**Assignment3.7**

**Problem Statement**

**1.What is NoSQL data base?**

A NoSQL (often interpreted as Not only SQL) is used to refer a non-SQL or non relational database that provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases. Motivations for this approach include simplicity of design, horizontal scaling, and finer control over availability.

NoSQL is an approach to database design that can accomodate a wide variety of data models, including key-value, document, columnar and graph formats. NoSQL, which stand for "not only SQL," is an alternative to traditional relational databases in which data is placed in tables and data schema is carefully designed before the database is built. NoSQL databases are especially useful for working with large sets of distributed data. NoSQL database doesn't use tables for storing data. It is generally used to store big data and real-time web applications.

NoSQL encompasses a wide variety of different database technologies that were developed in response to the demands presented in building modern applications:

* Developers are working with applications that create massive volumes of new, rapidly changing data types — structured, semi-structured, unstructured and polymorphic data.
* Long gone is the twelve-to-eighteen month waterfall development cycle. Now small teams work in agile sprints, iterating quickly and pushing code every week or two, some even multiple times every day.
* Applications that once served a finite audience are now delivered as services that must be always-on, accessible from many different devices and scaled globally to millions of users.
* Organizations are now turning to scale-out architectures using open source software, commodity servers and cloud computing instead of large monolithic servers and storage infrastructure.

Relational databases were not designed to cope with the scale and agility challenges that face modern applications, nor were they built to take advantage of the commodity storage and processing power available today.

**2.How does data get stored in NoSQl database?**

There are four general types of NoSQL databases, each with their own specific attributes:

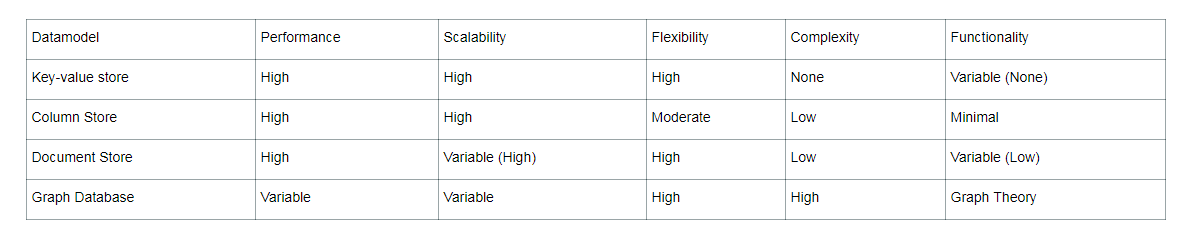
**Graph database** – Based on graph theory, these databases are designed for data whose relations are well represented as a graph and has elements which are interconnected, with an undetermined number of relations between them. Examples include: Neo4j and Titan.

**Key-Value store** – we start with this type of database because these are some of the least complex NoSQL options. These databases are designed for storing data in a schema-less way. In a key-value store, all of the data within consists of an indexed key and a value, hence the name. Examples of this type of database include:Cassandra, DyanmoDB, Azure Table Storage (ATS), Riak, BerkeleyDB.

**Column store** – (also known as wide-column stores) instead of storing data in rows, these databases are designed for storing data tables as sections of columns of data, rather than as rows of data. While this simple description sounds like the inverse of a standard database, wide-column stores offer very high performance and a highly scalable architecture. Examples include: HBase, BigTable and HyperTable.

**Document database** – expands on the basic idea of key-value stores where “documents” contain more complex in that they contain data and each document is assigned a unique key, which is used to retrieve the document. These are designed for storing, retrieving, and managing document-oriented information, also known as semi-structured data. Examples include: MongoDB and CouchDB.

The following table lays out some of the key attributes that should be considered when evaluating NoSQL databases.



**3.What is a column family in HBase?**

HBase tables are organized by column, rather than by row. The columns are organized in groups called column families. When creating a HBase table, we must define the column families before inserting any data. Column families should not be changed often, nor should there be too many of them, so it is important to think carefully about what column families will be useful for our particular data. Each column family, however, can contain a very large number of columns. Columns are named using the format family:qualifier.

Columns in Apache HBase are grouped into column families. All column members of a column family have the same prefix. For example, the columns courses:history and courses:math are both members of the courses column family. The colon character (:) delimits the column family from the . The column family prefix must be composed of printable characters. The qualifying tail, the column family qualifier, can be made of any arbitrary bytes. Column families must be declared up front at schema definition time whereas columns do not need to be defined at schema time but can be conjured on the fly while the table is up an running.

Physically, all column family members are stored together on the filesystem. Because tunings and storage specifications are done at the column family level, it is advised that all column family members have the same general access pattern and size characteristics.

**4.How many maximum number of columns can be added to HBase table?**

HBase currently does not do well with anything above two or three column families so keep the number of column families in your schema low. Currently, flushing and compactions are done on a per Region basis so if one column family is carrying the bulk of the data bringing on flushes, the adjacent families will also be flushed though the amount of data they carry is small. When many column families the flushing and compaction interaction can make for a bunch of needless i/o loading (To be addressed by changing flushing and compaction to work on a per column family basis).

Try to make do with one column family if you can in your schemas. Only introduce a second and third column family in the case where data access is usually column scoped; i.e. you query one column family or the other but usually not both at the one time.

Where multiple ColumnFamilies exist in a single table, be aware of the cardinality (i.e., number of rows). If ColumnFamilyA has 1 million rows and ColumnFamilyB has 1 billion rows, ColumnFamilyA's data will likely be spread across many, many regions (and RegionServers). This makes mass scans for ColumnFamilyA less efficient.

**5.Why columns are not defined at the time of table creation in HBase?**

Columns are usually physically co-located in column families. A column is identified by column family and column qualifier separated by a colon character (:). For example, *courses:math*. The column family prefix must be composed of printable characters. **The column qualifiers (columns) do not have to be defined at schema definition time and they can be added on the fly while the database is up and running.**

A column qualifier is an index for a given data and it is added to a column family. Data within a column family is addressed via the column qualifier**. Column qualifiers are mutable and they may vary between rows.** They do not have data types and they are always treated as arrays of bytes.

A row key, column family and column qualifier form a cell that has a value and timestamp that represents the value’s version. Values also do not have data types and they are always treated as arrays of bytes. A timestamp is recorded for each value and it is the time on the region server when the value was written.

All cell’s values are stored in a descending order by its timestamp. When values are retrieved and if the timestamp is not provided then HBase will return the cell value with the latest (the most recent) timestamp. If a timestamp is not specified during the write, the current timestamp is used.

The maximum number of versions (timestamps) for a given column to store is part of the column schema. It is specified at table creation. It can be specified via alter table command as well. The default value is 1. The minimum number of versions can be also set up per column family. You can also globally set up a maximum number of versions per column.

HBase does not overwrite row values. It stores different values per row by time and column qualifier. Extra versions above the current max version setup are removed during major compactions. If it is not necessary it is not recommended to have very high maximum number of versions since it will increase the HFile size significantly.

It is worth to mention that the column metadata is only stored in internal key/value instances for a column family. You have to keep track of the column names since HBase can support very high number of columns per row and columns can differ between the rows as well. If you do not record these column names by yourself and you forget them you will have to retrieve all rows from a column family in order to find out the column names.

**6.How does data get managed in HBase?**

Hbase is natively supported on Hadoop. The main characteristics that make Hbase an excellent data management platform are fault tolerance, speed and usability. Fault tolerance is provided by automatic fail-over, automatically sharded and load balanced tables, strong consistency in row level operations and replication. Speed is provided by almost real time lookups, in memory caching and server side processing. Usability is provided by a flexible data model that allows many uses, a simple Java API and ability to export metrics.

Hbase can run standalone on the local file system but this set up does not guarantee durability. Edits will be lost when daemons are not cleanly started and stopped. Such a set up is not suitable in a production environment but it provides a way of exploring how the database functions. Alternatively Hbase can be installed on a single or multi node cluster and use HDFS. This set up requires a working set up of Hadoop.

HBase is built on top of the distributed file system (DFS), which can store large files. HBase provides fast record lookups and updates for large tables. The ZooKeeper cluster acts as a coordination service for the entire HBase cluster.

HBase contains two primary services:

**Master server**

The master server co-ordinates the cluster and performs administrative operations, such as assigning regions and balancing the loads.

**Region server**

The region servers do the real work. A subset of the data of each table is handled by each region server. Clients talk to region servers to access data in HBase.

**Regions**

Region servers manage a set of regions.

An HBase table is made up of a set of regions. Regions are the basic unit of work in HBase. It is what is used as a split by the map reduce framework. The region contains store objects that correspond to column families. There is one store instance for each column family. Store objects create one or more *StoreFiles*, which are wrappers around the actual storage file called the *HFile*.

The region also contains a **MemStore**, which is in-memory storage and is used as a write cache. Rows are written to the **MemStore**. The data in the **MemStore** is ordered. If the **MemStore** becomes full, it is persisted to an HFile on disk

To improve performance, it is important to get an even distribution of data among regions, which ensures the best parallelism in map tasks.

**HFiles**

HFiles are the physical representation of data in HBase. Clients do not read HFiles directly but go through region servers to get to the data.

HBase internally puts the data in indexed *StoreFiles* that exist on HDFS for high-speed lookups.

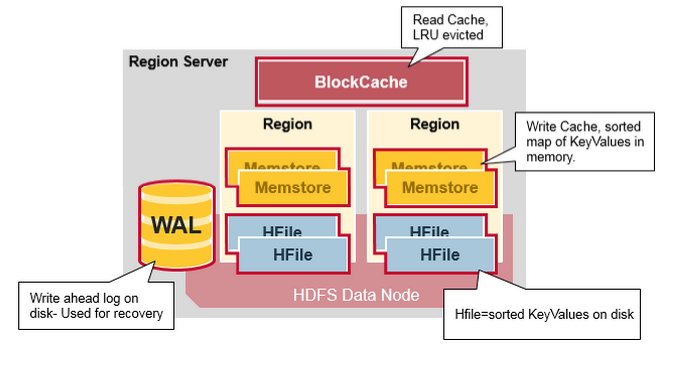
Everything in HBase is stored as bytes and there are no types. There is no schema since each row in HBase can have a different set of columns.

**7.What happens internally when new data gets inserted into HBase table?**

**Region Server Components**

A Region Server runs on an HDFS data node and has the following components:

* WAL: Write Ahead Log is a file on the distributed file system. The WAL is used to store new data that hasn't yet been persisted to permanent storage; it is used for recovery in the case of failure.
* BlockCache: is the read cache. It stores frequently read data in memory. Least Recently Used data is evicted when full.
* MemStore: is the write cache. It stores new data which has not yet been written to disk. It is sorted before writing to disk. There is one MemStore per column family per region.
* Hfiles store the rows as sorted KeyValues on disk.

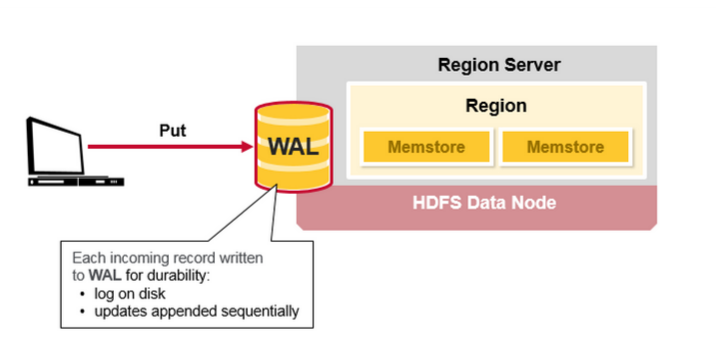


**HBase Write Steps (1)**

When the client issues a Put request, the first step is to write the data to the write-ahead log, the WAL:

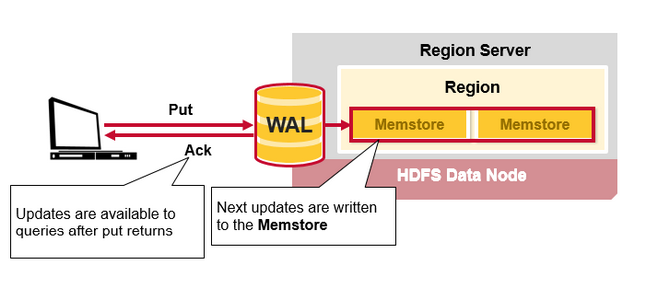
- Edits are appended to the end of the WAL file that is stored on disk.

- The WAL is used to recover not-yet-persisted data in case a server crashes.



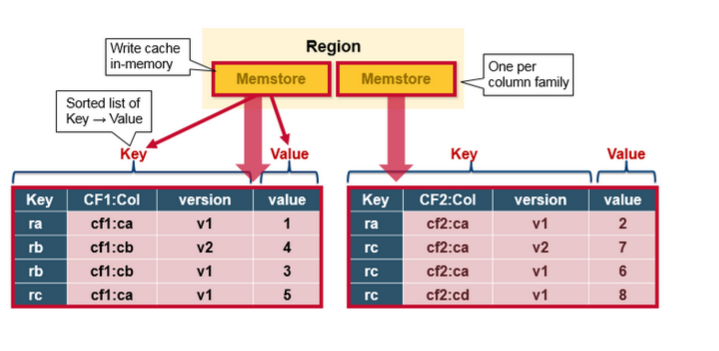
**HBase Write Steps (2)**

Once the data is written to the WAL, it is placed in the MemStore. Then, the put request acknowledgement returns to the client.



**HBase MemStore**

The MemStore stores updates in memory as sorted KeyValues, the same as it would be stored in an HFile. There is one MemStore per column family. The updates are sorted per column family.

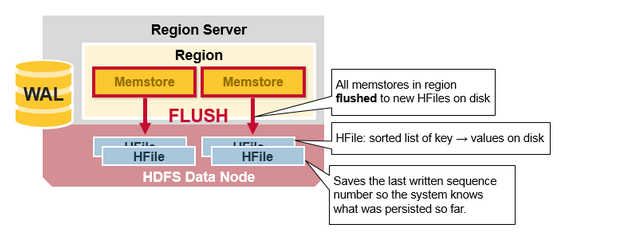


**HBase Region Flush**

When the MemStore accumulates enough data, the entire sorted set is written to a new HFile in HDFS. HBase uses multiple HFiles per column family, which contain the actual cells, or KeyValue instances. These files are created over time as KeyValue edits sorted in the MemStores are flushed as files to disk.

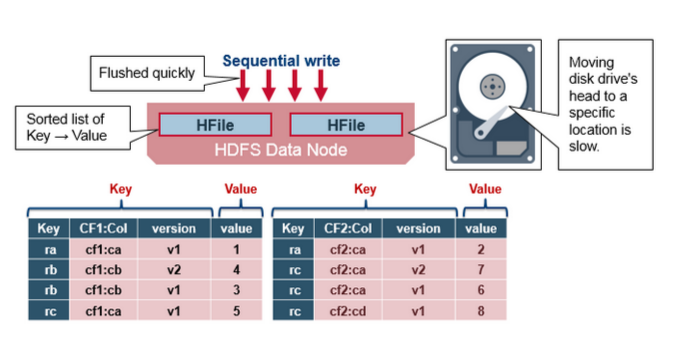
Note that this is one reason why there is a limit to the number of column families in HBase. There is one MemStore per CF; when one is full, they all flush. It also saves the last written sequence number so the system knows what was persisted so far.

The highest sequence number is stored as a meta field in each HFile, to reflect where persisting has ended and where to continue. On region startup, the sequence number is read, and the highest is used as the sequence number for new edits.



**HBase HFile**

Data is stored in an HFile which contains sorted key/values. When the MemStore accumulates enough data, the entire sorted KeyValue set is written to a new HFile in HDFS. This is a sequential write. It is very fast, as it avoids moving the disk drive head.

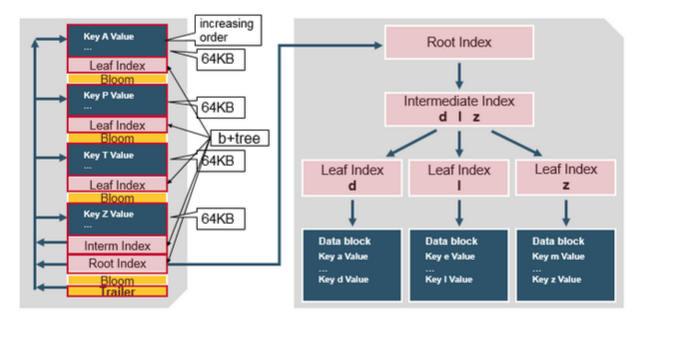


**HBase HFile Structure**

An HFile contains a multi-layered index which allows HBase to seek to the data without having to read the whole file. The multi-level index is like a b+tree:

* Key value pairs are stored in increasing order
* Indexes point by row key to the key value data in 64KB “blocks”
* Each block has its own leaf-index
* The last key of each block is put in the intermediate index
* The root index points to the intermediate index

The trailer points to the meta blocks, and is written at the end of persisting the data to the file. The trailer also has information like bloom filters and time range info. Bloom filters help to skip files that do not contain a certain row key. The time range info is useful for skipping the file if it is not in the time range the read is looking for.



**HFile Index**

The index, which we just discussed, is loaded when the HFile is opened and kept in memory. This allows lookups to be performed with a single disk seek.

