Module 3
NoSQL Big Data Management, MongoDB and Cassandra

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.

Features of distributed-computing architecture

- **Increased reliability and fault tolerance**: If a segment of machines in a cluster fails then the rest of the machines continue work.
- **Flexibility** makes it very easy to install, implement and debug new services in a distributed environment.
- **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.
- **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).
- **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.
- **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).

The **demerits o**f distributed computing are: (i) issues in troubleshooting in a larger networking infrastructure, (ii) additional software requirements and (iii) security risks for data and resources.

Big Data solutions require a scalable distributed computing model with shared-nothing architecture. A solution is Big Data store in HDFS files.

NoSQL data also store Big Data, and facilitate random read/write accesses. The accesses are sequential in HDFS data. HBase is a NoSQL solution .Examples of other solutions are MongoDB and Cassandra. MongoDB and Cassandra DBMSs create HDFS compatible distributed data stores and include their specific query processing languages.

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 RDBMSs. 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.
- Transactions on SQL databases exhibit ACID properties. ACID stands for atomicity, consistency, isolation and durability. 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.

Triggers, Views and Schedules in SQL Databases

- Trigger is a special stored procedure. Trigger executes when a specific action(s) occurs within a database, such as change in table data or actions such as UPDATE, INSERT and DELETE. For example, a Trigger store procedure inserts new columns in the columnar family data store.
- **View** refers to a logical construct, used in query statements. A View saves a division of complex query instructions and that reduces the query complexity. Viewing of a division is similar to a view of a table.
- **Schedul**e refers to a chronological sequence of instructions which execute concurrently. When a transaction is in the schedule then all instructions of the transaction are included in the schedule. Scheduled order of instructions is maintained during the transaction. Scheduling enables execution of multiple transactions in allotted time intervals.

Join in SQL Databases

- SQL databases facilitate combining rows from two or more tables, based on the related columns in them. Combining action uses Join function during a database transaction. Join refers to a clause which combines. Combining the products (AND operations) follows next the selection process.
- A Join operation does pairing of two tuples obtained from different relational expressions. Joins, if and only if a given Join condition satisfies. Number of Join operations specify using relational algebraic expressions. SQL provides JOIN clause, which retrieves and joins the related data stored across multiple tables with a single command, Join.
- Relational databases and RDBMS developed using SQL have issues of scalability and distributed design. This is because all tuples need to be on the same data node. The database has an issue of indexing over distributed nodes. They do not model the hierarchical, object-oriented, semi-structured or graph databases. Database Tables have relationships between them which are represented by related fields. RDBMS allows the Join operations on the related columns. The traditional RDBMS has a problem when storing the records beyond a certain physical storage limit. This is because RDBMS does not support horizontal scalability .For example, consider sharding a big table in a DBMS into two. Assume writing first 0.1 million records (1 to 100000) in one table and from 100001 in another table. Handling of the Joins and managing data in the other related tables are cumbersome processes, when using the sharding. The problem continues when data has no defined number of fields and formats. For example, the data associated with the choice of chocolate flavours of the users of ACVM in Example 1.6(i). Some users provide a single choice, while some users provide two choices, and a few others want to fill three best flavours of their choice. Defining a field becomes tough when a field in the database offers choice between two or many. This makes RDBMS unsuitable for data management in Big Data environments as well as data in their real forms.

- SQL compliant format means that database tables constructed using SQL and they enable processing of the queries written using SQL. ‘NoSQL’ term conveys two different meanings: (i) does not follow SQL compliant formats, (ii)“Not only SQL” use SQL compliant formats with variety of other querying and access methods.

➢ 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.
NoSQL data stores are considered as **semi-structured data.**
Characteristics are as follows:

- 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 JSON (CouchDB), JSON (PNUTS), JSON (MongoDB), Graph Store, Object Store, ordered keys and semi-structured data storage systems.
- 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 :
(i) Relax one or more of the ACID properties.
(ii) Characterize by two out of three properties (consistency, availability and partitions) of CAP theorem, two are at least present for the application/service/process.
(iii) 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.

NoSQL Data store	Description
Apache's HBase	HDFS compatible, open-source and non-relational data store written in Java; 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 with JSON-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 the JSON 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.

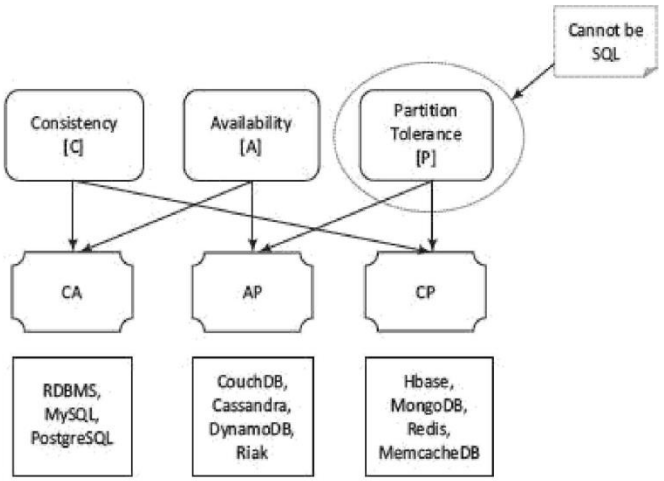
Consistency in distributed databases means that all nodes observe the same data at the same time. Therefore, the operations in one partition of the database should reflect in other related partitions in case of distributed database. Operations, which change the sales data from a specific showroom in a table should also reflect in changes in related tables which are using that sales data.
Availability means that during the transactions, the field values must be available in other partitions of the database so that each request receives a 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.
Partition means division of a large database into different databases without affecting the operations on them by adopting specified procedures.

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 (Consistency, Availability and Partition Tolerance) theorem demonstrates that any distributed system cannot guarantee C, A and P together. Consistency− All nodes observe the same data at the same time. Availability– Each request receives a response on success/failure. 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 can be:

- 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 and availability, 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.



➢ **Schema-less Models**

- 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.

- NoSQL data model offers relaxation in one or more of the ACID properties (Atomicity, consistence, isolation and durability) of the database. Distribution follows CAP theorem. CAP theorem states that out of the three properties, two must at least be present for the application/service/process.
- Figure 3.2 shows characteristics of Schema-less model for data stores. ER stands for entity-relation modelling. Relations in a database build the connections between various tables of data
- NoSQL data stores use non-mathematical relations but store this information as an aggregate called metadata. **Metadata** refers to data describing and specifying an object or objects. Metadata is a record with all the information about a particular dataset and the inter-linkages. Metadata helps in selecting an object, specifications of the data and, usages that design where and when. Metadata specifies access permissions, attributes of the objects and enables additions of an attribute layer to the objects. Files, tables, documents and images are also the objects.

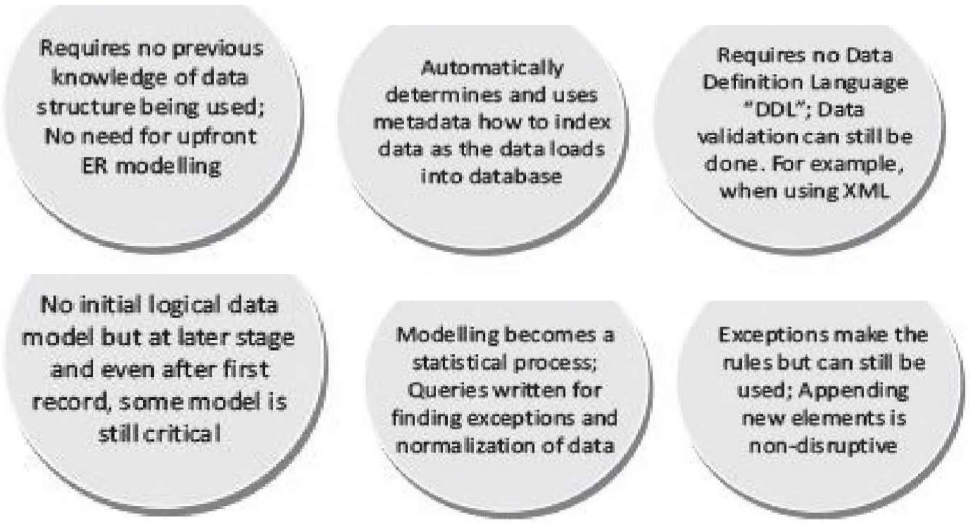


Figure 3.2 Characteristics of Schema-less model

➢ **Increasing Flexibility for Data Manipulation**

- BASE is a flexible model for NoSQL data stores. Provisions of BASE increase flexibility.
- BASE Properties :BA stands for basic availability,S stands for soft state and E stands for eventual consistency.
- **Basic availability** ensures by distribution of shards across many data nodes with a high degree of replication.
- **Soft state** ensures processing even in the presence of inconsistencies but achieving consistency eventually.
- **Eventual consistency** means consistency requirement in NoSQL databases meeting at some point of time in future..

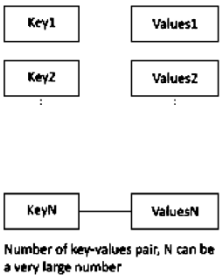
BASE model is not necessarily appropriate in all cases but it is flexible and is an alternative to SQL-like adherence to ACID properties.

3.3 NoSQL data architecture patterns

NoSQL data stores broadly categorize into architectural patterns described in the following subsections:

3.3.1 Key-Value Store

- Simplest way to implement.
- 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.
- The concept is similar to a hash table where a unique key points to a particular item(s) of data.



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

Figure 3.4 Example of key-value pairs in data architectural pattern

➢ **Advantages of a key-value store are as follows:**

- 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.
- A query just requests the values and returns the values as a single item.
- Values can be of any data type.
- Key-value store is eventually consistent.
- Key-value data store may be hierarchical or may be ordered key-value store.
- Returned values on queries can be used to convert into lists, table-columns, data-frame fields and columns. Have (i) scalability, (ii) reliability, (iii) portability and (iv) low operational cost.
- The key can be synthetic or auto-generated.

➢ The key-value store provides client to read and write values using a key as follows:

(i) Get(key), returns the value associated with the key.
(ii) Put (key, value), associates the value with the key and updates a value if this key is already present.
(iii) Multi-get (key1, key2, .., keyN), returns the list of values associated with the list of keys.
(iv) Delete(key), removes a key and its value from the data store.

➢ 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

➢ **Limitations of key-value store architectural pattern are:**

(i) No indexes are maintained on values, thus a subset of values is not searchable.
(ii) 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.
(iii) 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.
(iv) Queries cannot be performed on individual values. No clause like 'where' in a relational database usable that filters a result set.

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:

- ➢ Document stores unstructured data.
- ➢ Storage has similarity with object store.

- Data stores in nested hierarchies. For example, in JSON formats data model ,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.
- Querying is easy. For example, using section number, sub-section number and figure caption and table headings to retrieve document partitions.
- No object relational mapping enables easy search by following paths from the root of document tree.
- 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.

The demerits in Document Store are incompatibility with SQL and complexity for implementation. 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.

XML (eXtensible Markup Language) is 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 data exchanges over the network. An XML document is semi-structured

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 documents use to transform. CouchDB queries the indices using a web browser. CouchDB accesses the documents using HTTP API. HTTP methods are Get, Put and Delete .
- CouchDB data replication is the distribution model that results in fault tolerance and reliability.

Document JSON Format—MongoDB Database MongoDB Document database provides a rich query language and constructs, such as database indexes allowing easier handling of Big Data.

Example of Document in Document Store:

```
{
    "id": "1001"
    "Student Name":
{
"First": "Ashish",
"Middle": "Kumar",
"Last": "Rai"
    }
"Category": "Student",
"Class": "B.Tech.",
"Semester": "VII",
"Branch": "Computer Engineering",
"Mobile": "12345"
}
```

The document store allows querying the data based on the contents as well. For example, it is possible to search the document where student's first name is "Ashish". Document store can also provide the search value's exact location. The search is by using the document path. A type of key accesses the leaf values in the tree structure. Since the document stores are schema-less, adding fields to documents (XML or JSON) becomes a simple task.

Document Architecture Pattern and Discovering Hierarchical Structure

Following is example of an XML document in which a hierarchical structure discovers later. Figure 3.5 shows an XML document architecture pattern in a document fragment and document tree structure. Figure 3.5 XML document architecture pattern The document store follows a tree-like structure (similar to directory structure in file system). Beneath the root element there are multiple branches. Each branch has a related path expression that provides a way to navigate from the root to any given branch, sub-branch or value.

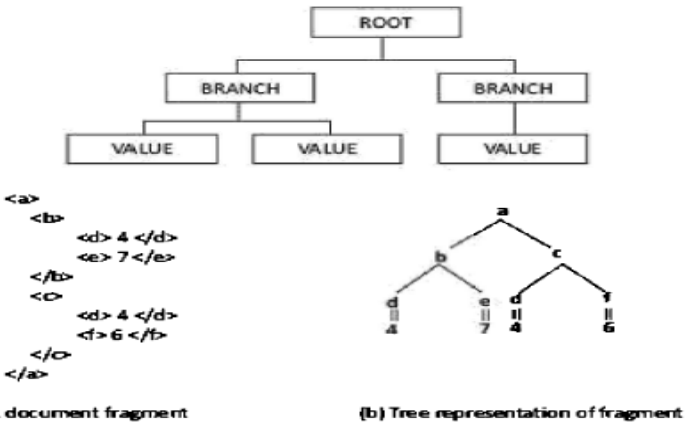


Figure 3.5 XML document architecture pattern

XQuery and XPath are query languages for finding and extracting elements and attributes from XML documents. The query commands use sub-trees and attributes of documents. The querying is similar as in SQL for databases. XPath treats XML document as a tree of nodes. XPath queries are expressed in the form of XPath expressions.

XML and JSON both are designed to form a simple and standard way of describing different kinds of hierarchical data structures. They are popularly used for storing and exchanging data. The following example explains the concept of Document Store in JSON and XML for hierarchical records.

EXAMPLE 3.5

Give the structures of XML and JSON document fragments for a student record.

SOLUTION

Following are the structures:

```
{
    "students": [
    {
        "name": "Ashish Jain",
        "rollNo": "12345"
    },
    {
        "name": "Sandeep Joshi",
        "rollNo": "12346"
    }
    ]
}
```

(a) JSON

```
<students>
    <student>
        <name>Ashish Jain</name>
        <rollNo>12345</rollNo>
    </student>
    <student>
        <name>Sandeep Joshi</name>
        <rollNo>12346</rollNo>
    </student>
</students>
```

(b) XML equivalent

When compared with XML, JSON has the following advantages:

- XML is easier to understand but XML is more verbose than JSON.
- XML is used to describe structured data and does not include arrays, whereas JSON includes arrays.
- JSON has basically key-value pairs and is easier to parse from JavaScript.

3.3.3 Tabular Data:

Tabular data stores use rows and columns. Row-head field may be used as a key which access and retrieves multiple values from the successive columns in that row. The OLTP is fast on in-memory row-format data. Oracle DBs provide both options: columnar and row format storages. Generally, relational DB store is in-memory row-based data

3.3.3.1 Column Family Store

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. All values in successive fields in a column consisting of multiple rows save at consecutive memory addresses. This enables fast accesses during in-memory analytics, which includes CPU accesses and analyses using memory addresses in which values are cached from the disk before processing. The OLAP (on-line AP) is also fast on in-memory column-format data. An application uses a combination of row head and a column head as a key for access to the value saved at the field.

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 of a column. A column-family head is also called a super-column head. Examples of columnar family data stores are HBase, BigTable, HyperTable and Cassandra.

Columns Families -Two of more columns in data-store group into one column family. Table 3.3 considered two families.

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.

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.

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:

- 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 Example 1.6(i). Number of processing instructions is proportional to the number of ACVMs due to scalable operations.
- 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.
- 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.
- 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.
- 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.

- 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.
- Replication of columns: HDFS-compatible column-family data stores replicate each data store with default replication factor = 3.
- No optimization for Join: Column-family data stores are similar to sparse matrix data. The data do not optimize for Join operations. Column-family data store in a format in which store set of column family field-values which are not empty (null or zero). Metadata of the matrix consists of hash keys that reference each set distinctly.

Typical uses of column store are: (i) web crawling, (ii) large sparsely populated tables and (iii) system that has high variance. HDFS is highly reliable for very long running queries.

3.3.3.2 BigTable 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

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.

3.3.3.3 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. A table such as that shown in Example 3.6 can be partitioned into row groups. Values at each column of a row group store as the RC record. The RC file records store data of a column in the row group (Serializability means query or transaction executable by series of instructions such that execution ensures correct results).

3.3.3.4 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 an HDFS 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.
- Lightweight indexing is an ORC feature. Those blocks of rows which do not match a query skip as they do not map on using indices data at metadata. Each index includes the aggregated values of minimum, maximum, sum and count using aggregation functions on the content columns. Therefore, contents-column key for accessing the contents from a column consists of combination of row-group key, column mapping key, min, max, count (number) of column fields of the contents column. Table 3.5 gives the keys used to access or skip a contents column during querying. The keys are Stripe_ID, Index-column key, and contents-column name, min, max and count. Table 3.5 Keys to access or skip a content column in ORC file format

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	
	...	...	...	...	
	...	...	...	...	

3.3.3.5 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.
- An ORC array <int> has two columns, one for array size and the other for contents. Parquet format file does not consist of extra column per nesting level. Similarly, ORC has two columns, one is for each Map, List size, min, max and the second is for the contents. Parquet format file does not consist of extra column per nesting level, just one column per leaf in the schema.
- [Parquet in English means 'a floor covering made of small rectangular wooden blocks (tiles) fitted together in a pattern. Similarly, Parquet objects have pages as the tiles. Pages build a column chunk. Column chunks build a row group. Row groups build the table. A page is like a tile consisting of column fields. The values read or write at an instance or used for encoding or compression. The values are not read separately from a page.] Table 3.6 gives the keys used to access or skip the contents page. Three keys are: (i) row-group_ID, (ii) column-chunk key and (iii) page key. Table 3.6 Combination of keys for content page in the Parquet file format

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'
	..	...	...	...
	..	...	...	...

Table 3.6 Combination of keys for content page in the Parquet file format

3.3.4 Object Data Store

An object store refers to a repository which stores the:

- Objects (such as files, images, documents, folders, and business reports)
- 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)
- Custom metadata which provides information, such as subject, category, sharing permissions.

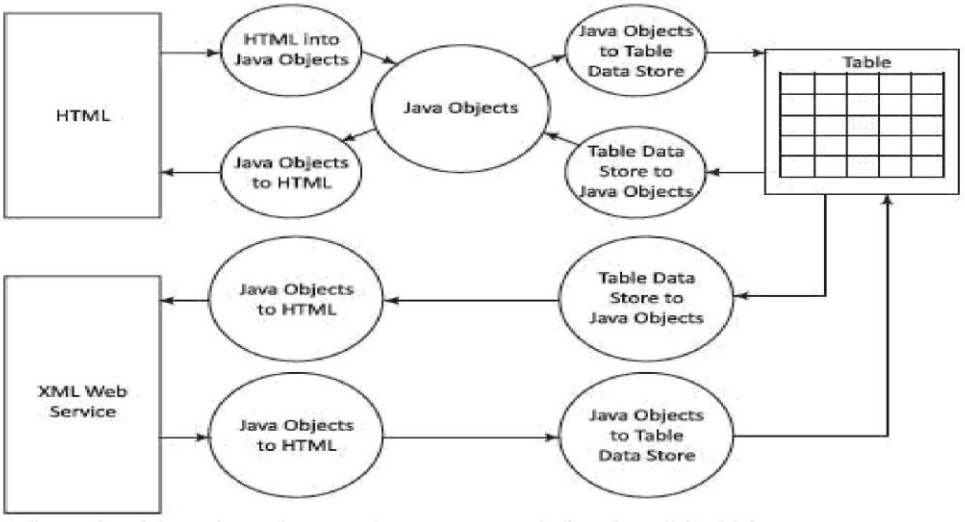
Metadata enables the gathering of metrics of objects, searches, finds the contents and specifies the objects in an object data-store tree. Metadata finds the relationships among the objects, maps the object relations and trends. Object Store metadata interfaces with the Big Data. API first mines the metadata to 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. Data definition and manipulation, DB schema design, database browsing, DB administration, application compilation and debugging use a programming language.

Eleven Functions Supporting APIs: An Object data store consists of functions supporting APIs for: (i) scalability, (ii) indexing, (iii) large collections, (iv) querying language, processing and optimization (s), (v) Transactions, (vi) data replication for high availability, data distribution model, data integration (such as with relational database, XML, custom code), (vii) schema evolution, (viii) persistency, (ix) persistent object life cycle, (x) adding modules and (xi) locking and caching strategy. Object Store may support versioning for collaboration.

Object Store can be created using IBM 'Content Platform Engine'. Creation needs installing and configuring the engine (Engine is software which drives forward.). Console of the engine makes creation of process easy. Amazon S3 and Microsoft Azure BLOB support the Object Store.

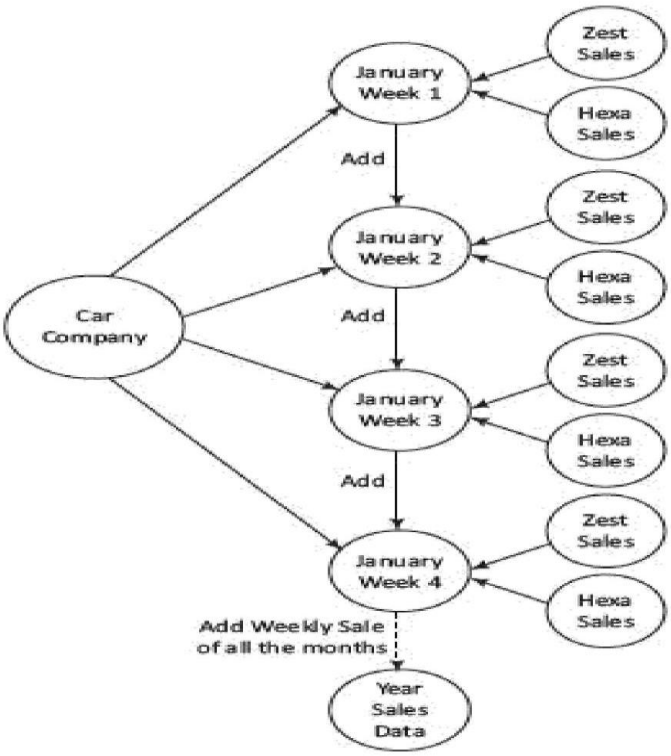
Amazon S3 (Simple Storage Service) S3 refers to Amazon web service on the cloud named S3. The S3 provides the Object Store. The Object Store differs from the block and file-based cloud storage. Objects along with their metadata store for each object store as the files. S3 assigns an ID number for each stored object. The service has two storage classes: Standard and infrequent access. Interfaces for S3 service are REST, SOAP and Bit Torrent. S3 uses include web hosting, image hosting and storage for backup systems. S3 is scalable storage infrastructure, same as used in Amazon e-commerce service. S3 may store trillions of objects.

3.3.4.1 Object Relational Mapping : The following example explains object relational mapping.
Example : How does an HTML object and XML based web service relate with tabular data stores?
Solution: Figure 3.7 shows the object relational mapping of HTML document and XML web services store with a tabular data store.



3.3.5 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. Data Store focuses on modeling interconnected structure of data. Data stores based on graph theory relation G = (E, V), where E is set of edges e1, e2, … and V is set of vertices, v1, v2, …, vn. 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. The following example explains the graph database application in describing entities relationships and relationship types
- **Example 3.11** Let us assume a car company represents a node entity, which has two connected nodes comprising two model entities, namely Hexa and Zest. Draw graph with directed lines, joining the car company with two entities. (i) How do four directed lines relate to four weeks and two directed lines? One directed line corresponds to a car model. Only directed line corresponds to weekly total sales. (ii) How will the yearly sales compute? (iii) Show the path traversals for computations exhibit BASE properties. **Solution** (i) Figure 3.8 shows section of a graph database for the sales of two car models.



- Figure 3.8 Section of the graph database for car-model sales (ii) The yearly sales compute by path traversals from nodes for weekly sales to yearly sales data. (iv) The path traversals exhibit BASE properties because during the intermediate paths, consistency is not maintained. Eventually when all the path traversals complete, the data becomes consistent.

  Graph databases enable fast network searches. Graph uses linked datasets, such as social media data. Data store uses graphs with nodes and edges connecting each other through relations, associations and properties. Querying for data uses graph traversal along the paths. Traversal may use single-step, path expressions or full recursion. A

relationship represents key. A node possesses property including ID. An edge may have a label which may specify a role.

Characteristics of graph databases are:

- Use specialized query languages, such as RDF uses SPARQL
- Create a database system which models the data in a completely different way than the key-values, document, columnar and object data store models.
- Can have hyper-edges. A hyper-edge is a set of vertices of a hypergraph. A hypergraph is a generalization of a graph in which an edge can join any number of vertices (not only the neighbouring vertices).
- Consists of a collection of small data size records, which have complex interactions between graph-nodes and hypergraph nodes.

When a new relationship adds in RDBMS, then the schema changes. The data need transfer from one field to another. The task of adding relations in graph database is simpler. The nodes assign internal identifiers to the nodes and use these identifiers to join the network. Traversing the joins or relationships is fast in graph databases. It is due to the simpler form of graph nodes. The graph data may be kept in RAM only. The relationship between nodes is consistent in a graph store. 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 the graph. 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 records (Sections 9.4 and 9.5). Rules and inference are used to run queries on complex structures such as class libraries, taxonomies and rule-based systems.

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. Spark provides a simple and expressive programming model that includes supports to a wide range of applications, including graph computation.

3.3.6 Variations of NoSQL Architectural Patterns

Six data architectures are SQL-table, key-value pairs, in-memory column-family, document, graph and object. Selected architecture may need variations due to business requirements. Business requirements are ease of using an architecture and long-term competitive advantage.

Kelly-McCreary, co-founder of 'NoSQL Now' suggested that when selecting a NoSQL-pattern, the pattern may need change and require variation to another pattern(s). Some reasons for this are:

- Focus changing from performance to scalability
- Changing from modifiability to agility
- Greater emphasis on Big Data, affording capacity, availability of support, ability for searching and monitoring the actions

Steps for selecting a NoSQL data architectural pattern can be as follows:

- Select an architecture
- Perform a use-case driven difficulty analysis for each of the six architectural patterns. Difficulties may be low, medium or high in the following processes: (i) ingestion, (ii) validation of structure and its fields, (iii) updating process using batch or record by record approach, (iv) searching process using full text or by changing the sorting order, and (v) export the reports or application results in HTML, XML or JSON.
- Estimate the total efforts for each architecture for all business requirements. Process the choice of architecture using trade-off. For example, between the MongoDB document data store and Cassandra column-family data store

3.4 NoSQL to manage Big Data

NoSQL

(i) limits the support for Join queries, supports sparse matrix like columnar-family,
(ii) Characteristics of easy creation and high processing speed, scalability and storability of much higher magnitude of data (terabytes and petabytes). NoSQL sacrifices the support of ACID properties, and instead supports CAP and BASE properties . NoSQL data processing scales horizontally as well vertically

NoSQL Solutions for Big Data: Big Data solution needs scalable storage of terabytes and petabytes, dropping of support 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:

- **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).
- **Support to replication**- Multiple copies of data store across multiple nodes of a cluster. This ensures high availability, partition, reliability and fault tolerance
- **Distributable:** Big Data solutions permit sharding and distributing of shards on multiple clusters which enhances performance and throughput.
- **Usages of NoSQL servers** which are less expensive. NoSQL data stores require less management efforts. It supports many features like automatic repair, easier data distribution and simpler data models that makes database administrator (DBA) and tuning requirements less stringent.
- **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.
- **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.
- **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.
- **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 need the scalable storage and use of distributed servers together as a cluster.
- 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
- 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 processing speed.

Comparison between NoSQL and SQL:

Features	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 of SQL	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

- Shared nothing (SN) is a cluster architecture. A node does not share data with any other node.
- Big Data store consists of SN architecture.
- Big Data store, therefore, easily partitions into shards. A partition processes the different queries on data of the different users at each node independently. Thus, data processes run in parallel at the nodes.
- A node maintains a copy of running-process data. A coordination protocol controls the processing at all SN nodes. An SN architecture optimizes massive parallel data processing.
- The features of SN architecture are as follows:

➢ Independence: Each node with no memory sharing; thus possesses computational self-sufficiency
➢ Self-Healing: A link failure causes creation of another link
➢ Each node functioning as a shard
➢ No network contention.

Choosing the Distribution Models

- Distribution gives the advantage of: (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:

➢ **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.

➢ **Sharding Very Large Databases**

Figure 3.9(b) 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 DBl are four shards.

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.

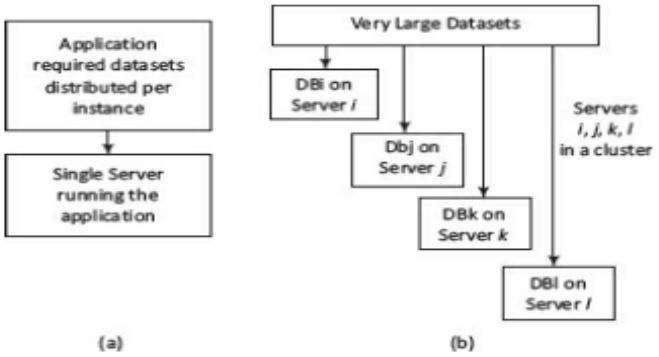


Figure 3.9 (a) Single server model (b) Shards distributed on four servers in a cluster.

➢ **Master-Slave Distribution Model**

- A node serves as a master or primary node and the other nodes are slave nodes.
- 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.
- 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.

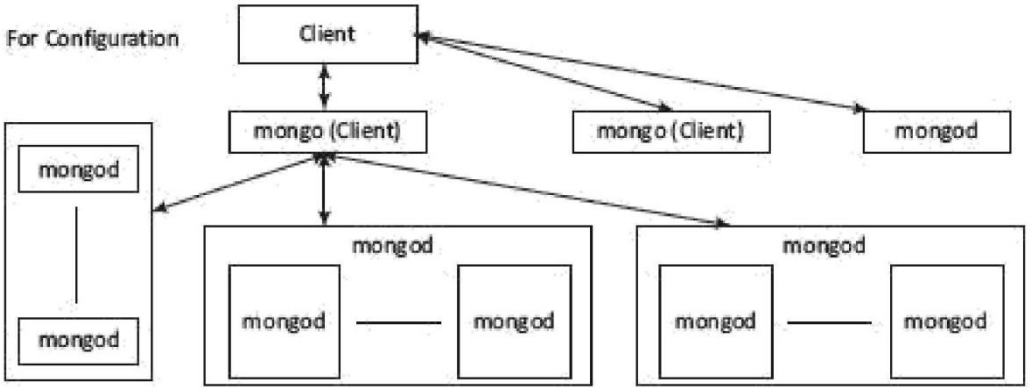


Figure 3.10 Master-slave distribution model. Mongo is a client and mongod is the server

➢ **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.
- Figure 3.11 shows the PPD model.

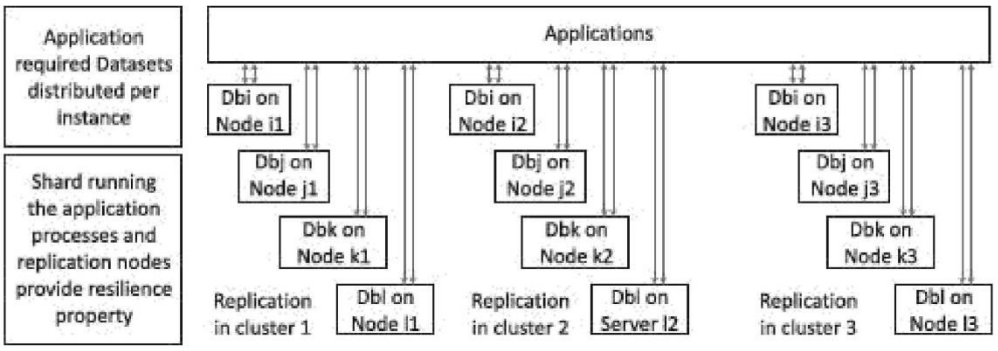


Figure 3.11 Shards replicating on the nodes, which does read and write operations both

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:

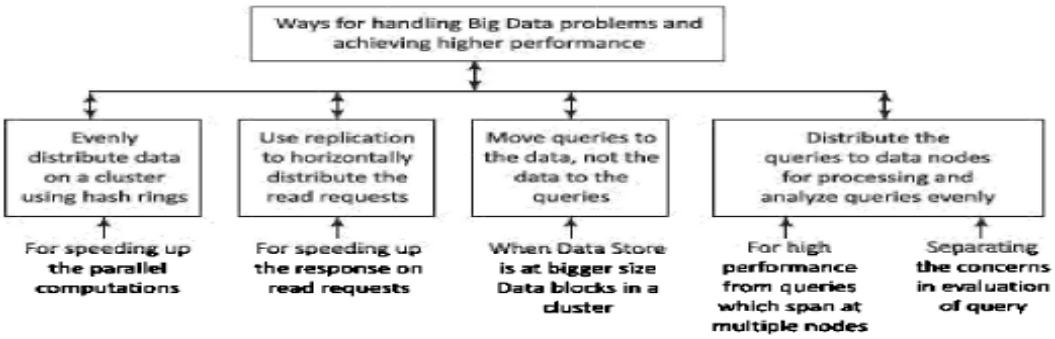


Figure 3.12 Four ways for handling big data problems

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: 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

- MongoDB is an open source DBMS. MongoDB programs create and manage databases.
- 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 JSON-like documents. The data store uses the dynamic schemas.

Features

- **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 of MongoDB is mongod and the client is mongo.
- **Collection stores a number of MongoDB documents.** A collection exists within a single DB to achieve a single purpose. 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**.**
- **Document model is well defined**. Structure of document is clear, Document is the unit of storing data in a MongoDB database. Insert, update and delete operations can be performed on a collection. Document use JSON (JavaScript 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.
- MongoDB is a **document data store in which one collection holds different documents**. Data store in the form of JSON-style documents. Number of fields, content and size of the document can differ from one document to another.
- 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.
- **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.
- **Querying, indexing, and real time aggregation** allows accessing and analyzing the data efficiently.
- **Deep query-ability**—Supports dynamic queries on documents using a document-based query language that's nearly as powerful as SQL.
- **No complex Joins**.
- Distributed DB makes **availability high, and provides horizontal scalability**.
- **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.
- **Atomic operations** on a single document can be performed even though support of multi-document transactions is not present.
- **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.
- **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.

- **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 gives the comparison with RDBMS.

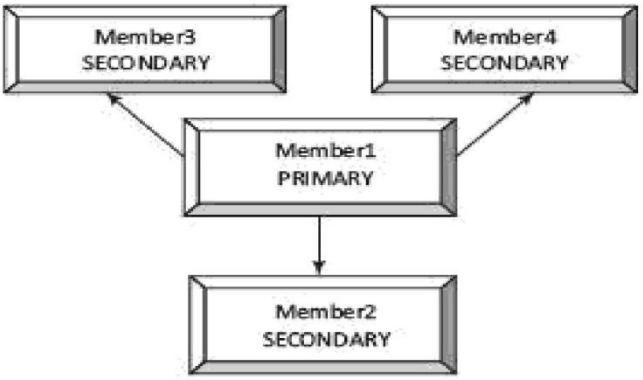
RDBMS	MongoDB
Database	Data store
Table	Collection
Column	Key
Value	Value
Records / Rows / Tuple	Document / Object
Joins	Embedded Documents
Index	Index
Primary key	Primary key (_id) is default key provided by MongoDB itself

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. Replica set starts a mongod instance by specifying – replSet option before running these commands from mongo (MongoDb Client). Table 3.9 gives the commands used for replication (Recoverability means even on occurrences of failures; the transactions ensure consistency).

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

- Figure 3.13 shows a replicated dataset after creating three secondary members from a primary member.



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**.**
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. (Figure 3.10) Basically, it splits the dataset and distributes them across multiple DBs, called shards on the different servers. Each shard is an independent DB. The whole collection of shards forms a single logical DB. If a DB has a 1 terabyte dataset distributed amongst 20 shards, then each shard contains only 50 Giga Byte of data.

Data Types

Table 3.10 Data types which MongoDB documents support

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	ObjectIds (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 ObjectId 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 (Jan 1, 1970).
Null	Represents a null value. A value which is missing or unknown is Null.
Regular Expression	RegExp maps directly to a JavaScript 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.

Rich Queries and other 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 and features for more diverse data types than a relational DB, and at scale

MongoDB querying commands

Command	Functionality
Mongo	Starts MongoDB; (*mongo is MongoDB client). The default database 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

- **To Create database :**Command use – *use command creates a database*;
  For example, Command use lego.
- **To see the existence of database:** Command db – *db command shows that lego database is created.*
- **To drop database:** Command db.dropDatabase() – *This command drops a database.*
- **To create a collection:** Command insert() – *To create 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, ProductId and ProductName) in the lego:*

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

- **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"
            }
      ]
)
```

- **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 objecteId ("_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 productId is 10725.

3.7 Cassandra DATABASES

- 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

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

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:

1. Simple Strategy: Specifies simply a replication factor for the cluster.
2. Network Topology Strategy: Allows setting the replication factor for each data center independently.

Data Types Table 3.14 gives the data types built into Cassandra, their usage and descriptions Table

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 IPv4 or IPv6 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	A JSON-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

Cassandra Data Model

Cassandra Data model is based on Google's BigTable (Section 3.3.3.2). Each value maps with two strings (row key, column key) and timestamp, similar to HBase (Example 2.4). 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 META1 tablet, maximum size is 200 MB, above which a compression algorithm used. META0 is the master-server. Querying by META0 server retrieves a META1 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, (iii) 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 and memtable on all replica nodes in the cluster.
2. EACH_QUORUM: A write must be written to commitlog and memtable on quorum of replica nodes in all data centers.
3. LOCAL_QUORUM: A write must be written to commitlog and memtable on quorum of replica nodes in the same center.
4. ONE: A write must be written to commitlog and memtable of at least one replica 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 outermost grouping 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.

Cassandra Query Language (CQL) Table 3.15 gives the CQL commands and their functionalities.

Command	Functionality
CQLSH	A command line shell for interacting with Cassandra through CQL
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