HW7

1. **Why was Hadoop platform for Big Data processing introduced and became so popular? What Features made it such a widely-adopted solution for data processing? What are its benefits over existing solutions? Explain briefly.**

Sol: The term 'Big Data' means innovative techniques and technologies to capture, store, distribute, manage and analyze petabyte or larger-sized datasets with high-velocity, volume and variety of data. In Big data the data can be structured, unstructured or semi-structured data which leads to the resulting in impotence of conventional data management methods.
In the present scenario data is generated from various peculiar sources and can reach in to the system at various rates. So these large amounts of data should be processed in an inexpensive and in an efficient manner parallelism is used. Mainly lots of amount of data is generated from web server logs and Internet clickstream data, social media content and social network activity reports, text from customer emails and survey responses, mobile-phone call detail records etc.

For any fast growing organization major problem it should face is storage of data. If the case of banking and financial services, the data security is more prior along with storage. Advancements in technology are targeting at storing data in secured way along with easy retrieval and with low cost and space. Idea evolved from this problem is "big data". Big data is term that indicates large data sets. These data sets are used to solve the challenges faced in data analysis, sharing, storage, transfer, querying and privacy. There by it reduced risk of handling data. Big data analytics helps in decision making, operational efficiency of an organization. The best advantage with big data is it can handle lots of data with low cost and less time. It has three main characteristics volume, velocity and variety. It can store any kind of data with more speed and volume.

Let us look into a real time scenario in where big data is used for handling sentimental data. Facebook is a giant social networking site from which daily data flows into our lives. This data is sentimental data because it handles opinions of the many users. Facebook have to handle lots of historical data besides collecting data on present happenings in the world. To store all this data, in traditional way we use main frames servers or high capacity databases. It involves cost in installation and day by day data increases, and then they have to install new servers and should maintain them. BIG DATA is the solution for this problem. Big data stores data in cloud so it installation and maintenance cost is reduced. To retrieve data from data base we use querying the time of data retrieval depends on load in server and connectivity, but in case of big data as it is storing data in cloud data retrieval happens in no time. One may think storing data in cloud is not safe but in case of Big data all the data is encrypted. Facebook has many chats and lots of personal data of the users so it should be kept safe. So cloud encrypts data.

Let us go into a deep dive and see how the data is stored in cloud and retrieved from it. In Facebook there is day by day data on all categories, so it has to store data based on "DATE". Once it collected the current data it is processed by some enterprise data warehouse and then refined data is obtained. The refined data is sent to cloud through some frame works. The most common frame work of big data is "Hadoop". It stores data in nodes (machine), for data retrieval we need to have a track of data where it is stored and how. So there are two kinds of nodes name node which is keeps track of data storage, and data node which stores data and processes them in the time of retrieval. Name node has 2 in memory tables, one will map which block is stored in which data node and the other tracks data node to block mapping. Data node sends some acknowledgement to name node in form of message where the name node knows that data node is alive. Big data has an efficient failure recovery system. Each node is replicated many times, in case of failure of any node; name node immediately takes action to replace it with its duplicate node. Data node gives block level information to name node in case of any mismatch of checksum it will let name node about the disk corruption for that block. It will maintain efficient disk usage without leaving any empty memory space. In some cases, newspapers will collect data and they will do some analysis and collect statistics, for these operations require doing some arithmetic and logical transformations on huge volume of data. To apply some arithmetic operations, big data will follow some basic steps:

- The data is stored in name node and data node
- The operation to be done as the required logic is written in any programming language.
- The data then enters to logical program as input and then required output is obtained which is again stored in data node

The data thus processed. Historical data is to be stored in cloud; most popular cloud is "Amazon web server." The processed data is encrypted using any scripting language and them compressed and sent to the cloud.

Big data is not replacement of relational data bases, enterprise data warehouse. But it is an advancement of data storage. It is highly secured as the data is encrypted and compressed in all possible ways. Organization needs only proper maintenance and skilled operating personnel to handle the data and cloud.

2. What is MapReduce and what are its characteristics? List its advantages and disadvantages. Compare it with MPI-based approaches to parallelize a certain task.

Sol: MapReduce is a programming model in the Hadoop architecture for the implementation of processing the large data sets and generating large data sets with the help of parallel and distributed algorithm on a particular cluster. Main theme of map reduce is to split the input large data sets in to independent chunks and these chunks are processed in a parallel manner with the help of map tasks.

The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as

an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

Let us take an example, large data collection have: {web, green, sun, moon, land, part, web, green…}.we need to Count the occurrences of the different words in the given set of collection

- We should start from scratch.
- We should add and relax the given constraints.
- We should do the incremental design, improving the solution for the performance scalability.

MAP: input data -> <key, value>pair

REDUCE: <key, value>pair>>><result>

Splitting the data to the multiple processors and compute data (Master node have two key functional pieces that make up Hadoop: storing lots of data (HDFS), and running parallel computations on all that data (Map Reduce). Slave Nodes make up the vast majority of machines and do all the dirty work of storing the data and running the computations. Each slave runs both a Data Node and Task Tracker daemon that communicate with and receive instructions from their master nodes.)

CHARACTERSTICS OF MAPREDUCE:

1. MapReduce uses very large scale data: peta,exa-bytes with high velocity.

2. Main important characteristics is write once and read the data many times: it also allows the data for parallelism without any mutexes.

3. It has two main Stages i.e MAP and Reduce.

   Map stage: The mapper's job is to process the input data. Generally the input data is in the form of file and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.

   Reduce stage: This stage is the combination of the Shuffle stage and the Reduce stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.

4. Reduce operation starts only after the completion of the Map.

5. Combine and partition are the other supporting operations for the map reduce.

6. Physical processor is used for performing both map and reduce operations.

7. Number of operation the processes map and reduce are configurable.

8. Commodity hardware and storage are the benefits for map reduce.

9. Run time will take care of splitting data and passing the data for the operations, and Operations are provisioned near the data.

Advantages of Map Reduce:

1. Scalability: It is highly scalable.

2. Cost-effective solution: since it is highly scalable structure also implies that it comes across as a very cost-effective solution.

3. Flexibility: Many organizations can make use of Hadoop MapReduce programming to have access to various new sources of data and also operate on different types of data.

4. Fast: MapReduce programming, are also generally located in the very same servers, which allows for faster processing of data.

5. Security and Authentication: MapReduce works with HDFS and HBase security that allows only approved users to operate on data stored in the system.

6. Parallel processing: It divides tasks in a manner that allows their execution in parallel.

7. Resilient nature: One of the biggest advantages is fault tolerance. Hadoop MapReduce has the ability to quickly recognize faults that occur and then apply a quick and automatic recovery solution.

Disadvantages of Map Reduce:

1. Biggest disadvantage of MapReduce is high latency.

2. Programming model is very restrictive.

3. Joining two large data sets with complex conditions.

4. If the computation of a value depends on previously computed values, then MapReduce cannot be used.

5. Cluster management is hard.

6. Managing job flow isn't trivial when intermediate data should be kept.

Now, we need to Compare MapReduce with MPI-based approaches to parallelize a certain task,

We know that MapReduce is suitable for non-iterative algorithms where nodes require little data exchange to proceed where as MPI is appropriate for iterative algorithms where nodes require data exchange to proceed . We compare two methods for implementing ranking and selection algorithms in parallel computing environments. The Message Passing Interface (MPI) provides the

programmer with complete control over sending and receiving messages between cores, and is fragile with regard to core failures. In contrast, MapReduce handles all communication and is quite robust, but is more rigid based on algorithm how it is coded. But MPI is more efficient of the two environments, although MapReduce is a reasonable choice. So, MapReduce might be attractive in environments since Map-reduce provides a fault-tolerant mechanism, when one node fails, map-reduce restarts the same task on another node. All MPI processes will exit if one of them fails.

3. **Provide pseudocode for sorting integers in an out-of-core fashion (i.e., integers cannot fit in memory) both in MapReduce (both mapper and reducer code) and in MPI. Think about the flow of sorting: reading the input, perform the sorting algorithm, merge all intermediate results, and write the final sorted output.**

Sol: In My MapReduce code, Initially It reads arguments number of mappers, number of reducers, input file and output file. And Input file contains line number and integers. It takes input file from hdfs .Mapper will map the all the keys and values and reducer will sort the data by using key and values. All the sorted data will be stored in output file.

MapReduce Code : :
```
public class Sorting {
  public static void main(String[] args) throws Exception {
/*Declaring number of mappers and reducers */
  int num_of_mappers=1;
  int num_of_reducers=1;
/* Reading number of mapper and reducers from user */
  num_of_mappers=Integer.parseInt(args[2]);
  num_of_reducers=Integer.parseInt(args[3]);
/*Reading Input file */
  Path input = new Path(args[0]);
/*Reading out file name */
  Path output = new Path(args[1]);

/*  Create configuration  */
Configuration conf = new Configuration(true);

/* Create job for sorting intergers */
Job job = new Job(conf, " Sorting");

/* Setting  Mappers and reducers to the job */
job.setNumMapTasks(num_of_mappers);
job.setNumReduceTasks(num_of_reducers);
  job.setJarByClass(Sorting.class);
```

```
        /*Mentioning key and value to the job */
         job.setMapOutputKeyClass(IntWritable.class);
         job.setMapOutputValueClass(IntWritable.class);
         job.setOutputKeyClass(IntWritable.class);
         job.setOutputValueClass(IntWritable.class);
         job.setSortComparatorClass(IntComparator.class);
        /*  Adding Input to the job */
         FileInputFormat.addinput(job, input);
         job.setInputFormatClass(TextInputFormat.class);

        /* Adding  Output to the job */
        FileOutputFormat.setOutputPath(job, output);
        job.setOutputFormatClass(TextOutputFormat.class);

       /* Executing the  job */
         int code = job.waitForCompletion(true) ? 0 : 1;
         System.exit(code);  }

   public static class IntComparator extends WritableComparator {
     /* comparing intergers */
          public int compare(byte[] b1, int s1, int l1,byte[] b2, int s2, int l2) {

           Integer v1 = ByteBuffer.wrap(b1, s1, l1).getInt();
           Integer v2 = ByteBuffer.wrap(b2, s2, l2).getInt();
            return v1.compareTo(v2) * (-1);  }
  }
    /*Mapping Jobs*/
  public static class MapTask extends
      Mapper<LongWritable, Text, IntWritable, IntWritable> {
      public void map(LongWritable key, Text value, Context context)
      throws java.io.IOException, InterruptedException {
      String line = value.toString();
      String[] tokens = line.split(","); // This is the delimiter between
      int keypart = Integer.parseInt(tokens[0]);
      int valuePart = Integer.parseInt(tokens[1]);
      context.write(new IntWritable(valuePart), new IntWritable(keypart)); }
}


 /* reducer takes the output from a map as an input and combines those data tuples*/
   public static class ReduceTask extends
```

```
                Reducer<IntWritable, IntWritable, IntWritable, IntWritable> {
            public void reduce(IntWritable key, Iterable<IntWritable> list, Context context)
                throws java.io.IOException, InterruptedException {
                    for (IntWritable value : list) {
                        context.write(value,key);}}}
    }
/* Code Ends */
```

Now, we have to write MPI Code. In MPI Code, we have used few MPI collective routine operations. Firstly, read the input and output file Name. Then, initialize MPI with MPI_Init, the size and rank are obtained respectively using MPI_Comm_size and MPI_Comm_rank.Read the data into the buffer array containing inputs is distributed proportionally to the size to all participating processes with MPI_Scatter by the root process which collect them again after they are sorted using MPI_Gather.

MPI Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <string.h>
#include <mpi.h>

int main(int argc, char **argv)
{
 /*Declaring MPI Intilization */
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &npes);
MPI_Status status;
int i;
int rank, size;
long  offset;
/*read the arguments with this function */
parameters(argc, argv);/* Process program parameters */

if(rank==master){
  printf("size is %d\n",size);
/*Allocating data */
data= malloc(size*sizeof(long));
FILE * input;
input = fopen(filenameR,"r");  //reading a file
```

```
if(input == NULL){
printf("Error : File not found\n");
return 0;
}
for(i=0;i<size;i++){
scan =fscanf(input,"%d",&t);
data[i]=t;
}
/*divide the input size by nuber  of processors  to get each chunk Size */
chunksize =size/npes;
/*Distribute data propotionally to all the processes*/
MPI_Scatter( data,chunksize, MPI_Long, localdata, chunksize, MPI_Long, MASTER, MPI_COMM_WORLD);
/*Each process sort the data of chunk size */
 sort(localdata,0,chunksize-1);
/*Master collects all the local dta from all the processes*/
 MPI_Gather(localdata, chunksize, MPI_Long, data,chunksize, MPI_LONG, MASTER, MPI_COMM_WORLD);
/*free the local data */
free(localdata);
/*Till now,each processor performs sorting its own data.Now,we have to sort again on whole data,its performed by master */
if(rank==master){
 sort(data,0,size-1);
}
/*Now write the output to the outputfile */
outputfile=fopen(outputf,"w");
if(outputfile ==NULL)
 {
  printf("No memory in a file");
 }
/*Storing final output to outputfile */
for(i=0;i<size;i++){
fprintf(outputfile,"%l",&data[i]);

}
/*closing output file */
fclose(outputfile);
MPI_Finalize();
return 0;
}
```

```
int sort(int* arr, int size, int per_rank,){
MPI_Status status;
int share_per = per_rank + pow(2, size);
int pivot = arr[size/2]; /* Select the pivot */
int partition_pt = sequential_quicksort(arr, pivot, size,(size/2) -1); /* partition array */
int offset = partition_pt + 1;

/* Send partition based on size, sort the remaining partitions, receive sorted partition */
if (offset > size - offset)
 {
  MPI_Send((arr + offset), size - offset, MPI::INT, share_per, offset, MPI_COMM_WORLD);
   Seuentialsort (arr, offset, per_rank);
  MPI_Recv((arr + offset), size - offset, MPI::INT, share_per,MPI_ANY_TAG, MPI_COMM_WORLD,
  &status);
}
else{
  MPI_Send(arr, offset, MPI::INT, ch_per , tag,MPI_COMM_WORLD);
   Seuentialsort((arr + offset), size - offset, per_rank);
  MPI_Recv(arr, offset, MPI::INT, ch_per, MPI_ANY_TAG,MPI_COMM_WORLD, &status);
}}
```

4. Provide a table of features for Distributed File Systems (HDFS) and Parallel File Systems (PVFS). Discuss briefly similarities and differences.

Sol: Distributed file system should have the following features:

1. Performance: It is the average amount of time required to satisfy client requests which includes CPU time, time for accessing secondary storage and network time. Explicit file placement decisions should not be needed to increase the performance.

2. Transparency: It refers to hiding details from a user. They are four types of transparency.

1. Structure transparency: Multiple file servers are used to provide better performance, scalability, and reliability. The multiplicity of file servers should be transparent to the client of a distributed file system. Clients should not know the locations of file servers or storage devices instead it should look like a conventional file system offered by a centralized, time sharing operating system.

2. Access transparency: Local and remote files should be accessible in the same way. The file system should automatically locate an accessed file and transport it to the client's site.

3. Naming transparency: The name of the file should not reveal the location of the file. The name of the file must not be changed while moving from one node to another.

4. Replication transparency: The existence of multiple copies and their locations should be hidden from the clients where files are replicated on multiple nodes.

3. User mobility: The user should not be forced to work on a specific node but should have the flexibility to work on different nodes at different times. This can be done by automatically bringing the users to the node where the user logs in.

4. It is Simple and easy of use.

5. User interface to the file system is simple. A DFS should be able to support the whole range of applications.

6. Scalability: A good DFS should cope with an increase of nodes and not cause any disruption of service. Scalability also includes the system to withstand high service load, accommodate growth of users and integration of resources.

7. High availability: A distributed file system should continue to function even in minute failures such as a link failure, a node failure, or a storage device crash. Replicating files at multiple servers can help achieve availability.

8. High reliability: Probability of loss of stored data should be minimized. System should automatically generate backup copies of critical files in event of loss.

9. Data integrity: Concurrent access requests from multiple users who are competing to access the file must be properly synchronized by the use of some form of concurrency control mechanism. Atomic transactions can also be provided to users by a file system for data integrity.

10. Security: A distributed file system must secure data so that its users are confident of their privacy. File system should implement mechanisms to protect data that is stored within DFS.

Parallel File Systems (PVFS) should have the following features:

1. Object-based design: In object based design, all PVFS server requests involved objects called dataspaces. A dataspace can be used to hold file data, file metadata, directory metadata, and directory entries. Every dataspace in a file system has a unique handle. Any client or server can look up which server holds the dataspace based on the handle.

A dataspace has two components: a byte stream and a set of key/value pairs. The byte stream is an ordered sequence of bytes, it is used to hold file data, and the key/value pairs are typically used to hold metadata.

2. Separation of data and metadata: client can access a server for metadata once, and then can access the data servers without further interaction with the metadata servers. This removes a critical bottleneck from the system and allows much greater performance.

3. MPI-based requests: When a client program requests data from PVFS it can supply a description of the data that is based on MPI_Datatypes. This facility allows MPI file views to be directly

implemented by the file system. MPI_Datatypes can describe complex non-contiguous patterns of data. The PVFS server and data codes implement data flows that efficiently transfer data between multiple servers and clients.

4. Multiple network support: PVFS uses a networking layer named BMI which provides a non-blocking message interface designed specifically for file systems. BMI has multiple implementation modules for a number of different networks used in high performance computing including TCP/IP and Portals.

5. Stateless servers: PVFS servers are designed, they do not share any state with each other or with clients. If a server crashes another can easily be restarted in its place. Updates are performed without using locks.

6. User-level implementation: PVFS clients and servers run at user level. Kernel modifications are not needed. There is an optional kernel module that allows a PVFS file system to be mounted like any other file system, or programs can link directly to a user interface such as MPI-IO or a Posix-like interface. It makes PVFS easy to install and less prone to causing system crashes.

7. System-level interface: The PVFS interface is designed to integrate at the system level. It has similarities with the Linux VFS, this making it easy to implement as a mountable file system, but is equally adaptable to user level interfaces such as MPI-I/O .

They are few similarities between Distributed File Systems (HDFS) and Parallel File Systems (PVFS). Both the file systems gives better performance, both the file systems have key/value pairs, highly available, have system level interface, easy to use and scalable. And they are differences between HDFS and PVFS. In PVFS, client can access a server for metadata once but in HDFS, client doesn't access a server for metadata. HDFS provides a fault-tolerant mechanism, when one node fails, map-reduce restarts the same task on another node. Where as in PVFS, All MPI processes will exit if one of them fails. In HDFS, Probability of loss of stored data should be minimized whereas in PVFS,data can be retrieved. HDFS provides replication where as PVFS doesn't. PVFS with the shim to be comparable in performance to HDFS for Hadoop MapReduce-workloads, except for write-heavy workloads.

5. How does data distribution works on HDFS? Who is responsible for distributing data? What would you optimize in the distribution policies to make the system faster and more reliable?

Sol: HDFS follows master/slave architecture. In HDFS cluster is built with one master server which is a Name Node that manages filesystem namespace and regulates access to file by clients. In addition, there are lot of Data Nodes which are slave nodes under one Name Node in the cluster, which also manages the storage attached to nodes that they run on. HDFS represents a file system name space and also allows the user data to be stored in files. A file is split into number of blocks depends up on the size of file and these blocks are stored in the data nodes with some replication factor depends up on the HDFS. The address of the files stored in the data nodes are maintained by Name Node in the form of metadata. The Name Node executes file system namespace

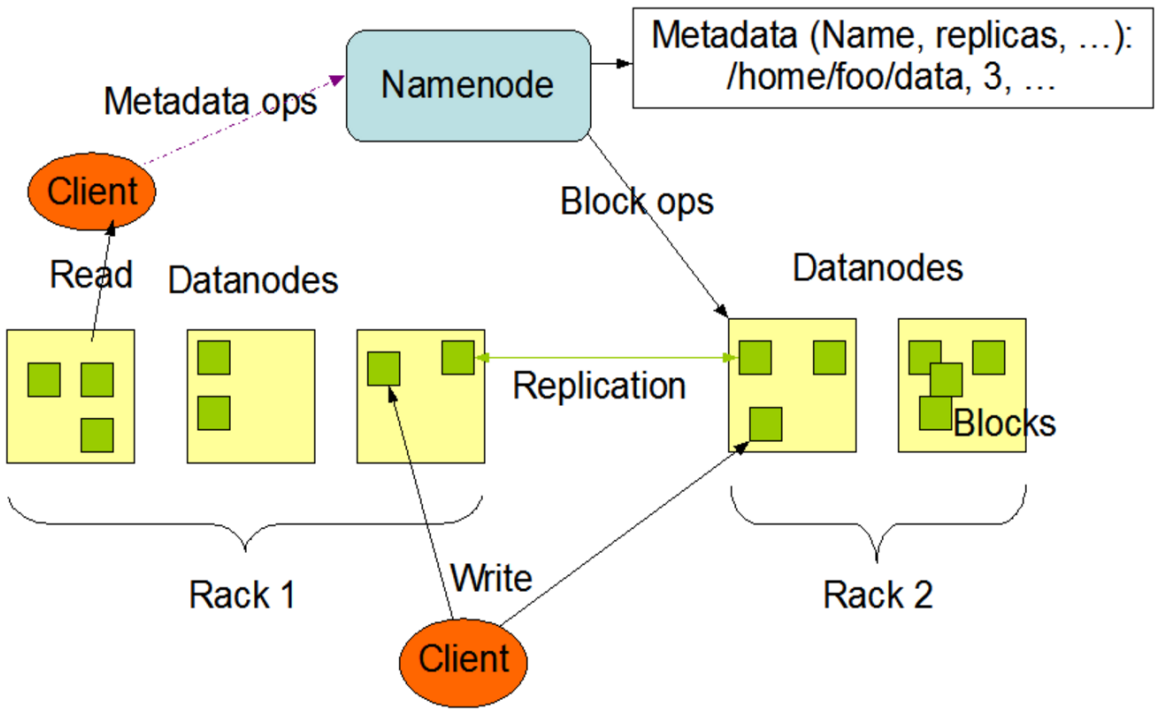
operations like opening, closing and renaming files and directories. It also determines the address of blocks to the Data Nodes. Data Nodes manages the serving read and write requests from the file system's clients. Name Node gives the instructions to perform block creation, deletion and replication for the Data Node.

Block Replication:

NameNode(Filename, numReplicas, block-ids,…)

/users/chiru/data/part-0, r:2, {1,3},…

/users/chiru/data/part-1, r:3, {2,4,5},…



In HDFS as we discussed above in the Hadoop distribution system we know that there will be a Master Node and Name Node. In here Master node nothing but the Name Node which distributes the data to the data nodes and stores the Meta data in it. Meta data is nothing but the address of the files distributed to the Name Nodes.

HDFS will not distribute the data files evenly across the data nodes properly. It depends up on the empty amount of space in the data node. Hence HDFS will rebalance the data blocks using various number of models. In particular model data node is moved from one data node to another data node depends up on the free space on the data falls to very low.

The placement of replicas is critical to HDFS data reliability and read/write performance. A good replica placement policy should improve data reliability, availability, and network bandwidth utilization. Currently HDFS provides a configurable block placement policy interface so that the

users and researchers can experiment and test alternate policies that are optimal for their applications.

The default HDFS block placement policy provides a tradeoff between minimizing the write cost, and maximizing data reliability, availability and aggregate read bandwidth. When a new block is created, HDFS places the first replica on the node where the writer is located. The second and the third replicas are placed on two different nodes in a different rack. The rest are placed on random nodes with restrictions that no more than one replica is placed at any one node and no more than two replicas are placed in the same rack, if possible. The choice to place the second and third replicas on a different rack better distributes the block replicas for a single file across the cluster. If the first two replicas were placed on the same rack, for any file, two-thirds of its block replicas would be on the same rack.

After all target nodes are selected, nodes are organized as a pipeline in the order of their proximity to the first replica. Data are pushed to nodes in this order. For reading, the Name Node first checks if the client's host is located in the cluster. If yes, block locations are returned to the client in the order of its closeness to the reader. The block is read from Data Nodes in this preference order.

This policy reduces the inter-rack and inter-node write traffic and generally improves write performance. Because the chance of a rack failure is far less than that of a node failure, this policy does not impact data reliability and availability guarantees. In the case of three replicas, it can reduce the aggregate network bandwidth used when reading data since a block is placed in only two unique racks rather than three.

6. **Discuss how the fault tolerance features in HDFS work? Compare it with hardware based (RAID) approaches. What other approaches for fault tolerance are out there (think erasure coding)**

Sol: One of the feature of HDFS is its high degree of fault tolerance. Individual nodes or network components may experience high rate of failure. Let us assume a file stored in a system, and due to some technical problem that file gets destroyed. Then there is no chance of getting the data back present in that file. To avoid these situations, Hadoop has introduced the feature of fault tolerance in HDFS. In Hadoop, when we store a file, it automatically gets replicated at two other locations also. Since HDFS replicates file blocks.

An application can specify the number of replicas of a file at the time it is created, and this number can be changed any time after that. The name node makes all decisions concerning block replication.

HDFS uses an intelligent replica placement model for reliability and performance. Optimizing replica placement makes HDFS unique from most other distributed file systems, and is facilitated by a rack-aware replica placement policy that uses network bandwidth efficiently. Large HDFS environments typically operate across multiple installations of computers. Communication between two data nodes in different installations is typically slower than data nodes within the

same installation. Therefore, the name node attempts to optimize communications between data nodes. The name node identifies the location of data nodes by their rack IDs.

When comparing different storage schemes, there are two important considerations: data durability and storage efficiency.

Fault tolerance of each of the possible Hardware based RAID as follows:

- All RAID levels except RAID-0 provide some degree of fault tolerance, but the effect of a failure and the ability to recover from subsequent failures can be different.

-If a drive in a RAID-1 mirror or duplex array fails, a full, complete, exact copy of the data remains. Access to your data or application is unimpeded, and performance degradation is minimal, although we will lose the benefit gained on read operations of being able to read from either disk. Until the failed disk is replaced, however, you will have no fault tolerance on the remaining disk.

In a RAID-3 or RAID-4 array, if one of the data disks fails, a significant performance degradation will occur since the missing data needs to be reconstructed from the parity information. Also, we'll have no fault tolerance until the failed disk is replaced. If it is the parity disk that fails, you'll have no fault tolerance until it is replaced, but also no performance degradation.

In a RAID-5 array, the loss of any disk will result in a significant performance degradation, and your fault tolerance will be gone until you replace the failed disk. Once you replace the disk, you won't return to fault tolerance until the entire array has a chance to rebuild itself, and performance will be seriously degraded during the rebuild process.

RAID systems that are arrays of arrays can provide for multiple failure tolerance. These arrays provide for multiple levels of redundancy and are appropriate for mission-critical applications that must be able to withstand the failure of more than one drive in an array, where as in Hadoop, file stored in a system, and due to some technical problem that file gets destroyed. Then there is no chance of getting the data back present in that file. To avoid these situations, Hadoop has introduced the feature of fault tolerance in HDFS by replicates file blocks. Hadoop have better performance than Hardware Raid approaches when disk fails, it takes more time to replace the disk, and performance will be degraded during the rebuild process. Replication is a simple and effective way of tolerating disk failures, at the cost of storage overhead. N-way replication can tolerate up to n-1 simultaneous failures with a storage efficiency of 1/n. For HDFS, the three-way replication scheme tolerates up to two failures with a storage efficiency of one-third (200% overhead).

Erasure coding is a branch of information theory which extends a message with redundant data for fault tolerance. An EC codec operates on units of uniformly-sized data termed cells. A codec can take as input a number of data cells and outputs a number of parity cells. This process is called encoding. Together, the data cells and parity cells are termed an erasure coding group. A lost cell can be reconstructed by computing over the remaining cells in the group; this process is called decoding. The simplest form of erasure coding is based on XOR operations.

While XOR can take any number of data cells as input, it is very limited since it can only produce at most one parity cell. So, XOR encoding with group size n can tolerate up to 1 failure with an efficiency of n-1/n , but is insufficient for systems like HDFS which need to tolerate multiple failures. With Reed-Solomon, the user can flexibly adjust data durability and storage cost by choosing different values for k and m. The number of parity cells (m) determines the number of simultaneous storage failures that can be tolerated. The ratio of data cells to parity cells determines the storage efficiency K/(K+m)

In Distributed Storage Systems, we have To manage potentially very large files, distributed storage systems usually divide files into fixed-size logical byte ranges called logical blocks. These logical blocks are then mapped to storage blocks on the cluster, which reflect the physical layout of data on the cluster.

The simplest mapping between logical and storage blocks is a contiguous block layout, which maps each logical block one-to-one to a storage block. Reading a file with a contiguous block layout is as easy as reading each storage block linearly in sequence. In contrast, a striped block layout breaks a logical block into much smaller storage units, typically called cells, and writes repeated stripes of cells round robin across a set of storage blocks. Reading a file with a striped layout requires querying the set of storage blocks of a logical block, then reading stripes of cells from the set of storage blocks. Erasure coding is advantageous over replication in terms of storage efficiency. However, this comes at the cost of additional complexity and more expensive failure recovery.

7. Provide a description for the ideal workload for HDFS. What about the worst-case scenario?

Sol: There are many causes of stragglers but one of the causes is data skew: some tasks take longer because they have been allocated more data. Ideally, if we assign the same amount of input data to tasks, then all tasks should take the same time to execute. We evaluate how close the workloads are to this ideal case by analyzing the relationship between input data size and task runtime.

For each phase of each job, we compute the ratio of the maximum task runtime in the phase to the average task runtime in that phase. We classify phases where this ratio is greater than 0.5 (meaning that at least one straggler took twice as long to process its data as the average) as Unbalanced in Time (UT). Otherwise, the phase is said to be balanced in Time (BT). We compute the same ratio for the input data and classify phases as either balanced or unbalanced in their data

Map Input: In the map phase, most jobs are balanced with respect to data allocated to tasks. However, a significant fraction of jobs remain unbalanced in runtime. Overall, allocating data to compute nodes simply based on data size alone is insufficient to eliminate skew. In Hadoop, the Input Format mechanism is responsible for generating Input Splits which describe the input data processed by a map task. And less number of jobs attempt to optimize the partitioning of the map input by using a custom input format. In addition, the user can manually specify the number of

bytes per split if the input is stored in HDFS. It is clear that users only rarely exploit these opportunities for optimizing the data allocation.

Reduce Key: Now, perform the reduce phase using the number of reduce keys as the input Measure. We group the jobs by the partition function, the partition function is responsible for redistributing the reduce keys among the reduce tasks. And users rely primarily on the default hash-partition function rather than manually trying to optimize the data allocation to reducers. Overall, Hash partitioning effectively redistributes reduce keys among reduce tasks For the jobs with a balanced data distribution, the runtime was still unbalanced, custom data partitioning is more effective than the default scheme in terms of balancing both the keys and the computation. Other partitioning schemes come with Hadoop distribution do not outperform hash partitioning in terms of balancing data and runtime. User's reluctance to use the manual tuning features provided by Hadoop, we recommend pursuing techniques that automatically reduce skew to achieve better overall performance.

In the worst case scenario, the execution time inequity for map task is much worse than that of reduce tasks. Data skew on the map phase is much worse due to some skewed map operations, such as with the table JOIN operations.

8. What storage needs led to the birth of NoSQL schema? Why did object storage become the de-facto solution in web services? Discuss the weaknesses and the strengths of a key-value store.

Sol: The world is undergoing massive change as industry after industry shifts to the Digital Economy. It's an economy powered by the Internet and other technologies such as the cloud, mobile, social media, and big data.

Digital Economy business are its web, mobile, and Internet of Things (IoT) applications: They are the primary way companies interact with customers today and how companies run more and more of their business. The experiences that companies deliver via those applications largely determine how satisfied. How are these applications different from legacy enterprise applications like ERP, HR and financial accounting?

Today's web, mobile, and IoT applications share one or more of the following characteristics:

1. They Support large numbers of concurrent users (millions)

2. They have to deliver highly responsive experiences to a globally distributed base of users.

3. They should be available all time without downtime.

4. They should handle semi- and unstructured data

5. Rapidly adapt to changing requirements with frequent updates and new features

Building and running these web, mobile, and IoT applications has created a new set of technology requirements. The new enterprise technology architecture needs to be far more agile than ever before, and requires an approach to real time data management that can accommodate unprecedented levels of scale, speed, and data variability. Relational databases are unable to meet these new requirements, and enterprises are therefore turning to NoSQL database technology.

We know that Relational databases were born in the era of mainframes and business applications, long before the Internet, the cloud, big data, mobile and the Digital Economy. These databases were engineered to run on a single server, the bigger and better. The only way to increase the capacity of these databases was to upgrade the servers, processors, memory, and storage to scale up.

NoSQL databases emerged as a result of the exponential growth of the Internet and the rise of web applications. Google released the Big Table research in 2006, and Amazon released the Dynamo research paper in 2007. These databases were engineered to meet a new generation requirements.

A NoSQL database provides a mechanism for storage and retrieval of data other than tabular relations used in relational databases. NoSQL databases are increasingly used in big data and real-time web applications. The data structures used by NoSQL databases (e.g. key-value, wide column, graph, or document) are different from those used by default in relational databases, making some operations faster in NoSQL. The particular suitability of a given NoSQL database depends on the problem it must solve. Sometimes the data structures used by NoSQL databases are also viewed as more flexible than relational database tables.

There are a number of factors that have come together to make object storage an attractive proposition:

1. Data retention demands: Organizations and individuals are storing more data than ever. At the consumer level, we create new data every day in the form of images, videos, text and log data, everything from where our iPhone is located to paying tolls on the roads. There are also large active archives generated by media organizations, energy and mining companies, plus an enormous amount of medical imaging information. Data is collected from sensors and other Internet of Things devices.

2. Software-defined storage: There has been an explosion in companies developing storage solutions based on commodity hardware running standard operating systems like Linux. The barriers to entry for developing new storage products is lower than ever before and there are open-source solutions looking to compete with commercial offerings. Pretty much all of today's object storage products are software-based, except for bespoke high performance computing (HPC) solutions.

3. The Amazon effect: Amazon Web Services introduced Simple Storage Service (S3) almost 10 years ago.S3 API has become a de-facto standard for object storage access and is supported by all of the major object storage vendors. The S3 API introduced simplicity, it can be accessed over HTTP using simple REST-based commands and interoperability. Code written to access S3 can easily be adapted to work with other solutions, making data much more portable than older days.

Widespread protocol support: Object storage vendors have accepted that REST-based APIs aren't for everyone. As a result, most support block- and file-based protocols, including data-specific access methods like Hadoop File System (HDFS). The use of traditional protocols, plus the ability to access the same data with each protocol type, means that object stores can offer more flexibility than ever before. This makes them more practical as active data stores rather than simply as archives.

Strengths of Key-value stores: Key value store is the simplest data model, it is just a distributed persistent associative array. The key is a unique identifier for a value, which can be any data application needs stored. This model is also the fastest way to get data by known key, but without the flexibility of more advanced querying. It may be used for data sharing between application instances like distributed cache or to store user session data. And Supports simple queries very efficiently.

Weaknesses of Key-value stores: There are various different weaknesses to each of them depending on lots of different factors, types of data model, level of persistence optimal workloads etc. and it is inefficient when you are only interested in querying or updating part of a value, they lack the ability to perform ACID transactions that span multiple pieces of data within the database. ACID Transactions, not SQL, is the huge advantage relational databases have over NoSQL in terms of ease of development.