Module -3

NoSQL

3.1 Introduction

Big Data uses distributed systems. A distributed system consists of multiple data nodes at clusters of machines and distributed software components. The tasks execute in parallel with data at nodes in clusters. The computing nodes communicate with the applications through a network.

Following are the features of distributed-computing architecture (Chapter

l. **Increased reliability and fault tolerance:** The important advantage of distributed computing system is reliability. If a segment of machines in a cluster fails then the rest of the machines continue work. When the datasets replicate at number of data nodes, the fault tolerance increases further. The dataset in remaining segments continue the same computations as being done at failed segment machines.

2. Flexibility makes it very easy to install, implement and debug new services in a distributed environment.

3. Sharding is storing the different parts of data onto different sets of data nodes, clusters or servers. For example, university students huge database, on sharding divides in databases, called shards. Each shard may correspond to a database for an individual course and year. Each shard stores at different nodes or servers.

4. Speed: Computing power increases in a distributed computing system as shards run parallelly on individual data nodes in clusters independently (no data sharing between shards).

5. Scalability: Consider sharding of a large database into a number of shards, distributed for computing in different systems. When the database expands further, then adding more machines

and increasing the number of shards provides horizontal scalability. Increased computing power and running number of algorithms on the same machines provides vertical scalability.Resources sharing: Shared resources of memory, machines and network architecture reduce the cost.

Open system makes the service accessible to all nodes.

6. Performance: The collection of processors in the system provides higher performance than a centralized computer, due to lesser cost of communication among machines (Cost means time taken up in communication).

3.2 NOSQL DATA STORE

SQL is a programming language based on relational algebra. It is a declarative language and it defines the data schema . SQL creates databases and RDBMS s. RDBMS uses tabular data store with relational algebra, precisely defined operators with relations as the operands. Relations are a set of tuples. Tuples are named attributes. A tuple identifies uniquely by keys called candidate keys.

ACID Properties in SQL Transactions

Atomicity of transaction means all operations in the transaction must complete, and if interrupted, then must be undone (rolled back). For example, if a customer withdraws an amount then the bank in first operation enters the withdrawn amount in the table and in the next operation modifies the balance with new amount available. Atomicity means both should be completed, else undone if interrupted in between.

Consistency in transactions means that a transaction must maintain the integrity constraint, and follow the consistency principle. For example, the difference of sum of deposited amounts and withdrawn amounts in a bank account must equal the last balance. All three data need to be consistent.

Isolation of transactions means two transactions of the database must be isolated from each other and done separately.

Durability means a transaction must persist once completed

NOSQL

A new category of data stores is NoSQL (means Not Only SQL) data stores. NoSQL is an altogether new approach of thinking about databases, such as schema flexibility, simple relationships, dynamic schemas, auto sharding, replication, integrated caching, horizontal scalability of shards, distributable tuples, semi-structures data and flexibility in approach.

Issues with NoSQL data stores are lack of standardization in approaches, processing difficulties for complex queries, dependence on eventually consistent results in place of consistency in all states.

Big Data NoSQL

NoSQL records are in non-relational data store systems. They use flexible data models. The records use multiple schemas. NoSQL data stores are considered as semi-structured data. Big Data Store uses NoSQL.

NoSQL data store characteristics are as follows:

1. NoSQL is a class of non-relational data storage system with flexible data model. Examples of NoSQL data-architecture patterns of datasets are key-value pairs, name/value pairs, Column family,Big-data store, Tabular data store, Cassandra (used in Facebook/Apache), HBase, hash table [Dynamo (Amazon S3)], unordered keys using]SON (CouchDB),]SON (PNUTS),]SON (MongoDB), Graph Store, Object Store, ordered keys and semi-structured data storage systems.
2. NoSQL not necessarily has a fixed schema, such as table; do not use the concept of Joins (in distributed data storage systems); Data written at one node can be replicated to multiple nodes. Data store is thus fault- tolerant. The store can be partitioned into unshared shards.

Features in NoSQL Transactions NoSQL transactions have following features:

1. **Relax one or more of the ACID properties.**
2. Characterize by two out of three properties (consistency, availability and partitions) of CAP theorem, two are at least present for the application/ service/process.
3. Can be characterized by BASE properties

Big Data NoSQL Solutions NoSQL DBs are needed for Big Data solutions. They play an important role in handling Big Data challenges. Table 3.1 gives the examples of widely used NoSQL data stores.

Table 3.1 NoSQL data stores and their characteristic features

Apache's HBase	HDFS compatible, open-source and non-relational data store written inJava; A column-family based NoSQL data store, data store providing BigTable-like capabilities (Sections 2.6 and 3.3.3.2); scalability, strong consistency, versioning, configuring and maintaining data store characteristics
Apache's MongoDB	HDFS compatible; master-slave distribution model (Section 3.5.1.3); document-oriented data store withJSON-like documents and dynamic schemas; open-source, NoSQL, scalable and non-relational database; used by Websites Craigslist, eBay, Foursquare at the backend
Apache's Cassandra	HDFS compatible DBs; decentralized distribution peer-to-peer model (Section 3.5.1.4); open source; NoSQL; scalable, non-relational, column-family based, fault-tolerant and tuneable consistency (Section 3.7) used by Facebook and Instagram

Apache's CouchDB	A project of Apache which is also widely used database for the web. CouchDB consists of Document Store. It uses theJSON data exchange format to store its documents,JavaScript for indexing, combining and transforming documents, and HTTP APis
Oracle NoSQL	Step towards NoSQL data store; distributed key-value data store; provides transactional semantics for data manipulation , horizontal scalability, simple administration and monitoring
Riak	An open-source key-value store; high availability (using replication concept), fault tolerance, operational simplicity, scalability and written in Erlang

CAP Theorem Among C, A and P, two are at least present for the application/service/process. Consistency means all copies have the same value like in traditional DBs. Availability means at least one copy is available in case a partition becomes inactive or fails. For example, in web applications, the other copy in the other partition is available. Partition means parts which are active but may not cooperate (share) as in distributed DBs.

1. *Consistency in distributed databases* means that all nodes observe the same data at the same time. Therefore, the operations in one partition of the database should reflect in other related partitions in case of distributed database. Operations, which change the sales data from a specific showroom in a table should also reflect in changes in related tables which are using that sales data.
2. *Availability* means that during the transactions, the field values must be available in other partitions of the database so that each request receivesa response on success as well as failure. (Failure causes the response to request from the replicate of data). Distributed databases require transparency between one another. Network failure may lead to data unavailability in a certain partition in case of no replication. Replication ensures availability.
3. *Partition* means division of a large database into different databases without affecting the operations on them by adopting specified procedures.
4. *Partition tolerance:* Refers to continuation of operations as a whole even in case of message loss, node failure or node not reachable.

Brewer's CAP (c.onsistency, Availability and fartition Tolerance) theorem demonstrates that any distributed system cannot guarantee C, A and P together.

1. Consistency- All nodes observe the same data at the same time.
2. Availability- Each request receives a response on success/failure.
3. Partition Tolerance-The system continues to operate as a whole even in case of message loss, node failure or node not reachable.

Partition tolerance cannot be overlooked for achieving reliability in a distributed database system. Thus, in case of any network failure, a choice canbe:

- Database must answer, and that answer would be old or wrong data (AP).
- Database should not answer, unless it receives the latest copy of the data(CP).

The CAP theorem implies that for a network partition system, the choice of consistency and availability are mutually exclusive. CA means consistency andavailability, AP means availability and partition tolerance and CP means consistency and partition tolerance. Figure 3.1 shows the CAP theorem usage in Big Data Solutions.

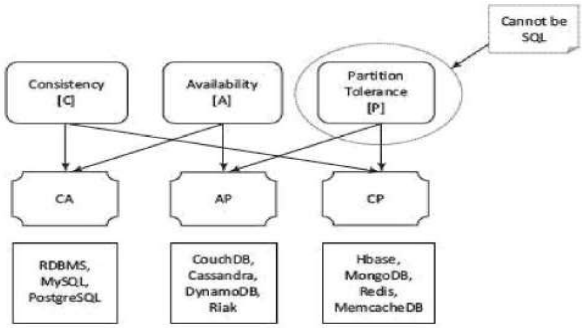


Figure 3.1 CAP theorem in Big Data solutions

Schema Less Database

Schema of a database system refers to designing of a structure for datasets and data structures for storing into the database. NoSQL data not necessarily have a fixed table schema. The systems do not use the concept of Join (between distributed datasets). A cluster-based highly distributed node manages a single large data store with a NoSQL DB. Data written at one node replicates to multiple nodes. Therefore, these are identical, fault-tolerant and partitioned into shards. Distributed databases can store and process a set of information on more than one computing nodes.

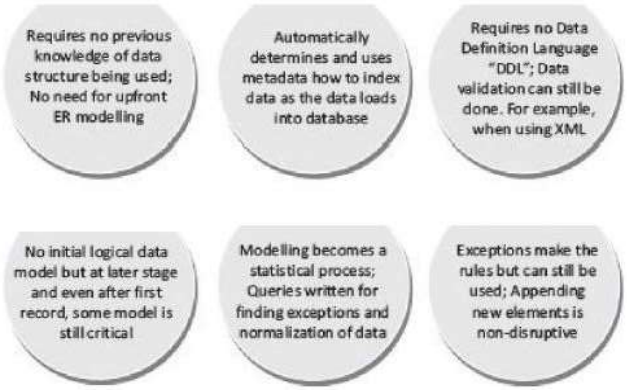


Figure 3.2 Characteristics of Schema-less model

Increasing Flexibility for Data Manipulation

NoSQL data store possess characteristic of increasing flexibility for data manipulation. The new attributes to database can be increasingly added. Late binding of them is also permitted.

BASE Properties BA stands for basic availability, S stands for soft state and E stands for eventual consistency.

l. Basic availability ensures by distribution of shards (many partitions of huge data store) across many data nodes with a high degree of replication. Then, a segment failure does not necessarily mean a complete data store unavailability.

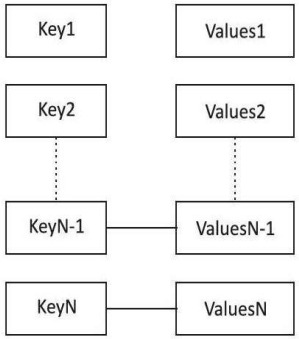
2. Soft state ensures processing even in the presence of inconsistencies but achieving consistency eventually. A program suitably takes into account the inconsistency found during processing. NoSQL database design does not consider the need of consistency all along the processing time.

3. Eventual consistency means consistency requirement in NoSQL databases meeting at some point of time in future. Data converges eventually to a consistent state with no time-frame specification for achieving that. ACID rules require consistency all along the processing on completion of each transaction. BASE does not have that requirement and has the flexibility.

3.3 NOSQL DATA ARCHITECTURE PATTERNS

3.3.1 Key-Value Store

The simplest way to implement a schema-less data store is to use key-value pairs. The data store characteristics are high performance, scalability and flexibility. Data retrieval is fast in key-value pairs data store. A simple string called, key maps to a large data string or BLOB (Basic Large Object). Key-value store accesses use a primary key for accessing the values. Therefore, the store can be easily scaled up for very large data. The concept is similar to a hash table where a unique key points to a particular item(s) of data. Figure 3.4 shows key-value pairs architectural pattern and example of students' database as key-value pairs



Number of key-values pair, N can be a very large number

Key	Value
"Ashish"	"Category: Student; Class: B.Tech.; Semester: VII; Branch: Engineering; Mobile:9999912345"
"Mayuri"	"Category: student; class: M.Tech.; Mobile:8888823456"

Advantages of a key-value store are as follows:

1. Data Store can store any data type in a value field. The key-value system stores the information as a BLOB of data (such as text, hypertext, images,video and audio) and return the same BLOB when the data is retrieved. Storage is like an English dictionary. Query for a word retrieves the meanings, usages, different forms as a single item in the dictionary. Similarly, querying for key retrieves the values.
2. A query just requests the values and returns the values as a single item. Values can be of any data type.
3. Key-value store is eventually consistent.
4. Key-value data store may be hierarchical or may be ordered key-value store.
5. Returned values on queries can be used to convert into lists, table- columns, data-

frame fields and columns.

6. Have (i) scalability, (ii) reliability, (iii) portability and (iv) low operationalcost.
7. The key can be synthetic or auto-generated. The key is flexible and can be represented in many formats: (i) Artificially generated strings created from a hash of a value, (ii) Logical path names to images or files, (iii) RESTweb-service calls (request response cycles), and (iv) SQL queries.

Limitations of key-value store architectural pattern are:

1. No indexes are maintained on values, thus a subset of values is not searchable.
2. Key-value store does not provide traditional database capabilities, such as atomicity of transactions, or consistency when multiple transactions are executed simultaneously. The application needs to implement such capabilities.
3. Maintaining unique values as keys may become more difficult when the volume of data increases. One cannot retrieve a single result when a key- value pair is not uniquely identified.
4. Queries cannot be performed on individual values. No clause like 'where' in a relational database usable that filters a result set.

Table 3.2 Traditional relational data model vs. the key-value store model

Traditional relational model	Key-value store model
Result set based on row values	Queries return a single item
Values of rows for large datasets are indexed	No indexes on values
Same data type values in columns	Any data type values

Typical uses of key-value store are:

(i) Image store,

(ii) Document or file store,

(iii) Lookup table, and

(iv) Query-cache.

Riak is open-source Erlang language data store. It is a key-value data store system. Data auto-

distributes and replicates in Riak. It is thus, fault tolerant and reliable. Some other widely used key-value pairs in NoSQL DBs are Amazon's DynamoDB, Redis (often referred as Data Structure server), Memcached and its flavours, Berkeley DB, upscaledb (used for embedded databases), project Voldemort and Couchbase.

3.3.2 Document Store

Characteristics of Document Data Store are high performance and flexibility. Scalability varies, depends on stored contents. Complexity is low compared to tabular, object and graph data stores.

Following are the features in Document Store:

1. Document stores unstructured data.
2. Storage has similarity with object store.
3. Data stores in nested hierarchies. For example, inJSON formats data model[Example 3.3(ii)], XML document object model (DOM), or machine-readable data as one BLOB. Hierarchical information stores in a single unit called *document tree*. Logical data stores together in a unit.
4. Querying is easy. For example, using section number, sub-section number and figure caption and table headings to retrieve document partitions.
5. No object relational mapping enables easy search by following paths fromthe root of document tree.
6. Transactions on the document store exhibit ACID properties.

Typical uses of a document store are: (i) office documents, (ii) inventory store, (iii) forms data, (iv) document exchange and (v) document search.

Examples of Document Data Stores are CouchDB and MongoDB.

CSV and JSON File Formats CSV data store is a format for records CSV does not represent object-oriented databases or hierarchical data records. *JSON* and XML represent semistructured data, object- oriented records and hierarchical data records. *JSON* (Java Script Object Notation) refers to a language format for semistructured data. *JSON* represents object-oriented and hierarchical data records, object, and resource arrays in JavaScript.

Semester, Subject Code, Subject Name, Grade
1, CS101, ""Theory of Computations"", 7.8.
1, CS102,1, ""Computer Architecture"", 7.8.
.....
2, CS204, ""Object Oriented Programming"", 7.2.
2, CS205, ""Data Analytics"", 8.1.

JSON Files

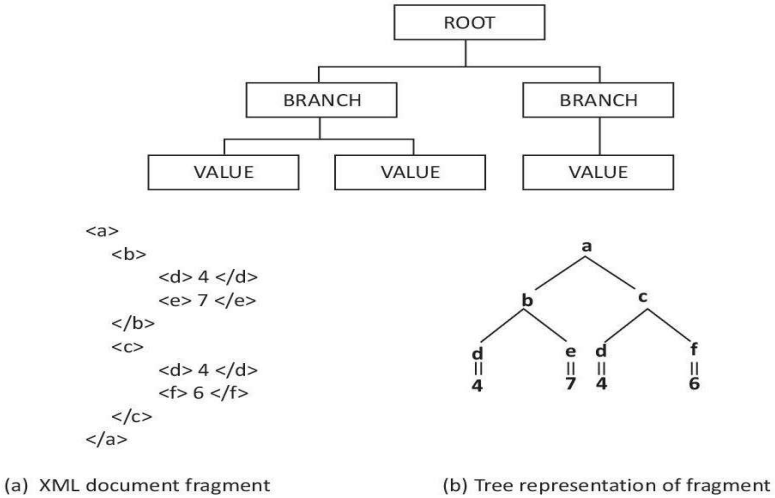
- Semi-structured data
- object-oriented records and hierarchical data records
- JSON refers to a language format for semistructured data. JSON represents object-oriented and hierarchical data records, object, and resource arrays in JavaScript

Document JSON Format CouchDB Database Apache CouchDB is an open- source database. Its features are:

- CouchDB provides mapping functions during querying, combining and filtering of information.
- CouchDB deploys JSON Data Store model for documents. Each document maintains separate data and metadata (schema).
- CouchDB is a multi-master application. Write does not require field locking when controlling the concurrency during multi-master application.
- CouchDB querying language is JavaScript. Java script is a language which

XML

- An extensible, simple and scalable language. Its self-describing format describes structure and contents in an easy to understand format
- XML is widely used. The document model consists of root element and their sub-elements. XML document model has a hierarchical structure. XML document model has features of object-oriented records. XML format finds wide uses in data store and
- XML document model has a hierarchical structure. XML document model has features of object-oriented records. XML format finds wide uses in data store



(a) XML document fragment (b) Tree representation of fragment

Tabular data stores use rows and columns. Row-head field may be used as a keywhich access and retrieves multiple values from the successive columns in that row. The OLTP is fast on in-memory row-format data.

Columnar Data Store A way to implement a schema is the divisions into columns. Storage of each column, successive values is at the successive memory addresses. Analytics processing (AP) In-memory uses columnar storage in memory. A pair of row-head and column-head is a key-pair. The pair accesses a field in the table.

Column-Family Data Store Column-family data-store has a group of columns as a column family. A combination of row-head, column-family head and table- column head can also be a key to access a field in a column of the table during querying. Combination of row head, column families head, column-family head and column head for values in column fields can

also be a key to access fields ofa column. A column-family head is also called a super-column head.

Sparse Column Fields A row may associate a large number of columns but contains values in few column fields. Similarly, many column fields may not have data. Columns are logically grouped into column families. Column-family data stores are then similar to sparse matrix data. Most elements of sparse matrix are empty. Data stores at memory addresses is columnar-family based rather than as row based. Metadata provide the column-family indices of not empty column fields.

That facilitates OLAP of not empty column families faster. For example, assume hash key in a column heading field and values in successive rows at one column family. For another key, the values will be in another column family.

Grouping of Column Families Two or more column-families in data store form a super group, called super column. Table 3.3 consists of one such group (super column), 'Nestle Chocolate Flavours Group'.

Grouping into Rows When number of rows are very large then horizontal partitioning of the table is a necessity. Each partition forms one row-group. For example, a group of 1 million rows per partition. A row group thus has all column data store in the memory for in-memory analytics. Practically, row groups are chosen such that memory required for the group is above, say 10 MB and below the maximum size which can cached and buffered in memory, say 1 GB for in-memory analytics.

Characteristics of Columnar Family Data Store Columnar family data store imbibes characteristics of very high performance and scalability, moderate level

of flexibility and lower complexity when compared to the object and graph databases. Advantages of column stores are:

1. *Scalability:* The database uses row IDs and column names to locate a column and values at the column fields. The interface for the fields is simple. The back-end system can distribute queries over a large number of processing nodes without performing any Join operations. The retrieval of data from the distributed node can be least complicated by an intelligent plan of row IDs and columns, thereby increasing performance. Scalability means addition of number of rows as the number of ACVMs increase in Example1.6(i). Number of processing instructions is proportional to the number of ACVMs due to scalable operations.

2. *Partitionability:* For example, large data of ACVMs can be partitioned into datasets of size, say 1 MB in the number of row-groups. Values in columns of each row-group,process in-memory at a partition. Values in columns of each row-group independently parallelly process in-memory at the partitioned nodes.
3. *Availability:* The cost of replication is lower since the system scales on distributed nodes efficiently. The lack of Join operations enables storing a part of a column- family matrix on remote computers. Thus, the data is always available in case of failure of any node.
4. *Tree-like columnar* structure consisting of column-family groups, column families and columns. The columns group into families. The column families group into column groups (super columns). A key for the column fields consists of three secondary keys: column-families group ID, column-family ID and column-head name.
5. *Adding new data at* ease: Permits new column *Insert* operations. Trigger operation creates new columns on an Insert. The column-field values can add after the last address in memory if the column structure is known in advance. New row-head field, row-group ID field, column-family group, column family and column names can be created at any time to add new data.
6. Querying all the field values in a column in a family, all columns in the family or a group of column-families, is fast in in-memory column-family data store.
7. Replication of columns: HDFS-compatible column-family data stores replicate each data store with default replication factor= 3.
8. No optimization for Join: Column-family data stores are similar to sparse matrix data. The data do not optimize for Join operations.

Big Table Data Store

Examples of widely used column-family data store are Google's BigTable, HBase and Cassandra. Keys for *row key, column key, timestamp* and *attribute* uniquely identify the values in the fields

Following are features of a BigTable:

- Massively scalable NoSQL. BigTable scales up to 100s of petabytes.
- Integrates easily with Hadoop and Hadoop compatible systems.
- Compatibility with MapReduce, HBase APis which are open-source Big Data platforms.
- Key for a field uses not only row_ID and Column_ID (for example, ACVM_ID and KitKat

in Example 3.6) but also timestamp and attributes. Values are ordered bytes. Therefore, multiple versions of values may be present in the BigTable.

- Handles million of operations per second.
- Handle large workloads with low latency and high throughput
- Consistent low latency and high throughput
- APis include security and permissions
- BigTable, being Google's cloud service, has global availability and its service is seamless.

RC *File Format*

Hive uses Record Columnar (RC) file-format records for querying. RC is the best choice for intermediate tables for fast column-family store in HDFS with Hive.Serializability of RC table column data is the advantage. RC file is DeSerializable into column data.

ORC File Format

An ORC (Optimized Row Columnar) file consists of row-group data called stripes. ORC enables concurrent reads of the same file using separate RecordReaders. Metadata store uses Protocol Buffers for addition and removal of fields. [1]

ORC is an intelligent Big Data file format for HDFS and Hive.[2] An ORC file stores a collections of rows as a row-group. Each row-group data store in columnar format. This enables parallel processing of multiple row-groups in anHDFS cluster.

An ORC file consists of a stripe the size of the file is by default 256 MB. Stripe consists of indexing (mapping) data in 8 columns, row-group columns data (contents) and stripe footer (metadata). An ORC has two sets of columns data instead of one column data in RC. One column is for each map or list size and other values which enable a query to decide skipping or reading of the mapped columns. A mapped column has contents required by the query. The columnar layout in each ORC file thus, optimizes for compression and enables skipping of data in columns. This reduces read and decompression load.

Stripe_ID	Index Column 1				Index Column 2
	Index column 1 key 1				Index column 2 key 1
	Contents-Column name	Contents Minimum value	Contents Maximum value	Count (number) of content-column fields	
	...	...	...	...	
	...	...	...	...	
	Index column 1 key 2				Index column 2 key 2
	Column-name	Minimum value	Maximum value	Count of number of column fields	
	...	...	...	...	
	...	...	...	...	

Keys to access or skip a content column in ORC file format

Parquet File Formats

Parquet is nested hierarchical columnar-storage concept. Nesting sequence is the table, row group, column chunk and chunk page. Apache Parquet file is columnar-family store file. Apache Spark SQL executes user defined functions (UDFs) which query the Parquet file columns. A programmer writes the codes for an UDF and creates the processing function for big long queries.

A Parquet file uses an HDFS block. The block stores the file for processing queries on Big Data. The file compulsorily consists of metadata, though the file need not consist of data.

The Parquet file consists of row groups. A row-group columns data process in memory after data cache and buffer at the memory from the disk. Each row group has a number of columns. A row group has Ncol columns, and row group consists of Ncol column chunks. This means each column chunk consists of values saved in each column of each row group.

A column chunk can be divided into pages and thus, consists of one or more pages. The column chunk consists of a number of interleaved pages, Npg• A page is a conceptualized unit which can be compressed or encoded together at an instance. The unit is minimum portion of a chunk which is read at an instance for in-memory analytics.

Row-group_ID	Column Chunk 1 key			
	Page 1 key	Page 2 key	‘’’’	Page m key
	..	...	...	...
	..	...	...	...
	Column Chunk 2 key			
	Page 1key	Page 2 key	‘’’’	Page key m’
	..	...	...	...
	..	...	...	...

Combination of keys for content page in the Parquet file format

Object Data Store

An object store refers to a repository which stores the:

1. Objects (such as files, images, documents, folders, and business reports)
2. System metadata which provides information such as filename, creation_date, last_modified, language_used (such as Java, C, C#, C++, Smalltalk, Python), access_permissions, supported query languages)
3. Custom metadata which provides information, such as subject, category, sharing permissions.

Metadata enables the gathering of metrics of objects, searches, finds the contents and specifies the objects in an object data-store tree. Metadata finds the relationships among the objects, maps the object relations and trends. Object Store metadata interfaces with the Big Data. API first mines the metadatato enable mining of the trends and analytics. The metadata defines classes and properties of the objects. Each Object Store may consist of a database. Document content can be stored in either the object store database storage area or in a file storage area. A single file domain may contain multiple Object Stores.

Object Relational Mapping

The following example explains object relational mapping

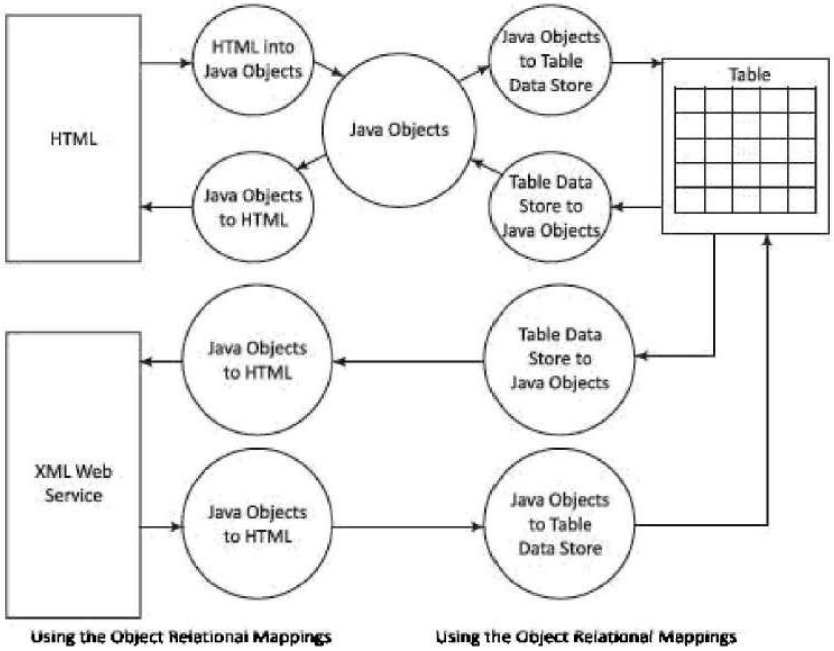
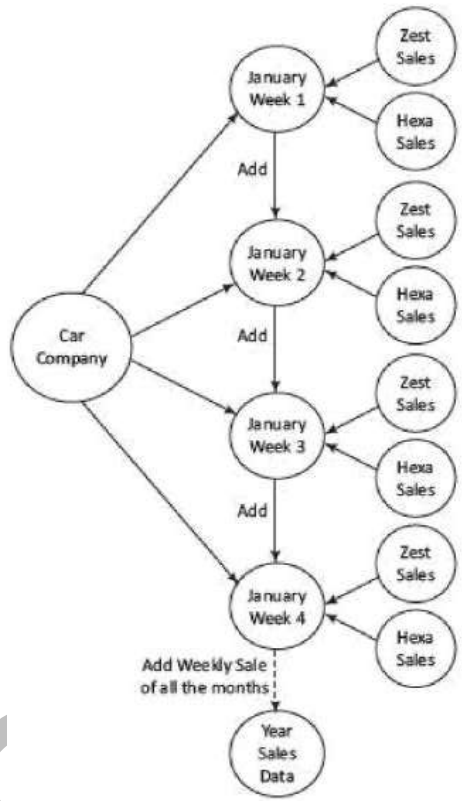


Fig : HTML document and XML web services

Graph Data Base

One way to implement a data store is to use graph database. A characteristic of graph is high flexibility. Any number of nodes and any number of edges can be added to expand a graph. The complexity is high and the performance is variable with scalability. Data store as series of interconnected nodes. Graph with data nodes interconnected provides one of the best database system when relationships and relationship types have critical values.

Nodes represent entities or objects. Edges encode relationships between nodes. Some operations become simpler to perform using graph models. Examples of graph model usages are social networks of connected people. The connections to related persons become easier to model when using the graph model.



Graph Data base for Car Model Sale

Characteristics of graph databases are:

1. Use specialized query languages, such as RDF uses SPARQL
2. Create a database system which models the data in a completely different way than the key-values, document, columnar and object data store models.
3. Can have hyper-edges. A hyper-edge is a set of vertices of a hypergraph. Ahypergraph is a generalization of a graph in which an edge can join any number of vertices (not only the neighbouring vertices).
4. Consists of a collection of small data size records, which have complex interactions between graph-nodes and hypergraph nodes. Nodes represent the entities or objects. Nodes use Joins. Node identification can use URI or other tree-based structure. The edge

encodes a relationship between the nodes.

Graph databases have poor scalability. They are difficult to scale out on multiple servers. This is due to the close connectivity feature of each node in thegraph. Data can be replicated on multiple servers to enhance read and the query processing performance. Write operations to multiple servers and graph queries that span multiple nodes, can be complex to implement.

Typical uses of graph databases are: (i) link analysis, (ii) friend of friend queries, (iii) Rules and inference, (iv) rule induction and (v) Pattern matching. Link analysis is needed to perform searches and look for patterns and relationships in situations, such as social networking, telephone, or email

Examples of graph DBs are Neo4J, AllegroGraph, HyperGraph, Infinite Graph, Titan and FlockDB. Neo4J graph database enable easy usages by Java developers. Neo4J can be designed fully ACID rules compliant. Design consists of adding additional path traversal in between the transactions such that data consistency is maintained and the transactions exhibit ACID properties.

3.4 NO SQL to Manage Big Data

NoSQL Solutions for Big Data

Big Data solution needs scalable storage of terabytes and petabytes, dropping ofsupport for database Joins, and storing data differently on several distributed servers (data nodes) together as a cluster. A solution, such as CouchDB, DynamoDB, MongoDB or Cassandra follow CAP theorem (with compromising the consistency factor) to make transactions faster and easier to scale. A solution must also be partitioning tolerant

Characteristics of Big Data NoSQL solution are:

1. *High and easy scalability:* NoSQL data stores are designed to expand horizontally. Horizontal scaling means that scaling out by adding more machines as data nodes (servers) into the pool of resources (processing, memory, network connections). The design scales out using multi-utility cloud services.
2. *Support to replication:* Multiple copies of data store across multiple nodes ofa cluster. This ensures high availability, partition, reliability and fault tolerance.
3. Distributable: Big Data solutions permit sharding and distributing of shards on multiple clusters which enhances performance and throughput.
4. *Usages ofNoSQL servers* which are less expensive. NoSQL data stores requireless management efforts. It supports many features like automatic repair, easier data distribution and simpler

data models that makes database administrator (OBA) and tuning requirements less stringent.

5. Usages of open-source tools: NoSQL data stores are cheap and open source. Database implementation is easy and typically uses cheap servers to manage the exploding data and transaction while RDBMS databases are expensive and use big servers and storage systems. So, cost per gigabyte data store and processing of that data can be many times less than the cost ofRDBMS
6. *Support to schema-less data model:* NoSQL data store is schema less, so data can be inserted in a NoSQL data store without any predefined schema. So, the format or data model can be changed any time, without disruption of application. Managing the changes is a difficult problem in SQL.
7. *Support to integrated caching:* NoSQL data store support the caching in system memory. That increases output performance. SQL database needs a separate infrastructure for that.
8. *No inflexibility* unlike the SQL/RDBMS, NoSQL DBs are flexible (not rigid) and have no structured way of storing and manipulating data. SQL stores in the form of tables consisting of rows and columns. NoSQL data stores have flexibility in following ACID rules.

Types* of *Big Data Problems

Big Data problems arise due to limitations of NoSQL and other DBs. The following types of problems are faced using Big Data solutions.

1. Big Data need the scalable storage and use of distributed servers together as a cluster. Therefore, the solutions must drop support for the database Joins
2. NoSQL database is open source and that is its greatest strength but at the same time its greatest weakness also because there are not many defined standards for NoSQL data stores. Hence, no two NoSQL data stores are equal. For example:

(i) No stored procedures in MongoDB (NoSQL data store)

(ii) GUI mode tools to access the data store are not available in the market

(iii) Lack of standardization

(iv) NoSQL data stores sacrifice ACID compliancy for flexibility and processingspeed.

Comparison of NOSQL/RDBMS

Feature	NOSQL Data Store	SQL/RDBMS
Model	Schema-less model	Relational
Schema	Dynamic schema	Predefined
Types of data architecture patterns	Key/value based, column-family based, document based, graph based, object based	Table based
Scalable	Horizontally scalable	Vertically scalable
Use ofSQL	No	Yes
Dataset size preference	Prefers large datasets	Large dataset not preferred
Consistency	Variable	Strong
Vendor support	Open source	Strong
ACID properties	May not support, instead follows Brewer's CAP theorem or BASE properties	Strictly follows

3.5 SHARED-NOTHING ARCHITECTURE FOR BIG DATA TASKS

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 are using the partitioning and processing are Hadoop, Flink and Spark.

The features of SN architecture are as follows:

1. *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

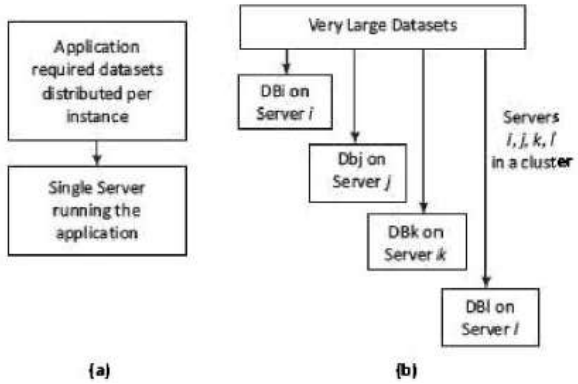
Big Data requires distribution on multiple data nodes at clusters. Distributed software components give advantage of parallel processing; thus providing horizontal scalability. Distribution gives (i) ability to handle large-sized data, and (ii) processing of many read and write operations simultaneously in an application. A resource manager manages, allocates, and schedules 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 or BigTable data stores which require 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 the SSD model. Process and datasets distribute 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



(a) Single server model (b) Shards distributed on four servers in a cluster

The application programming model in SN architecture is such that an application process runs on

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.

Master Slave Distribution

Master directs the slaves. Slave nodes data replicate on multiple slave servers in Master Slave Distribution (MSD) model. When a process updates the master, it updates the slaves also. A process uses the slaves for read operations. Processing performance improves when process runs large datasets distributed onto the slave nodes. Figure 3.10 shows an example of MongoDB. MongoDB database server is mongod and the client is mongo.

Master-Slave Replication Processing performance decreases due to replication in MSD distribution model. Resilience for read operations is high, which means if in case data is not available from a slave node, then it becomes available from the replicated nodes. Master uses the distinct write and read paths.

Complexity Cluster-based processing has greater complexity than the other architectures. Consistency can also be affected in case of problem of significant time taken for updating

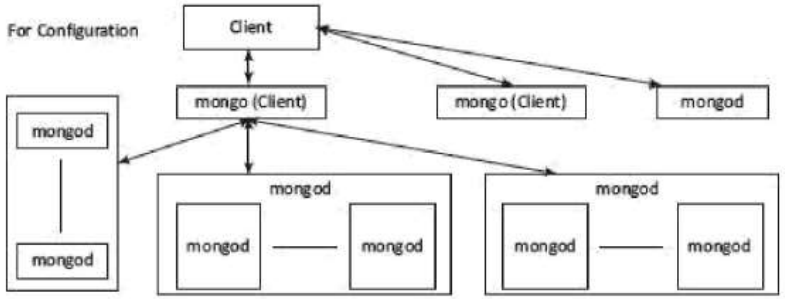


Figure 3.10 Master-slave distribution model. Mongo is a client and mangod is the server

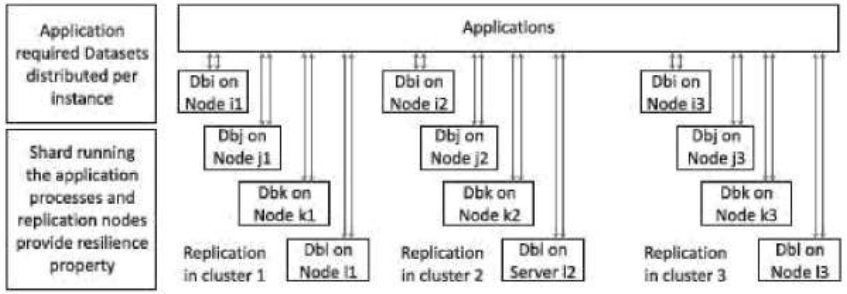
Peer-to-Peer Distribution Model

Peer-to-Peer distribution (PPD) model and replication show the following characteristics: (1) All replication nodes accept read request and send the responses. (2) All replicas function equally. (3) Node failures do not cause loss of write capability, as other replicated node responds.

Cassandra adopts the PPD model. The data distributes among all the nodes in a cluster.

Performance can further be enhanced by adding the nodes. Since nodes read and write both, a replicated node also has updated data. Therefore, the biggest advantage in the model is

consistency. When a write is on different nodes, then write inconsistency occurs.



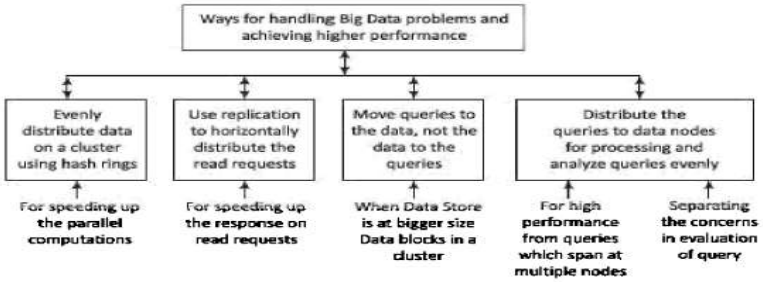
Shards replicating on the nodes, which does read and writeoperations both

Choosing Master-Slave versus Peer-to-Peer

Master-slave replication provides greater scalability for read operations. Replication provides resilience during the read. Master does not provide resilience for writes. Peer-to-peer replication provides resilience for read and writes both.

Sharing Combining with Replication Master-slave and sharding creates multiple masters. However, for each data a single master exists. Configuration assigns a master to a group of datasets. Peer-to-peer and sharding use same strategy for the column-family data stores. The shards replicate on the nodes, which does read and write operations both.

Ways of Handling Big Data Problems



Four ways for handling big data problems

Following are the ways:

1. **Evenly distribute the data** on a cluster using the hash rings: Consistent hashing refers to a process where the datasets in a collection distribute using a hashing algorithm which generates the pointer for a collection. Using only the hash of Collection_ID, a Big Data solution client node determines the data location in the cluster. Hash Ring refers to a map of hashes with locations. The client, resource manager or scripts use the hash ring for data searches and Big Data solutions. The ring enables the consistent assignment and usages of the dataset to a specific processor.

2. Use replication to horizontally distribute the client read-requests: Replication means creating backup copies of data in real time. Many Big Data clusters use replication to make the failure-proof retrieval of data in a distributed environment. Using replication enables horizontal scaling out of the client requests.

3. Moving queries to the data, not the data to the queries: Most NoSQL data stores use cloud utility services (Large graph databases may use enterprise servers). Moving client node queries to the data is efficient as well as a requirement in Big Data solutions.

4. Queries distribution to multiple nodes: Client queries for the DBs analyze at the analyzers, which evenly distribute the queries to data nodes/ replica nodes. High performance query processing requires usages of multiple nodes. The query execution takes place separately from the query evaluation (The evaluation means interpreting the query and generating a plan for its execution sequence).

3.6 MONGODB DATABASE

MongoDB is an open source DBMS. MongoDB programs create and manage databases. MongoDB manages the collection and document data store. MongoDB functions do querying and accessing the required information. The functions include viewing, querying, changing, visualizing and running the transactions. Changing includes updating, inserting, appending or deleting.

MongoDB is (i) non-relational, (ii) NoSQL, (iii) distributed, (iv) open source, (v) document based (vi) cross-platform, (vii) Scalable, (viii) flexible data model, (ix) Indexed, (x) multi-master and (xi) fault tolerant. Document data store in SON-like documents. The data store uses the dynamic schemas.

The typical MongoDB applications are content management and delivery systems, mobile

applications, user data management, gaming, e-commerce, analytics, archiving and logging.

Features of Mango D B

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. DB is default DB in MongoDB that stores within a data folder. The database server ofMongoDB is *mongod* 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 singlepurpose. Collections may store documents that do not have the same fields. Thus, documents of the collection are schema-less. Thus, it is possible to store documents of varying structures in a collection. Practically, in an RDBMS, it is required to define a column and its data type, but does not need them while working with the MongoDB.
3. *Document model* is well defined. Structure of document is clear, Document is the unit of storing data in a MongoDB database. Documents are analogousto the records of RDBMS table. Insert, update and delete operations can beperformed on a collection. Document use *JSON* OavaScript Object Notation) approach for storing data. *JSON* is a lightweight, self-describing format used to interchange data between various applications. JSON data basically has key-value pairs. Documents have dynamic schema.
4. MongoDB is a document data store in which one collection holds differentdocuments. Data store in the form of]SON-style documents. Number of fields, content and size of the document can differ from one document to another.
5. *Storing of data* is flexible, and 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 scalable way of describing hierarchical data (Example 3.3(ii)).
6. *Storing of 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 adocument-based query language that's nearly as powerful as SQL.
9. No complexJoins.
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 new memory location and write a full new copy of the object in case of data updates. This results into 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 operating systems use the similar approach with their file system caches). 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

Dynamic Schema Dynamic schema implies that documents in the same collection do not need to have the same set of fields or structure. Also, the similar fields in a document may contain different types of data. Table 3.8 givesthe comparison with RDBMS

RDBMS	MongoDB
Database	Data store
Table	Collection
Column	Key
Value	Value
Records / Rows / Tuple	Document/ Object
Joins	Embedded Documents
Index	Index
Primary key	Primary key (_id) is default key provided by MongoDB itself

Comparison of Mango DB and RDBMS

Replication: Replication ensures high availability in Big Data. Presence of multiple copies increases on different database servers. This makes DBs fault- tolerant against any database server failure. Multiple copies of data certainly help in localizing the data and ensure availability of data in a distributed system environment.

MongoDB replicates with the help of a replica set. A replica set in MongoDB is a group of mongod (MongoDb server) processes that store the same dataset. Replica sets provide redundancy but high availability. A replica set usually has minimum three nodes. Any one out of them is called primary. The primary node receives all the write operations. All the other nodes are termed as secondary. The data replicates from primary to secondary nodes. A new primary node can be chosen among the secondary nodes at the time of automatic failover or maintenance. The failed node when recovered can join the replica set as secondary node again.

Commands	Description
rs.initiate()	To initiate a new replica set
rs.conf ()	To check the replica set configuration
rs.status ()	To check the status of a replica set
rs.add ()	To add members to a replica set

S

Figure shows a replicated dataset after creating three secondary membersfrom a primary member.

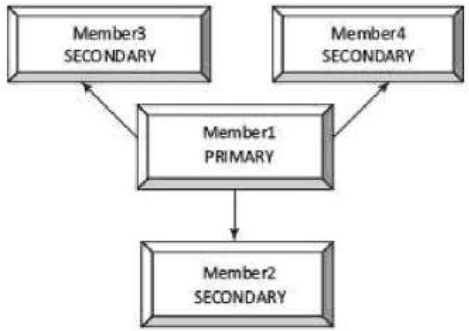


Figure 3.13 Replicated set on creating secondary members

Auto-sharding :Sharding is a method for distributing data across multiple machines in a distributed application environment. MongoDB uses sharding to provide services to Big Data

applications.

A single machine may not be adequate to store the data. When the data size increases, do not provide data retrieval operation. Vertical scaling by increasing the resources of a single machine is quite expensive. Thus, horizontal scaling of the data can be achieved using sharding mechanism where more database servers can be added to support data growth and the demands of more read and write operations.

Sharding automatically balances the data and load across various servers. Sharding provides additional write capability by distributing the write load over a number of mongod (MongoDB Server) instances.

Type	Description
Double	Represents a float value.
String	UTF-8 format string.
Object	Represents an embedded document.
Array	Sets or lists of values.
Binary data	String of arbitrary bytes to store images, binaries.
Object id	Objectlds (MongoDB document identifier, equivalent to a primary key) are: small, likely unique, fast to generate, and ordered. The value consists of 12-bytes, where the first four bytes are for timestamp that reflects the instance when Objectld creates.
Boolean	Represents logical true or false value.
Date	BSON Date is a 64-bit integer that represents the number of milliseconds since the Unix epoch Oan 1, 1970).
Null	Represents a null value. A value which is missing or unknown is Null.
Regular Expression	RegExp maps directly to aJavaScript RegExp
32-bit integer	Numbers without decimal points save and return as 32-bit integers.
Timestamp	A special timestamp type for internal MongoDB use and is not associated with the regular date type. Timestamp values are a 64-bit value, where first 32 bits are time, t (seconds since the Unix epoch), and next 32 bits are an incrementing ordinal for operations within a given second.
64-bit integer	Number without a decimal point save and return as 64-bit integer .

Min key	MinKey compare less than all other possible BSON element values, respectively, and exist primarily for internal use.
Max key	MaxKey compares greater than all other possible BSON element values, respectively, and exist primarily for internal use.

Data Types which Mango DB document Supports

Rich Queries and Other DB Functionalities MongoDB offers a rich set of features and functionality compared to those offered in simple key-value stores. They can be comparable to those offered by any RDBMS. MongoDB has a complete query language, highly-functional secondary indexes (including text search and geospatial), and a powerful aggregation framework for data analysis. MongoDB provides functionalities comparison of features.

Features	RDBMS	MongoDB
Rich Data Model	No	Yes
Dynamic Schema	No	Yes
Typed Data	Yes	Yes
Data Locality	No	Yes
Field Updates	Yes	Yes
Complex Transactions	Yes	No
Auditing	Yes	Yes
Horizontal Scaling	No	Yes

Comparison of features MongoDB with respect to RDBMS

Command	Functionality
Mongo	Starts MongoDB; (*mongo is MongoDB client). The defaultdatabase **in** MongoDB is test.
db.help()	Runs help. This displays the list of all the commands.
db.stats()	Gets statistics about MongoDB server.

Use <database name)	Creates database
Db	Outputs the names of existing database, if created earlier
Dbs	Gets list of all the databases
db.dropDatabase ()	Drops a database
db.database name.insert ()	Creates a collection using insert ()
db.<database name>. find()	Views all documents in a collection
db.<database name>.update ()	Updates a document
db.<database name>.remove ()	Deletes a document

MongoDB querying commands

Following explains the sample usages of the commands:

To Create database Command use - use command creates a database; For example, Command use lego creates a database named lego. (A sample database is created to demonstrate subsequent queries. The Lego is an international toy brand). Default database in MongoDB is test.

To see the existence of database Command db - db command shows that lego database is created.

To get list of all the databases Command show dbs - This command shows the names of all the databases.

To drop database Command db. dropDatabase () - This command drops a database. Run use lego command before the db. dropDatabase () command to drop lego Database. If no database is selected, the default database test will be dropped.

To create a collection Command insert () -Tocreate a collection, the easiest way is to insert a record (a document consisting of keys (Field names) and Values) into a collection. A new collection will be created, if the collection does not exist. The following statements demonstrate the creation of a collection with three fields (ProductCategory, Productld and ProductName) in the lego:


```
db.lego.insert
(
        {
                "ProductCategory": "Airplane",
                "ProductId": 10725,
                "ProductName": "Lost Temple"
        }
)
```

To view all documents in a collection Command db. <database name>. find ()-Find command is equivalent to select query of RDBMS. Thus, "Select * from lego" can be written as db. lego. find () in MongoDB. MongoDB created unique objecteld ("_id") on its own. This is the primary key of the collection. Command db. <database name>. find() .pretty() gives a prettier look.

To update a document Command db. <database name>. update ()-Update command is used to change the field value. By default, multi attribute is false. If {multi: true} is not written then it will update only the first document.

To delete a document Command db. <database name>. remove () - Remove command is used to delete the document. The query db. <database name>. remove (("ProdctID": 10725)) removes the document whose productld is 10725.

To add array in a collection Command insert () - Insert command can also be used to insert multiple documents into a collection at one time.

```
db.lego.insert
(
        [
                {
                        "ProductCategory": "Airplane",
                        "ProductId": 10725,
                        "ProductName": "Lost Temple"
                },
                {
                        "ProductCategory": "Airplane",
                        "ProductId": 31047,
                        "ProductName": "Propeller Plane"
                },
                {
                        "ProductCategory": "Airplane",
                        "ProductId": 31049,
                        "ProductName": "Twin Spin Helicopter"
                }
        ]
)
```

CASSANDRA DATA BASE

Cassandra was developed by Facebook and released by Apache. Cassandra was named after

Trojan mythological prophet Cassandra, who had classical allusions to a curse on oracle. Later on, IBM also released the enhancement of Cassandra, as open

source version. The open source version includes an IBM Data Engine which processes No SQL data store. The engine has improved throughput when workload of read-operations is intensive.

Cassandra is basically a column family database that stores and handles massive data of any format including structured, semi-structured and unstructured data.

Apache Cassandra DBMS contains a set of programs. They create and manage databases. Cassandra provides functions (commands) for querying the data and accessing the required information. Functions do the viewing, querying and changing (update, insert or append or delete), visualizing and perform transactions on the DB.

Apache Cassandra has the distributed design of Dynamo. Cassandra is written in Java. Big organizations, such as Facebook, IBM, Twitter, Cisco, Rackspace, eBay, Twitter and Netflix have adopted Cassandra.

Characteristics of Cassandra are (i) open source, (ii) scalable (iii) non- relational (v) NoSQL (iv) Distributed (vi) column based, (vii) decentralized, (viii) fault tolerant and (ix) tuneable consistency.

Features of Cassandra are as follows:

1. Maximizes the number of writes - writes are not very costly (time consuming)
2. Maximizes data duplication
3. Does not support Joins, group by, OR clause and aggregations
4. Uses Classes consisting of ordered keys and semi-structured data storage systems
5. Is fast and easily scalable with write operations spread across the cluster. The cluster does not have a master-node, so any read and write can be handled by any node in the cluster.
6. Is a distributed DBMS designed for handling a high volume of structured data across multiple cloud servers

Has peer-to-peer distribution in the system across its nodes, and the data is distributed among all the nodes in a cluster.

Data Replication Cassandra stores data on multiple nodes (data replication) and thus has no single point of failure, and ensures availability, a requirement in CAP theorem. Data replication uses a replication strategy. Replication factor determines the total number of

replicas placed on different nodes. Cassandra returns the most recent value of the data to the client. If it has detected that some of the nodes responded with a stale value, Cassandra performs a read repair in the background to update the stale values.

Components at Cassandra Table 3.13 gives the components at Cassandra and their description

Component	Description
Node	Place where data stores for processing
Data Center	Collection of many related nodes
Cluster	Collection of many data centers
Commit log	Used for crash recovery; each write operation written to commit log
Mem-table	Memory resident data structure, after data written in commit log, data write in mem-table temporarily
SSTable	When mem-table reaches a certain threshold, data flush into an SSTable disk file
Bloom filter	Fast and memory-efficient, probabilistic-data structure to find whether an element is present in a set, Bloom filters are accessed after every query.

Scalability Cassandra provides linear scalability which increases the throughput and decreases the response time on increase in the number of nodes at cluster.

Transaction Support Supports ACID properties (Atomicity, Consistency, Isolation, and Durability).

Replication Option Specifies any of the two replica placement strategy names. The strategy names are Simple Strategy or Network Topology Strategy. The replica placement strategies are:

Simple Strategy: Specifies simply a replication factor for the cluster.

Network Topology Strategy: Allows setting the replication factor for each data center independently.

Table 3.14 Data types built into Cassandra, their usage and description

CQL Type	Description
ascii	US-ASCII character string
bigint	64-bit signed long integer
blob	Arbitrary bytes (no validation), BLOB expressed in hexadecimal
boolean	True or false
counter	Distributed counter value (64-bit long)

decimal	Variable-precision decimal integer, float
double	64-bit IEEE-754 *double precession* floating point integer, float
float	32-bit IEEE-754 single precession floating point integer, float
inet	IP address string in 1Pv4 or 1Pv6 format, used by the python-cql driver and CQL native protocols
int	32-bit signed integer
list	A collection of one or more ordered elements
map	AJSON-style array of literals: {literal: literal, literal: literal ...}
set	A collection of one or more elements
text	UTF-8 encoded string
timestamp	Date plus time, encoded as 8 bytes since epoch integers, strings
varchar	UTF-8 encoded string
varint	Arbitrary-precision integer

Cassadra Data Model Cassandra Data model is based on Google's BigTable Each value maps with two strings (row key, column key) and timestamp, similar to HBase.The database can be considered as a sparse distributed multi-dimensional sorted map. Google file system splits the table into multiple tablets (segments of the table) along a row. Each tablet, called METAl tablet, maximum size is 200 MB, above which a compression algorithm used. META0 is the master-server. Querying by META0 server retrieves a METAl tablet. During execution of the application, caching of locations of tablets reduces the number of queries.

Cassandra Data Model consists of four main components: (i) Cluster: Made up of multiple nodes and keyspaces, (ii} Keyspace: a namespace to group multiple column families, especially one per partition,

Column: consists of a column name, value and timestamp and (iv) Column- family: multiple columns with row key reference. Cassandra does keyspace management using partitioning of keys into ranges and assigning different key- ranges to specific nodes.

Following Commands prints a description (typically a series of DDL statements) of a schema element or the cluster:

DESCRIBE CLUSTER

DESCRIBE SCHEMA

DESCRIBE KEYSPACES

DESCRIBE KEYSPACE <keyspace name>

DESCRIBE TABLES

DESCRIBE TABLE <table name>

DESCRIBE INDEX <index name>

DESCRIBE MATERIALIZED VIEW <view name>DESCRIBE

TYPES

DESCRIBE TYPE <type name>

DESCRIBE FUNCTIONS

DESCRIBE FUNCTION <function name>DESCRIBE

AGGREGATES

DESCRIBE AGGREGATE <aggregate function name>

Consistency Command CONSISTENCY shows the current consistency level. CONSISTENCY <LEVEL> sets a new consistency level. Valid consistency levels are ANY, ONE, TWO, THREE,QUORUM, LOCAL_ONE, LOCAL_QUORUM, EACH_QUORUM, SERIAL AND LOCAL_SERIAL. Following are their meanings:

1. ALL: Highly consistent. A write must be written to commitlog andmemtable on all replica nodes in the cluster.
2. EACH_QUORUM: A write must be written to commitlog and memtable onquorum of replica nodes in all data centers.
3. LOCAL_QUORUM: A write must be written to commitlog and memtable onquorum of replica nodes in the same center.
4. ONE: A write must be written to commitlog and memtable of at least onereplica node.
5. TWO, THREE: Same as One but at least two and three replica nodes, respectively.
6. LOCAL_ONE: A write must be written for at least one replica node in the local data center.
7. ANY: A write must be written to at least one node.
8. SERIAL: Linearizable consistency to prevent unconditional update.
9. LOCAL_SERIAL: Same as Serial but restricted to the local data center.

Keyspaces A keyspace (or key space) in a NoSQL data store is an object that contains all column families of a design as a bundle. Keyspace is the outermostgrouping of the data in the data store. It is similar to relational database. Generally, there is one keyspace per application. Keyspace in Cassandra is a namespace that defines data replication on nodes. A cluster contains one keyspace per node.

Create Keyspace Command CREATE KEYSPACE <Keyspace Name> WITH replication = {'class': '<Strategy name>', 'replication_factor': '<No. of replicas>'}AND durable_writes= '<TRUE/FALSE>';

CREATE KEYSPACE statement has attributes replication with option class and replication factor, and durable_write.

Default value of *durable_ writes* properties of a table is set to true. That commands the Cassandra to use Commit Log for updates on the current Keyspace true or false. The option is not compulsory.

1. ALTER KEYSPACE command changes (alter) properties, such as the number of replicas and the durable_writes of a keyspace: ALTER KEYSPACE <Keyspace Name> WITH replication = {'class': '<Strategy name>', 'replication_factor': '<No. of replicas>'};
2. DESCRIBE KEYSPACE command displays the existing keyspaces.
3. DROP KEYSPACE command drops a keyspace:
4. Re-executing the drop command to drop the same keyspace will result in configuration exception.
5. Use KEYSPACE command connects the client session with a keyspace.

Command	Functionality
CQLSH	A command line shell for interacting with Cassandra throughCQL
HELP	Runs help. This displays the list of all the commands
CONSISTENCY	Shows the current consistency level
EXIT	Terminate the CQL shell
SHOW HOST	Displays the host

SHOW VERSION	Displays the details of current cqlsh session such as host,Cassandra version, or data type assumptions
CREATE KEYSPACE <Keyspace Name>	Creates keyspace with a name
DESCRIBE KEYSPACE <Keyspace Name>	Displays the keyspace with a name
ALTER KEYSPACE <Keyspace Name>	Modifies keyspace with a name
DROP KEYSPACE <Keyspace Name>	Deletes keyspace with a name
CREATE (TABLE | COLUMNFAMILY)	Creates a table or column family
COLLECTIONS	Lists the Collections

CQL commands and their functionalities

Give the examples of usages of various CQL commands.

SOLUTION

(1) Create Table Command: CREATE TABLE command creates a table in the current keyspace:

CREATE (TABLE COLUMNFAMILY) <tablename> ('<column-definition>', '<column-definition>')(WITH <option> AND <option>);

***Primary key* is a column used to uniquely identify a row. Therefore, defining a primary key is compulsory while creating a table. A primary key is made of one or more columns of a table.**

Example:* Create a table *Productinfo* in the keyspace *lego,* with primary key field *Productid.

Use lego;

Create table Productinfo(Productid int primarykey, ProductType text);

(2) Describe Tables Command: DESCRIBE TABLE Command displays all the tables in the current keyspace:

DESCRIBE TABLE <TABLE NAME>;

***Example:* Display the details of a table Productinfo:**

DESCRIBE TABLE Productinfo;

(3) Alter Tables Command:

ALTER TABLE Command ALTER (TABLE COLUMNFAMILY) <tablename> (ADD I DROP) <column name>

(4) Cassandra CURD Operations: (CURD-Create, Update, Read and Delete data into tables) :

(a) Insert Command:

INSERT INTO <tablename> (<columnl name>, <column2name>....) VALUES (<valuel>, <value2>....) USING

<option>

(a) Update Command:

UPDATE command updates data in a table. The following keywords are used while updating data in a table:

Where - This clause is used to select the row to be updated.

Set - Set the value using this keyword.

Must- Includes all the columns composing the primary key.

If a given row is unavailable, then UPDATE creates a new row.

UPDATE <tablename> SET <column name>= <new value> <column name>= <value>.... WHERE <condition>

(a) Select Command

SELECT command reads the data from a table. The command can read a whole table, a single column, or a particular cell:

SELECT <column name(s)> FROM <Table Name>

To select all records:

SELECT* FROM <Table Name>

To select records that fulfils required condition:

SELECT <columnl, column2,..> FROM <Table Name>where <Condition>

(b) **Delete Command**

DELETE command deletes data from a table:

DELETE FROM <identifier> WHERE <condition>; ***Example:*** **Delete row from a table where Product id is 31047:** DELETE FROM Productinfo WHERE Productid = 31047;

(5) Creating a Table with List

CREATE Table command is used for creating a table with a list.

The following query creates a table with two columns, one is the primary key and the other has multiple items (List):

CREATE TABLE data (<column name>, <data type>PRIMARY KEY, <column name list<data type>);

Example : **Create a sample table *ContactInfo* with three columns: *Sno, name* and *Emailld.* To store multiple Email Ids, use a list:**

create table Contactinfo (Sno int Primary key,Name text, emailid list <text>);

(6) Update Command for updating Data into a List

UPDATE command also updates data into a list:

UPDATE <table Name> SET <New data> where <condition>.

Example : Add one more email Id to the *emailld* list in *ContactInfo* table :

UPDATE Contactinfo SET emailid = emailid + ['preeti@ymail.com'] where SNo=l.