**1) Nosql Databases**

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

**2) Types of NoSQL databases**

There are 4 basic types of NoSQL databases:

1. **Key-Value Store** – It has a Big Hash Table of keys & values {Example- Riak, Amazon S3 (Dynamo)}
2. **Document-based** **Store- It**stores documents made up of tagged elements. {Example- CouchDB}
3. **Column-based Store-**Each storage block contains data from only one column, {Example- HBase, Cassandra}
4. **Graph-based**-A network database that uses edges and nodes to represent and store data. {Example- Neo4J}

1.     **Key Value Store NoSQL Database**

The schema-less format of a key value database like Riak is just about what you need for your storage needs. The key can be synthetic or auto-generated while the value can be String, JSON, BLOB (basic large object) etc.

The key value type basically, uses a hash table in which there exists a unique key and a pointer to a particular item of data. A bucket is a logical group of keys – but they don’t physically group the data. There can be identical keys in different buckets.

Performance is enhanced to a great degree because of the cache mechanisms that accompany the mappings. To read a value you need to know both the key and the bucket because the real key is a hash (Bucket+ Key).

There is no complexity around the Key Value Store database model as it can be implemented in a breeze. Not an ideal method if you are only looking to just update part of a value or query the database.

When we try and reflect back on the CAP theorem, it becomes quite clear that key value stores are great around the Availability and Partition aspects but definitely lack in Consistency.

2.     **Document Store NoSQL Database**

The data which is a collection of key value pairs is compressed as a document store quite similar to a key-value store, but the only difference is that the values stored (referred to as “documents”) provide some structure and encoding of the managed data. XML, JSON (Java Script Object Notation), BSON (which is a binary encoding of JSON objects) are some common standard encodings.

3.     **Column Store NoSQL Database**–

In column-oriented NoSQL database, data is stored in cells grouped in columns of data rather than as rows of data. Columns are logically grouped into column families. Column families can contain a virtually unlimited number of columns that can be created at runtime or the definition of the schema. Read and write is done using columns rather than rows.

In comparison, most relational DBMS store data in rows, the benefit of storing data in columns, is fast search/ access and data aggregation. Relational databases store a single row as a continuous disk entry. Different rows are stored in different places on disk while Columnar databases store all the cells corresponding to a column as a continuous disk entry thus makes the search/access faster.

For example:   To query the titles from a bunch of a million articles will be a painstaking task while using relational databases as it will go over each location to get item titles. On the other hand, with just one disk access, title of all the items can be obtained.

**Data Model**

* **ColumnFamily**:  ColumnFamily is a single structure that can group Columns and SuperColumns with ease.
* **Key**: the permanent name of the record. Keys have different numbers of columns, so the database can scale in an irregular way.
* **Keyspace**:  This defines the outermost level of an organization, typically the name of the application. For example, ‘3PillarDataBase’ (database name).
* **Column**:  It has an ordered list of elements aka tuple with a name and a value defined.

The best known examples are Google’s BigTable and HBase & Cassandra that were inspired from BigTable.

BigTable, for instance is a high performance, compressed and proprietary data storage system owned by Google. It has the following attributes:

* **Sparse**– some cells can be empty
* **Distributed**– data is partitioned across many hosts
* **Persistent**– stored to disk
* **Multidimensional**– more than 1 dimension
* **Map**– key and value
* **Sorted**– maps are generally not sorted but this one is
* Google’s BigTable, HBase and Cassandra are the most popular column store based databases.

4.     **Graph Base NoSQL Database**

In a Graph Base NoSQL Database, you will not find the rigid format of SQL or the tables and columns representation, a flexible graphical representation is instead used which is perfect to address scalability concerns. Graph structures are used with edges, nodes and properties which provides index-free adjacency. Data can be easily transformed from one model to the other using a Graph Base NoSQL database.

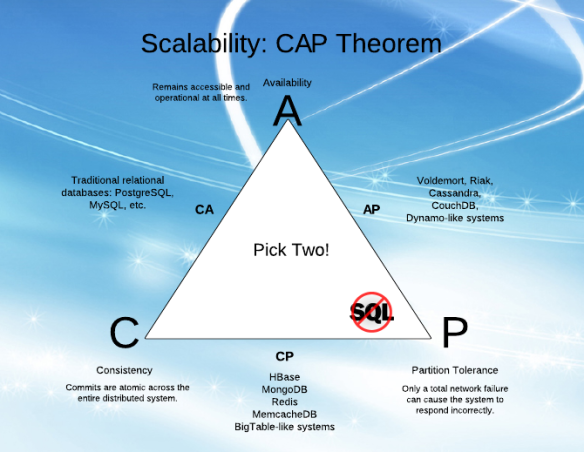
These databases that uses edges and nodes to represent and store data.

* These nodes are organised by some relationships with one another, which is represented by edges between the nodes.
* Both the nodes and the relationships have some defined properties.

**3) CAP Theorem**

* Consistency: every read would get you the most recent write
* Availability: every node (if not failed) always executes queries
* Partition-tolerance: even if the connections between nodes are down, the other two (A & C) promises, are kept.

Usually its depicted in a nicely equilaterl triangle, as this one from [Ofirm](http://ofirm.wordpress.com/2013/01/22/classical-big-data-reading-cap-theorem/" \l "more-65):



There's a nice proof and explanation of it in this 4 minute video  [here](http://www.youtube.com/watch?v=Jw1iFr4v58M). But if we think about it, and also see some of Brewer's (the theorem author) later  [remarks](http://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed), we'll see that the 2 out of 3 is really 1 out of 2:

#### It's really just A vs C!

And this is simply because:

1. Availability is achieved by replicating the data across different machines
2. Consistency is achieved by updating several nodes before allowing further reads
3. Total partitioning, meaning failure of part of the system is rare. However, we could look at a delay, a latency, of the update between nodes, as a *temporary partitioning*. It will then cause a temporary decision between A and C:
   1. On systems that allow reads before updating all the nodes, we will get high Availability
   2. On systems that lock all the nodes before allowing reads, we will get Consistency

**4) Hbase Architecture :**

HBase provides low-latency random reads and writes on top of HDFS. In HBase, tables are dynamically distributed by the system whenever they become too large to handle (Auto Sharding). The simplest and foundational unit of horizontal scalability in HBase is a Region. A continuous, sorted set of rows that are stored together is referred to as a region (subset of table data).  HBase architecture has a single HBase master node (HMaster) and several slaves i.e. region servers. Each region server (slave) serves a set of regions, and a region can be served only by a single region server. Whenever a client sends a write request, HMaster receives the request and forwards it to the corresponding region server.

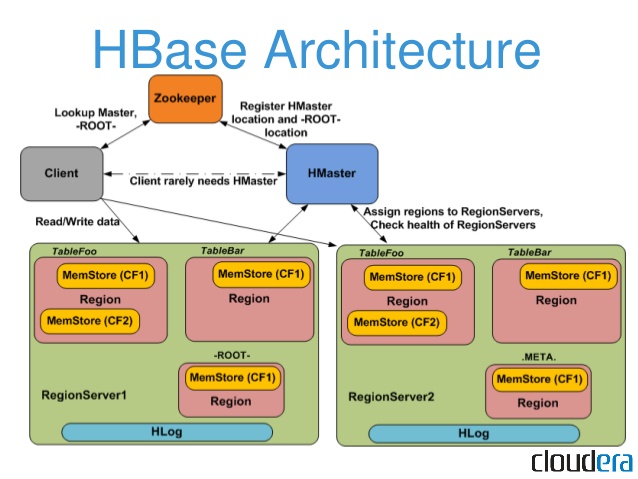


Image Credit : Cloudera

HBase can be run in a multiple master setup, wherein there is only single active master at a time. HBase tables are partitioned into multiple regions with every region storing multiple table’s rows.

### ****Components of Apache HBase Architecture****

HBase architecture has 3 important components- HMaster, Region Server and ZooKeeper.

#### ****HMaster****

HBase HMaster is a lightweight process that assigns regions to region servers in the Hadoop cluster for load balancing. Responsibilities of HMaster –

* Manages and Monitors the Hadoop Cluster
* Performs Administration (Interface for creating, updating and deleting tables.)
* Controlling the failover
* DDL operations are handled by the HMaster
* Whenever a client wants to change the schema and change any of the metadata operations, HMaster is responsible for all these operations.

#### ****Region Server****

These are the worker nodes which handle read, write, update, and delete requests from clients. Region Server process, runs on every node in the hadoop cluster. Region Server runs on HDFS DataNode and consists of the following components –

* Block Cache – This is the read cache. Most frequently read data is stored in the read cache and whenever the block cache is full, recently used data is evicted.
* MemStore- This is the write cache and stores new data that is not yet written to the disk. Every column family in a region has a MemStore.
* Write Ahead Log (WAL) is a file that stores new data that is not persisted to permanent storage.
* HFile is the actual storage file that stores the rows as sorted key values on a disk.

#### ****Zookeeper****

HBase uses ZooKeeper as a distributed coordination service for region assignments and to recover any region server crashes by loading them onto other region servers that are functioning. ZooKeeper is a centralized monitoring server that maintains configuration information and provides distributed synchronization. Whenever a client wants to communicate with regions, they have to approach Zookeeper first. HMaster and Region servers are registered with ZooKeeper service, client needs to access ZooKeeper quorum in order to connect with region servers and HMaster. In case of node failure within an HBase cluster, ZKquoram will trigger error messages and start repairing failed nodes.ZooKeeper service keeps track of all the region servers that are there in an HBase cluster- tracking information about how many region servers are there and which region servers are holding which DataNode. HMaster contacts ZooKeeper to get the details of region servers. Various services that Zookeeper provides include –

Establishing client communication with region servers.

* Tracking server failure and network partitions.
* Maintain Configuration Information
* Provides ephemeral nodes, which represent different region servers.

**5) HBase vs RDBMS**

|  |  |
| --- | --- |
| **HBASE** | **RDBMS** |
| It is distributed, column oriented, versioned data storage system. | It is designed to follow FIXED schema. It is row-oriented databases and doesn’t natively scale to distributed storage. |
| HDFS is underlying layer of HBase and provides fault tolerance and linear scalability. It doesn’t support secondary indexes and support data in key-value pair. | It supports secondary indexes and improvises data retrieval through SQL language. |
| It supports dynamic addition of column in table schema. It is not relational database like RDBMS. | It has slow learning curve and support complex joins and aggregate functions. |
| HBASE helps Hadoop overcome the challenges in random read and write. |  |