Session 09: HBase

Assignment 3

**Problem Statement**

1. Nosql Databases
2. Types of Nosql Databases
3. CAP Theorem
4. HBase Architecture
5. HBase vs RDBMS

**Solution**

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.

1. **Types of Nosql Databases**

TYPES OF NOSQL DATABASES:-

Several different varieties of NoSQL databases have been created to support specific needs and use cases. These fall into four main categories:

* **Key-value data stores:** Key-value NoSQL databases emphasize simplicity and are very useful in accelerating an application to support high-speed read and write processing of non-transactional data. Stored values can be any type of binary object (text, video, JSON document, etc.) and are accessed via a key. The application has complete control over what is stored in the value, making this the most flexible NoSQL model. Data is partitioned and replicated across a cluster to get scalability and availability. For this reason, key value stores often do not support transactions. However, they are highly effective at scaling applications that deal with high-velocity, non-transactional data.
* **Document stores:** Document databases typically store self-describing JSON, XML, and BSON documents. They are similar to key-value stores, but in this case, a value is a single document that stores all data related to a specific key. Popular fields in the document can be indexed to provide fast retrieval without knowing the key. Each document can have the same or a different structure.
* **Wide-column stores:** Wide-column NoSQL databases store data in tables with rows and columns similar to RDBMS, but names and formats of columns can vary from row to row across the table. Wide-column databases group columns of related data together. A query can retrieve related data in a single operation because only the columns associated with the query are retrieved. In an RDBMS, the data would be in different rows stored in different places on disk, requiring multiple disk operations for retrieval.
* **Graph stores:** A graph database uses graph structures to store, map, and query relationships. They provide index-free adjacency, so that adjacent elements are linked together without using an index.

Multi-modal databases leverage some combination of the four types described above and therefore can support a wider range of applications.

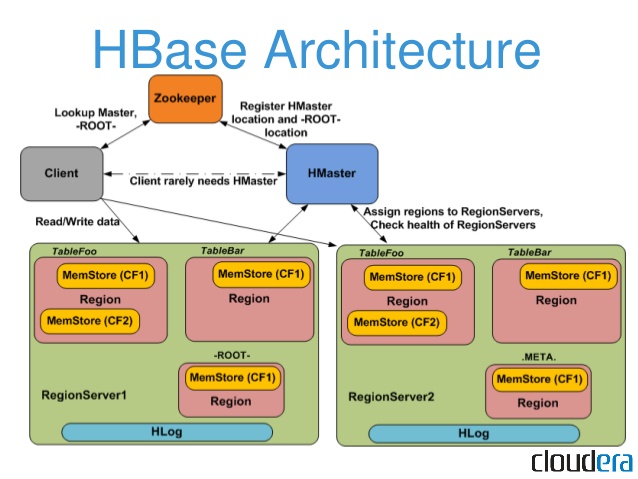
1. **CAP Theorem**

For any distributed system, CAP Theorem reiterates the need to find balance between Consistency, Availability and Partition tolerance. Consistency means all the nodes see the same data at the same time. Availability implies that every request receives a response about whether it was successful or failed. It’s more of a handshaking mechanism in computer network methodology.

Coming to partition tolerance, the system continues to operate despite arbitrary message loss or failure of part of the system. Systems with partition tolerance feature works well despite physical network partitions.

According to CAP Theorem distributed systems can satisfy any two features at the same time but not all three features. Traditional systems like RDBMS provide consistency and availability. Column oriented databases like MongoDB, Hbase and Big Table provide features consistency and partition tolerance. Let us have a look at some the differences between RDBMS and HBase.

1. **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.

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.

(HBase vs RDBMS on next page)

1. **HBase vs RDBMS**

|  |  |
| --- | --- |
| **HBASE** | **RDBMS** |
| Schema-less in database. | Having fixed schema in database. |
| Column oriented database. | Row oriented data store. |
| Designed to store De-normalized data. | Designed to store Normalized data. |
| Wide and sparsely populated tables present in Hbase. | Contains thin tables in database. |
| Supports automatic partitioning. | Has no built in support for partitioning. |
| Well suited for OLAP systems. | Well suited for OLTP systems. |
| Read only relevant data from database. | To retrieve one row at a time and hence could read unnecessary data if only some of the data in a row is required. |
| Structured and semi structure data can be stored and processed using Hbase. | Structured data can be stored and processed using an RDBMS. |
| Enables aggregation over many rows and columns. | Aggregation is an expensive operation. |