"COMP41720 - Distributed Systems Lab 2" "Hairui Wang (23226138)"

Exploring Replication and Consistency in MongoDB

Table of Contents

1. Introduction

This experiment focuses on understanding how MongoDB handles replication, write concerns, and data consistency in a distributed environment.
The goal was to build a small replica set using Docker containers, observe how data propagates between nodes, and explore how different write concerns affect performance and reliability.

Through this lab, I aimed to connect theoretical knowledge about **CAP principles** with practical system behavior — seeing how a real system balances consistency, availability, and partition tolerance during replication and failover events.

2. Experimental Setup

All experiments were conducted using a three-node MongoDB replica set and a Spring Boot application running inside a Docker environment.

Component	Configuration
MongoDB Nodes	`mongodb1`, `mongodb2`, `mongodb3`
Replica Set Name	`rs0`
Application	Spring Boot (Java 17, Maven)
Network	Docker bridge network `mongo-network`
Dataset	Generated user profiles (id, username, email, last login time)

Each node was bound to a separate port (27017–27019).
The Spring Boot app interacted with the replica set through the `MongoTemplate` interface, executing batch insertions and read operations with different write concerns.

Before running the tests, I verified that all three MongoDB instances joined the same replica set and could elect a primary successfully.

3. Experiments

3.1 Write Concern Comparison

I compared the latency and durability of three levels of write concern:

- `w:1` (acknowledged by primary only)
- `w:majority` (acknowledged by a majority of nodes)
- `w:all` (acknowledged by all replicas)

Each configuration inserted batches of 400 user documents repeatedly.
The total latency for each run was recorded in milliseconds, and the test was repeated 500 times to reduce noise.

Observations:

- `w:1` consistently produced the lowest latency, around 6–8 ms on average.
- `w:majority` was slightly slower (roughly 9 ms), but the difference was smaller than expected.
- `w:all` had the highest latency, though still under 15 ms in most cases.

Interestingly, the latency gap between `w:majority` and `w:all` became more visible when the network was under load, which suggests additional replication coordination overhead.

Interpretation:
Higher write concerns clearly improve durability at the cost of slight latency increases.
For most applications, `w:majority` offers a reasonable balance between reliability and performance.

3.2 Replication Behavior and Failover

To study replication, I first wrote a series of user records to the primary node and observed their appearance on secondaries using `ReadPreference.secondary()`.
Most documents became visible on the secondaries within 100–200 ms.

Next, I triggered a **primary step-down** using the MongoDB admin command:

```
{ replSetStepDown: 15, force: true }
```

During this short period, write operations temporarily failed until a new primary was elected.
In my environment, the election usually completed within 3–5 seconds.
After recovery, no data loss was observed — the new primary contained all previously committed documents.

Reflection:
This part of the experiment made it clear that MongoDB prioritizes **consistency** over availability during elections.
Writes were blocked briefly, but data integrity was fully preserved.

3.3 Consistency Models

Finally, I tested three types of read/write consistency through controlled operations:

- **Strong Consistency:** Writes with `WriteConcern.MAJORITY` and reads using `ReadConcern.MAJORITY`.
  Reads from secondary nodes returned the updated value immediately after the write, confirming synchronous replication.
- **Eventual Consistency:** Writes with `WriteConcern.W1` and reads from secondary nodes.
  Occasionally, new documents appeared on the secondary after a short delay (typically 200–400 ms).
- **Causal Consistency:** Using causally consistent sessions, dependent operations respected ordering — the “cause” document always appeared before the “effect”.
  This worked as expected under normal conditions, though sometimes verification required several read attempts.

Insight:
The small propagation delay in eventual consistency is noticeable but tolerable for non-critical data.
Strong consistency guarantees immediate visibility but sacrifices a bit of throughput.

---

4. Results and Analysis

Metric	w:1	w:majority	w:all
Average Latency (ms)	~7	~9	~13
Data Durability	Low	High	Very High
Availability During StepDown	Partial	Partial	Limited

The results confirmed MongoDB’s design philosophy:
it leans toward **Consistency + Partition Tolerance (CP)** in the CAP trade-off, temporarily reducing availability during elections to ensure data correctness.

From a developer’s perspective, this means MongoDB is safe for applications that cannot tolerate data loss, even if it briefly pauses writes during failover.

---

5. Challenges and Observations

- At first, my replica set initialization failed due to hostname mismatches; I fixed it by explicitly binding to `--bind_ip_all`.
- When I forced a step-down, some test threads threw exceptions — this helped me realize that transient errors are normal during elections.
- I also noticed that `w:1` operations can complete before the data reaches secondaries, which explains why a read from `ReadPreference.secondary()` might not find the latest record.
- Overall, MongoDB handled recovery gracefully without manual intervention.

These details helped me understand the difference between **logical data safety** (what the client expects) and **physical replication delay** (what the system actually guarantees).

---

6. Conclusion

This lab gave me practical experience in how distributed databases achieve replication and consistency under different settings.
By running real tests and observing behavior during failover, I learned how MongoDB implements trade-offs in the CAP theorem.

If I repeated this experiment, I would focus more on automating measurements to capture delay variations over time, and possibly extend the setup to simulate network partitions.

Overall, this lab helped me build a clearer, experience-based understanding of distributed data systems — not only through theory but through actual debugging, measurement, and observation.

---

7. Code Repository

All source code, configurations, and Docker setup files used in this experiment are available at:
https://github.com/harryw712/COMP41720

End of Report
Hairui Wang – 23226138