USN										

CELEBRATING 25 YEARS CMRIT CMR INSTITUTE OF TECHNOLOGY, BENGALURU. ACCREDITED WITH A+ GRADE BY NAAC

Internal Assessment Test 2 Scheme and Solution– Nov 2024

Sub:	Big Data Analytics				Sub Code:	18CS71	Branch:	CSE
Date:	19/11/2024	Duration:	90 mins	Max Marks: 50	Sem / Sec:	7 –A/B/C		OBE

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1	How will you consider MongoDB as a complete data store which imbibes highly functional secondary indices and provides a powerful framework for data analysis? Enlist business use cases for use of MongoDb.	[10]	CO 3	L4

MongoDB as powerful framework: justification – 6 marks
At least four use cases- 4 marks

Features of MongoDB:
1. MongoDB data store is a physical container for collections. Each DB gets its own set of files on the file system . A number of DBs can run on a single MongoDB server. The database server of MongoDB is mongodb and the client is mongo
2. Collection stores a number of MongoDB documents. It is analogous to a table of RDBMS . A collection exists within a single DB to achieve a single purpose. Documents of the collection are schema less . Thus, it is possible to store documents of varying structures in a collection
3 .Document model is well defined . Document is the unit of storing data in a MongoDB database. Insert , update and delete operations can be performed on a collection . Documents have dynamic schema
4 MongoDB is a document data store in which one collection holds different documents . Number of fields, content and size of the document can differ from one document to another.
5 . Storing of data is flexible, and the data store consists of JSON-like documents . This implies that the fields can vary from document to document and data structure can be changed over time ; JSON has a standard structure, and a scalable way of describing hierarchical data 6. Storing documents on disk is in BSON serialization format. BSON is a binary representation of JSON documents. The mongo JavaScript shell and MongoDB language drivers perform translation between BSON and language specific document representation.
7. Querying , indexing, and real time aggregation allows accessing and analyzing the data efficiently
8. Deep query ability Supports dynamic queries on documents using a document based query language that's nearly as powerful as SQL.
9. No complex Joins
10. Distributed DB makes availability high, and provides horizontal scalability
11. Indexes on any field in a collection of documents : Users can create indexes on any field in a document. Indices support queries and operations . By default, MongoDB creates an index on the _id field of every collection.

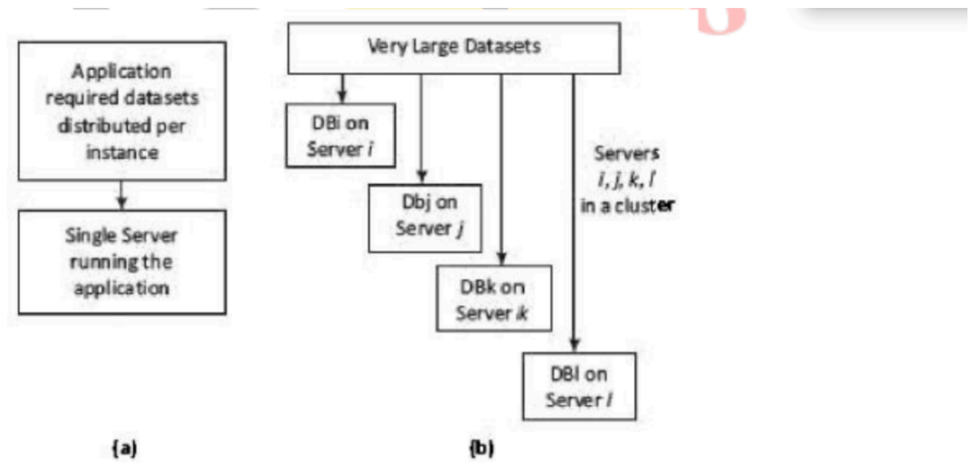
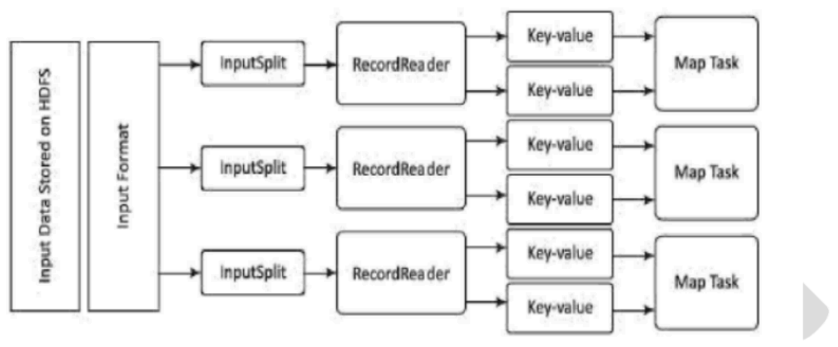
	12. Atomic operations on a single document can be performed even though support of multi document transactions is not present . The operations are alternate to ACID transaction requirement of a relational DB. 13 . Fast in-place updates: The DB does not have to allocate a new memory location and write a full new copy of the object in case of data updates . This results in high performance for frequent update use cases. For example , incrementing a counter operation does not fetch the document from the server. Here, the increment operation can simply be set 14 . No configurable cache: MongoDB uses all free memory on the system automatically by way of memory mapped files . The most recently used data is kept in RAM . If indexes are created for queries and the working dataset fits in RAM, MongoDB serves all queries from memory. 15. Conversion/mapping of application objects to data store objects not needed MongoDB Use Case 1: Product Data Management MongoDB is perfect for Product Data Management. It enables product data and related information to be managed and processed in a single, central system. This allows for Detailed Cost Analysis, Increased Productivity, and Improved Collaboration. MongoDB Use Case 2: Operational Intelligence Another real-world MongoDB use case is Operational Intelligence, as it aids in Real-time Decision-making. It allows companies to seamlessly gather various data feeds representing their ongoing business operations and information of related external factors. They can then analyze these feeds as the data arrives for developing profitable and functional business strategies. MongoDB Use Case 3: Product Catalog Product catalogs have been in existence for years in the ever-evolving digital space. However, with the rapid evolution in technology, product catalogs sometimes feel like a new digital experience. This is because the richness and volume of data feed product catalogs' interactions today are remarkable. MongoDB is useful in such applications, as it provides an excellent tool for storing different types of objects. In addition, its Dynamic Schema Capability ensures that product documents only contain attributes relevant to that product. MongoDB Use Case 4: Scaling And Application Mobility For most Mobile Application Development, it is expected that the companies involved will have to deal with different data structures from several sources and potentially highly dynamic growth. Interestingly, MongoDB provides High Flexibility and Scalability that serves as an excellent database solution for such challenges. In addition, it allows developers to focus on developing a better customer experience, instead of spending time adjusting the database.			
2	Write queries in Cassandra for CRUD operations by taking the ProductInfo table. Consider suitable columns and decide the partition key. Table structure and partition key: 2 marks 4 queries- each 2.5 marks 1.CREATE TABLE ProductInfo (product_id UUID PRIMARY KEY, product_name TEXT,	[10]	CO 3	L4

	```
	    price DECIMAL,
	    category TEXT,
	    brand TEXT,
	    stock_quantity INT,
	    created_at TIMESTAMP
	);
	2.INSERT INTO ProductInfo (product_id, product_name, price, category,
	brand, stock_quantity, created_at)
	VALUES (uuid(), 'Smartphone XYZ', 499.99, 'Electronics', 'BrandA', 150,
	toTimestamp(now()));

	3.SELECT * FROM ProductInfo WHERE product_id = <some-uuid>;

	4.UPDATE ProductInfo
	SET stock_quantity = 120
	WHERE product_id = <some-uuid>;

	5.DELETE FROM ProductInfo WHERE product_id = <some-uuid>;
	```
3	Explain Shared-Nothing Architecture for Big Data Tasks. Diagram -5 Marks Explanation -5 Marks The columns of two tables relate by a relationship. A relational algebraic equation specifies the relation. Keys share between two or more SQL tables in RDBMS. Shared nothing (SN) is a cluster architecture. A node does not share data with any other node.Data of different data stores partition among the number of nodes (assigning different computers to deal with different users or queries). Processing may require every node to maintain its own copy of the application's data, using a coordination protocol. Examples of partitioning and processing are Hadoop, Flink and Spark. The features of SN architecture are as follows: l. Independence: Each node with no memory sharing; thus possesses computational self-sufficiency 2. Self-Healing: A link failure causes creation of another link 3. Each node functioning as a shard: Each node stores a shard (a partition of large DBs) 4. No network contention Choosing the Distribution Models Big Data requires distribution on multiple data nodes at clusters. Distributed software components give advantage of parallel processing; thus providing horizontal scalability. Distribution gives (i) ability to handle large-sized data, and (ii) processing of many read and write operations simultaneously in an application. A resource manager manages, allocates, andschedules the resources of each processor, memory and network connection. Distribution increases the availability when a network slows or link fails. Four models for distribution of the data store are given below: Single Server Model Simplest distribution option for NoSQL data store and access is Single Server Distribution (SSD) of an application. A graph database processes the relationships between nodes at a server. The SSD model suits well for graph DBs. Aggregates of datasets may be key-value, column-family orBigTable data stores which require	[10]	CO 1	L3

	sequential processing. These data stores also use the SSD model. An application executes the data sequentially on a single server. Figure 3.9(a) shows theSSD model. Process and datasets are distributed to a single server which runs the application.Sharding Very Large Databases Figure shows sharding of very large datasets into four divisions, each running the application on four i,j, k and l different servers at the cluster. DBi, DBj, DBk and DB1 are four multiple shards in parallel. Sharding provides horizontal scalability. A data store may add an auto-sharding feature. The performance improves in the SN. However, in case of a link failure with the application, the application can migrate the shard DB to another node.  (a) Single server model (b) Shards distributed on four servers in a cluster			
4	Explain MapReduce execution steps with suitable examples. Diagram - 5 Marks Explanation -5 Marks Big Data Processing employs the Map Reduce Programming Model. A job means a Map Reduce Program. Each job consists of several smaller units, called MapReduce Tasks.A software execution framework in MapReduce programming defines the parallel tasks. The Hadoop MapReduceimplementation usesJava framework.  **Figure 4.5** Key-value pairing in MapReduce	[10]	CO 4	L3

Key-Value Pair
Each phase (Map phase and Reduce phase) of MapReduce has key-value pairs as input and output. Data should be first converted into key-value pairs before it is passed to the Mapper, as the Mapper only understands key-value pairs of data.
Key-value pairs in Hadoop MapReduce are generated as follows:
InputSplit - Defines a logical representation of data and presents a Split data for processing at individual map(). RecordReader - Communicates with the InputSplit and converts the Split into.
records which are in the form of key-value pairs in a format suitable for reading by the Mapper.
RecordReader uses TextinputFormat by default for converting data into key-value pairs. RecordReader communicates with the InputSplit until the file is read.

Generation of a key-value pair in MapReduce depends on the dataset and the required output. Also, the functions use the key-value pairs at four places: map() input, map() output, reduce() input and reduce() output.

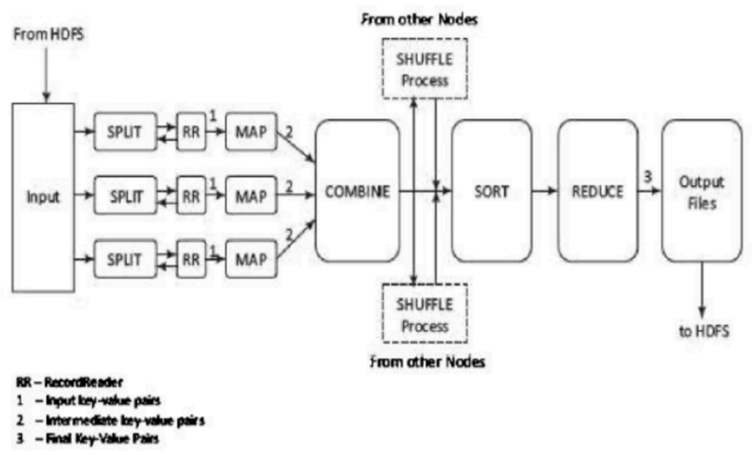


Figure 4.6 MapReduce execution steps

1. Input Data:Hello World

- Hadoop MapReduce
- Hello Hadoop
- MapReduce Framework

2. Map Phase (Mapper) - Splitting and Mapping:

- In this phase, the mapper reads each line of text and splits it into words.
- For each word in a line, the mapper emits a key-value pair: the word as the key, and 1 as the value, indicating that the word was found once.
- Output of the mapper for each word is a pair (word, 1).

Example: For the line "Hello World", the mapper will emit:

("Hadoop", 1)
("MapReduce", 1)

3. Shuffle and Sort Phase:

- After the map phase, all the emitted key-value pairs are shuffled and sorted by the key (the word).
- The shuffle groups all occurrences of the same word together, and the sort organizes them.
- After this phase, we get groups of keys with their corresponding values (counts).

Example: The key-value pairs after the shuffle and sort phase might look like:

("Hello", [1, 1])
("World", [1])
("Hadoop", [1, 1])
("MapReduce", [1, 1])
("Framework", [1])

. Reduce Phase (Reducer) - Summing Counts:

- In the reduce phase, the reducer processes each group of key-value pairs (where the key is a word and the value is a list of counts).
- The reducer sums the values for each key to get the total count of occurrences of that word.
- The output of the reducer is a final key-value pair: the word and its total count.

Example: For the key ("Hello", [1, 1]), the reducer sums the counts to get 2, and outputs:

("Hello", 2)

("Hadoop", 2)

5. Final Output:

- The final output is the result of the reduce phase, which consists of words and their total counts.
- The output is typically stored in a file or database, such as HDFS.

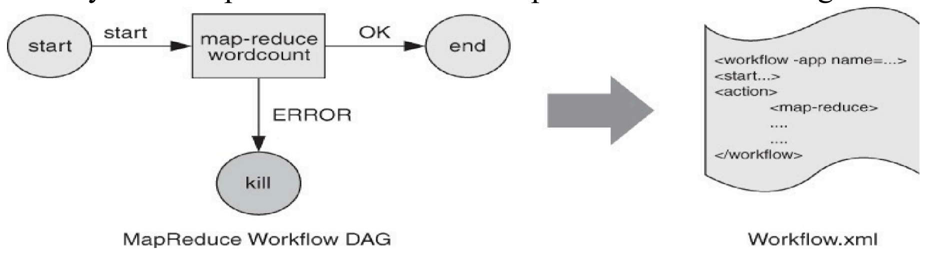
Final Output:

("Hello", 2)

("World", 1)

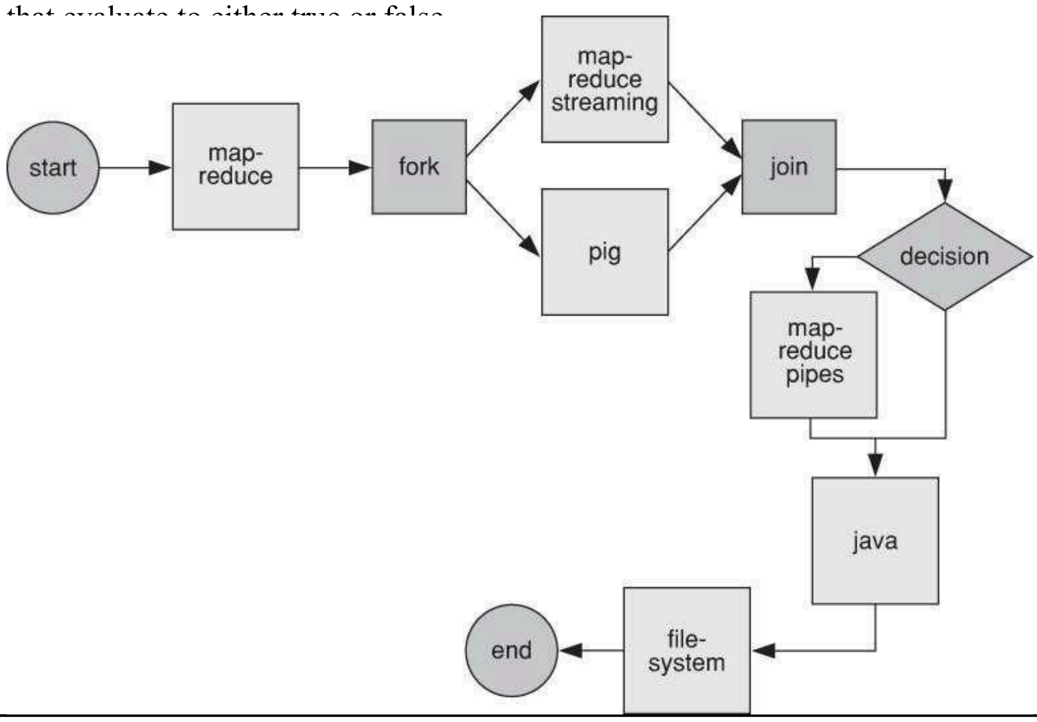
("Hadoop", 2)

("MapReduce", 2)

	("Framework", 1)			
5	Identify and explain the context. Explain the below diagram in detail.  start start map-reduce wordcount OK end ERROR kill MapReduce Workflow DAG <workflow -app name=...> <start...> <action> <map-reduce> .... </workflow> Workflow.xml Explanation -10 Marks Identification and explanation of the Context- 4 mark Diagram node explanation - 6 marks ● Oozie is a workflow director system designed to run and manage multiple related Apache Hadoop jobs. ● Oozie is designed to construct and manage these workflows. Oozie is not a substitute for the YARN scheduler. ● YARN manages resources for individual Hadoop jobs, and Oozie provides a way to connect and control Hadoop jobs on the cluster. ● Oozie workflow jobs are represented as directed acyclic graphs (DAGs) of actions. Three types of Oozie jobs are permitted: Workflow—a specified sequence of Hadoop jobs with outcome-based decision points and control dependency. Progress from one action to another cannot happen until the first action is complete. Coordinator—a scheduled workflow job that can run at various time intervals or when data becomes available. Bundle—a higher-level Oozie abstraction that will batch a set of coordinator jobs. Oozie is integrated with the rest of the Hadoop stack, supporting several types of Hadoop jobs out of the box Figure depicts a simple Oozie workflow. In this case, Oozie runs a basic MapReduce operation. If the application was successful, the job ends; if an error occurred, the job is killed. Such workflows contain several types of nodes: **Control flow nodes** define the beginning and the end of a workflow. They include start, end, and optional fail nodes. **Action nodes** are where the actual processing tasks are defined. When an action node finishes, the remote systems notify Oozie and the next node in the workflow is executed. Action nodes can also include HDFS commands. **Fork/join nodes** enable parallel execution of tasks in the workflow. The fork node enables two or more tasks to run at the same time. A join node represents a rendezvous point that must wait until all forked tasks	[10]	CO 2	L4

complete.

Control flow nodes enable decisions to be made about the previous task. Control decisions are based on the results of the previous action (e.g., file size or file existence). Decision nodes are essentially switch-case statements that use JSP EL (Java Server Pages—Expression Language) that evaluate to either true or false.



6	Explain working of Flume agent and Apache Pig, Flume Agent- 5 Marks Apache Pig -5 Marks	[10]	CO 2	L3

Apache Flume is an independent agent designed to collect, transport, and store data into HDFS.

• Often data transport involves a number of Flume agents that may traverse a series of machines and locations. Flume is often used for log files, social media-generated data, email messages, and just about any continuous data source.

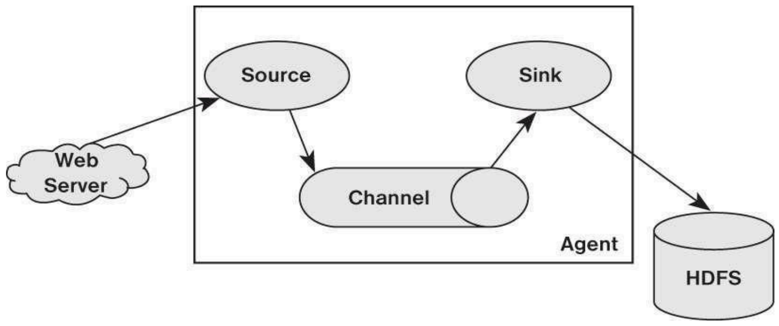
• Flume agent is composed of three components.

✔ ***Source***. The source component receives data and sends it to a channel. It can send the data to more than one channel. The input data can be from a real-time source (e.g., weblog) or another Flume agent.

✔ ***Channel.*** A channel is a data queue that forwards the source data to the sink destination. It can be thought of as a buffer that manages input (source) and output (sink) flow rates.

✔ ***Sink.*** The sink delivers data to destination such as HDFS, a local

file, or another Flume agent.



Apache Pig is a high-level platform built on top of Hadoop for analyzing large datasets in a distributed environment. Pig uses a scripting language called Pig Latin, which abstracts the complexity of writing raw MapReduce programs, making it easier to process and analyze data.

Working of Apache Pig:

Pig operates in two modes: Local mode (for smaller datasets) and MapReduce mode (for large datasets running on Hadoop).

1. Pig Latin Script:
    - Pig Latin is a language designed to express data flows and transformations. It consists of commands that define operations such as loading, transforming, filtering, grouping, and storing data.
    - A Pig Latin script may contain a sequence of operations that read data from the file system (HDFS), process it, and then store the results back.
2. Execution:

	○ Once the Pig Latin script is written, it is compiled into a series of MapReduce jobs. These jobs are executed on the Hadoop cluster. ○ Pig translates the Pig Latin code into MapReduce jobs that can be run on Hadoop, allowing you to perform transformations like joins, filters, and group-by operations without manually writing complex MapReduce code.			