Session 11: ADVANCED HBASE

Assignment 1

Task 1:

Explain the below concepts with an example in brief.

1. **NoSQL data bases**

Solution:

NoSQL is an approach to database design that can accommodate 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.

2. **Types of NoSQl databases**

Solution:

There are 4 basic types of NoSQL databases:

- **Key-Value Store –** It has a Big Hash Table of keys & values {Example- Riak, Amazon S3 (Dynamo)}

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 3Pilllar 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”}

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 are the most popular key-value store NoSQL databases.

- **Document-based Store-** It stores documents made up of tagged elements. {Example-CouchDB}
  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.

{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 is an example of a document store. CouchDB uses JSON to store data, JavaScript as its query language using MapReduce and HTTP for an 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.

- **Column-based Store-** Each storage block contains data from only one column, {Example- HBase, Cassandra}
  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.
  Google’s BigTable, HBase and Cassandra are the most popular column store based databases.

- **Graph-based-**A network database that uses edges and nodes to represent and store data. {Example- Neo4J}

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.

3. CAP Theorem

Solution:

CAP Theorem is a concept that a distributed database system can only have 2 of the 3: **Consistency, Availability and Partition Tolerance.**CAP Theorem is very important in the Big

Data world, especially when we need to make trade off's between the three, based on our unique use case.

In the past, when we wanted to store more data or increase our processing power, the common option was to scale vertically (get more powerful machines) or further optimize the existing code base. However, with the advances in parallel processing and distributed systems, it is more common to expand horizontally, or have more machines to do the same task in parallel. We can already see a bunch of data manipulation tools in the Apache project like Spark, Hadoop, Kafka, Zookeeper and Storm. However, in order to effectively pick the tool of choice, a basic idea of CAP Theorem is necessary.

Partition Tolerance

This condition states that the system continues to run, despite the number of messages being delayed by the network between nodes. A system that is partition-tolerant can sustain any amount of network failure that doesn't result in a failure of the entire network. Data records are sufficiently replicated across combinations of nodes and networks to keep the system up through intermittent outages. When dealing with modern distributed systems, Partition Tolerance is not an option. It's a necessity. Hence, we have to trade between Consistency and Availability.

High Consistency

This condition states that all nodes see the same data at the same time. Simply put, performing a read operation will return the value of the most recent write operation causing all nodes to return the same data. A system has consistency if a transaction starts with the system in a consistent state, and ends with the system in a consistent state. In this model, a system can (and does) shift into an inconsistent state during a transaction, but the entire transaction gets rolled back if there is an error during any stage in the process. In the image, we have 2 different records ("Bulbasaur" and "Pikachu") at different timestamps. The output on the third partition is "Pikachu", the latest input. However, the nodes will need time to update and will not be Available on the network as often.

High Availability

This condition states that every request gets a response on success/failure. Achieving availability in a distributed system requires that the system remains operational 100% of the time. Every client gets a response, regardless of the state of any individual node in the system. This metric is trivial to measure: either you can submit read/write commands, or you cannot. Hence, the databases are time independent as the nodes need to be available online at all times. This means that, unlike the previous example, we do not know if "Pikachu" or "Bulbasaur" was added first. The output could be either one. Hence why, high availability isn't feasible when analyzing streaming data at high frequency.

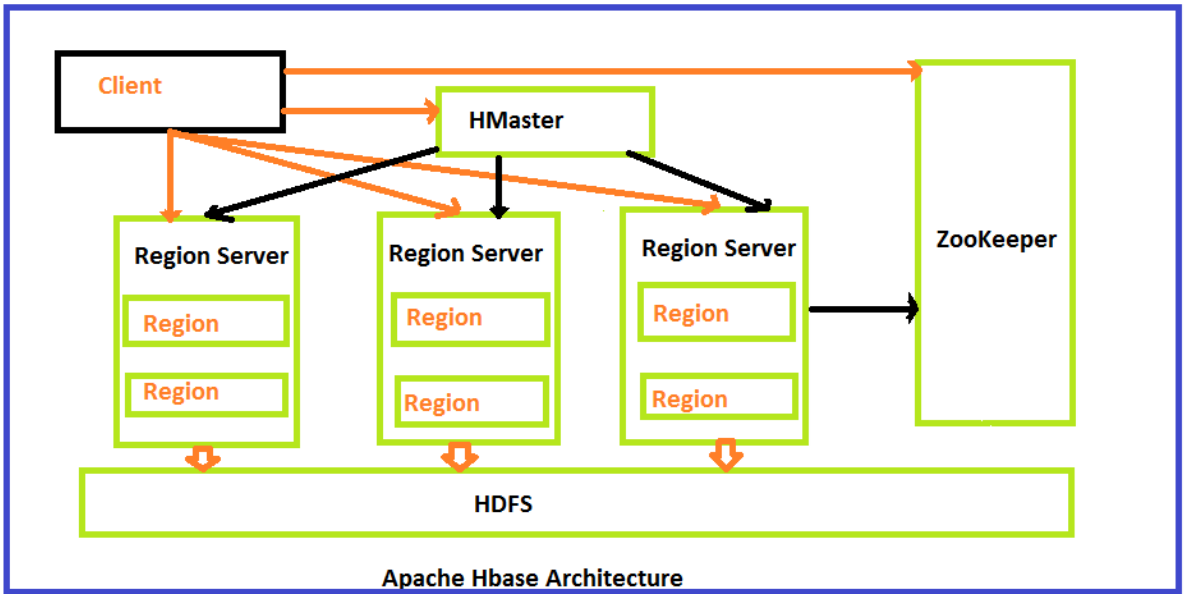
Conclusion

Distributed systems allow us to achieve a level of computing power and availability that were simply not available in the past. Our systems have higher performance, lower latency,

and near 100% up-time in data centers that span the entire globe. Best of all, the systems of today are run on commodity hardware that is easily obtainable and configurable at affordable costs. However, there is a price. Distributed systems are more complex than their single-network counterparts. Understanding the complexity incurred in distributed systems, making the appropriate trade-offs for the task at hand (CAP), and selecting the right tool for the job is necessary with horizontal scaling.

4. HBase Architecture

Solution:



HBase architecture consists mainly of four components

- **HMaster**
- **HRegionserver**
- **HRegions**
- **Zookeeper**

HMaster:

HMaster is the implementation of Master server in HBase architecture. It acts like monitoring agent to monitor all Region Server instances present in the cluster and acts as an interface for all the metadata changes. In a distributed cluster environment, Master runs on NameNode. Master runs several background threads.

The following are important roles performed by HMaster in HBase.

- Plays a vital role in terms of performance and maintaining nodes in the cluster.
- HMaster provides admin performance and distributes services to different region servers.
- HMaster assigns regions to region servers.

- HMaster has the features like controlling load balancing and failover to handle the load over nodes present in the cluster.
- When a client wants to change any schema and to change any Metadata operations, HMaster takes responsibility for these operations.

Some of the methods exposed by HMaster Interface are primarily Metadata oriented methods.

- Table (createTable, removeTable, enable, disable)
- ColumnFamily (add Column, modify Column)
- Region (move, assign)

The client communicates in a bi-directional way with both HMaster and ZooKeeper. For read and write operations, it directly contacts with HRegion servers. HMaster assigns regions to region servers and in turn check the health status of region servers.

In entire architecture, we have multiple region servers. Hlog present in region servers which are going to store all the log files.

HRegions Servers:

When Region Server receives writes and read requests from the client, it assigns the request to a specific region, where actual column family resides. However, the client can directly contact with HRegion servers, there is no need of HMaster mandatory permission to the client regarding communication with HRegion servers. The client requires HMaster help when operations related to metadata and schema changes are required.

HRegionServer is the Region Server implementation. It is responsible for serving and managing regions or data that is present in distributed cluster. The region servers run on Data Nodes present in the Hadoop cluster.

HMaster can get into contact with multiple HRegion servers and performs the following functions.

- Hosting and managing regions
- Splitting regions automatically
- Handling read and writes requests
- Communicating with the client directly

HRegions:

HRegions are the basic building elements of HBase cluster that consists of the distribution of tables and are comprised of Column families. It contains multiple stores, one for each column family. It consists of mainly two components, which are Memstore and Hfile.

5. HBase vs RDBMS

Solution:

Hadoop and RDBMS are varying concepts of processing, retrieving and storing the data or information. While Hadoop is an open-source Apache project, RDBMS stands for Relational Database Management System. Hadoop framework has been written in Java which makes it scalable and makes it able to support applications that call for high performance standards. Hadoop framework enables the storage of large amounts of data on files systems of multiple computers. Hadoop is configured to allow scalability from a single computer node to several thousands of nodes or independent workstations in a manner that the individual nodes utilize local computer storage CPU processing power and memory.

HBase	RDBMS
Column-oriented	Row oriented (mostly)
Flexible schema, add columns on the fly	Fixed schema.
Good with sparse tables,	Not optimized for sparse tables.
Joins using MR –not optimized	Optimized for joins.
Tight integration with MR	Not really...
Horizontal scalability –just add hardware	Hard to shard and scale
Good for semi-structured data as well as Un-structured data	Good for structured data

Task 2:

Execute blog present in below link

https://acadgild.com/blog/importtsv-data-from-hdfs-into-hbase/