Scaling MongoDB

Percona Webinar - Wed October 18th 11:00 AM PDT
Adamo Tonete
MongoDB Senior Service Technical Service Engineer
Me and the expected audience

- @adamotonete
- Intermediate - At least 6+ months MongoDB experience
Agenda

- Overview on MongoDB
- The adamo.com story scaling out
- The adamo.com story scaling down
- Review
- Q&A
Overview on MongoDB

- Fast;
- Document-oriented database;
- Easy to deploy and manage;
- Secure database;
- HA by default;
- Easy to scale.
Internals

- Document size limit is 16 MB

- Different storage engines
  - MMAPv2
  - WiredTiger
  - RocksDB
  - In Memory
Internals

MongoDB shares features with Relational Databases such as:

- Indexes;
- Query Optimizer - Explain;
- Cache Management;
- Backups;
- Restores.
The adamo.com company story

adamo.com is a startup that started using MongoDB as a single instance and was using a single machine on a cloud provider.

In this webinar, I’m going to tell you the story of how the company evolved from one instance to a sharded environment.

Single instance

8 GB RAM, 2 processors - WiredTiger as Storage Engine
The adamo.com company story

After a few months, the database started answering queries very slowly. The company noticed high processor usage and a very active disk, so they decided to increase the instance type.

16 GB RAM, 4 processors - WiredTiger as Storage Engine
Even after increasing the instance type, the company still noticed slow reads and writes. Processor usage was still high and the company still noticed high disk I/O. To make matters worse, the database failed after a few weeks.
The adamo.com company story

Cache

Cache evicted by ETL

Free memory

Single instance

ETL process

16 GB RAM, 4 processors - WiredTiger as Storage Engine
The adamo.com company story

So, why was this database still slow even after increasing the machine resources? What could be done to avoid outages?

Now the adamo.com company had learned both how to use replica-sets and the advantages of having multiple instances with High Availability.

The HA was solved, the single instance became a replica-set with 3 members, but they were still having issues on the primary instance. The same problem as they had faced before.
The adamo.com company story

A new environment. The replica-set was working properly but still facing issues while ETL was running, and the application auto fail-over was not working properly.
The adamo.com company story

A few names you should know for the next step:

- Read Preference
- Write Concern
- Replication Lag
- Oplog Window
The adamo.com company story

A new environment. The replica-set was working properly but still facing issues while ETL was running, and the application auto fail-over was not working properly.
A few names you should know for the next steps:

- **Read Preference**: Most application drivers feature the read preference option. This is, in simple words, where the application will get data from.

  All writes will be routed to the primary, but you can change the read preference option to:
  - Primary, secondary_prefered, secondary_only, nearest, and tags.
The adamo.com company story

A few names you should know for the next steps:

- **Write Concern**: MongoDB offers eventual consistency, but we can tune it up and have more than one instance confirming a write command. This is very useful to guarantee data consistency across the replica-set and/or datacenters.

  The write concerns are: 1,2,<n> and majority, where majority means ½ + 1 instance from the replica-set.
The adamo.com company story

A few names you should know for the next steps:

- **Replication Lag**: MongoDB replication between instances doesn't occur in real time. It is asynchronous. A secondary instance with fewer resources than the primary can easily fall behind to receive new writes and updates.

  The most common issues are slow disks, limited bandwidth, and slow processors.

  Replication lag is a critical metric when you run queries on secondaries.
The adamo.com company story

A few names you should know for the next steps:

- **Oplog Window**: Per design, the replication is asynchronous and all the primary commands are saved on the oplog.rs collection. The oplog collection is a fixed-size collection that saves such operations, so that the secondaries can pull and apply the same commands. As it is a fixed-size collection, the old documents will be replaced by new ones when it gets full. The period of time between the first and the last command in this collection is called oplog window.
A new environment, spreading the read to all the secondaries, and primary only receives writes.
The adamo.com company story

The new environment seemed to be really robust, but after a while the company started facing new issues.

The understanding of the concepts below is required to discuss the next issues.

- Hidden Instance
- Priority
- Votes
- Arbiters
The adamo.com company story

When someone from BI was running their ETL, a few clients started noticing slowness, why?

App is reading from here as well
The adamo.com company story

PRIMARYP

P

S

S

HS

ETL

Hidden secondary
The adamo.com company story

The hidden secondary was completely hidden from the replica-set, which means that the application was not connected to the instance to perform reads.

In the case above, only the ETL process read data from this secondary.

If we need more reads, it is ok to add new secondaries. However, we must make sure we don't add too many of them, as issues such as replication delay may occur.
The adamo.com company story
The replication delay can occur for several reasons. A replica-set may, for example, perform chained replication, which in other words means the database is replicating to a secondary, and that secondary is replicating to another secondary.

The chained replication can be disabled on the replica-set configuration.

- https://docs.mongodb.com/manual/tutorial/manage-chained-replication/
The adamo.com company story
The adamo.com company story

Not all the instances will be eligible to vote. Replica-sets can only have up to 7 instances voting. Therefore, if there are more than 7 instances in the replica-set, only 7 can vote. As good practice, keep those close to the "primary" and choose the best ones.

Sometimes we need a mongodb instance that will only check whether the other instances are available. These are called arbiters and arbiters don’t keep data, they only perform votes.
The adamo.com company story
The adamo.com company story

It is time to consider sharding.

Even with the replica-sets, the number of writes we need as well as the working set are making it too expensive to keep everything in a single replica-set. Sharding will create a virtual database among replicas and split the load between those replica sets, which will be called shards.
A simple 2-shard cluster
Shards - Important words

**MongoS** is the proxy process that the application will connect to. As the database is sharded, there is no guarantee that all the data is in just one shard.

**ConfigServers** are the servers that store the data location. These special instances save all the data location in order to enable the mongos to find the expected data. This data is called cluster metadata.

A **Shard** is a replica set that belongs to a cluster.

A **Cluster** is a combination of 1 or more shards along with a config server and mongos.
Shards - Important words

A **Shard Key** is the field where the collection will be split.

**Chunks** are small amounts of data, usually 64M, based on the shard key. There are several chunks in a database after it is sharded.

A **Balancer** is the process that moves data between shards or even inside shards to different chunks.
adamo.com was now one of the most popular websites. Even though the replica-set helped them to handle reads, writes became a problem. It was time to start developing a cluster strategy.
Shards - 1-shard cluster
The adamo.com company story - shards
Shards - Adding new shards

Even after a new shard was added, the sample company didn’t see any improvement in the write performance.

In fact, the main database was not yet configured as a sharded database, so all the reads and writes were going to shard 1 only.
Shards - Adding new shards
It is necessary to tell the config servers that we want to shard a database. After running:

```
sh.shardDatabase('mydatabase')
```

We need to start sharding the collections using a shard key. The most used collection is called *posts*, and for this collection we shard based on the hash of the `_id` field.

We can change shards by range, but we choose the hash `_id` to speed up the writes and make them random.
Shards - Sharding a database
Shards - Balancer

After the balancer funs for a while, both shards keep data. Each part has its data and the information about where the data is stored is in the config database.

adamo.com noticed that the etl was only moving "half" of the expected data because the data was now split between the shards.
Shards - Balancer
The adamo.com company story - shard

ETL couldn't connect to the hidden secondaries because all the connections were made through the mongos.
To work around this situation, the company decided to use tag-aware reads.

In order to perform the reads, each instance had to have a tag, for different purposes: read or ETL.

In this case, the application will perform reads from instances with the "read" tag and the ETL will only read from the ETL tag.

https://docs.mongodb.com/v3.2/core/tag-aware-sharding/
Shards - Read Tag Aware
The adamo.com company story

With such architecture, the example company could have 10 shards running at the same time, but unfortunately the success of the website didn't last too long.

After 2 years running on a 10-shard replica-set, the company decided to start shrinking the environment to just 2 shards again.
The adamo.com company story

The process was the inverse of sharding, where the company needed to remove shards from the cluster and wait until all the data moved to different shards.

The balancer process works the other way round. Data from the shard was removed to the remaining shards and the config servers were updated to the new data location.

All of those steps can be done online meaning that no downtime is required.
Review

- Single Instances are not recommended;
- Replica-sets must have at least 3 instances storing data;
- Hidden instances can't be read by drivers;
- A cluster can have only one shard but there is no performance improvement;
- Adding new shards is easy, but you need to pick the right shard key to split the data among the instances;
- Use read tags to connect through the mongos in order to read from a specific instance in the shard.
- Scale down is as easy as scale out in mongoDB

Some subjects, such as backups and security, were omitted from this presentation to keep it simple and concise.
Review

Resources:

https://www.percona.com/blog/2017/10/16/when-should-i-enable-mongodb-sharding/

https://docs.mongodb.com/manual/sharding/

https://www.percona.com/blog/2016/12/16/mongodb-pit-backups-in-depth/