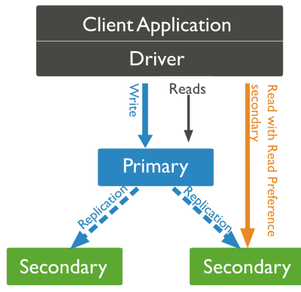

snipped from: https://docs.mongodb.com/manual/replication/

MONGODB CLUSTER

MongoDB: Read from secondary to boost performance

MongoDB Read from secondaries, Use cases, reading stale-data/up-to-date-data from secondaries pro's and con's with example.

Arun PratapApr 18, 2020 · 3 min read

> *mongodb version 3.6+ , link to mongodb documentation wherever necessary for further reading.*

Why do we want to read from Secondary node's ?

By default mongodb read/write on the primary for consistency(considering all preferences and configuration).

Use Cases to read from secondaries:

Reporting/analytics where in stale data is acceptable.

Write is significantly low compared to **Reads**(still have to take care of stale data).

Providing local reads for geographically distributed applications

. . .

Read from secondary (stale-DATA):

All read preference modes except `primary` may return stale data because secondaries replicate operations from the primary in an asynchronous process. [1] Ensure that your application can tolerate stale data if you choose to use a non-`primary` mode, but there is a way you can control how much stale data you want to read by giving `maxStalenessSeconds`, value "0" doesn't mean's no-staleness, it mean's it won't look for staleness.

> All we have to do is give read-preference — [`secondary`, `secondaryPreferred`, `nearest`, `primaryPreferred`] ex — `db.collection.find({}).readPref( "secondary", [ { "region": "South" } ] )`
> java: spring-data — we can specify that in mongoTemplate bean.

```
@Bean
public MongoTemplate mongoTemplate() throws Exception {
    MongoTemplate template = new
MongoTemplate(this.mongoDbFactory(), this.mappingMongoConverter());
    int maxStaleness = 91;
template.setReadPreference(ReadPreference.secondary(maxStaleness,
TimeUnit.SECONDS));
    return template;
}
```

This change is application level will apply to all the read's.

. . .

Reading from secondary (up-to-date-data) Method 1:

Don't want to read stale data, keep all your secondary up-to-date. To achieve this we have to get write concern acknowledged by all the secondaries i.e write-concern = total number of replicaset including primary.

> *Don't forget to give wtimeout when mentioning write-concern = number of nodes, as in case of failover write will wait forever.*

Pros:

No replication lags.

failover will be smooth, as there will be no roll-back, neither sync is needed for secondaries(will always be in sync).

Will have performance gain as the read load is distributed(more read oriented system).

Cons:

Will lose the failover capability, write's will start failing as one of the secondary will not be available to acknowledge.

write will be slower as it has to acknowledge all the secondary, but if in same network with same specs it won't be that significant as this is done asynchronously.

Ex.

mongo shell

```
db.products.insert(
   { item: "envelopes", qty : 100, type: "Clasp" },
   { writeConcern: { w: "majority" , wtimeout: 5000 } }
)
```

spring-data

```
@Bean
public MongoTemplate mongoTemplateWriteAll() throws Exception {
    MongoTemplate template = new
MongoTemplate(this.mongoDbFactory(), this.mappingMongoConverter());
    int maxStaleness = 91;

template.setReadPreference(ReadPreference.secondary(maxStaleness,
TimeUnit.SECONDS));
    int wTimeoutMS = 5000;
    int totalNoOfReplicaServer = 3;
    template.setWriteConcern(new
WriteConcern(totalNoOfReplicaServer, wTimeoutMS));
    return template;
}
```

. . .

Reading from secondary (up-to-date-data) Method 2:

Read in a client session — Causal Consistency.

First we need to have our **readConcern** and **writeConcernt** as **Majority** and for **readPreference** we can choose anything, it still guarantee's causal consistency and for most important part all our operation should be performed under a client session.

Ex.

```
// Example 1: Use a causally consistent session to ensure that the
update occurs before the insert.
ClientSession session1 =
client.startSession(ClientSessionOptions.builder().causallyConsisten
t(true).build());
Date currentDate = new Date();
MongoCollection<Document> items = client.getDatabase("test")
        .withReadConcern(ReadConcern.MAJORITY)
        .withWriteConcern(WriteConcern.MAJORITY.withWTimeout(1000,
TimeUnit.MILLISECONDS))
        .getCollection("test");

items.updateOne(session1, eq("sku", "111"), set("end",
currentDate));

Document document = new Document("sku", "nuts-111")
        .append("name", "Pecans")
        .append("start", currentDate);
items.insertOne(session1, document);
```

. . .

There are pro's and con's associated specially when you choose to go with up-to-date-date read from secondary OR define boundaries where stale data is acceptable. There will be added complexity introduced in your system that you have to handle. — Happy Coding :-)