Accadgild_Session_9_Assignement_9.3_Solution

Problem Statement

Explain the below concepts with an example in brief.

➔ <u>Nosql Databases :</u>

A NoSQL database environment is, simply put, a non-relational and largely distributed database system that enables rapid, ad-hoc organization and analysis of extremely high-volume, disparate data types. NoSQL databases are sometimes referred to as cloud databases, non-relational databases, Big Data databases and a myriad of other terms and were developed in response to the sheer volume of data being generated, stored and analyzed by modern users (user-generated data) and their applications (machine-generated data).

In general, NoSQL databases have become the first alternative to relational databases, with scalability, availability, and fault tolerance being key deciding factors. They go well beyond the more widely understood legacy, relational databases (such as Oracle, SQL Server and DB2 databases) in satisfying the needs of today's modern business applications. A very flexible and schema-less data model, horizontal scalability, distributed architectures, and the use of languages and interfaces that are "not only" SQL typically characterize this technology.

From a business standpoint, considering a NoSQL or 'Big Data' environment has been shown to provide a clear competitive advantage in numerous industries. In the 'age of data', this is compelling information as a great saying about the importance of data is summed up with the following "if your data isn't growing then neither is your business".

➔ Types of Nosql Databases :

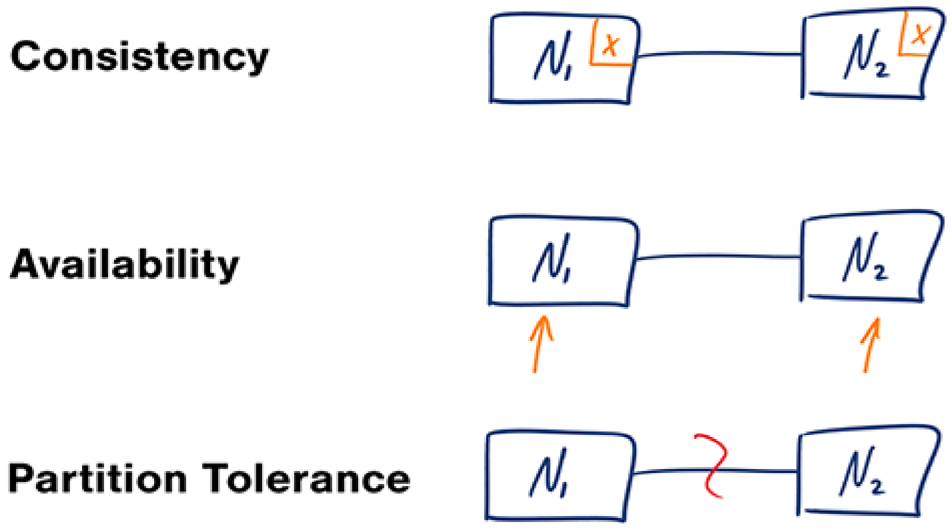
There are four general types of NoSQL databases, each with their own specific attributes:

- Graph database – Based on graph theory, these databases are designed for data whose relations are well represented as a graph and has elements which are interconnected, with an undetermined number of relations between them. Examples include: Neo4j and Titan.
- Key-Value store – we start with this type of database because these are some of the least complex NoSQL options. These databases are designed for storing data in a schema-less way. In a key-value store, all of the data within consists of an indexed key and a value, hence the name. Examples of this type of database include:Cassandra, DyanmoDB, Azure Table Storage (ATS), Riak, BerkeleyDB.
- Column store – (also known as wide-column stores) instead of storing data in rows, these databases are designed for storing data tables as sections of columns of data, rather than as rows of data. While this simple description sounds like the inverse of a standard database, wide-column stores offer very high performance and a highly scalable architecture. Examples include: HBase, BigTable and HyperTable.

- Document database – expands on the basic idea of key-value stores where “documents” contain more complex in that they contain data and each document is assigned a unique key, which is used to retrieve the document. These are designed for storing, retrieving, and managing document-oriented information, also known as semi-structured data. Examples include: MongoDB and CouchDB.

➔ CAP Theorem :

The CAP Theorem states that, in a distributed system (a collection of interconnected nodes that share data.), you can only have two out of the following three guarantees across a write/read pair: Consistency, Availability, and Partition Tolerance - one of them must be sacrificed. However, as you will see below, you don’t have as many options here as you might think.



- **Consistency** – A read is guaranteed to return the most recent write for a given client.
- **Availability** – A non-failing node will return a reasonable response within a reasonable amount of time (no error or timeout).
- **Partition Tolerance** – The system will continue to function when network partitions occur.

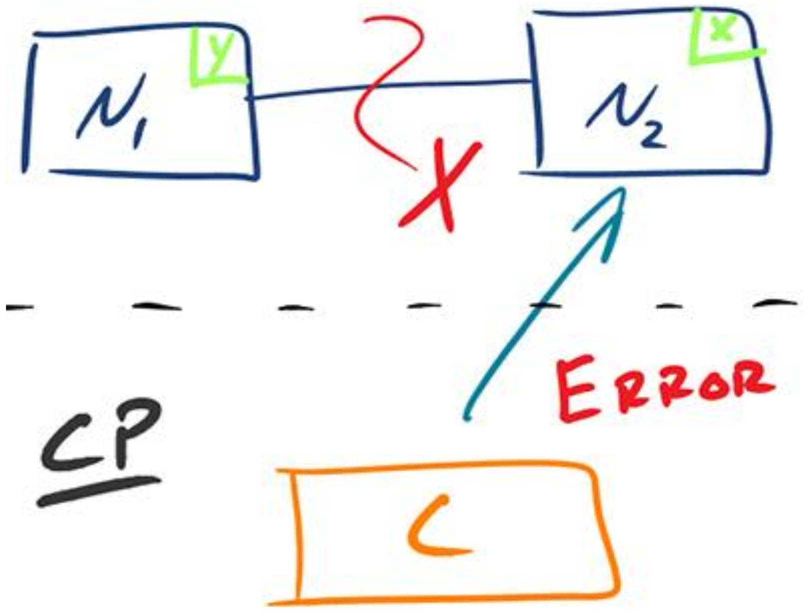
Before moving further, we need to set one thing straight. Object Oriented Programming != Network Programming! There are assumptions that we take for granted when building applications that share memory, which break down as soon as nodes are split across space and time.

One such fallacy of distributed computing is that networks are reliable. They aren’t. Networks and parts of networks go down frequently and unexpectedly. Network failures happen to your system and you don’t get to choose when they occur.

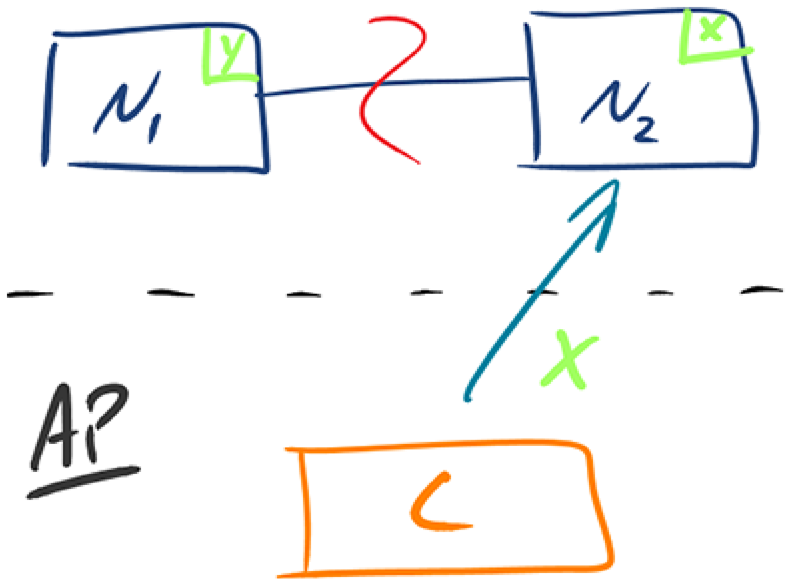
Accadgild_Session_9_Assignement_9.3_Solution

Given that networks aren't completely reliable, you must tolerate partitions in a distributed system, period. Fortunately, though, you get to choose what to do when a partition does occur. According to the CAP theorem, this means we are left with two options: Consistency and Availability.

- **CP** – Consistency/Partition Tolerance - Wait for a response from the partitioned node which could result in a timeout error. The system can also choose to return an error, depending on the scenario you desire. Choose Consistency over Availability when your business requirements dictate atomic reads and writes.



- **AP** – Availability/Partition Tolerance - Return the most recent version of the data you have, which could be stale. This system state will also accept writes that can be processed later when the partition is resolved. Choose Availability over Consistency when your business requirements allow for some flexibility around when the data in the system synchronizes. Availability is also a compelling option when the system needs to continue to function in spite of external errors (shopping carts, etc.)



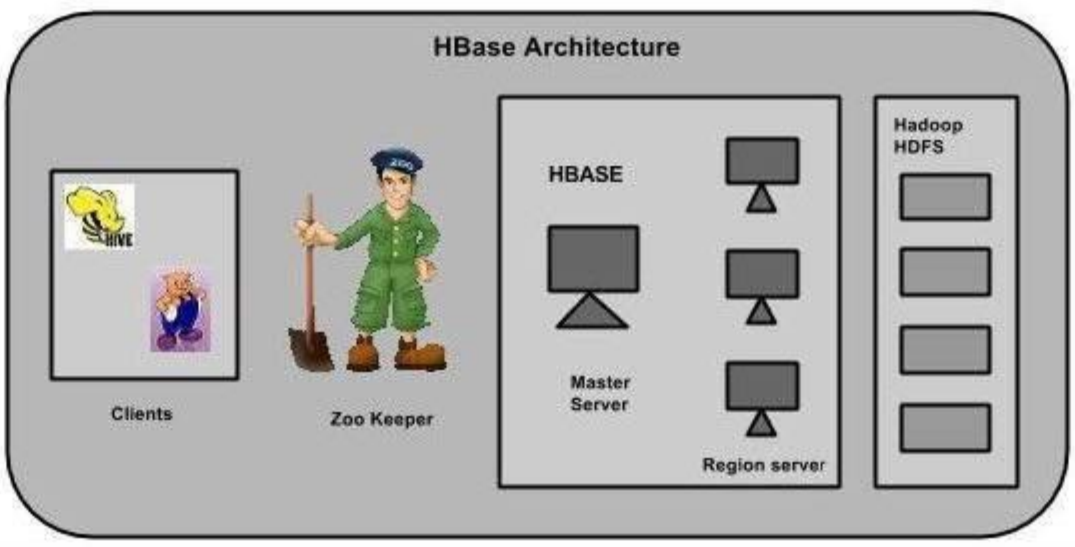
The decision between Consistency and Availability is a software trade off. You can choose what to do in the face of a network partition - the control is in your hands. Network outages, both temporary and permanent, are a fact of life and occur whether you want them to or not - this exists outside of your software.

Building distributed systems provide many advantages, but also adds complexity. Understanding the trade-offs available to you in the face of network errors, and choosing the right path is vital to the success of your application. Failing to get this right from the beginning could doom your application to failure before your first deployment.

➔ HBase Architecture :

In HBase, tables are split into regions and are served by the region servers. Regions are vertically divided by column families into "Stores". Stores are saved as files in HDFS. Shown below is the architecture of HBase.

➔ **Note:** The term 'store' is used for regions to explain the storage structure.



➔ HBase has three major components: the client library, a master server, and region servers. Region servers can be added or removed as per requirement.

MasterServer

The master server -

- Assigns regions to the region servers and takes the help of Apache ZooKeeper for this task.
- Handles load balancing of the regions across region servers. It unloads the busy servers and shifts the regions to less occupied servers.
- Maintains the state of the cluster by negotiating the load balancing.
- Is responsible for schema changes and other metadata operations such as creation of tables and column families.

Regions

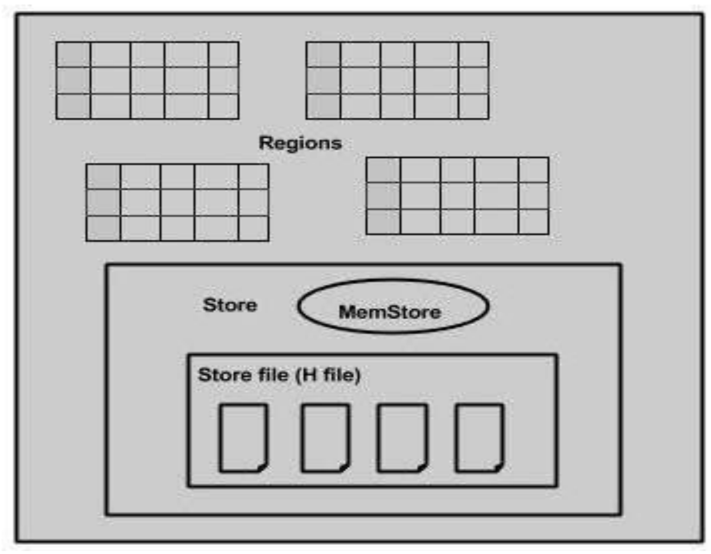
- Regions are nothing but tables that are split up and spread across the region servers.

Region server

The region servers have regions that -

- Communicate with the client and handle data-related operations.
- Handle read and write requests for all the regions under it.
- Decide the size of the region by following the region size thresholds.

When we take a deeper look into the region server, it contain regions and stores as shown below:



The store contains memory store and HFiles. Memstore is just like a cache memory. Anything that is entered into the HBase is stored here initially. Later, the data is transferred and saved in Hfiles as blocks and the memstore is flushed.

Zookeeper

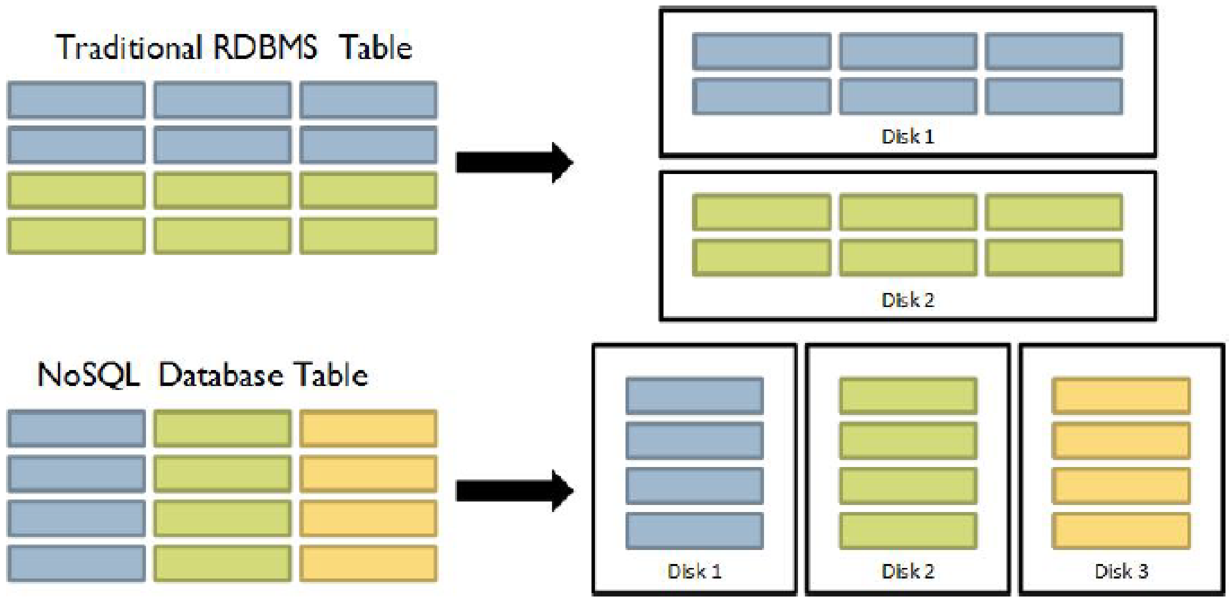
- Zookeeper is an open-source project that provides services like maintaining configuration information, naming, providing distributed synchronization, etc.
- Zookeeper has ephemeral nodes representing different region servers. Master servers use these nodes to discover available servers.
- In addition to availability, the nodes are also used to track server failures or network partitions.
- Clients communicate with region servers via zookeeper.
- In pseudo and standalone modes, HBase itself will take care of zookeeper.

➔ HBase vs RDBMS

There differences between RDBMS and HBase are given below.

- Schema/Database in RDBMS can be compared to namespace in Hbase.
- A table in RDBMS can be compared to column family in Hbase.
- A record (after table joins) in RDBMS can be compared to a record in Hbase.
- A collection of tables in RDBMS can be compared to a table in Hbase..

Accadgild_Session_9_Assignement_9.3_Solution



Submitted By:

Shishir Kumar Jha