Day_2

Mongodb Query Architecture

Ajit Yadav DBA

Preparation :

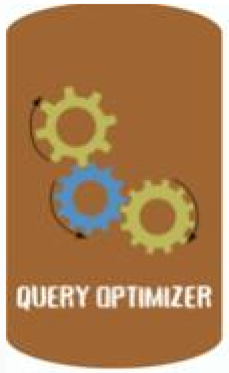
Parse and analyzed to determine the optimal execution plan .
check the query syntax validating the query against the scheme

Optimization :

query optimizer evaluate the available indexes .
select the most efficient execution plan based on the query predicates

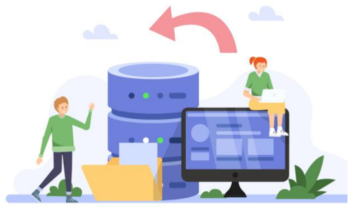
Execution :

execute the query it scan the select indexes or collections to locate the matching document based on the query filter



mongod(daemon)
This process which take care of everything

Retrive :



Retrive the matching document from disk / memory in batches

Processing :

involves iterating over the result extracting the data

Completion :

the matching document have been retried & processed the find query is completed or close the db connection

Query Preparation:

When you execute a find query in MongoDB, the query is first parsed and analyzed to determine the optimal execution plan.
This step involves checking the query syntax, validating the query against the schema, and determining the indexes that can be utilized.

Query Optimization:

MongoDB’s query optimizer evaluates the available indexes and selects evaluates the available indexes and selects the most efficient execution plan based on the query predicates, index statistics, and data distribution. The goal is to minimize the number of documents examined and maximize the use of indexes.

Query Execution:

Once the optimal execution plan is determined, MongoDB executes the query. It scans the selected indexes or collections to locate the matching documents based on the query filters and projections.

Document Retrieval:

As MongoDB executes the query, it retrieves the matching documents from disk or memory, depending on the storage configuration. The retrieved documents are returned in batches, where the batch size can be configured by the client.

Result Processing:
The client application receives the batches of documents and processes them as desired. This may involve iterating over the results, applying additional filtering or transformations, and extracting the required data.

Query Completion:
Once all the matching documents have been retrieved and processed, the find query is considered complete. The client can then continue with further operations or close the database connection.

It’s important to note that the execution time of a find query can vary depending on factors such as the size of the collection, the complexity of the query, the presence of suitable indexes, the available system resources, and the network latency between the client and the database server.

Additionally, MongoDB provides various query modifiers, such as sort, limit, and skip, which can affect the query execution and result retrieval process. These modifiers allow you to control the ordering of results, limit the number of documents returned, and skip a certain number of documents.

Overall, MongoDB’s find query life cycle involves query preparation, optimization, execution, document retrieval, result processing, and query completion. Understanding this process can help you optimize your queries and improve their performance.

The MongoDB query optimizer is a component of the MongoDB database that is responsible for selecting the most efficient query execution plan for a given query. It analyzes the query and determines the best approach to retrieve and manipulate the requested data from the database.

Here is a high-level overview of how the MongoDB query optimizer works:

Query Parsing:
When a query is received, the query optimizer first parses the query to understand its structure, including the fields being queried, the filtering conditions, and any sorting or aggregation operations.

Query Analysis:
The optimizer then performs an analysis of the query to gather relevant statistics and metadata about the collections and indexes involved. This information includes the size of the collections, the distribution of values in the fields, and the indexes available.

Query Planning:
Based on the analysis, the optimizer generates multiple potential query execution plans. Each plan represents a different way to execute the query, utilizing different indexes and operations.

Cost Estimation:
The optimizer estimates the execution cost for each query plan. The cost estimation takes into account factors such as the number of disk reads, network transfers, memory usage, and CPU operations required for each plan.

Plan Selection:
The optimizer compares the estimated costs of the query plans and selects the one with the lowest cost. The goal is to minimize the overall resource usage and execution time.

Execution and Feedback:
The chosen query plan is executed, and during the execution, the optimizer collects additional runtime statistics. These statistics help improve future query planning by providing information about the actual performance of the chosen plan.

Plan Caching:
If a query plan is executed multiple times, MongoDB caches the plan in memory to avoid the need for re-optimization. However, if the data or indexes change significantly, the query optimizer may re-evaluate the cached plan and potentially select a different plan.

By selecting the most efficient query execution plans, the query optimizer in MongoDB helps improve the overall performance and responsiveness of database queries.

In MongoDB, flushing refers to the process of writing data from memory to disk. This is an important aspect of database management to ensure data durability and consistency, especially in the event of a system crash or restart. The flushing process in MongoDB is closely related to the concept of write operations and data persistence. Here’s how flushing happens in MongoDB:

Write Operations and Memory:

When you perform write operations (such as inserts, updates, or deletes) in MongoDB, the data is initially written to memory (RAM) in a data structure called the WiredTiger cache. This cache holds recently modified or inserted data before it is persisted to disk.

Write Concern and Durability:

MongoDB provides a concept called “write concern,” which allows you to control the level of data durability you want to achieve. A write concern specifies how many nodes need to acknowledge the write before it is considered successful. The default write concern, often referred to as “acknowledged,” ensures that the write operation is acknowledged by the primary node.

Journaling:

MongoDB uses a journaling mechanism to ensure data durability. Journaling involves writing changes to an on-disk journal before they are written to the main data files. This helps in recovering data in the event of a crash. The journal is flushed to disk periodically, and MongoDB also has options to control the frequency of journal commits.

Flush to Disk:

The data in the WiredTiger cache is periodically flushed to disk. This flushing process is managed by the storage engine (WiredTiger in most cases). The frequency of flushing and the amount of data flushed depend on various factors, including system resources, workload, and write concern settings.

Checkpoints:

WiredTiger maintains checkpoints, which are consistent snapshots of the data files. During a checkpoint, modified data in memory is written to the data files on disk. This process helps ensure that the data on disk is consistent and can be used for recovery.

Background Threads:
MongoDB uses background threads to manage various tasks, including flushing data to disk. These threads ensure that data is periodically written from memory to disk, while also optimizing the I/O operations to minimize performance impact on the main application threads.

WAL (Write-Ahead Logging):

MongoDB’s storage engine uses a write-ahead logging mechanism to record changes before they are applied to the data files. This helps maintain the durability of writes and provides a way to recover data after a crash.

Periodic Maintenance:

MongoDB performs periodic maintenance tasks, such as cleaning up unused space and optimizing storage structures. These tasks can also involve flushing data to disk as part of the optimization process.

Overall, the flushing process in MongoDB is designed to balance the performance of write operations with data durability. The frequency and mechanisms of flushing can be influenced by configuration settings, write concern options, and workload characteristics. It’s important to configure your MongoDB deployment appropriately based on your application’s requirements and the desired level of data durability.