Problem Statement

Understanding the concepts of Hbase

Problem-1:

What is NoSQL Database?

Solution:

NoSQL is an approach to databases that represents a shift away from traditional relational database management systems (RDBMS). To define NoSQL, it is helpful to start by describing SQL, which is a query language used by RDBMS. Relational databases rely on tables, columns, rows, or schemas to organize and retrieve data. In contrast, NoSQL databases do not rely on these structures and use more flexible data models. NoSQL can mean "not SQL" or "not only SQL." As RDBMS have increasingly failed to meet the performance, scalability, and flexibility needs that next-generation, data-intensive applications require, NoSQL databases have been adopted by mainstream enterprises. NoSQL is particularly useful for storing unstructured data, which is growing far more rapidly than structured data and does not fit the relational schemas of RDBMS. Common types of unstructured data include: user and session data; chat, messaging, and log data; time series data such as IoT and device data; and large objects such as video and images.

BENEFITS OF NOSQL

NoSQL databases offer enterprises important advantages over traditional RDBMS, including:

Scalability: NoSQL databases use a horizontal scale-out methodology that makes it easy to add or reduce capacity quickly and non-disruptively with commodity hardware.

Performance: By simply adding commodity resources, enterprises can increase performance with NoSQL databases. This enables organizations to continue to deliver reliably fast user experiences.

High Availability: NoSQL databases are generally designed to ensure high availability and avoid the complexity that comes with a typical RDBMS architecture that relies on primary and secondary nodes.

Global Availability: By automatically replicating data across multiple servers, data centers, or cloud resources, distributed NoSQL databases can minimize latency and ensure a consistent application experience wherever users are located.

Flexible Data Modeling: NoSQL offers the ability to implement flexible and fluid data models. Application developers can leverage the data types and query options that are the most natural fit to the specific application use case rather than those that fit the database schema.

Problem-2:

How does data get stored in NoSQl database?

Solution:

In the in-memory databases like Redis/CouchBase/Tarantool/Aerospike everything is stored in RAM in balanced trees like RB-Tree or in hash tables. All the writes are applied on both RAM and disk, but on disk it goes in an append-only way. A file append can be done as fast as 100Mbytes per second on a normal magnetic disk. If a record size is, say, 1K, then the data will be written at 100krps.

In the on-disk NoSQL databases and db-engines like Cassandra/ HBase/ RocksDB/ LevelDB/ Sophia the main idea is that you have a snapshot file and a write ahead log (WAL) file. Snapshot contains already prepared data in a form of B-Tree with upper levels of that tree being permanently in RAM that can be accesses for reading by doing only one disk seek. A WAL contains all the new changes on top of a current snapshot. A snapshot file is being totally rebuilt on a regular basis using current snapshot and a WAL. All the writes are done nearly as fast as with in-memory databases. "Nearly" because disk is partially busy by doing regular snapshot converting that was described earlier. Reads are significantly slower than that are in in-memory databases, because they take at least one disk seek, but good news is that they can be cached in optimized in-memory structures like RB-Trees/hash tables.

Problem-3:

What is a column family in HBase?

Solution:

Columns in Apache HBase are grouped into *column families*. It is a NoSQL object that contains columns of related data. It is a tuple (pair) that consists of a key-value pair, where the key is mapped to a value that is a set of columns.

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 and 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.

Problem-4:

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

Solution:

There is no hard/special limit to number of columns in HBase, we can have more than 1 million columns but usually three column families are recommended (not more than three).

Note:

But there is a potential issue where we could have 'too wide' rows (many columns)

And if you don't specify exact qualifiers any scan will result whole rows so you could get much more data than you actually need. Also, it could possibly slow intra-row scanning to get needed column inside row.

Problem-5:

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

Solution:

HBase has the characteristic of **Schema-Less design:**

Columns in HBase don't need to be defined up front so they provide a flexible way of managing evolving schemas. However columns can't be renamed or assigned easily from one column-family to the other. Making such changes requires creation of new columns, migration of data from existing columns to the new column and then potentially deletion of old columns.

HBase is effectively a map of a map, so it has no schema. The user can define the columns at runtime, and each row can have its own columns. The responsibility is on the application to interpret the values stored in the HBase. This makes HBase very suitable for applications in which the schema is flexible. Also, unlike a database in which a separate metadata table is needed to describe the page identifiers, each column key describes itself in HBase.

Problem-6:

How does data get managed in HBase?

Solution:

Data manipulation commands that include COUNT, DELETE, DELETEALL and SCAN. Some

table management commands like: ALTER, CREATE, DESCRIBE, DROP, and DROPALL.

Some general commands like VERSION and STATUS all comprise a very small list of

commands available for managing data in Hbase.

The Hbase data model is different from the model provided by relational databases. Hbase is referred to by many terms like a key-value store, column oriented database and versioned map of maps which are correct. The easiest way of visualizing an Hbase data model is a table that has rows and tables.

Data in Hbase is organized into tables. Any characters that are legal in file paths are used to name tables. Tables are further organized into rows that store data. Each row is identified by a unique row key which does not belong to any data type but is stored as a bytearray. Column families are further used to group data in rows. Column families define the physical structure of data so they are defined upfront and their modification is difficult. Each row in a table has same column families. Data in a column family is addressed using a column qualifier. It is not necessary to specify column qualifiers in advance and there is no consistency requirement between rows. No data types are specified for column qualifiers, as such they are just stored as bytearrays. A unique combination of row key, column family and column qualifier forms a cell. Data contained in a cell is referred to as cell value. There is no concept of data type when referring to cell values and they are stored as bytearrays. Versioning happens to cell values using a timestamp of when the cell was written.

Tables in Hbase have several properties that need to be understood to come up with an effective data model. Indexing and sorting only happens on the row key. The concept of data types is absent and everything is stored as bytearray. Only row level atomicity is enforced so multi row transactions are not supported.

Problem-7:

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

Solution:

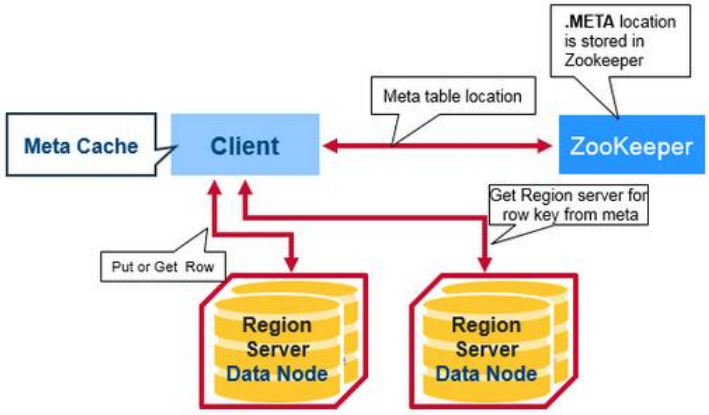
The following takes place when new data is inserted into Hbase:

HBase Write

This is what happens the first time a client reads or writes to HBase:

1. The client gets the Region server that hosts the META table from ZooKeeper.
2. The client will query the .META. server to get the region server corresponding to the row key it wants to access. The client caches this information along with the META table location.
3. It will get the Row from the corresponding Region Server.

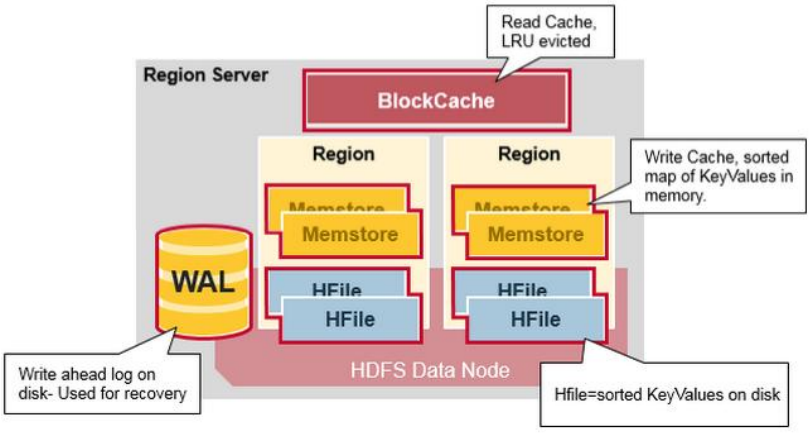
For future reads, the client uses the cache to retrieve the META location and previously read row keys. Over time, it does not need to query the META table, unless there is a miss because a region has moved; then it will re-query and update the cache.



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.

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.

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.