7. The primary replies to the client. Any errors encountered at any of the replicas are reported to the client.

Fault Tolerance in GFS

Garbage Collection

- Whenever the client deletes the file, the file is not deleted but renamed to hidden file with delete timestamp.
- During regular scan of file namespace, hidden files are removed if existed for more than 3 days.
- Until that time, it is possible to undelete the file.
- When it is removed, the memory metadata stored by master is erased.
- Master and chunk server exchanges information about the files with the Heartbeat message.

Advantages:
- Simple and reliable in distributed system
- Uniform and dependable way to clean up replicas
- Performed in batches only when the master is free
- Provides protection against accidental deletion

Disadvantages:
- The storage cannot be used immediately

Stale Replica

- A replica is said to be stale if a chunk server fails and misses mutations to the chunk while it is down.
- The master is responsible to maintain a chunk version number to distinguish between up-to-date replica and stale replica.

Chapter-3

Map-Reduce Framework

Functional Programming

- Functional programming is the declarative programming paradigm that treats computation as the evaluation of mathematical functions.
- It means programming is done with expressions or declarations rather than statements.
- The output value of a function only depends on the arguments passed to that function. It implies that passing the same argument value yields the same output each time the function is executed.

Benefits of Functional Language

1. It has no side effects.
2. The function does not rely upon any external states, so it is easier to maintain.
3. It does not have any assignment statements.
4. It does not make use of variables, so there is no change in state.
5. It has simple and uniform syntax.
6. It can be implemented via interpreters rather than compiler.
7. It is mathematically easier to handle.
8. The programs can be made concurrent automatically.
9. The overhead in computation such as loops are not present, making the execution faster.
10. It encourages safe way of programming.
11. It is easier to refactor and changes in design are easier to implement.
12. It is easy to test and debug in isolation.

Example of Functional program in Python

The given example shows the implementation of functional programming in Python to print a list with first 10 Fibonacci number.

```
        Fibonacci = (lambda n, first = 0, second = 1:
                    [ ] if n == 0 else
                    [first] + Fibonacci (n-1, second, fir
s   t + second) )


        print (Fibonacci (10))
```

Functional programming Vs. Imperative programming

1. In imperative programming, the programmer focuses on how to perform task and how to track changes. In functional programming, the programmer focuses on what information is desired and what transformations are required.

2. State changes in imperative programming. State change do not exist in functional programming.

3. Order of execution is important in imperative programming but not important in functional programming.

4. Flow control is managed using loops, conditions in imperative programming while using function call and recursion in functional programming.

5.

Map Reduce Fundamentals

Map Reduce

- MapReduce is a programming model used as an implementational tool for processing and generating big data sets using a parallel and distributed algorithm in a cluster.
- It generally uses split-apply-combine strategy for data analysis.
- It uses map and reduce functions commonly used in functional programming.

Basic Components of Map Reduce

- MapReduce is composed of two components. They are Mapper and Reducer.
- Mapper is the component that execute map () function. The real input to the system is the input for the mapper. The map () function when executed, the given input is converted to a key/value pair to generate intermediate key/value pairs.
- Reducer is the component that execute reduce () function. The intermediate key/value pair generated from mapper is the input to reduce () function. It merges all the intermediate values associated to the same key.

Usage of MapReduce

1. Distributed sort
2. Web link-graph reversal
3. Web access log stats
4. Inverted index construction
5. Document clustering
6. Machine learning
7. Statistical machine translation

Data Flow (Architecture)

Assumptions

Let us consider that:
M represents number of map task.
R represents number of reduce task.
W represents number of machines.

Data Distribution

- The input data files are distributed into M number of pieces on the distributed file system.
- The intermediate output of map function is stored in the local disk of each machine as an intermediate file.
- The final output file is written to distributed file system.

Execution Overview

1. Map
- It reads the contents of corresponding input partition.
- It performs user defined map computation to create intermediate pairs.
- It periodically writes the output pairs to the local disk.

2. Reduce / Fold
- It iterates over the ordered intermediate data obtained as the output of map function.
- When each unique key is encountered, the values are passed to the user's reduce function.
- The output of reduce function is written to the output file on global file system.

Distributed Execution

- The MapReduce library in the user program splits the input files into M pieces. It then starts up many copies of the program on the cluster of machines.

- One of the copies acts as a master and others as workers. There are M map tasks and R reduce tasks. The master selects the idle workers and assigns each one either a map task or a reduce task.

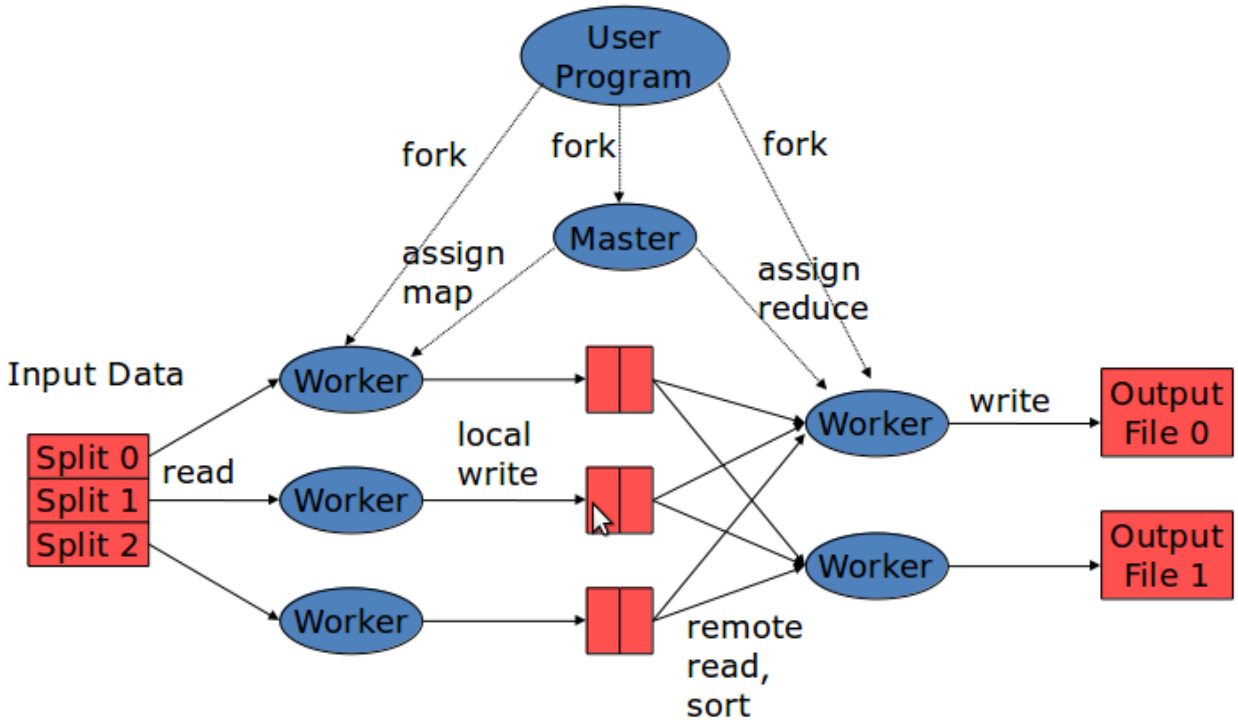
- A worker who is assigned with the map task reads the content of the input split. The user defined map function generates key/value pairs from the input split. The intermediate key/value pair produced are stored in the local disk partitioned into R regions by the partitioning function.

- The locations of the pairs in the local disks are passed to the master, which is responsible to forward to the reduce worker.

- When the reduce worker get notified from the master about the locations, it uses remote procedure calls to read the data from the local disks of map workers. After reading all the intermediate data, it sorts them by the intermediate keys.

- The reduce worker iterates over the sorted intermediate data and for each unique key it encountered, it passes the key and the corresponding set of intermediate values to the user's Reduce function. The output of the Reduce function is appended to a final output file for this reduce partition.

- When all map and reduce tasks are completed, the master wakes up the user program.



Combiner Function

- Combiner function is the function that is responsible to accept the output of the Map function, summarize the output records with the same key and pass each key data to one of the Reduce function.
- It helps to segregate data into multiple groups for Reduce phase.
- It helps in easier processing of the Reduce task.
- It produces output as key-value collection pairs.

Real World Problems

Map Reduce Program to find word frequency

Implementation in JAVA

```
public class WordCount {
         public static class WordMapper extends Mapper
         {
                        private final static IntWritable on
e = new IntWritable(1);
                        private Text word = new Text();

                        public void map(Object key, Text va
lue, Context context) throws IOException, InterruptedExc
eption
                        {
                                       StringTokenizer itr = new
StringTokenizer(value.toString());
                                       while (itr.hasMoreTokens(
))
                                       {
                                                      word.set(itr.n
extToken());
                                                      context.write(
word, one);
                                        }
                         }
          }

          public static class CountReducer extends Reduc
er
         {
                         private IntWritable result = new I
```

```
ntWritable();
                        public void reduce(Text key, Itera
ble values, Context context) throws IOException, Interru
ptedException
                        {
                                int sum = 0;
                                for (IntWritable val : val
ues)
                                {
                                        sum += val.get();
                                }
                                result.set(sum);
                                context.write(key, result)
;
                         }
            }


            public static void main(String[] args) throw
s Exception
            {
                        Configuration conf = new Configurat
ion();
                        Job job = Job.getInstance(conf, "wo
rd count");

                        job.setJarByClass(WordCount.class);
                        job.setMapperClass(WordMapper.class
);
                        job.setCombinerClass(CountReducer.c
lass);
                        job.setReducerClass(CountReducer.cl
ass);

                        job.setOutputKeyClass(Text.class);
                        job.setOutputValueClass(IntWritable
```

```
.class);

                    FileInputFormat.addInputPath(job, ne
w Path(args[0]));
                    FileOutputFormat.setOutputPath(job,
new Path(args[1]));

                   System.exit(job.waitForCompletion(tru
e) ? 0 : 1);
          }
}
```

Implementation in Python

```
import sys


class mapper(object):


      def __init__(self, text):
            super(mapper, self).__init__()
            self.text = text


      def map(self):
            self.words = self.text.split()
            self.map_output = []
            for word in self.words:
                  self.map_output.append({word : 1})


class reducer(object):
```

```
      def __init__(self, map_list):
            super(reducer, self).__init__()
            self.map_list = map_list


      def reduce(self):
            self.reduce_output = {}
            for length in range(0,len(self.map_list)):
                  current_pair = self.map_list[length]
                  current_word = current_pair.keys()
                  current_count = 0
                  for i in self.map_list:
                        key = i.keys()
                        if current_word == key:
                              current_count = current_coun
t + 1
                  self.reduce_output[current_word[0]] = cu
rrent_count


text = 'Ram is a good boy He is one of the best boy in t
he class'


mapper_obj = mapper(text)
mapper_obj.map()


map_output = mapper_obj.map_output
```

```
reducer_obj = reducer(map_output)
reducer_obj.reduce()


for frequency in reducer_obj.reduce_output.keys():
      print '%s = %s' %(frequency, reducer_obj.reduce_out
put[frequency])
```

Fault Tolerance in Map Reduce

Machine Failure

- The master pings workers regularly to detect the failures.
- If no response is returned by the worker, the worker is considered to be faulty.
- When the map worker failure is encountered, the map task to that worker is reset to idle and resheduled to another map worker. The reduce workers are notified about this resheduling.
- When the reduce worker failure is encountered, the task that are not completed i.e. in progress task are reset to idle and rescheduled to another reduce worker.
- In case of failure of the master, the complete MapReduce task is aborted and is notified to the client.

Chapter-4

NoSQL

Structured and Unstructured Data

Structured Data

- Structured data is defined as the data that can fit into the fixed record or file.
- Such data can be stored in relational databases and spreadsheet.
- It is easily searchable by the basic algorithm or basic queries.
- It is written in the format that is easy for machine to understand.
- It is simple to enter, store, query and analyze.
- It must be strictly defined in terms of field name and type.

-The example of structured data is:

```
 Employee            Age                Salary
   ---------------------------------------
Ram                   34                  30000
Shyam                 20                  12000
Hari                  24                  20000
```

Unstructured Data

- Unstructured data is defined as the data that cannot be classified and cannot fit in a fixed record.
- Such data cannot be stored in relational databases as they do not possess any well-known structure.
- It generally allows keyword based queries or sophisticated conceptual queries..
- It is written in the format that is easy for humans to understand.
- It is being increasingly valuable and available.

- It is difficult to unbox, understand and analyze.
- It refers to the free text data.

- The example of structured data includes personal messaging, business documents, web content and so on.

Scaling of Database

The scaling techniques used in traditional databases are as follows:
1. Vertical Scaling
- It is also known as scaling up.
- It is achieved by upgrading the new hardware requirements to fulfill the processing demands.
- It is generally configured in the single machine.
- It may hamper availability of resources at the time of upgrade.

2. Horizontal Scaling
- It is also known as scaling out.
- It is achieved by adding the necessary hardware parallel to the currently available hardware.
- It is generally configured in multiple machines.
- It do not hamper availability of resources.
- It involves database sharing and database replication.
- Database sharing refers to the stripping of data and storing in multiple machine to allow concurrent access to the database.
- Database replication refers to the storing of copies of database in multiple machines to ensure availability and prevent from single point failure.

CAP Theorem

- CAP stands for Consistency, Availability and Partition Tolerance.
- CAP theorem is used to describe the limitations of distributed databases.
- Consistency refers to the process of maintaining the same state of data in all the replicas at any instance.
- Availability refers to the process of successfully operate at any instance even if some node crashes.
- Partition tolerance refers to the process of operation of the system in presence of network partition.

- CAP theorem states that any distributed database with shared data can have it must two of the three desirable properties (Consistency, Availability and Partition Tolerance).

- So, CAP theorem can be summarized as: For any distributed database, one of the following can hold:
1. If a database guarantees availability and partition tolerance, it must forfeit consistency. Egg: Cassandra, CouchDB and so on.
2. If a database guarantees consistency and partition tolerance, it must forfeit availability. Egg: HBase, Mongo DB and so on.
3. If a database guarantees availability and consistency, there is no possibility of network partition. Egg: RDBMS like MySQL, Postgres and so on.

Why absolute consistency is generally sacrificed?

- The large companies generally scales out vertically, that means the network partition is present that is spread over thousands of nodes.
- Among consistency and availability, such companies have to choose only one as per the CAP theorem.
- Due to large number of network partitioning, there is high chance of failure of some nodes.
- If the data is not become available on time, it means a huge loss to the company.
- So, the company choose availability in terms of financial gain and

trust from the client.
- In this sense, strict consistency is generally sacrificed.

- In other to maintain consistency, the company follow eventual consistency process instead of strict or absolute consistency.

Eventual Consistency

- A database is said to be eventually consistent if all the replicas will gradually become consistent in the absence of the updates.
- It means that the system will be consistent eventually not at the same instance.

Tunable Consistency

- In some distributed databases that forfeits consistency for availability like Cassandra, it extends the concepts of eventual consistency by offering tunable consistency for any given read or write operations.
- In case of tunable consistency, the client application is responsible to decide how consistent the requested data should be.

Properties of Distributed Database

1. Basically Available (guarantees availability)
2. Soft-state (state of system may change over time)
3. Eventual consistency (system becomes consistent over time)

Generally, it is known as BASE properties.

Taxonomy of NoSQL Implementation

NoSQL

- NoSQL stands for Not only SQL.
- It is a class of database management system with the following properties:
1. It does not use SQL as query language.
2. It has distributed and fault tolerant architecture.
3. It is schema less.
4. It follows BASE properties rather than ACID properties.
5. Consistency is traded in favor of Availability.

RDBMS vs NoSQL

- RDBMS is a table-based database. NoSQL is a schema less database.
- RDBMS stores data in rows and columns in the form of table. NoSQL stores data in multiple collections and nodes.
- RDBMS must be scaled up vertically. NoSQL can be scaled out horizontally.
- If the data does not fit in RDBMS table, the table must be restructured. NoSQL can handle unstructured data in efficient manner.

NoSQL Taxonomy

NoSQL databases are of four types. They are as follows:
1. Document Store
2. Graph Database
3. Key-Value Store
4. Columnar Database

Document Store

- It is used to store documents in some standard format such as PDF, JSON and so on.
- It is referred as Binary Large Objects (BLOB).
- The documents which are stored can be indexed.
- Egg: MongoDB, CouchDB

Graph Database

- The data are represented in the form of graph (i.e. nodes and vertices).
- The node represents the data.
- The vertex represents the relation of nodes.
- Egg: Vertex DB, Neo4j

Key-Value Store

- The data are represented in the form of keys and complex values like list.
- Keys are stored in a hash table which can be distributed easily.
- It supports regular CRUD operations.
- Egg: Amazon DynamoDB, Redis

Columnar Database

- It is the hybrid form of Relational database system and key-value store.
- The values are stored in a group of zero or more columns.
- Values are queried by matching keys.
- Egg: HBase, Cassandra

Basic Architecture of HBase, Cassandra and Mongo DB

Cassandra

- Cassandra is a distributed database that is used to manage large amount of structured data spread out across the world.

- The reasons for choosing Cassandra are as follows:
1. value availability over consistency
2. require high write throughput
3. high scalability required
4. no single point of failure

- The key properties are as follows:
1. It is column-oriented database.
2. It is scalable, fault tolerant and consistent.
3. It supports ACID properties.

- The key components of Cassandra are as follows:
1. Node (place where data is stored)
2. Data center (collection of related nodes)
3. Cluster (collection of one or more data centers)
4. Commit logs (write operation is written for crash recovery)
5. Mem table (after commit log, data is written to mem table)
6. Stable (disk file to which data is flushed from mem table when its contents reach threshold value)
7. Bloom filter (algorithm for testing whether the element is a member of a set)

- Cassandra is accessed through its node using Cassandra Query Language (CQL). It treats database (Key space) as a container of tables.

Data Modeling in Cassandra

- Key space is the outermost container for data. It consists of one or more column families.
- A column family is a container of a collection of rows.
- Each row contains ordered columns.
- A column is the basic data structure of Cassandra with three values; key or column name, value and time stamp.

Tunable Consistency - Write Consistency Level

- It indicates the number of replicas on which write must succeed before returning acknowledgement to the client application.
- It consists of following levels:

1. ANY (must be written to at least one node)
2. ALL (must be written to commit log and mem-table on all replica nodes in the cluster for that row)
3. EACH_QUORUM (written to commit log and mem-table on a quorum of replica nodes in all data centers.)
4. LOCAL_ONE (send to and acknowledged by at least one replica node in the local data center)
5. LOCAL_QUORUM (written to commit log and mem-table on a quorum of replica nodes in the same data center as coordinator node.)
6. LOCAL_SERIAL (written conditionally to the commit log and memory table on a quorum of replica nodes in the same data center.)
7. ONE (written to the commit log and memory table of at least one replica node.)

Tunable Consistency - Read Consistency Level

1. ALL (Returns the record with the most recent timestamp after all replicas have responded)
2. EACH_QUORUM (Returns the record with the most recent timestamp once a quorum of replicas in each data center of the cluster has responded)
3. LOCAL_SERIAL (confined to the data center)
4. LOCAL_QUORUM (Returns the record with the most recent timestamp once a quorum of replicas in the current data center as the coordinator node has reported)
5. LOCAL_ONE (Returns a response from the closest replica, as determined by the snitch, but only if the replica is in the local data

center)
6. ONE (Returns a response from the closest replica, as determined by the snitch)
7. QUORUM (Returns the record with the most recent timestamp after a quorum of replicas has responded regardless of data center)
8. TWO (Returns the most recent data from two of the closest replicas)
9. THREE (Returns the most recent data from three of the closest replicas)

MongoDB

- MongoDB is a cross platform and document-oriented database.
- It uses JSON format.
- It works on the concept of collection and document.
- Collection is a group of MongoDB documents. It is equivalent to table in RDBMS. It does not enforce schema.
- Document is a set of key-value pairs. Each document has dynamic schema.

- The basic considerations while designing schema in MongoDB are as follows:
1. Design according to user requirements.
2. Combine objects into one document if they are used together.
3. Duplicate the data.
4. Do joins while write, not on read.

- Example:

Create a collection named "posts", insert following records: title: MongoDB, description: MongoDB is a NoSQL database, by: Tom, Comments: We use MongoDB for unstructured data, likes:100.
Write a query to search title of the post written by Tom.

```
> use test //create database name test and use that data
base
> db.createCollection("posts") //create collection named
posts
> db.posts.insert({
         title : "MongoDB",
         description : "MongoDB is a NoSQL database",
         by : "Tom",
         comments : "We use MongoDB for unstructured dat
a",
         likes : 100
    })
> db.posts.find({"by" :  "Tom"}, {"title" : 1, _id : 0})
.pretty() // title of the post written by Tom
> db.posts.find({"likes" : {$lt : 150}}).pretty() // lik
es less than 150
```

Chapter-5

Searching and Indexing Big Data

Full Text Indexing and Searching

Indexing

- The process of converting text or original data into a format that allows highly efficient cross reference lookup to facilitate rapid searching is known as indexing.
- It is the first step in search application.
- It eliminates the slow sequential scan during searching.

Searching

- Searching is the process of looking up the words in an index to find documents where they appear.

Indexing Process

1. Crawl all the pages of the seed list and persist them to disk.
2. Extract the file content and persist it to disk.
3. Crawl the seed list page from the disk.
4. Index the seed list entries into documents.
5. Write the documents to the index.
6. Repeat until all the seed list page have been crawled.

Document Search

- Document search is the component of Document Management System that is used to search for documents based on various criteria.
- It also supports text search within the original document.

Indexing with Lucene

Lucene

- Lucene is a high performance and scalable JAVA based information retrieval library that adds indexing and searching capabilities to the application.
- It is the core of any search application that provides all the functionality required by a search application except for the user interaction.
- It is an open source library.
- It is competitive in engine performance, relevancy and code maintenance.

Components of Search Application

1. Acquire Raw Content
- It is the first process of any search application.
- It helps to collect the target contents on which search application needs to be performed.
- All these contents are in raw format.

2. Build Document
- Document in Lucene is a collection of fields, which is easy to understand and interpret.
- The collected raw contents are built into documents so that the search application can easily interpret it.
- It is the process of converting raw contents into application understandable format.

3. Analyze Document
- The document is analyzed to find out which part of the document is the candidate for indexing.

4. Indexing Document
- Indexing is done for a document.
- It helps in retrieving document based on keys instead of entire document content.

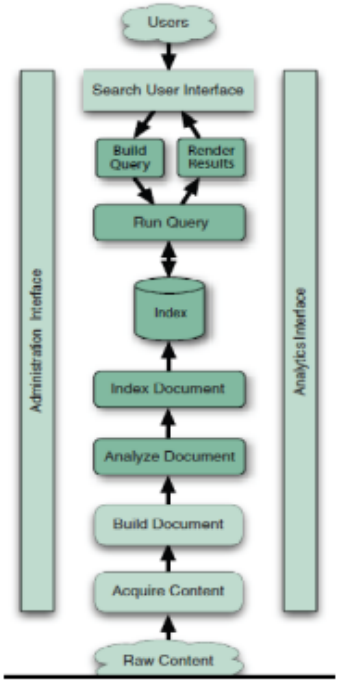
- It creates index as the output.
- All the indexes are stored in the database.

5. User Search Interface
- The search application can make queries once the index is stored in database.
- It is the interface provided to the user to facilitate user to make query.

6. Build Query
- Once a user make request to search a text, the application prepare a Query object using that text.
- This object is used to inquire index database.

7. Search Query
- The index database is then checked using the query object to obtain the relevant details or contents.

8. Render Request
- After receiving the result from the search query, the application should show the results to the user.
- The user interface is used for this process.
-



Lucene Indexing Process

1. Document Analyzer
2. Index Writer
3. Index Store

Analyzers

1. White Space Analyzer
- It splits the tokens based on white spaces.

2. Simple Analyzer
- It splits the tokens based on non-letters and then lowercase it.

3. Stop Analyzer
- It splits the tokens based on non-letters and then lowercase it and also removes the stop words.

4. Standard Analyzer
- It splits the tokens based on certain token types like name, e-mail address and so on, lower case it, removes the stop words, common words, punctuation and so on.

```
For example: Consider the text:
     "The quick brown fox jumped over the lazy dog"


1. [The] [quick] [brown] [fox] [jumped] [over] [the] [la
zy] [dog]
2. [the] [quick] [brown] [fox] [jumped] [over] [the] [la
zy] [dog]
3. [quick] [brown] [fox] [jumped] [over] [lazy] [dog]
4. [quick] [brown] [fox] [jumped] [over] [lazy] [dog]
```

Distributed Searching with Elastic Search

Elastic Search

- Elastic search is the highly scalable open source full text search and analytic engine based on Lucene.
- It provides a distributed, multitenant-capable full-text search engine with HTTP web interface and schema free JSON documents.
- It is developed in JAVA.
- It is used to search all kinds of documents.
- It achieves fast search response due to index searching.

Benefits of Elastic Search

- It is open source.
- It is easy to deploy.
- It can be scaled vertically as well as horizontally.
- It provides easy to use API and supports various programming and scripting languages.
- It has active community and well formatted documentation.

Write the equivalent Elasticsearch query to find the unique employee_id and corresponding salary.

```
GET /_search
{
        "size" : 0,
        "aggs" : {
             "distinct_employee" : {
                    "terms" : {
                           "script" : "[doc['employe
e.employee_id'].value, doc['employee.salary'].value]",
                           "size" : 1000
                      }}}}
```

Chapter-6

Case Study: Hadoop

Introduction to Hadoop Environment

What is Hadoop?

- The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models.
- It is designed to scale up from single servers to thousands of machines, each offering local computation and storage.
- Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.
- It makes use of MapReduce programming model to process data sets.

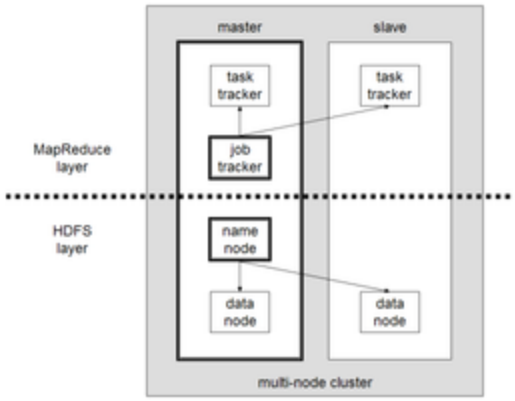
Why to use Hadoop?

- It facilitates processing of huge data sets easily on large clusters of computers.
- It provides efficient, reliable and easy to use common infrastructure.
- It supports processing of data stored in each node, without necessity of moving data for initial processing.
- It does not rely up on hardware to provide fault tolerant and high availability.
- Servers can be added or removed dynamically without interrupting the operation of the system.
- It is open source, so is available freely.
- It is compatible on almost all the platforms.

Data Flow in Hadoop

Hadoop Master/Slave Architecture

- A basic master and slave architecture of Hadoop is shown in given figure:



- The above architecture of Hadoop depicts four daemons:
1. Name Node
2. Data Node
3. Job Tracker
4. Task Tracker

Name Node

- The name node is the master of HDFS that helps to direct the data node daemons of slaves to perform the low level I/O tasks.
- It is responsible to keep track of how the files are broken into file blocks, which nodes store those blocks and overall performance of the distributed file system.
- It is memory and I/O intensive. It means that the name node functionality mainly performs memory and I/O tasks.
- It has a single point of failure.

Data Node

- The data node is contained by all the slave nodes of a cluster.
- This daemon is used to perform work of the distributed file system such as read and write HDFS blocks to actual files on local disk.
- Job communicates directly with data node daemon to process the local files corresponding to that block.
- Data replication can be initiated by communication of data nodes.
- It is responsible to report back to name node about the local changes.

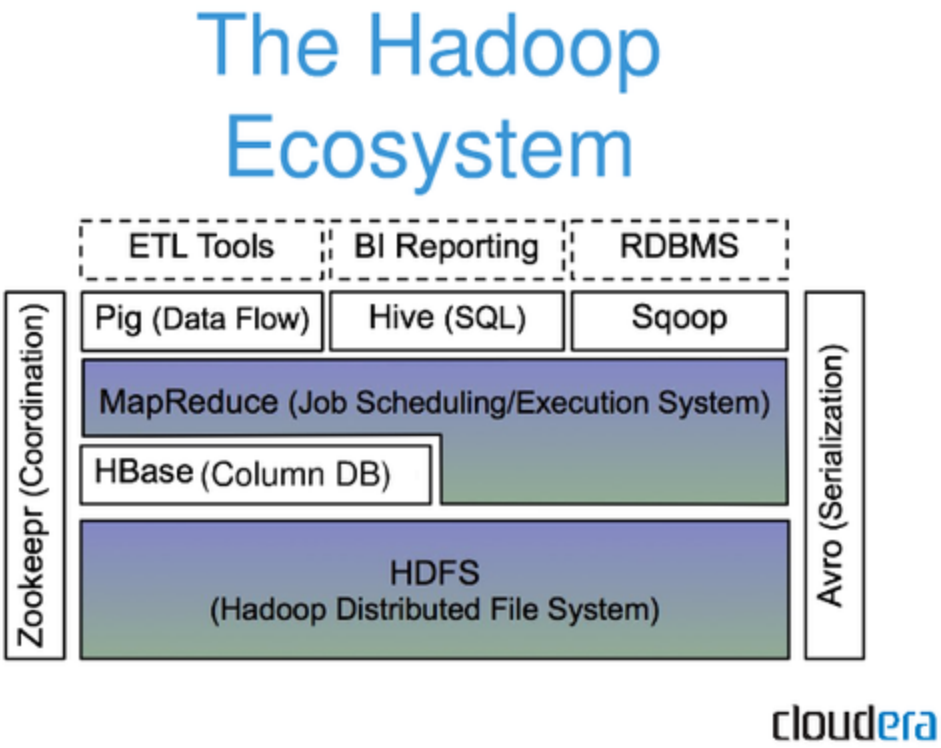
Job Tracker

- Job tracker is the mediator between the application and Hadoop.
- It is the master of MapReduce layer of Hadoop.
- It determines the execution plan by determining which files to process, assigns nodes to different tasks and monitors all the running tasks.
- If task on any node fails, job tracker is responsible to re-launch the job on another node of the cluster.
- It is responsible for overall work of MapReduce job.

Task Tracker

- Task tracker are present in the MapReduce layer of each nodes.
- It manages the execution of individual MapReduce task on each slave nodes.
- It performs the individual task assigned by the Job tracker.
- If the task tracker fails to execute the assigned job, the job is rescheduled by the job tracker.
- A task tracker on each slave node spawns a separate JVM process to prevent itself from failing if the running job crashes its JVM.

Hadoop Ecosystem



- The Hadoop ecosystem is shown in given figure:

- The Hadoop ecosystem consists of following components:
1. Hadoop Common
2. Hadoop Distributed File System (HDFS)
3. YARN
4. Hadoop MapReduce

- Pig is a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs.

- Sqoop is a tool designed for efficiently transferring bulk data between Apache Hadoop and structured datastores such as relational databases.

Hadoop Common

- It consists of the common utilities that support other Hadoop modules.
- It is also known as Hadoop core.
- It provides file system and operating system level abstractions.
- It contains Java archive files and scripts needed to start Hadoop.

Hadoop Distributed File System (HDFS)

- It is a file system written in JAVA based on Google's GFS.
- It is a distributed, scalable and portable file system used by Hadoop.
- It is responsible to store data in the cluster.
- It provides redundant storage for huge data sets.
- The data files are split into blocks and are distributed across the nodes in the cluster.
- The details of which blocks forms a file and where they are stored is tracked by the name node.
- The default replication of data in HDFS is 3 fold.
- When a client wants to retrieve data, it communicates with name node to determine the blocks and node storing them; and then the client directly communicates with the data node to read the data.

Yet Another Resource Negotiator (YARN)

- It provides the solution to the scalability problems of classical MapReduce.
- It is a framework for job scheduling and cluster resource management.
- It enables Hadoop to support more varied processing approach and a broader array of applications.

Map Reduce

- It is a method of distributing computations across multiple nodes.
- Each node processes the data that is stored at that node.

Query Language for Hadoop

Hive

- Hive is the query language used by Hadoop.
- It is used to query and manage large data sets of a Hadoop cluster using an SQL like language called HiveQL.

GFS and HDFS

1. File Structure:
- In GFS:
Divided into 64 MB chunks
Chunks identified by 64-bit handle
Chunks replicated (default 3 replicas)
Chunks divided into 64KB blocks
Each block has 32-bit checksum

- In HDFS:
Divided into 128 MB blocks
Name node holds block replica as 2 files (one for data and other for checksum)

2. Architecture Comparison
- In GFS, leases at primary default to 60 seconds. In HDFS, no leases (client decides where to write.)
- The read and write process by client is similar in both GFS and HDFS:

```
Read Operation:
-----------------------------
1. Client sends request to master
2. Caches lists of replicas
3. Locations provided for limited time


Write Operation:
------------------------------
1. Client obtains replica locations and identify primary
replica
2. Client pushes data to replica
3. Client issues update request to primary replica
4. Primary replica forwards write request.
5. Primary receives replies from replicas
6. Primary replica replies to the client
```

3. Replica Management:
- No more than one replica on one node and no more than two replicas on same rack.
- For reliability and availability, replicas are stored in different nodes present at different racks.

Q) For a Hadoop cluster with 128MB block size, how many mappers will Hadoop infrastructure form while performing mapper functions on 1 GB of data?

Determining the number of mappers for Hadoop
---
1. If the file is split table:
- Calculate total size of input file
- No of mapper = Total input file size / Block size (Split size)

2. If the file is not split table:
- No of mapper = No of input files

- If the file size is too huge, it becomes bottle neck to the performance of MapReduce.

Solution to numerical problem:
---
1. Assuming the file to be split table:
No of mappers = 1024 / 128 = 8

2. Assuming the file to be not split table and data is contained in single file:
No of mappers = 1

Hadoop and Amazon Cloud

- Amazon cloud provides an interface to create and manage fully configured, elastic clusters of instances running Hadoop and other applications in the Hadoop ecosystem.
- Amazon EMR securely and reliably handles a broad set of big data use cases, including log analysis, web indexing, data transformations (ETL), machine learning, financial analysis, scientific simulation, and bioinformatics.
- It uses Amazon EMR to easily install and configure tools such as Hive, Pig, HBase and many other tools on the personal cluster.
- In the Hadoop project, Amazon EMR programmatically installs and configures applications including Hadoop MapReduce, YARN and HDFS, across the nodes in the cluster.
- By using EMR file system on Amazon cluster, one can leverage Amazon S3 as the data layer for Hadoop.
- By storing data on Amazon S3, one can decouple compute layer from storage layer, allowing one to size the Amazon EMR cluster for the amount of CPU and memory required for the workloads instead of having extra nodes in the cluster to maximize on-cluster storage.
- EMR file system is customized for Hadoop to directly read and write in parallel to Amazon S3 performance.
- Hadoop in Amazon EMR can be used as an elastic query layer.

Advantages of Hadoop on Amazon EMR

1. Increased speed and agility
- One can initialize a new Hadoop cluster dynamically and quickly, or add servers to the existing Amazon cluster.
- It reduces time taken for the resources to be available to the users.
- Using Hadoop on Amazon AWS lowers cost and time it takes to allocate resources.

2. Reduced administrative complexity
- Amazon EMR automatically addresses the Hadoop infrastructure requirements, that reduce the complexity to administration.

3. Integration to other cloud services
- The Hadoop environment in Amazon cloud can be integrated with other cloud services like Amazon S3, Amazon DynamoDB and so on easily.

4. Flexible capacity
- With Amazon EMR, one can create clusters with the required capacity within minutes and use auto scaling to dynamically scale out and scale in nodes.