PAGE NO. :
DATE :

1 Explain the role of distributed system in Big Data. Who are the data scientists, list out their roles and skills. [5+5]

⇒ A distributed system is a collection of independent computers that appears to its users as a single coherent system.

A distributed system is one in which components located at networked computers communicate and coordinate their actions only by passing messages. It play an important role in managing the big data problems that prevails in todays world. In the distributed approach, data are placed in multiple machines and are made available to the user as if they are in a single system. It makes the proper use of hardware and resources in multiple location and multiple machines.

& Roles and skills of data scientists are as follow:-

i Recognize and reflect the two-phased nature of analytic process.

ii Provide guidance for companies about how to establish that their use of data for knowledge discovery is a legitimate business purpose.

iii Take into account that analytics may be an iterative process using data from a variety of sources.

8 Explain the term "Big Data". How could you say that your organization suffers from Big Data problem?

⇒ Big data are those data sets with sizes beyond the ability of commonly used software tools to capture, curate, manage, and process the data within a tolerable elapsed time Big data is the term for a collection of data sets so large and complex that it becomes difficult to process using on-hand database management tools or ~~trada~~ traditional data processing applications.

Big data is often defined along three dimensions - volume, velocity and variety.

↳ Big data is data that can be manipulated (slices and diced) with massive speed.

PAGE NO.:
DATE:

↳ Big data is the not the standard fare that we use, but the more complex and intricate data sets.
↳ Big data is the unification and integration of diverse data sets (kill the data ghettos).
↳ Big data is based on much larger amount of data sets than what we're used to and how they can be resolved with both speed and variety.
↳ Big data extrapolates the information in a different (three dimensional) way.

Data sets grow in size in part because they are increasingly being gathered by ubiquitous information-sensing mobile devices, aerial sensory technologies (remote sensing), software logs, cameras, microphones, radio-frequency identification readers, and wireless sensor networks. The world's technological per-capita capacity to store information has roughly doubled every 40 months since the 1980s; as of 2012, every day 2.5 quintillion (2.5×10^{18}) bytes ~~using most~~ of data were created. As the data collection is increasing day by day, is difficult to work with using most relational database management systems and desktop statistics and ~~visualizing~~ visualization

PAGE NO. :
DATE :

packages, requiring instead "massively parallel software running on tens, hundreds, or even thousands of servers. The challenges include capture, duration, storage, search, sharing, transfer, analysis, and visualization. So such large gathering of data suffers the organization forces the need to big data management with distributed approach.

3. What are the technical challenges and characteristics of big data? Explain the data analytics process in terms of big data. [3+7]

⇒ The technical challenges and characteristics of big data are:

Challenges:

1. Lack of proper understanding of Big data.
2. Data growth issues
3. Confusion while Big data tool selection
4. Lack of data professionals.
5. Securing data.
6. Integrating data from a variety of sources.

Characteristics:

1. Variety
2. Velocity

PAGE NO.:
DATE:

3 Volume.

Data analysis is a process of collecting, transforming, cleaning and modeling data with the goal of discove-ring the required information. The result so obtaine are communicated, suggesting conclusions, and supporting decision-making. Data visualization is of times used to portray the data for the ease of discovering the useful patterns in the data. The terms Data modeling and data analysis mean the same.

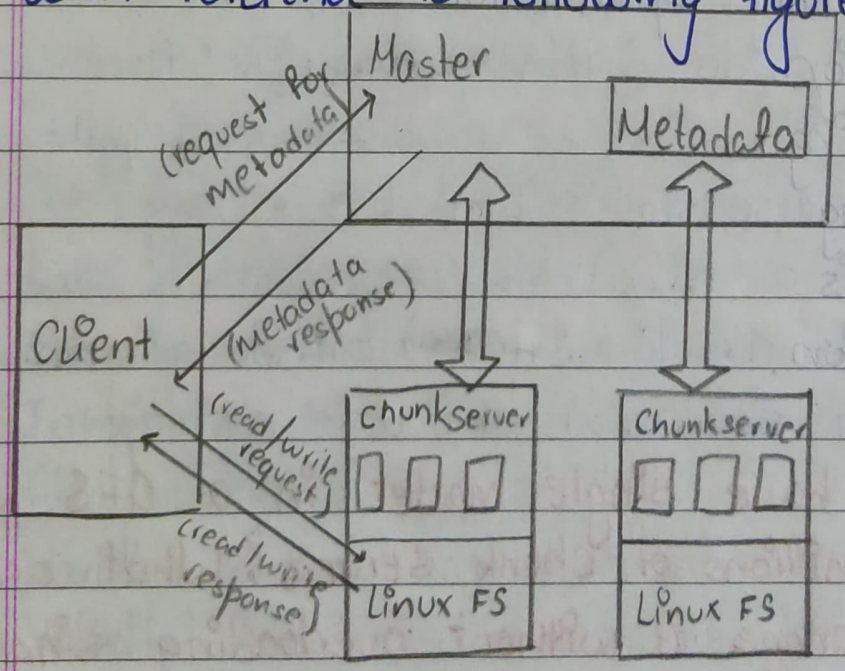
Data analysis process consists of the following phases that are iterative in nature:

- Data Requirements Specification
- Data Collection
- Data Processing
- Data cleaning
- Data Analysis
- Communication.

4. Why do we have single master in a GFS many managing millions of chunk servers? What are done to manage it without overloading single

master? [10]

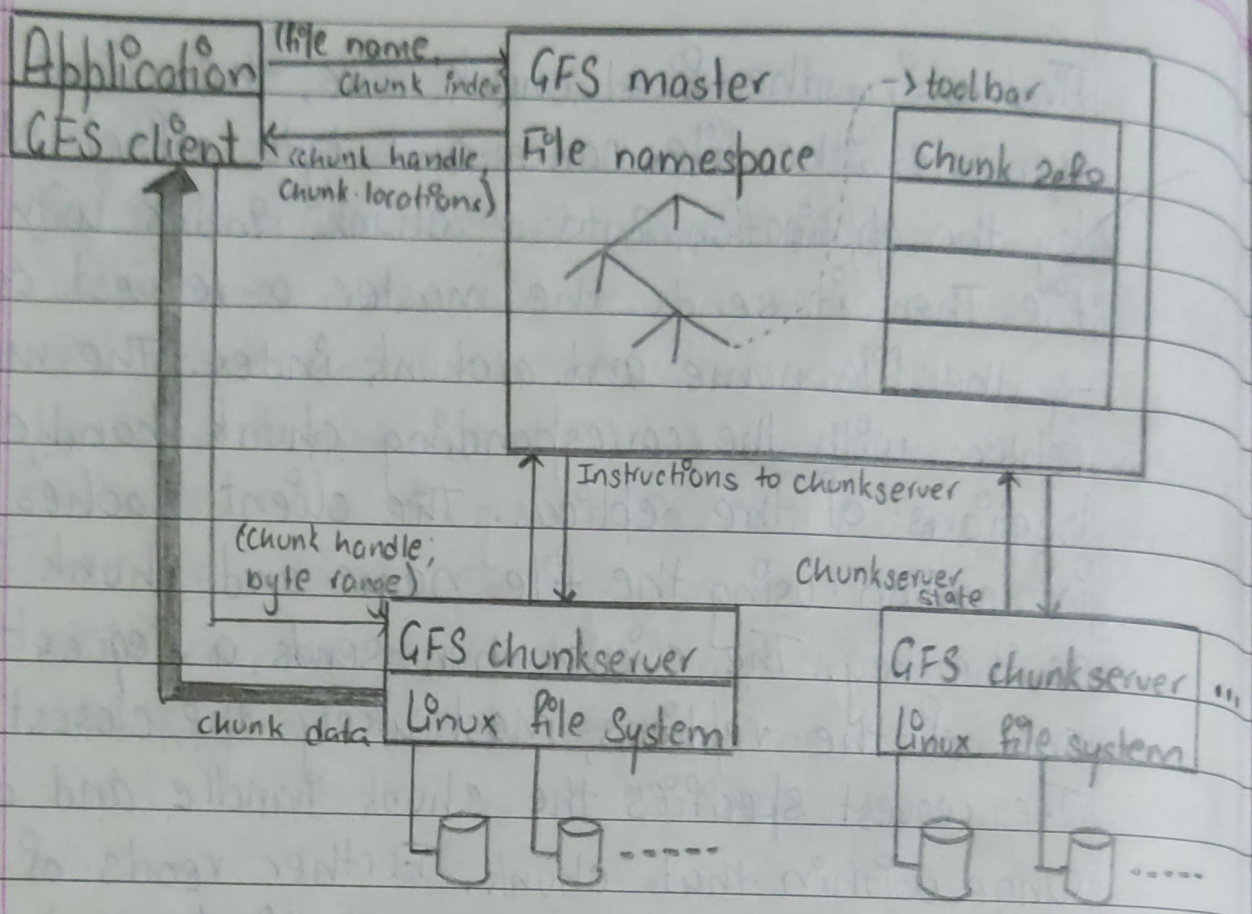
⇒ Having a single master vastly simplifies the design of GFS and enables the master to make sophisticated chunk placement and replication decisions using global knowledge. However, the involvement of master in reads and write ~~file data th~~ must be minimize so that it does not become a bottleneck. Clients never read and write file data through the master. Instead, a client asks the master which chunk servers it should contact. It caches this information for a limited time and interacts with the chunk servers directly for many subsequent operations. Let's explain the interactions for a simple read with reference to following figure:



PAGE NO.:
DATE:

First, using the fixed chunk size, the client translates the file name and byte offset specified by the application into a chunk index within the file. Then, it sends the master a request containing the file name and a chunk index. The master replies with the corresponding chunk handle and locations of the replicas. The client caches this information using the file name and chunk index as the key. The client then sends a request to one of the replicas, most likely the closest one. The request specifies the chunk handle and a byte range within that chunk. Further reads of the same chunk require no more client master interaction until the cached information expires or the file is reopened. In fact, the client typically asks for multiple chunks in the same request and the master can also include the information for chunks immediately following those requested. This extra information sidesteps several future client-master interactions at ~~part~~ practically no extra cost.

5. Explain the architecture of Google File System (GFS) with neat and clean diagram.



Legend

→ : Control messages

➡ : Data messages

fig: GFS Architecture.

Google organized the GFS into clusters of computers. Each cluster might contain hundreds or even thousands of machines. Within GFS clusters there are three kinds of entities: clients, master servers and chunkservers. In the world of GFS, the term "client" refers to any

entity that makes a file request. Requests can range from retrieving and manipulating existing files to creating new files on the system. You can think of client as the customers of the GFS. The master server acts as the coordinator for the cluster. The master's ~~client~~ duties include maintaining an operation log, which keeps track of the activities of the master's cluster. The operation log helps keep service interruption to a minimum. If the master server crashes, a replacement server that has monitored the operation log can take its place. The metadata tells the master server to which files the chunks belong and where they fit within the overall file. Upon startup, the master polls all the chunkservers in its cluster. The chunkserve respond by telling the master server the contents of their inventories. From that moment on, the master server keeps track of the location of chunks within the cluster. There's only one active master server per cluster at any one time. The GFS gets around this sticky situation by keeping the message the master server sends and receives very small. The master server doesn't actually handle file data at all. It leaves that up to the chunkservers. Chunkservers are the workhorses of the GFS. The chunkservers

don't send chunks to the master server. Instead, they send requested chunks directly to the client. The GFS copies every chunk multiple times and stores it on different chunkservers. Each copy is called a replica. By default, the GFS makes three replicas per chunk, but users can change the setting and make more or fewer replicas if desired.

6 What is availability and fault tolerance in Google File system? Explain garbage ~~centre~~ collection implemented by GFS. [5+5]

⇒ Availability and fault tolerance in Google file system are:

- Fast Recovery
- Chunk Replication
- Master Replication.

- It uses checksumming to detect corruption of stored data.
- Chunk is broken up into 64kB blocks with corresponding 32 bit checksum
- Chunkserver verifies the checksum before returning data.

Garbage Collection

Whenever the client deletes the file, the file is not deleted ~~by~~ but renamed to hidden file with delete ~~timestat~~ 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 chunkserver 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 can not be used immediately.

7. How map-reduce works in distributed fashion? Describe the parallel efficiency of map-reduce with suitable block diagram.

⇒ At the crux of MapReduce are two functions: Map and Reduce. They are sequenced one after the other.

PAGE NO. :
DATE :

- The Map function takes input from the disk as <key, value> pairs, processes them, and produces another set of intermediate <key, value> pairs as output.

- The Reduce function also takes inputs as <key, value> pairs, and produces <key, value> pairs as output.

PAGE NO. :
DATE :

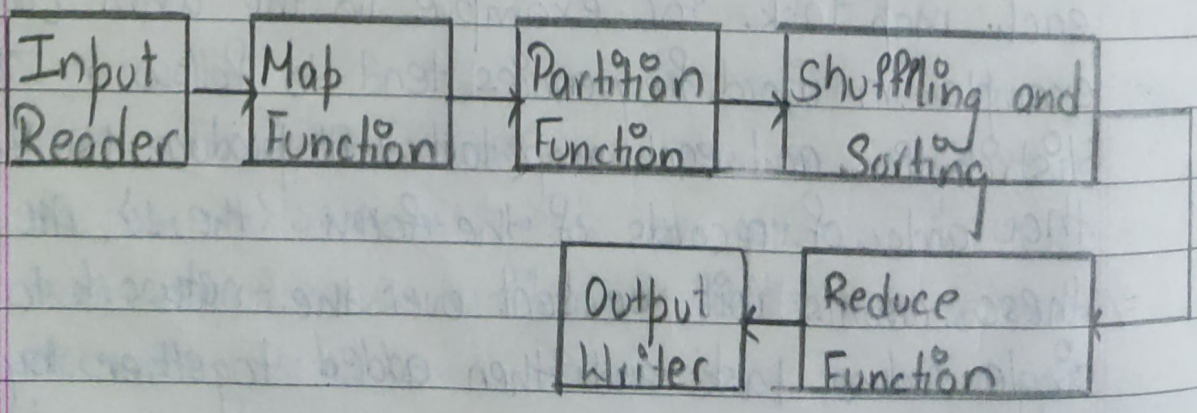
8. What is the combiner function? Explain its purpose with suitable diagram. [10]

⇒ Combiner is a user specified function that does partial merging of the data before it is sent over the network. In some cases, there is significant repetition in the inter-mediate keys produced by each map task. for example in the word counting example in word frequencies tend to follow a Zipf distribution and each map task will produce hundreds or thousands of records of the form <the, 1>. All of these counts will be sent over the network to a single reduce task and then added together by the

Reduce function to produce one number. So to decentralize the count of reduce, user are allowed to specify an optional Combiner function that does partial merging of this data before it is sent over the network. The combiner function is executed on each machine that perform a map task. No extra effort is necessary to implement the combiner function since the same code is used to implement both the combiner and the reduce functions.

9. Explain in brief Data Flow technique of Map-Reduce Framework. [10]

⇒ MapReduce is used to compute the huge amount of data. To handle the upcoming data in a parallel and distributed form, the data has to flow from various phases.



Phases of MapReduce data flow

i Input Reader:

The input reader reads the upcoming data and splits it into the data blocks of the appropriate size (64 MB to 128 MB). Once input reads the data, it generates the corresponding key-value pairs.

ii Map function:

The map function process the upcoming key-value pairs and generated the corresponding output key-value pairs. The map input and output type may be different from each other.

iii Partition function:

The partition function assigns the output of each Map function to the appropriate reducer. The available key and value provide the function. It returns the index of reducers.

iv Shuffling and Sorting:

The data are shuffled between/within nodes so that it moves out from the map and get ready to process for reduce function. Sometimes, the shuffling of data can take much computation

time.
The sorting operation is performed on input data & Reduce function. Here, the data is compared using comparison function and arranged in a sorted form.

v. Reduce function
The reduce function is assigned to each unique key. These keys are already arranged in sorted order. The values associated with the keys can iterate the Reduce and generates the corresponding output.

vi. Output Writer:
Once the data flow from all the above phases, Output writer executes. The role of output writer is to write the Reduce output to the stable storage.

10. What is map reduce? Explain the execution overview of the map reduce. What is Optimization and Data Locality in Map Reduce? [6+4]

⇒ MapReduce is a programming model designed for processing large volumes of data in parallel by dividing the work into a set of independent tasks.

PAGE NO. :
DATE :

The Map invocations are distributed across multiple machines by automatically partitioning the input data into a set of M splits or shards, which are what will be processed across the machines. Reduce invocations are distributed by partitioning the intermediate key space into R pieces using a partitioning function specified by the user.

Execution Notes :

- After successful completion, the output of the MapReduce execution is available in the R output files.
- To detect failure, the master pings every worker periodically. If no worker response after a certain point, the worker is marked a "failed" and all previous task work by that worker is reset, to become eligible for rescheduling on other workers.
- Completed map tasks are re-executed when failure occurs because their output is stored on the local disk(s) of the failed machine and therefore inaccessi-ble. Completed reduce tasks do not need to be re-executed since their output is stored in a global file system.

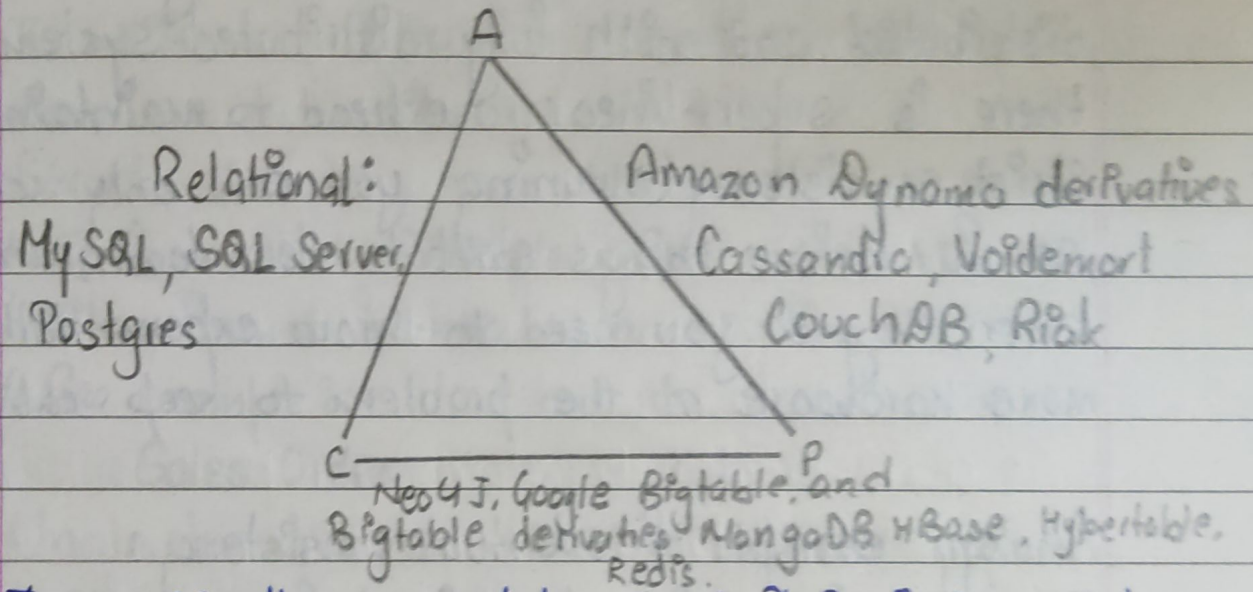
Optimization in MapReduce

Data Locality in MapReduce refers to the ability to move the computation close to where the actual data resides on the nodes, instead of moving large data to computation. This minimizes network congestion and increases the overall throughput of the system.

11. Explain CAP theorem with suitable block diagram and Eventual consistency. Also, explain the reason why some NoSQL databases like Cassandra sacrifice absolute consistency for absolute availability.

⇒ The CAP theorem was formally proved to be true by Seth Gilbert and Nancy Lynch of MIT in 2002. In distributed databases like NoSQL, however, it is very likely that we will have network partitioning, and that at some point,

machines will fail and cause others to become unreachable. Figure below illustrates visually that there is no overlapping segment where all three are obtainable, thus explaining the concept of CAP theorem:



* The CAP theorem states that it is impossible for any shared-data system to guarantee simultaneously all of the following three properties: consistency, availability and partition tolerance.

Eventual Consistency is a consistency model used in distributed computing that informally guarantees that, if no new updates are made to given data then, eventually all accesses to that item will return the last updated value. It is widely deployed in distributed systems and has origins in early mobile computing

projects. A system that has achieved eventual consistency is often said to have converged, or achieved replica convergence. The reason why so many NoSQL systems have eventual consistency is that virtually all of them are designed to be distributed, and with fully distributed systems there is super-linear overhead to maintaining strict consistency (meaning you can only scale so far before things start to slow down, and when they do you need to throw exponentially more hardware at the problem to keep scaling).

Analogy example of eventual consistency:

1. I tell you it's going to rain tomorrow.
2. Your neighbor tells his wife that it's going to be sunny tomorrow.
3. You tell your neighbor that it is going to rain tomorrow.

Eventually, all of the servers (you, me, your neighbor) know the truth (that it's going to rain tomorrow) but in the meantime the client (his wife) came away thinking it is going to be sonny, even though she asked after one of the servers

knew the truth.

12. Differentiate between structured and unstructured data and discuss the Taxonomy of NoSQL.

⇒ The difference between structured and unstructured data are as follow:-

Structured data:

↳ It is normal RDBMS data.

↳ Its format is known and defined.

↳ for example:-

Sales Order

Unstructured data:

↳ Structure is merely encoding; metadata may be in the structure.

↳ for example:-

Audio files, Word documents, PDF, Movies, etc.

Taxonomy of NoSQL implementation

The current NoSQL world fits into 4 basic categories.

- Key-values Stores

The key-value database is a very simple structure

based on Amazon's Dynamo DB. Data is indexed and queried based on its key. It provide consistent hashing so they can scale incrementally as your data scales. They communicate node structure through a gossip-based membership protocol to keep all the nodes synchronized. If you are looking to scale very large sets of low complexity data, key-value stores are the best option.
Examples: Riak, Voldemort, etc.

- **Column Family Stores**
  It were created to store and process very large amounts of data distributed over many machines. There are still keys but they point to multiple columns. The columns are arranged by column family. Example: Cassandra, HBase, etc. Column family database are still extremely scalable but less-so than keyvalue stores. However, they work better with more complex data sets.

- **Document Database**
  It were inspired by Lotus Notes and are similar to keyvalue stores. The model is basically versioned documents that are collections of other key-value

collections. The semi-structured documents are stored in formats like JSON e.g: MongoDB, Couch DB. It is not a new idea. It was used to power one of the more prominent communication platforms of the 90's and still in service today. It has a fairly simple data model based on collections of key-value pairs. It improve on handling more complex structures but are slightly less scalable than column family databases.

- **Graph Databases**

They are built with nodes, relationships between notes and the properties of nodes. Instead of tables of rows and columns. and the rigid structure of SQL, a flexible graph model is used which can scale across many machines. It takes document databases to the extreme by introducing the concept of type relationships between documents or nodes. The most commo example is the relationship between people on a social network such as fact Facebook. It is a big dense network structure. It uses sophisticated shortest path algorithms to make

data queries more efficient.

13 Explain the term NO-SQL. Differentiate between a RDBMS and a NoSQL Database.

⇒ A NoSQL (originally referring to "non-SQL" or "non-relational") database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases. NoSQL database are increasingly used in big data and real-time web applications.

The difference between a RDBMS and a NoSQL database are as follow:-

RDBMS	NoSQL
1. RDBMS applications store data in the form of toble structured manner.	NoSQL is a non-relational database system. It stores data in the form of unstructured manner.
2. It uses tabular structures to store data. In table headers	It uses to store data in structured, semi-structured

	are the column names and the rows contains corresponding values.	and unstructured forms.
3.	It supports the normali-zation and joining of tables.	It does have table form, so it does not support normalization.
4.	Open-source application.	Open-source program.
5.	It was developed in the 1970s to deal with the issues of flat file storage.	It developed in the late 2000s to overcome the issues and limitations of the SQL database.
6.	This database system deals with a large quantity of data.	NoSQL database mainly designed for Big data and real-time web data.
7.	RDBMS program support client-server architecture.	NoSQL storage system supports multi-server. It also supports client-server architecture.

16. What are the data indexing steps? ~~Describe the components~~

⇒ The data indexing steps are as follows:-

1. Index by workload, not by table.
2. Build indexes based on predicates
3. Index most-heavily used queries
4. Index important queries.
5. Index to avoid sorting (GROUP BY, ORDER BY)
6. Create indexes for uniqueness (PK, U)
7. Create indexes for foreign keys
8. Consider adding columns for index only access.
9. Don't arbitrarily limit number of indexes.
10. Be aware of data modification implications.

OR

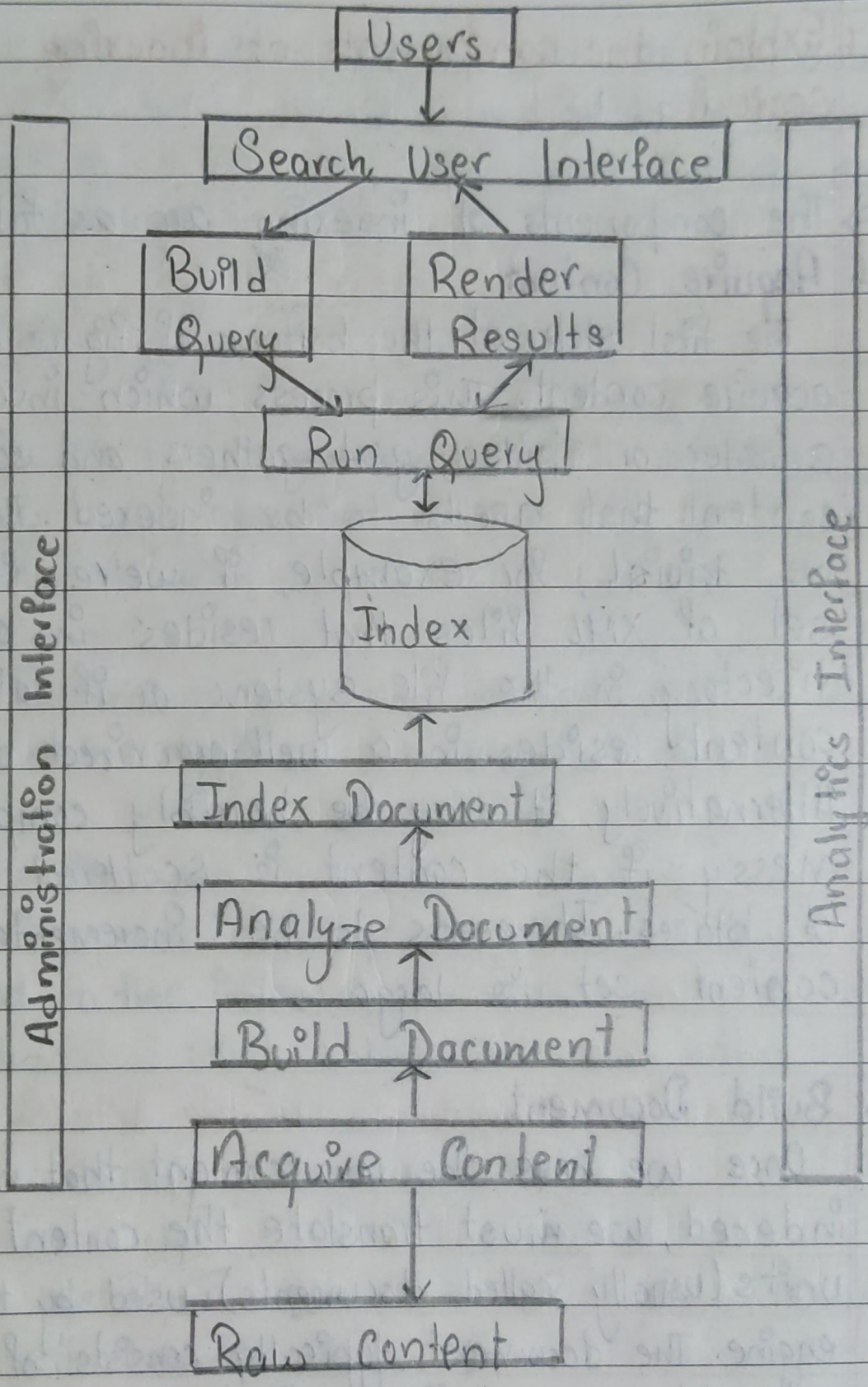
1. The first step in implementing full-text searching with Lucene is to build an index.
2. To create an index, the first thing that need to do is to create an IndexWriter Object.
3. The IndexWriter object is used to create the

index and to add new index entries (i.e. Documents) to this index.
We can create an IndexWriter as follows:-
IndexWriter indexWriter = new IndexWriter ("Index-directory", new StandardAnalyzer (), true);

15. Describe the typical components involved in search application.

⇒ The following figure illustrates the typical components of a search application:

An enterprise search application starts with an indexing chain, which in turn requires separate steps to retrieve the raw content; create documents from the content, possibly extracting text from binary documents; and index the documents. Once the index is built, the components required for searching are equally diverse, including a user interface, a means for building up a programmatic query, query execution (to retrieve matching documents), and result rendering.
The following figure illustrates the typical components of a search application:


Users
Search User Interface
Build Query
Render Results
Run Query
Index
Index Document
Analyze Document
Build Document
Acquire Content
Raw Content
Administration Interface
Analytics Interface

17. Explain the components of indexing and searching. [8]

=> The components of indexing are as follows:

1. Acquire Content

The first step, ~~at the bottom of figure~~ is to acquire content. This process, which involves using crawler or spider, ~~grat~~ gathers and scope the content that needs to be indexed. That may be trivial, for example, if we're indexing a set of XML files that resides in a specific directory in the file system or if all our content resides in a well-organized database. Alternatively, it may be horribly complex and messy if the content is scattered in all sorts of places. It needs to be incremental if the content set is large.

2. Build Document

Once we have the raw content that need to be indexed, we must translate the content into the units (usually called documents) used by the search engine. The document typically consists of several separately named fields with values, such as

title, body, abstract, author, and URL. A common part of building the document is to inject boosts to individual documents and fields that are deemed more or less important.

3. Analyze Document

No search engine indexes text directly: rather, the text must be broken into a series of individual atomic elements called tokens. This is what happens during this step. Each token corresponds roughly to a "word" in the language, and this step determines how the textual fields in the document are divided into a series of tokens.

4. Index Document

During the indexing step, the document is added to the index.

18. Explain the various components of Hadoop in brief.

⇒ There are three components of Hadoop. They are explained as follows :-

PAGE NO.:
DATE:

1. Hadoop HDFS (Hadoop distributed File System)

It is the storage unit of Hadoop. Data is stored in a distributed manner in HDFS. There are two components of HDFS - name node and data node. It is specially designed for storing huge datasets in commodity hardware.

Features of HDFS

- Provides distributed storage
- Can be implemented on commodity storage.
- Provides data security

2. Hadoop MapReduce

It is the processing unit of Hadoop. In the MapReduce approach, the processing is done at the slave nodes, and the final result is sent to the master node.

A data containing code is used to process the entire data. This coded data is usually very small in comparison to the data itself. We only need to send a few kilobytes worth of code to perform a heavy duty process on computers. The input data is first split into chunks of data.

PAGE NO. :
DATE :

3. Hadoop YARN

It is the resource management unit of Hadoop. and is available as a component of Hadoop version 2. Hadoop YARN stands for Yet Another Resource Negotiator. It acts like an OS to Hadoop. It is a file system that is built on top of HDFS. It is responsible for managing cluster resources to make sure we don't overload one machine. It also performs job scheduling to make sure that the jobs are scheduled in the right place.

19. List out the Hadoop daemons. How Hadoop and GFS are similar interms of design architecture. [2+8]

=> Hadoop consist of five daemons. They are:-

1. NameNode
2. Data Node
3. Secondary nameNode
4. Job tracker
5. Task tracker.

PAGE NO. :
DATE :

20. What are the different daemons in Hadoop cluster? Explain each in details.

=> The different daemons in Hadoop cluster are explained as follow:-

1. NameNode

The NameNode is the master of HDFS that directs the slave DataNode ~~dam~~ daemons to perform the low-level I/O tasks. It is the bookkeeper of HDFS; it keeps track of how our files are broken down into file blocks, which nodes store those blocks, and the overall health of the distributed file system. The function of the NameNode is memory and I/O intensive. It's a single point of failure of our Hadoop cluster. For any of the other daemons, if their host node fail for software or hardware reasons, the Hadoop cluster will likely continue to function smoothly or we can quickly restart it but not so for the NameNode.

2. DataNode

Each slave machine in cluster host a DataNode daemon to perform work of the distributed file system, reading and writing HDFS blocks to actual

files on the local file system. Read or write a HDFS file, the file is broken into blocks and the NameNode will tell your client which DataNode each block resides in. A DataNode may communicate with other DataNode to replicate its data blocks for redundancy.

3. Secondary NameNode

It is a specially dedicated node in HDFS cluster whose main function is to take checkpoints of the file system metadata present on NameNode. It just checkpoints namenode's file system namespace. The Secondary NameNode is a helper to the primary NameNode but not replace for primary namenode. It is not a true backup NameNode and cann't serve primary NameNode's operations.

4. Job Tracker

It is the liaison (mediator) between your application and Hadoop. Sould a task fail, the Job Tracker will automatically re launch the task, possibly on a different node, up to a predefined limit of retries. There is only one Job tracker daemon per Hadoop cluster. It's typically run on a

Server as a master node of the cluster.

5. Task tracker

It is a node in the cluster that accepts tasks. Every TaskTracker is configured with a set of slots, these indicate the number of tasks that it can accept. ~~When the~~ The TaskTracker spawns a separate JVM processes to do the actual work; this is to ensure that process failure does not take down the task tracker. It monitors these spawned processes, capturing the output and exit codes.

21. Explain in brief five daemons of Hadoop. [5]

⇒ Done in Question Number 20

22. What is the ~~role~~ goal of Hadoop Distributed File System in Hadoop? [4]

⇒ The goal of Hadoop Distributed File System in Hadoop are as follows:-

1. Fault detection and recovery
2. Huge datasets
3. Hardware at data

4. Fast recovery from hardware failures.
5. Access of streaming data.
6. Accommodation of large data sets.
7. Portability.

23. Write short Notes on: (any two)

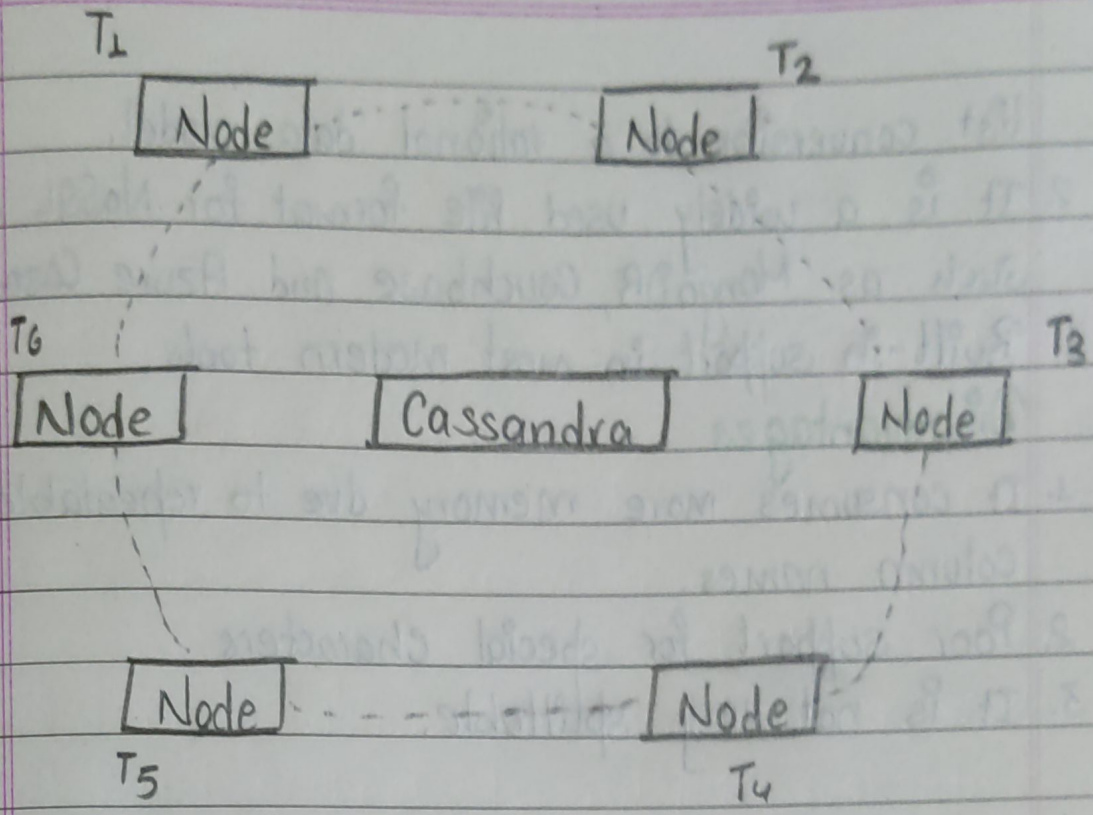
i Basic Architecture of Cassandra

Cassandra was designed to address many architecture requirements. Some of the features of cassandra architecture are as follows:

- It is designed such that it has no master or slave nodes.
- It has a ring-type architecture, that is, its node are logically distributed like a ring.
- Data is automatically distributed across all the nodes.

Additional features of cassandra architecture are:-

- It supports multiple data centers.
- Data can be replicated across data centers.



v. JSON

JSON (Java Script object notation) data are presented as key-value pairs in a partially structured format. JSON is often compared to XML because it can store data in a hierarchical format. Both formats are user-readable, but JSON documents are typically much smaller than XML. It is used to represent data structures, exchange formats for hot data, and cold data warehouses.

Advantages

1. It supports lists of objects, helping to avoid chaotic

list conversion to a rational data model.

2. It is a widely used file format for NoSQL databases such as MongoDB, Couchbase and Azure Cosmos DB.
3. Built-in support in most modern tools.

Disadvantages

1. It consumes more memory due to repeatable column names.
2. Poor support for special characters.
3. It is not very splittable.