Explain the below concepts with an example in brief.

● 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.

● Types of Nosql Databases

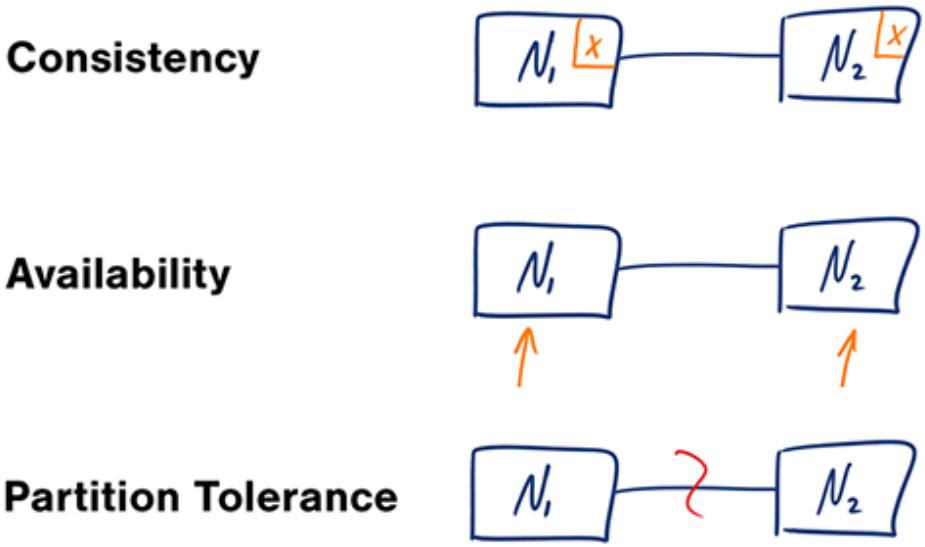
Several different varieties of NoSQL databases have been created to support specific needs and use cases. These fall into four main categories:

- 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.

● 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.

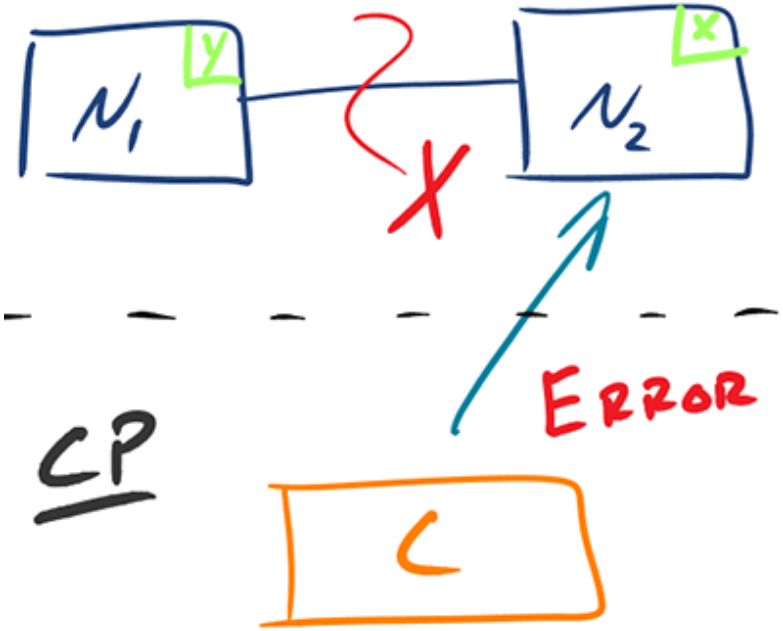


- Consistency - A read is guaranteed to return the most recent write for a given client.
- Availability - A non-failing node will return a reasonable response within a reasonable amount of time (no error or timeout).
- Partition Tolerance - The system will continue to function when network partitions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

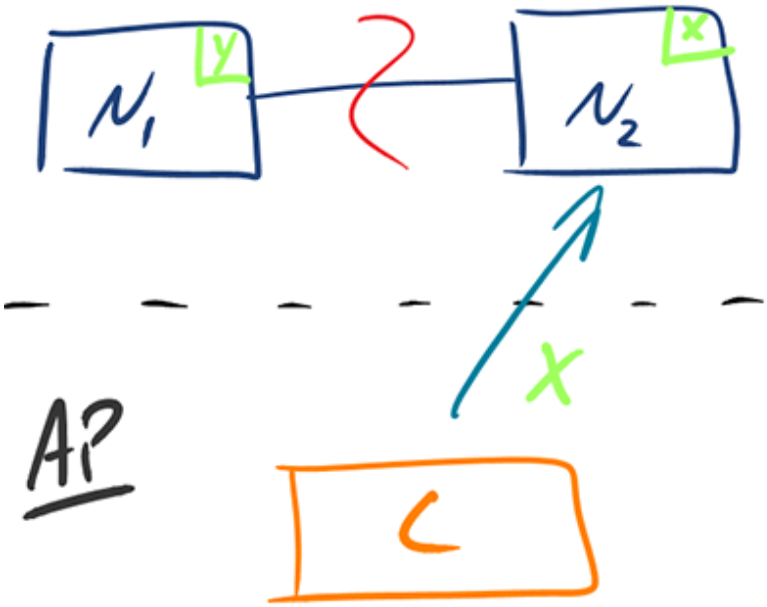
Before moving further, we need to set one thing straight. Object Oriented Programming != Network Programming! There are assumptions that we take for granted when building applications that share memory, which break down as soon as nodes are split across space and time.
One such fallacy of distributed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

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.



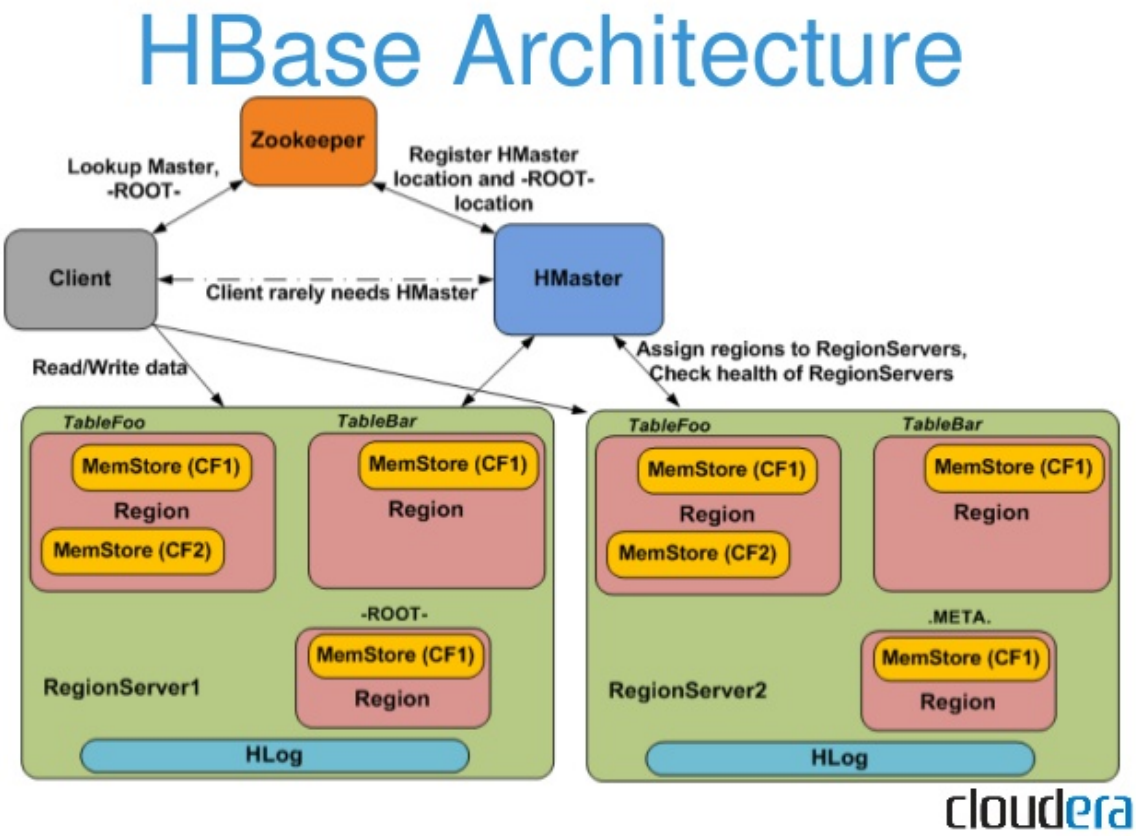
- AP - Availability/Partition Tolerance - Return the most recent version of the data you have, which could be stale. This system state will also accept writes that can be processed later when the partition is resolved. Choose Availability over Consistency when your business requirements allow for some 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

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.

i 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.

i 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.

i 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

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..