Tribhuvan University

Institute of Engineering

Pulchowk Campus

Big Data Technologies

No SQL database architectures

Hbase, Casasndra, MongoDB

Submitted by:

Bishal Katuwal

075BCT028

Submitted to:

Department of Electronics and Computer Engineering

Pulchowk Campus

Submitted on:

9th March, 2023

1 Hbase

1.1 Introduction

HBase is a distributed NoSQL database that runs on top of the Hadoop Distributed File System (HDFS). It is a column-oriented database management system that is designed to handle large amounts of structured data and provide real-time read/write access to that data. It is highly scalable. The architecture of HBase consists of three main components. They are :

- ZooKeeper
- RegionServers
- HMaster

1.2 Architecture

The architecture of HBase consists of the following components:

- **Zookeeper**:
  HBase uses Apache ZooKeeper to manage the distributed coordination of the HBase cluster. ZooKeeper ensures that all the nodes in the cluster are synchronized and up to date. It keeps track of the state of the cluster and helps in leader election.
- **Region Server**:
  HBase is designed to scale horizontally by distributing data across multiple nodes in a cluster. Each node is called a Region Server and is responsible for managing a set of regions, which are portions of the overall dataset. They are the nodes that store the actual data. Each regionServer manages one or more regions of the table. A region is a contiguous range of rows in the table.
- **Master Server**:
  Master Server is responsible for monitoring and managing the Region Servers in the cluster. It performs tasks such as load balancing, failover,and region assignment. It is responsible for managing the metadata of the tables and the RegionServers. It handles operations like table creation, deletion, and splitting.
- **HDFS**:
  HBase stores its data in HDFS, which provides a fault-tolerant and scalable distributed file system for storing large datasets.
- **HBase client**:
  Applications that need to access data stored in HBase interact with the database through the HBase client API. The HBase client communicates with the HBase cluster to retrieve and store data.

The data in HBase is organized into tables, which are divided into rows and columns. Each row in a table has a unique row key that identifies it, and each column in a row is identified by a column family and a column qualifier. HBase supports automatic sharding of tables, which means that as the size of the dataset grows, it can be split across multiple Region Servers for improved performance and scalability.

To understand the architecture of Hbase, following concepts should be understood.

- **Write-Ahead Log** The Write-Ahead Log is a file that HBase uses to ensure data durability. Whenever a write operation occurs, HBase writes the data to the WAL first, before the data is written to the MemStore. The WAL is stored on disk and provides a mechanism for recovering data in the event of a node failure or crash. When a RegionServer starts up, it reads the contents of the WAL and applies any outstanding write operations to the MemStore. This ensures that data is not lost due to a node failure or crash.

- **MemStore** The MemStore is an in-memory buffer that holds the most recent writes to a region. Whenever a write operation occurs, HBase writes the data to the MemStore first. The MemStore is a write-intensive cache, optimized for high-throughput write operations. When the MemStore is full, its contents are flushed to disk, creating a new HFile on the local file system. The HFiles are immutable and provide efficient random read access, making them optimized for read-intensive workloads.

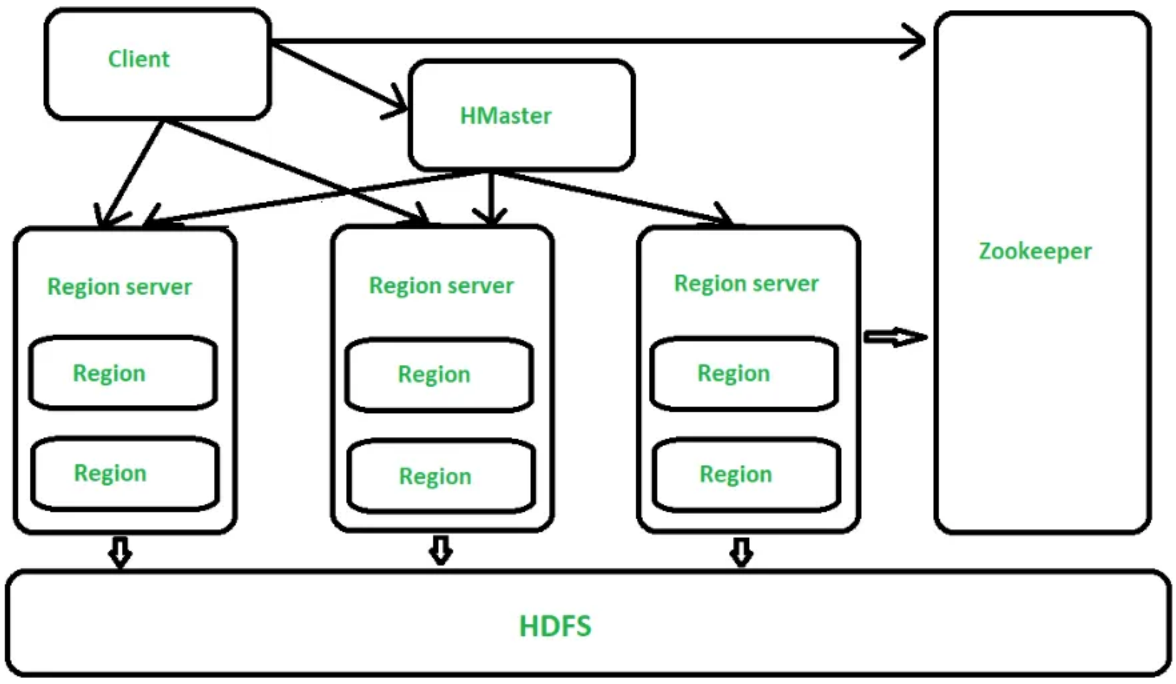

Figure 1: Hbase architecture

1.3 Working of HBase

1.3.1 Write

When a client wants to write data, it sends a write request to the RegionServer responsible for the region that contains the data. The RegionServer validates the write request and, if the request is valid, writes the data to the write-ahead log (WAL) on the local disk for durability. The RegionServer then applies the write to the MemStore. When the MemStore is full, its contents are flushed to disk, creating a new HFile on the local file system. Once the data is written to the local disk, the RegionServer replicates the data to other RegionServers in the cluster.

1.3.2 Read

When a client wants to read data, it sends a read request to the RegionServer responsible for the region that contains the data. The RegionServer reads the data from the HFiles on its local file system or from the MemStore if the data has not yet been flushed to disk. If the requested data is not in the RegionServer's local cache, it retrieves the data from other RegionServers in the cluster through a process called data locality. The RegionServer returns the data to the client.

1.4 Conclusion

Overall, HBase uses a distributed architecture where data is partitioned into regions and stored on multiple RegionServers. This architecture provides high availability, fault tolerance, and scalability. HBase also provides efficient data locality to minimize the network traffic required for read operations.

2 Casasndra

2.1 Introduction

Cassandra is a distributed, non-relational database management system designed to handle large amounts of structured and unstructured data across multiple data centers. It is based on a peer-to-peer architecture that allows nodes to communicate with each other without a central coordinator. It is a distributed database that is designed to handle large amounts of data across multiple commodity servers. It has a masterless architecture, where all nodes are equal and communicate with each other. The architecture of Cassandra consists of:

- Node
- Datacenter
- Cluster
- Coordinator

2.2 Architecture

The architecture of Cassandra consists of following components:

- **Node :**
  A node in Cassandra is a physical or virtual machine that runs the Cassandra software. It is the basic building block of the cluster. Each node is responsible for storing a portion of the data, and can communicate with other nodes in the cluster to retrieve or update data.
- **Datacenter :**
  A datacenter is a logical grouping of nodes that are geographically close to each other and are configured to be highly available. Each datacenter in a cluster can be configured with its own replication factor, consistency level, and other settings.
- **Cluster :**
  A cluster in Cassandra is a collection of one or more datacenters that work together to provide fault-tolerant and scalable storage for data.
- **Coordinator :**
  It is responsible for handling client requests and routing them to the appropriate nodes.

The architecture of Cassandra also involves:

- **Commit Log :**
  Cassandra uses a commit log to ensure data durability. When data is written to Cassandra, it is first written to the commit log on disk before it is written to the in-memory data structures.

- **Memtable :**
  The memtable is an in-memory data structure that stores data that has been recently written to Cassandra. Data in the memtable is periodically flushed to disk to ensure data durability.

- **SSTable :**
  An SSTable (Sorted String Table) is a disk-based data structure that stores data that has been flushed from the memtable. SSTables are immutable, meaning that once they are written to disk, they cannot be modified.

- **Partitioner :**
  Cassandra uses a partitioner to distribute data evenly across nodes in the cluster. The partitioner is responsible for generating a token for each row in a table, which determines which node the data should be stored on.

- **Replication :**
  Cassandra uses replication to ensure data availability and fault tolerance. Each datacenter in the cluster can be configured with its own replication factor, which determines how many copies of each piece of data should be stored in the datacenter.

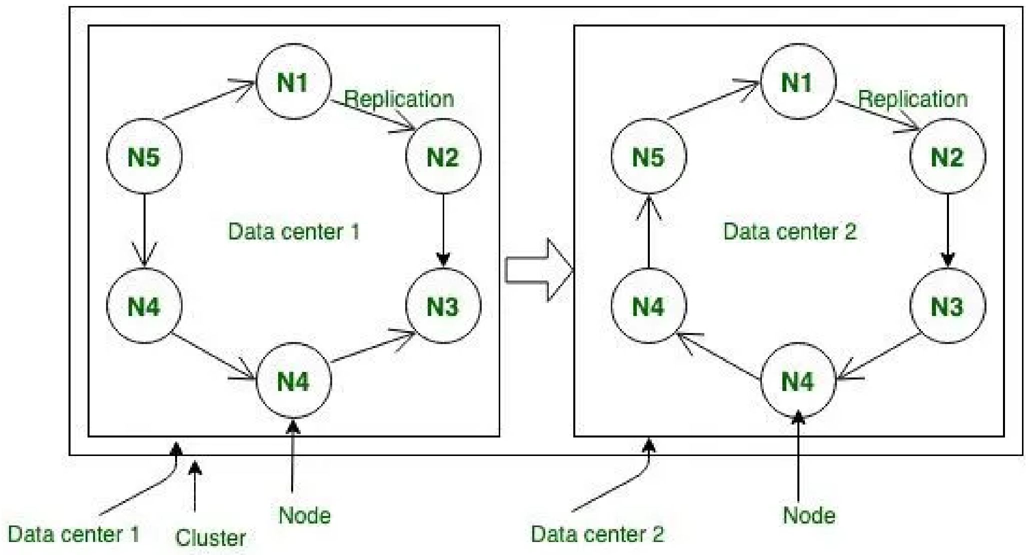

Figure 2: Cassandra Architecture

2.3 Working of Cassandra

2.3.1 Write

When a client wants to write data, it sends a write request to one of the nodes in the cluster, called a coordinator node. The coordinator node determines the nodes responsible for storing the data based on the hash value of the partition key of the data. The coordinator node then sends a write request to all the replica nodes that are responsible for storing the data. The replica nodes acknowledge the write operation once the data has been successfully written to their local disk.

Once enough replicas has acknowledged the write, the coordinator node sends an acknowledgment to the client, indicating that the write was successful.

2.3.2 Read

When a client wants to read data, it sends a read request to one of the nodes in the cluster, called a coordinator node. The coordinator node determines the nodes responsible for storing the data based on the hash value of the partition key of the data. The coordinator node then sends a read request to all the replica nodes that are responsible for storing the data. The replica nodes return the data to the coordinator node. The coordinator node waits for the required number of replicas to respond based on the consistency level specified by the client. Once the required number of replicas has responded, the coordinator node returns the data to the client.

2.4 Conclusion

Overall, Cassandra uses a decentralized architecture and consistent hashing to distribute data across multiple nodes in the cluster. Each node acts as a coordinator for some data and as a replica for other data. This architecture provides high availability, fault tolerance, and scalability.

3 MongoDB

3.1 Introduction

MongoDB is a NoSQL document-oriented database that is designed to store and manage unstructured data. Its architecture is based on a distributed cluster model, and it supports horizontal scaling through the use of sharding. It stores data in collections and documents. It is designed to be highly scalable and flexible. The architecture of MongoDB consists of the following components:

- Replica Set
- Shard
- Config Servers
- Router

3.2 Architecture

- **Replica Set :**
  A replica set is a group of MongoDB servers that maintains the same data set. They work together to provide redundancy and high availability. A replica set typically consists of three or more servers, with one server designated as the primary and the others as secondary servers.
- **Shard :**
  A shard is a horizontal partition of data across multiple servers. Each shard is responsible for storing a subset of the data.
- **Config Servers :**
  Config Servers are responsible for maintaining the metadata of the cluster, such as the location of the shards and the mapping of data to shards.
- **Router :**
  It is responsible for routing client requests to the appropriate shards and maintaining a connection to the config servers.

To understand the architecture of MongoDB following concepts should be understood.

- **MongoDB Cluster :**
  A MongoDB cluster consists of one or more MongoDB servers that are deployed on different physical or virtual machines. The servers in the cluster communicate with each other to manage data and provide high availability and fault tolerance.
- **Sharding :**
  Sharding is the process of partitioning a large dataset across multiple servers. In MongoDB, sharding is used to horizontally scale data across multiple servers in a cluster. Sharding is achieved by splitting a dataset into smaller chunks called shards and distributing them across multiple servers.

- **Documents :**
  MongoDB stores data in the form of documents, which are JSON-like objects that contain data and metadata. Documents are grouped into collections, which are similar to tables in a relational database.

- **Indexes :**
  MongoDB supports the creation of indexes to improve the performance of data retrieval operations. Indexes are created on specific fields in a collection and are used to quickly locate data in the collection.

- **WiredTiger Storage Engine:**
  MongoDB uses the WiredTiger storage engine as the default storage engine. WiredTiger uses a document-level concurrency control approach to provide high concurrency and performance.

- **Drivers :**
  MongoDB provides drivers for many programming languages, which allow applications to interact with the MongoDB database.

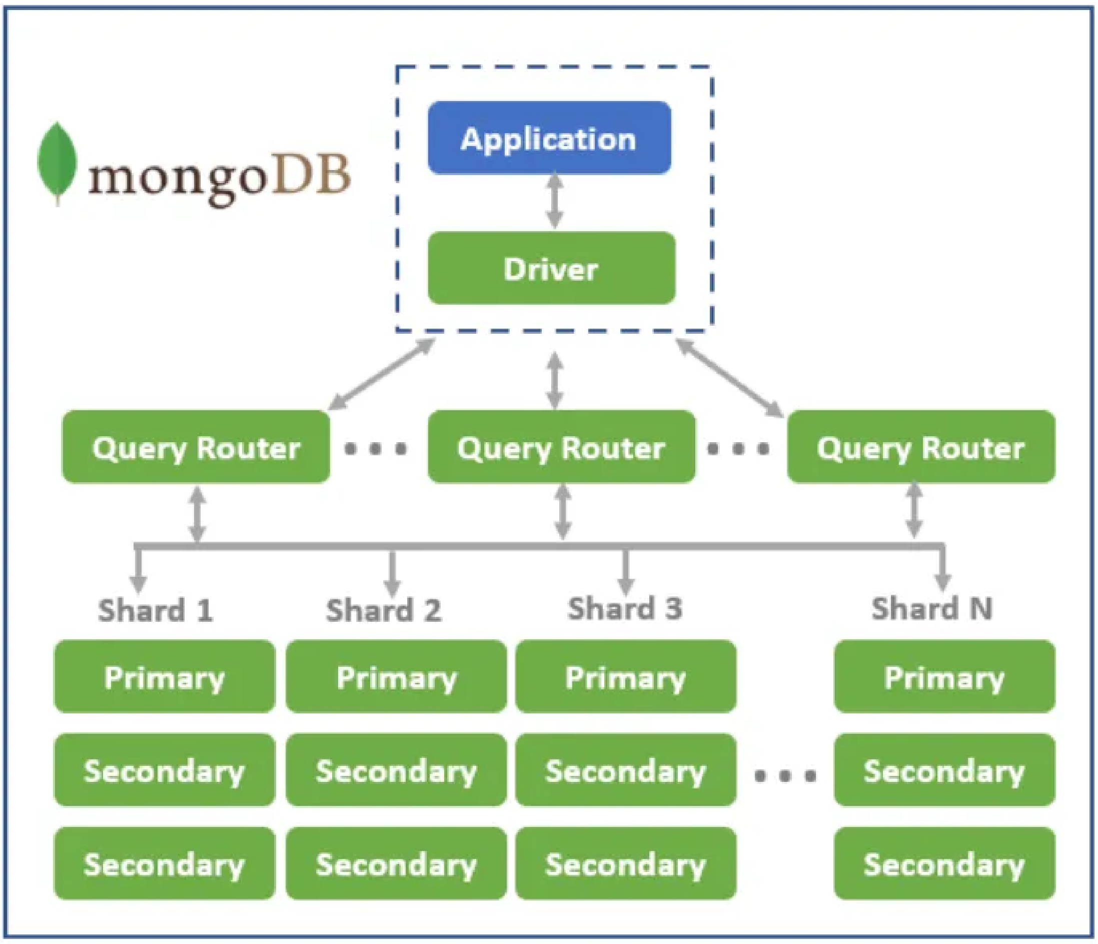

Figure 3: Architecture of MongoDB

3.3 Working of MongoDB

3.3.1 Write

When a client wants to write data, it sends a write request to the primary node in the replica set. The primary node validates the write request and, if the request is valid, writes the data to its local storage engine. Once the data is written to the primary node, the primary node replicates the data to the secondary nodes in the replica set. Once enough nodes has acknowledged the write, the primary node sends an acknowledgment to the client, indicating that the write was successful.

3.3.2 Read

When a client wants to read data, it sends a read request to any node in the replica set. If the client specifies a read preference, the request will be sent to a node that satisfies the preference. If the requested data is not in the node's memory, the node reads the data from the local storage engine or, if the node is a secondary node, from the primary node. The node returns the data to the client.

3.4 Conclusion

Overall, MongoDB uses a replica set architecture to provide high availability and fault tolerance. The primary node handles all write operations and replicates data to secondary nodes, while any node in the replica set can handle read operations. MongoDB also supports sharding, which allows data to be distributed across multiple nodes in a cluster, providing scalability.

4 Conclusion

In conclusion, HBase, Cassandra, and MongoDB are NoSQL databases that have different architectures and usage. HBase is column-oriented and built on Hadoop, Cassandra is a masterless architecture, and MongoDB is document-oriented. Understanding the architecture of each database can help in selecting the appropriate one for given use case. It is important to note that each database has its strengths and weaknesses. HBase is ideal for read-heavy workloads, Cassandra is designed for write-heavy workloads, and MongoDB is great for flexibility and ease of use. When selecting a database, it is important to consider specific use case and the requirements of given application. NoSQL databases are becoming increasingly popular due to them being designed to be highly scalable and flexible, making them ideal for modern use cases.