Ministry of Higher Education and Scientific Research
University of Abdelhafid Boussouf - Mila
Institute of Mathematics and Computer Science
Department of Computer Science
Master 2 I2A – Big Data
2025/2026

Directed Works TD 9 – Advanced MongoDB Architecture & Integration

Exercise 1: High Availability & Replica Sets

1. Failover

- **Steps:**
    1. B and C stop receiving "heartbeats" from A.
    2. B and C hold an **Election**.
    3. If B has the latest data (oplog), it becomes the new Primary.
- **Write Availability: NO.** Writes are paused during the election (usually 2-10 seconds).
- **Read Availability: YES**, *if* the application is configured to read from secondaries (secondaryPreferred). If reading from Primary (default), reads pause too.

2. Split Brain (Network Partition)

- **Node A (Isolated):** It cannot see a majority (1 out of 3 is not a majority). It typically **steps down** and becomes a Secondary (Read-only).
- **Cluster B+C:** They have a quorum (2 out of 3). They elect a new Primary. Service continues.

3. Write Concern (w: "majority")

- **Count:** In a 3-node set, majority is **2**. The write must be written to the Primary AND replicated to at least 1 Secondary.
- **Latency:** Higher latency than w:1. The application waits for the network round-trip to the secondary.
- **Benefit:** Guarantees data is safe even if Primary crashes immediately after.

Exercise 2: Sharding Design (The TechStore)

Goal: Avoid "Hot Spots" (all writes going to one shard) and "Scatter-Gather" queries (querying all shards for one piece of data).

1. _id (Hashed)

- **Verdict: Mediocre.**
- *Pros:* Great distribution of writes (random hash).
- *Cons:* Bad for queries. "Find User X" requires checking *every* shard (Scatter-Gather) because _id has no relation to user_id.

2. order_date (Ranged)

- **Verdict: BAD (Critical Fail).**

- *Why:* **Monotonically Increasing.** All new orders come in with "today's date". They will all go to the *last* chunk on the *last* shard.
- *Result:* 1 shard takes 100% of the write load (Hot Spot), while the others sit idle.

3. user_id (Hashed)

- **Verdict: WINNER (Best Choice).**
- *Pros:*
  - **Write Distribution:** Hashing ensures User A goes to Shard 1, User B to Shard 2. Even load.
  - **Query Isolation:** The main query "Find orders for User X" can be targeted directly to a single shard. The mongos router knows exactly where User X lives.

Exercise 3: Consistency vs. Availability

1. Analytics Dashboard

- **Read Preference:** secondary (or secondaryPreferred).
- **Reason:** Offload work from the Primary. We accept "Eventual Consistency" (data might be 1s old) to gain performance.

2. Billing System

- **Write Concern:** w: "majority" (and j: true for Journaling).
- **Reason:** Durability is paramount. We cannot risk losing a payment record if a server crashes.

3. User Profile

- **Read Preference:** primary.
- **Reason:** "Read your own writes" consistency. Only the Primary is guaranteed to have the data you *just* wrote. Secondaries might lag by a few milliseconds.

Exercise 4: Spark-MongoDB Integration

1. Predicate Pushdown

- **Answer:** Spark **asks MongoDB to filter first**.
- **Mechanism:** The connector translates df.filter(age > 25) into a Mongo Query find({age: {$gt: 25}}).
- **Impact:** Massive performance gain. Only relevant data is sent over the network. Without this, we would transfer 5TB just to keep 1GB.

2. Partitioning

- **Answer: 100 Partitions.**
- **Logic:** The connector maps 1 MongoDB Chunk/Split = 1 Spark Partition. This allows 100 Spark tasks to read in parallel.

3. Locality

- **Optimization:** If the Spark Worker is on the same machine as the Mongo Shard, the connector prefers **Data Locality**. It reads directly from the local interface/memory rather than clogging the network switch.