MongoDB Replication and Sharding- A Complete Introduction

BEGINNER DATA ENGINEERING DATABASE NOSQL PROGRAMMING SQL

This article was published as a part of the Data Science Blogathon.

Introduction

A NoSQL database is a non-relational database that does not use the traditional table-based schema of a relational database. NoSQL databases are often used for big data and real-time web applications.

The main advantages of using a NoSQL database are that NoSQL databases can handle large amounts of data much more efficiently than relational databases. These databases are highly scalable, meaning they can be easily scaled up or down to meet the needs of a particular application.

Further, NoSQL databases are generally much easier to use and more flexible than relational databases, making them a good choice for applications that require a high degree of flexibility. They are often much faster than relational databases, making them a good choice for applications that require real-time response times.

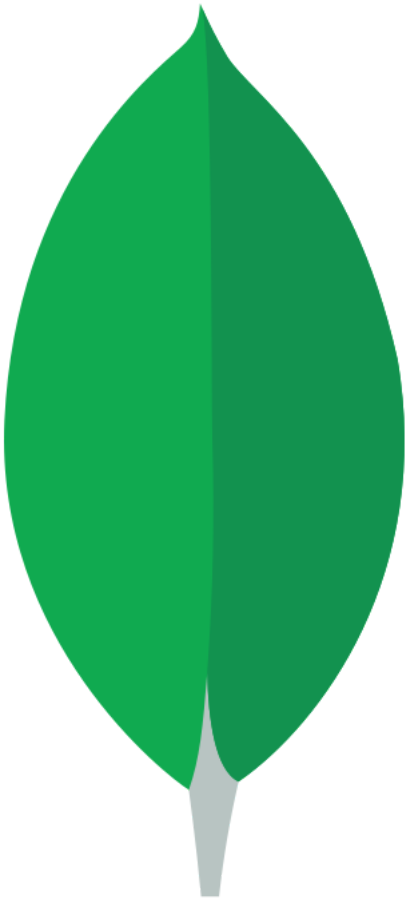
What is MongoDB Database?

MongoDB is a powerful document-oriented database system. It has an index-based search feature that makes data retrieval quick and easy. MongoDB also offers a scalability feature, handling extensive scale data. To get an in-depth knowledge of MongoDB, click here.

Unlike SQL databases, which store data in tables of rows or columns, each record in a MongoDB database is a document specified in BSON, a binary representation of the data. This information can then be retrieved by applications in JSON format.

Document databases are extremely adaptable, allowing for variations in document structure and the storage of partially completed documents. Others can be embedded in a single document. Fields in a document function similarly to columns in a SQL database, and they, too, can be indexed to improve search performance.

MongoDB was founded on a scale-out design, which allows many small machines to collaborate to develop fast systems and process massive volumes of data.



Source – commons.wikimedia.org

In this article, we will discuss Database Replication and Sharding. We will also discuss how MongoDB implements these techniques to make it Available, Consistent, and Partition Tolerant.

What is Database Replication?

Database replication is the process of copying data from a database on one server to a database on another server. We can do this for several reasons, such as to keep a backup of the data in case the original is lost or corrupted or to allow multiple people to access the same data from different locations.

There are a few different ways to replicate databases, but the most common is to use a tool like MySQL replication. This allows the data on the original server to be copied to the new server without any manual intervention.

There are several benefits to replicating databases, but the most important is that it helps keep data safe. If the original database is lost or corrupted, the replica can be used to restore it. This is especially important for businesses that rely on their data to function.

Another benefit is that it can improve performance. If multiple people need to access the same data, replicating it can help reduce the load on the original server.

What is the Need for Database Replication?

As your business grows, you will inevitably need to scale your database to accommodate the increased load. One way to scale a database is to replicate it. Database replication is the process of copying data from one database to another.

There are several reasons why you should replicate your database. Replication can improve performance by distributing the load across multiple servers. It can also improve availability by providing a backup in

case one server fails. Finally, it can provide disaster recovery in a complete system failure.

No matter your reasons for replicating your database, it is essential to ensure that the process is done correctly. Incorrect replication can lead to data loss or corruption. Therefore, it is crucial to understand the basics of database replication before attempting to implement it.

What is CAP Theorem in Database?

In database systems, the CAP theorem states that it is impossible for a distributed system to simultaneously provide more than two out of the following three guarantees:
1. **Consistency:** All nodes see the same data simultaneously.
2. **Availability:** Every request receives a success or failure response.
3. **Partition tolerance:** The system continues to operate despite an arbitrary number of failed nodes.

The CAP theorem often explains the trade-offs between different database systems. For example, a database system that prioritizes consistency over availability is said to be “CP,” while a system that prioritizes availability over consistency is “AP.”

How MongoDB Performs Replication?

MongoDB uses a simple replication strategy called “master-slave” replication. In this strategy, one server is designated as the primary server, and all other servers are slaves. The primary server receives all write operations, which are then replicated to the slaves.

MongoDB uses an oplog to keep track of changes to the data on the master node. The oplog is a capped collection containing all the operations applied to the data on the master node. When a slave node starts up, it reads the oplog from the master node and applies the operations to its own copy of the data.

Read operations can be performed on any server, but slaves generally have slightly outdated data compared to the primary. This is because writes are only replicated to slaves after being committed to the primary’s journal.

If the primary server goes down, one of the slaves will be automatically promoted to primary, and write operations will resume as normal.

MongoDB uses a vote-based replication protocol to determine which node is the primary node. The primary node is the node that receives all the write operations. The other nodes in the replica set are called secondary nodes. Secondary nodes can serve read operations but cannot process write operations.

Overall, this replication strategy is quite simple and effective, but it does have some drawbacks. For example, if the primary server is down, all write it will block operations until it comes back up. Additionally, if there is a large amount of write traffic, slaves can fall behind and may have outdated data.

Source – progressivecoder.com

What is Database Sharding?

Database sharding is a horizontal partitioning of data in a database. Each partition is known as a shard. Database sharding is a popular approach to scaling out data stores. It is often used with NoSQL databases and extensive data systems.

One of the critical benefits of database sharding is that it allows for horizontal scalability. This means more nodes can be added to support increasing data and traffic. Database sharding can improve performance by distributing data and queries across multiple nodes.

A few challenges need to be considered when using database sharding. One is that Sharding can introduce data inconsistencies if not done correctly. Another is that it can be challenging to change the shard key (the attribute used to determine which shard a piece of data belongs to) once it is stored in the database.

Overall, database sharding is a powerful tool for scaling out data stores. When used correctly, it can provide significant performance improvement.

What is the need for Database Sharding?

The need for database sharding arises because as data grows, it becomes more challenging to store it on a single server. It is because the server has limited storage and processing power. As a result, the performance of the database can start to deteriorate.

Database sharding allows for horizontal scaling, which means more servers can be added to the system as needed. It will enable the system to handle more data and improve performance.

There are a few things to consider when sharding a database:

1. The data must be divided up in a way that makes sense.
2. The system must handle queries that span multiple servers.
3. The system must be able to recover from failures.

Database sharding is a powerful tool for scaling out databases. When used correctly, it can handle a vast number of queries without any downtime or inconsistency.

How Does MongoDB Database Perform Sharding?

MongoDB database sharding stores data records in multiple machines and distributes the data evenly across these machines. In MongoDB, Sharding is done at the collection level, and a collection can be sharded across multiple devices.

Source – medium.com

Shards:

In database sharding, a shard is a horizontal partition of data in a database or search engine. Each partition is referred to as a shard or database shard. Horizontal partitioning is a data-sharding strategy where rows from a database table are stored in different database servers.

Config Servers:

A config server is a server that stores configuration data for a system. In a sharded system, a config server is a server that stores configuration data for all of the shards in the system. The system uses the config server to determine which shard a particular data is stored.

A config server can be a dedicated server, or it can be a shared server. In a sharded system, it is essential to have a dedicated config server so that the system can continue to function even if one of the shards goes down.

The config server is also responsible for handling failover in a sharded system. If one of the shards goes down, the config server will automatically route traffic to the next available shard.

Config servers are an essential part of a sharded system. Without a config server, the system would not be able to function.

Query Routers:

Query routers are an essential component of Sharding, as they are responsible for routing queries to the appropriate shard. With query routers, it would be easier to determine which shard a particular query should be sent to, and data would be more difficult to retrieve.

Query routers typically use a consistent hashing algorithm to determine which shard a particular query should be routed to. This algorithm considers the key of the data being queried and the number of shards in the system. By mapping the key to a particular shard, the query router can ensure that all queries for a given key will be routed to the same shard.

There are a few different implementations of query routers, but they all serve the same purpose: to route queries to the appropriate shard. The system's specific needs will likely determine the choice of query router.

Conclusion

In this article, we have discussed how we can achieve replication and Sharding in MongoDB Database. These two techniques are essential to take care of when designing systems that can handle millions of users. This makes our system available 24×7, consistent, and partition tolerant to some extent.

Companies like Google is pretty much the gold standard when it comes to handling downtime and consistency. They have a team of engineers who are constantly monitoring their systems and working to keep them up and running. If there is ever an issue, they are quick to communicate it to users and work to resolve the issue as quickly as possible. In terms of consistency, Google is also top-notch. They work hard to keep their systems running smoothly and efficiently and always look for ways to improve.

Major Points of this article:
1. Discussed NoSQL Databases and their importance.
2. Then we discussed Database Replication, its importance, and how MongoDB replicates in the database.
3. Finally, we discussed Database Sharding and how MongoDB performs and concluded the article.

It is all for today. Thanks for reading.

Article Url - https://www.analyticsvidhya.com/blog/2022/12/mongodb-replication-and-sharding-a-complete-introduction/

Aditya Garg