Write Concern

Write concern' in **MongoDB** describes the level of write acknowledgment you can expect from it. It's a rather important setting to remember in your write operations and its behavior is useful to understand, especially in distributed MongoDB deployments (i.e. **replica sets** and **sharded clusters**). In this post, we discuss 3 gotchas when using MongoDB write concern.

MongoDB Write Concern

MongoDB's documentation defines write concern as "*the level of acknowledgment requested from MongoDB for write operations to a standalone mongod or to replica sets or to sharded clusters.*"

Simply put, a write concern is an indication of 'durability' passed along with write operations to MongoDB. To clarify, let us look at the syntax:

```
{ w: <value>, j: <boolean>, wtimeout: <number> }
```

Where*,

w can be an integer | "majority" | , it represents the number of members that must acknowled ge the write. **Default value is 1.**

j Requests that a write be acknowledged after it is written to the on-disk journal as opposed to just the system memory. Unspecified by default.

wtimeout specifies timeout for the applying the write concern. Unspecified by default.

* You can find the detailed syntax in the **Write Concern Specification documentation**.

Example:

```
db.inventory.insert(
   { sku: "abcdxyz", qty : 100, category: "Clothing" },
   { writeConcern: { w: 2, j: true, wtimeout: 5000 } }
)
```

The above insert's write concern can be read as follows: acknowledge this write when 'at least 2 members of the replica set have written it to their journals within 5000 msecs or return an error'. A write concern value for option was **majority,** meaning "r*equests acknowledgment that write operations have propagated to the majority of **voting nodes**, including the primary.*"

The importance of write concern is apparent. Increasing values of w increases the latency of writes while also decreasing their probability of **getting lost.** Choosing the correct values for write concern depends on the latency and durability requirements of writes being performed.

With that as the background on what a write concern is, let's move on to the three caveats to remember when using write concern.

CAVEAT 1: Setting write concern on replica sets without a wtimeout can cause writes to block indefinitely

The **majority definition** (applicable MongoDB 3.0 onwards) above states that acknowledgment is requested from a majority of the "voting nodes". **Note** that *"If you do not specify the wtimeout option and the level of write concern is unachievable, the write operation will block indefinitely. "*

This can have unexpected consequences, for example, consider a 2+1 replica set (i.e. a primary, a secondary and an arbiter). If your sole read replica goes down, then all writes with a write concern w option of "majority" will block indefinitely. The same will happen if the w option is set to 2. Another extreme example is in the case of a 3+2 replica set (primary, 2 secondaries and 2 arbiters, **not** a recommended configuration). All "majority" writes will block even if a single data node is unavailable as the majority number, in this case, is 3.

The simplest way to alleviate this issue is to always specify a wtimeout value so the query can timeout if the write concern can't be enforced. However, in case of such timeout errors, MongoDB **doesn't undo** already successful writes made to some of the members before the timeout occurred.

There is also currently no setting to ensure a write reaches the majority of nodes that are currently reachable, so be careful about setting the value of write concern w based on the topology, desired durability, and availability.

CAVEAT 2: You might lose data even with w: majority

It seems intuitive that once a write has been acknowledged by the majority of voting members, its durability is guaranteed. However, that isn't the case! Remember that when the j option is unspecified, a write is acknowledged right after it has been written to memory.

So, such a write can be lost if a freak power outage takes out the majority of the nodes to which the write had propagated (and before **syncPeriodSecs** i.e. before it could be flushed to disk).

In order to ensure the durability of writes, it's best not to turn off **journaling** on your database and set the j option to true. In fact, starting MongoDB 3.6, the --nojournal flag has been **deprecated** for replica set members using the WiredTiger storage engine.

With a w value of "majority" and the j option unspecified on a replica set, the exact durability behavior depends on the value of the replica set configuration **writeConcernMajorityJournalDefault**. When set to true (and when **journaling** is

enabled), it acknowledges writes after they have been written to the journals of a majority of voting members.

Aside: Even with **journaling turned on,** your writes might still get lost on the **MMAPv1 storage engine** if an outage occurs within **commitIntervalMs** duration. The WiredTiger storage engine, on the other hand, **forces a sync of journal files** when it receives a write with j option set to true. And, even with j set to false, an acknowledged "majority" write to a **latest WiredTiger based deployment** can be lost only when majority of the data nodes crash simultaneously.

CAVEAT 3: w: 0 while setting j: true doesn't improve write performance

This is easy enough to reason once you think about it, but equally easy to forget. Setting w option to 0 is usually done to write to the database in a "fire-and-forget" fashion – when you have a fair amount of confidence on the database infrastructure and care more about latency than the durability of every write. However, if you set the j option to true, your w option will effectively be overridden as the database will ensure that the write is written to the on-disk journal before returning.