UNIT -III

MongoDB Best Practices:

MongoDB Limitations: Deployment, Hardware Suggestions from the MongoDB Site, Few Points to be Noted, Coding, Application Response Time Optimization, Data Safety, Administration, Replication Lag, Sharding, Monitoring

PYQ : (Previous Year Mumbai University Question)

Nov – 18

1 : - ------

Apr -19

1 : Explain Hardware requirement for MongoDB

2 : Write a short note on Deployment

Nov – 19

1: Define Monitoring . Explain the factors to be considered while using Monitoring services

Nov – 22

1. Write a short note on Deployment
2. What are the tips need to be considered when coding with the MongoDB Database

Dec – 23

1. Write a short note on performance monitoring of MongoDB query

Nov – 24

1. How can application response time be optimized to ensure instantaneo responses us for better user experience?
2. Explain the monitoring mechanism of MongoDB .

"Getting started with MongoDB is easy, but once you start developing applications you will come across scenarios where you may need best practices to achieve particular use cases."

Deployment :

- While deciding on the deployment strategy , keep the following tips in mind so that the hardware sizing is done appropriately. These tips will also help you decide whether to use sharding and replication .

- **Data set size:** It's crucial to know the current and future size of your data. This helps you choose the right resources for each server and plan for sharding if needed.

- **Data importance:** The next crucial step is to figure out how important the data is and how much data loss or delay you can tolerate, especially when using replication.

- **Memory sizing:** The next step is to figure out how much memory is needed and then make sure the RAM meets those needs.

    - "MongoDB works best when all the data can fit in memory, so it doesn't need to access the disk."
    - Page faults show that your deployment might be running out of memory, so you should think about adding more. You can track page faults using monitoring tools like MongoDB Cloud Manager.
    - It's best to choose a platform with more memory than the amount of data you work with.
    - If your data is too large for one server's memory, you should use sharding to spread the data across multiple servers. This way, you can increase the available memory and improve the system's performance.
- **Disk Type :**
    - If speed isn't your top concern or your data is too large for memory, pick the right disk. Disk speed is measured by IOPS (input/output operations per second)—higher IOPS means better MongoDB performance. Use local disks for faster access, as network storage can cause delays. If possible, use RAID 10 for better performance.
- **CPU :**
    - If you plan to use map-reduce, the CPU's clock speed and the number of processors matter. A higher clock speed improves performance, especially when most data is in memory. To maximize operations per second, choose a CPU with a high clock/bus speed.

- **Replication :** ensures high availability by using multiple copies of data. In MongoDB, it's common to set up a replica set with at least three nodes. A typical setup involves two nodes in one data center and one backup node in another data center

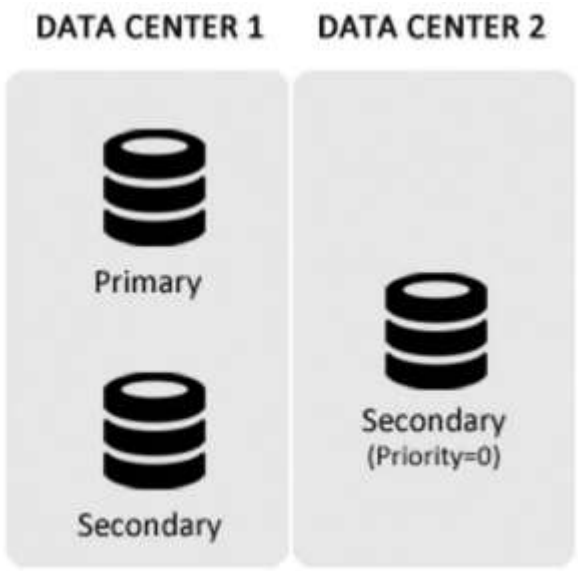

Figure 12-1. *MongoDB 2*1 deployment*

Hardware Suggestions from the MongoDB Site:

These are general guidelines for setting up MongoDB. Your actual hardware needs depend on your data, how you need it to be available, the types of queries, and the hardware's capabilities.

- **Memory:** More memory means better performance, as MongoDB uses a lot of it.

- **Storage:** Use SSDs for faster performance since they handle MongoDB's disk access patterns better. SSDs also help if your data doesn't fit in memory. RAID-10 is recommended for most setups. If you use the WiredTiger storage engine, XFS file system is best. Avoid huge pages; default virtual memory pages work better.

- **CPU:** For MMAPv1, faster clock speeds are better than many cores. For WiredTiger, more cores improve performance.

Few Points to be Noted :

Sure, here's a simplified explanation for each point:

1. **Faster CPUs and more RAM**: Faster processors and more memory make MongoDB work better.

2. **More cores and MMAPv1**: Adding more CPU cores helps but doesn't significantly boost performance with the MMAPv1 storage engine.

3. **SATA SSDs and PCI drives**: SATA SSDs and PCI drives provide good performance for their price.

4. **SATA spinning drives**: Regular SATA hard drives are a budget-friendly option that works well.

5. **NUMA hardware and MongoDB**: MongoDB doesn't perform well on NUMA hardware, so it's best to turn off NUMA and use an interleave memory policy on Linux.

Coding :

Once the hardware is acquired, consider the following tips when coding with the database:

Coding Tips for Databases:

1. **Data Model**: Choose whether to embed or reference data based on performance needs. Avoid patterns that cause documents to grow beyond 16MB.

2. **Document Size**: Avoid designs that make documents grow indefinitely. MongoDB can move documents if they exceed size limits, which slows performance.

3. **Future-Proofing**: Create all potential fields in documents upfront if possible. This avoids delays when MongoDB needs to find new space.

4. **Anticipated Size**: Insert documents with a temporary field of expected size, then remove it. This helps manage space effectively.

5. **Subdocuments vs Arrays**: Use subdocuments if you always know the field names; otherwise, use arrays.

6. **Explicit Information**: Include necessary computed data in documents rather than relying on client-side computation.

7. **Avoid $Where**: It's slow and resource-heavy.

8. **Data Types**: Use appropriate data types (e.g., store numbers as numbers, not strings).

9. **Case Sensitivity**: MongoDB string searches are case-sensitive. Normalize cases or use regex for searches.

10. **Unique _id**: Use a unique key as _id to save space and aid indexing, but ensure it's unique and consider insertion order.

11. **Retrieve Only Needed Fields**: Fetch only the fields you need to improve performance.

12. **GridFS**: Use for large files (e.g., videos) that can't fit in a single document.

13. **TTL**: Automatically delete documents after a set time using TTL (Time To Live).

14. **Capped Collections**: Use for high-throughput data that requires a rolling window (e.g., logs).

15. **Data Consistency**: Monitor for consistency issues due to MongoDB's flexible schema.

16. **Handle Failures**: Ensure your application can handle exceptions and failovers gracefully.

Optimizing Application Response Time :

To ensure your application responds quickly, follow these tips:

- **Minimize Disk Access:** Avoid frequent disk reads by keeping data in memory and increasing memory size to prevent page faults.

- **Create Indexes:** Index fields that are frequently queried to reduce memory usage and speed up queries.

- **Use Covering Indexes:** For queries that return a few fields, create indexes that cover those fields to speed up performance.

- **Use Compound Indexes:** A single compound index for multiple queries saves memory compared to several individual indexes.

- **Apply Trailing Wildcards:** Use trailing wildcards in regular expressions to improve index performance.

- **Be Selective with Indexes:** Indexes should be balanced; remove unused ones to avoid unnecessary costs, especially for write-heavy operations.

- **Design Hierarchical Documents:** Group related data together to help MongoDB find information efficiently.

- **Query Efficiently:** Use small result sets first with AND queries and large result sets first with OR queries to reduce search time.

- **Fit Working Set in Memory:** Ensure your working set (active data) fits in memory to improve performance.

- **Use WiredTiger for Heavy Writes:** Opt for the WiredTiger storage engine for applications with lots of writes and I/O operations due to its efficient compression features.

Data Safety Tips :

- **Use Replication and Journaling**: For better data safety, use replication rather than a single server. If you can't use replication, make sure journaling is enabled on a single server.

- **Repair as a Last Resort**: Only use repair if absolutely necessary. It might not restore all your data.

- **Set W for Safe Writes**: In a replica setup, set the write concern (W) to a majority of the servers to ensure data is safely replicated, even if it slows down writes.

- **Always Use wtimeout**: Specify a timeout with your write concern to avoid long waits.

- **Secure Your MongoDB**: Run MongoDB in a secure environment to prevent unauthorized access.

Administrations :

Here’s a simplified version of the administration tips:

- **Backups**: For durable servers, take backups by either using a file system snapshot or fsync+lock and then dump the database. Don't just copy files without locking, as copying isn't instant.

- **Repair**: Use the repair command to compact databases. It performs a mongodump and mongorestore to create a clean copy of your data and remove any empty spaces.

- **Database Profiler**: MongoDB's profiler logs detailed information on database operations. It can log all events or only those taking more than 100ms by default.

- **Explain Plan**: Use explain plans to see how a query is processed. It shows details like the index used, number of documents returned, and query execution time. If a query completes in under 1ms, the plan shows 0. When you call an explain plan, it tests indexes to find the best one.

Replication Lag :

Replication Lag Simplified

Replication lag is the delay between when an operation happens on the primary server and when it appears on the secondary server. If this lag is high and keeps increasing, there may be a problem. You might need to pause the system, manually fix issues, or deal with outdated data.

To check the current replication lag, use:

```shell
rs.printSlaveReplicationInfo()
```

For more details, use:

```shell
rs.printReplicationInfo()
```

You can also view replication lag in MongoDB Cloud Manager under the Status tab of each secondary node.

Tips to reduce replication lag:

- Ensure your secondary server is as powerful as the primary and has sufficient network bandwidth.
- Adjust the write concern of your application.
- Schedule index builds on the secondary during low write times.
- Take backups without blocking if the secondary is used for that.
- Check for errors using `rs.status()` and review secondary server logs.

Sharding :

When data is too large for one server, sharding spreads it across multiple servers so that performance remains good.

- Choose an effective shard key.
- Use three config servers for backup.
- Shard collections before they get too big (256GB).

Monitoring MongoDB:

- **Proactive Monitoring:** Keep an eye on MongoDB to spot issues early and fix them. Use tools like MongoDB Cloud Manager for a detailed view or create your own with Nagios, SNMP, or Munin.

- **MongoDB Tools:** Use `mongostat` and `mongotop` to check performance.

Key Metrics to Watch:

- **Op Counters:** Track operations like inserts, deletes, reads, and updates.

- **Resident Memory:** Ensure allocated memory is less than physical memory to avoid performance slowdowns.

- **Working Set Size:** The data actively used should fit into memory. Optimize queries or increase memory if needed.

- **Queues and Locks:** Watch for queues and lock percentages to avoid contention. MongoDB 3.0+ has better concurrency with collection and document-level locking.

- **Performance Testing:** Test with a full-size database to identify potential issues before going live.