**ASSIGNMENT 3.9**

**1. Nosql Databases**

*NoSQL* refers to a database that is not based on SQL (Structured Query Language), which is the language most commonly associated with **relational** databases. Essentially, NoSQL data isn't relational, NoSQL databases usually do not have schema, and they come with looser consistency models than traditional relational databases do.

The term "NoSQL" refers to the fact that traditional relational databases are not adequate for all solutions, particularly ones involving large volumes of data. But the term has been extended to also mean "Not only SQL", indicating support for potential SQL-based interfaces even if the core database isn't relational. Software developers that use NoSQL solutions don't necessarily advocate dismissing relational databases, but instead see value in using the right data store for the job.

Dozens of NoSQL data stores are available; the following are among the most popular:

* [MongoDB](https://www.mongodb.org/). Open-source document database.
* [CouchDB](https://couchdb.apache.org/). Database that uses JSON for documents, JavaScript for MapReduce queries, and regular HTTP for an API.
* [GemFire](http://www.springsource.org/spring-gemfire). Distributed data management platform providing dynamic scalability, high performance, and database-like persistence.
* [Redis](http://redis.io/). Data structure server wherein keys can contain strings, hashes, lists, sets, and sorted sets.
* [Cassandra](https://cassandra.apache.org/). Database that provides scalability and high availability without compromising performance.
* [memcached](https://memcached.org/). Open source high-performance, distributed-memory, and object-caching system.
* [Hazelcast](http://www.hazelcast.com/). Open source highly scalable data distribution platform.
* [HBase](https://hbase.apache.org/). Hadoop database, a distributed and scalable big data store.
* [Mnesia](http://www.erlang.org/doc/man/mnesia.html). Distributed database management system that exhibits soft real-time properties.
* [Neo4j](http://www.neo4j.org/). Open source high-performance, enterprise-grade graph database.

**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.

Example: Consider the data subset represented in the following table. Here the key is the name of the 3Pillar country name, while the value is a list of addresses of 3PiIllar centers in that country.

|  |  |
| --- | --- |
| **Key** | **Value** |
| “India” | {“B-25, Sector-58, Noida, India – 201301” |
| “Romania” | {“IMPS Moara Business Center, Buftea No. 1, Cluj-Napoca, 400606″,City Business Center, Coriolan Brediceanu No. 10, Building B, Timisoara, 300011”} |
| “US” | {“3975 Fair Ridge Drive. Suite 200 South, Fairfax, VA 22033”} |

The key can be synthetic or auto-generated while the value can be String, JSON, BLOB (basic large object) etc.

This key/value type database allow clients to read and write values using a key as follows:

* Get(key), returns the value associated with the provided key.
* Put(key, value), associates the value with the key.
* Multi-get(key1, key2, .., keyN), returns the list of values associated with the list of keys.
* Delete(key), removes the entry for the key from the data store.

While Key/value type database seems helpful in some cases, but it has some weaknesses as well. One, is that the model will not provide any kind of traditional database capabilities (such as atomicity of transactions, or consistency when multiple transactions are executed simultaneously). Such  capabilities must be provided by the application itself.

Secondly, as the volume of data increases, maintaining unique values as keys may become more difficult; addressing this issue requires the introduction of some complexity in generating character strings that will remain unique among an extremely large set of keys.

* Riak and [Amazon’s Dynamo](http://en.wikipedia.org/wiki/Amazon_DynamoDB) are the most popular key-value store NoSQL databases.

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.

The following example shows data values collected as a “document” representing the names of specific retail stores. Note that while the three examples all represent locations, the representative models are different.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | {officeName:”3Pillar Noida”,  {Street: “B-25, City:”Noida”, State:”UP”, Pincode:”201301”}  }  {officeName:”3Pillar Timisoara”,  {Boulevard:”Coriolan Brediceanu No. 10”, Block:”B, Ist Floor”, City: “Timisoara”, Pincode: 300011”}  }  {officeName:”3Pillar Cluj”,  {Latitude:”40.748328”, Longitude:”-73.985560”}  } |

One key difference between a key-value store and a document store is that the latter embeds attribute metadata associated with stored content, which essentially provides a way to query the data based on the contents. For example, in the above example, one could search for all documents in which “City” is “Noida” that would deliver a result set containing all documents associated with any “3Pillar Office” that is in that particular city.

[Apache CouchDB](http://en.wikipedia.org/wiki/CouchDB) is an example of a document store. CouchDB uses [JSON](http://en.wikipedia.org/wiki/JSON) to store data, [JavaScript](http://en.wikipedia.org/wiki/JavaScript) as its query language using [MapReduce](http://en.wikipedia.org/wiki/MapReduce) and [HTTP](http://en.wikipedia.org/wiki/HTTP) for an [API](http://en.wikipedia.org/wiki/API).  Data and relationships are not stored in tables as is a norm with conventional relational databases but in fact are a collection of independent documents.

The fact that document style databases are schema-less makes adding fields to JSON documents a simple task without having to define changes first.

* Couchbase and MongoDB are the most popular document based databases.

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

A 2-dimensional table comprising of rows and columns is part of the relational database system.

|  |  |  |  |
| --- | --- | --- | --- |
| **City** | **Pincode** | **Strength** | **Project** |
| Noida | 201301 | 250 | 20 |
| Cluj | 400606 | 200 | 15 |
| Timisoara | 300011 | 150 | 10 |
| Fairfax | VA 22033 | 100 | 5 |

For above RDBMS table a BigTable map can be visualized as shown below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43 | {  3PillarNoida: {  city: Noida  pincode: 201301  },  details: {  strength: 250  projects: 20  }  }  {  3PillarCluj: {  address: {  city: Cluj  pincode: 400606  },  details: {  strength: 200  projects: 15  }  },  {  3PillarTimisoara: {  address: {  city: Timisoara  pincode: 300011  },  details: {  strength: 150  projects: 10  }  }  {  3PillarFairfax : {  address: {  city: Fairfax  pincode: VA 22033  },  details: {  strength: 100  projects: 5  }  } |

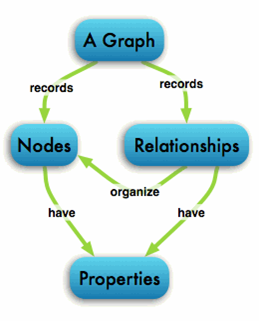
* The outermost keys 3PillarNoida, 3PillarCluj, 3PillarTimisoara and 3PillarFairfax are analogues to rows.
* ‘address’ and ‘details’ are called **column families**.
* The column-family ‘address’ has **columns**‘city’ and ‘pincode’.
* The column-family details’ has **columns**‘strength’ and ‘projects’.

Columns can be referenced using CloumnFamily.

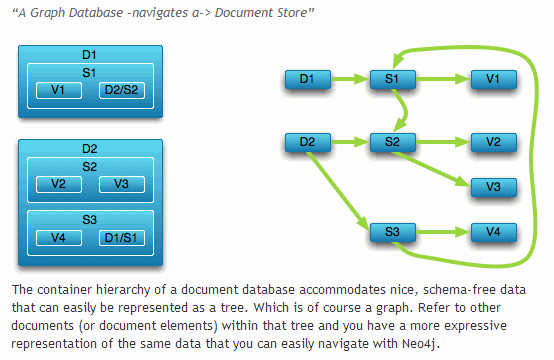
* Google’s BigTable, HBase and Cassandra are the most popular column store based databases.

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.



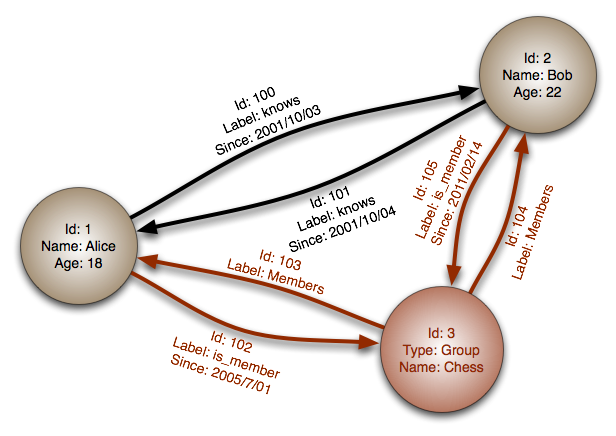
* These databases that uses edges and nodes to represent and store data.
* These nodes are organised by some relationships with one another, which is represented by edges between the nodes.
* Both the nodes and the relationships have some defined properties.



The following are some of the features of the graph based database, which are explained on the basis of the example below:

Labeled, directed, attributed multi-graph : The graphs contains the nodes which are labelled properly with some properties and these nodes have some relationship with one another which is shown by the directional edges. For example: in the following representation, “Alice knows Bob”  is shown by an edge that also has some properties.

While relational database models can replicate the graphical ones, the edge would require a join which is a costly proposition.

  
**UseCase**–

Any ‘Recommended for You’ rating you see on e-commerce websites (book/video renting sites) is often derived by taking into account how other users have rated the product in question. Arriving at such a UseCase is made easy using Graph databases.

[InfoGrid](http://infogrid.org/) and [Infinite Graph](http://www.infinitegraph.com/) are the most popular graph based databases.  InfoGrid allows the connection of as many edges (Relationships) and nodes (MeshObjects), making it easier to represent hyperlinked and complex set of information.

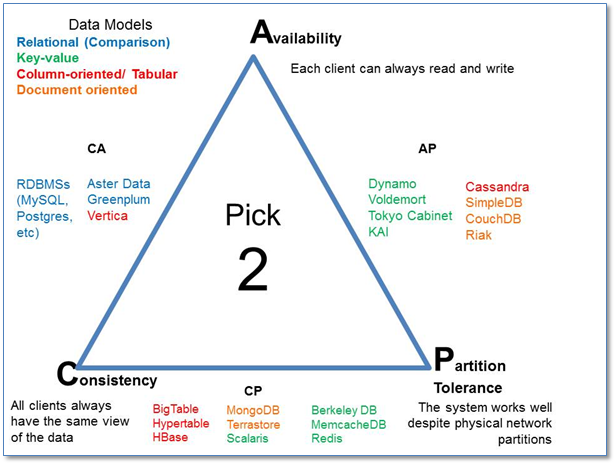
There are two kinds of GraphDatabase offered by InfoGrid, these include the following:

[MeshBase](http://infogrid.org/trac/wiki/MeshBase)–   It is a perfect option where standalone deployment is required.

[NetMeshBase](http://infogrid.org/trac/wiki/NetMeshBase) – It is ideally suited for large distributed graphs and has additional capabilities to communicate with other similar NetMeshbase.

**3) CAP Theorem**

According to **CAP Theorem** distributed systems can satisfy any two features at the same time but not all three features.



* Consistency - This means that the data in the database remains consistent after the

execution of an operation. For example after an update operation, all clients see the

same data.

• Availability - This means that the system is always on (service guarantee availability),

no downtime.

• Partition Tolerance - This means that the system continues to function even if the

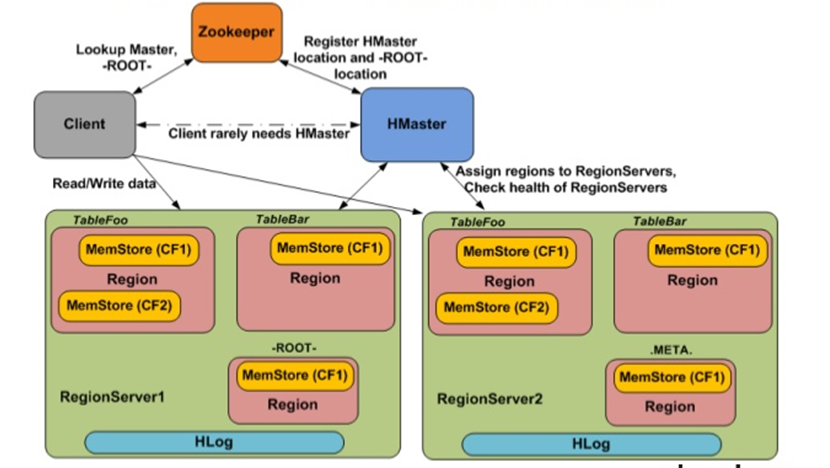
communication among the servers is unreliable, i.e. the servers may be partitioned into

multiple groups that cannot communicate with one another.

Duplicate Copy of same data is maintained on Multiple Machines.

• This increases availability, but decreases consistency.

**4) HBase Architecture**

****

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.

Understanding the fundamental of HBase architecture is easy but running HBase on top of HDFS in production is challenging when it comes to monitoring compactions, row key designs manual splitting, etc.

**5) HBase vs RDBMS**

