**Week 1** – MongoDB Sharding Pitfalls

**4. Under-provisioning Config Servers**

Provisioning enough resources without being wasteful is always tricky, and all the more so in a complicated distributed system like a MongoDB sharded cluster. Everyone wants to use their hardware, virtual instances, virtual machines, containers and the like in the most efficient way possible, and get the best bang for their buck. Hence it is only natural to take a look at the various pieces of a distributed cluster and look for lower utilized pieces that could be put on less expensive resources.

The most common pitfall here with MongoDB are the config servers, which are often neglected when stress testing a cluster. In testing environments and smaller deployments (unless specific measures are taken to stress them) they are relatively lightly loaded and usually identified as candidates for lesser instances/hardware.

The problem is that these are critical pieces of infrastructure. They may not be heavily loaded all the time, but when they do see load and struggle to service requests, that can impact all queries (reads, writes, authentication) and add latency to all requests made of the cluster in question.

In particular, the first config server in the list supplied to your mongos processes is vital. This is the config server that all mongos processes will default to read from when fetching or refreshing their view of the data distribution in your cluster. Similarly, this is the server that will be hit when attempting to authenticate a user. If it is under-provisioned and cannot service queries, or if it has problems with networking (packet loss, congestion), then the effects will be significant.

**Possible Mitigation Strategies**

* Ensure the config servers are load tested, slightly over-provisioned (the first config server in particular)
* If using virtual machines or cloud based instances, investigate increasing available resources
* Turning off the balancer, disabling chunk splitting will reduce the chances of high read traffic to the config servers (no migrations, no meta data refresh) but this is only a temporary fix unless you have a perfect write distribution and may not eliminate issues completely.

**5. Using the count() command on sharded collections**

This pitfall is very common, and it seems to hit somewhat randomly in terms of how long someone has been running a sharded environment. At some point, a question will arise along the lines of:

“How are we tracking/verifying/checking how many documents we have in each collection on each shard, how balanced are they and do they agree with <some other system that holds the same data>?”

Hopefully no one is actually constructing questions this way in your organization, but you get the basic idea. The most obvious way to do a quick check on this type of thing is to count the documents and see if the numbers make sense and/or agree with counts elsewhere. That thinking naturally leads people to the count command and they proceed to use it to gather figures for their documents and collections.

Unfortunately, on a busy, mature sharded cluster, the results will very rarely be what is expected. The reason for this is that the count command as implemented today has several optimizations in place to make it faster to run in general and those speed optimizations essentially bypass a key piece of the sharding functionality needed to return accurate results in this case. This is a known bug and is being tracked in SERVER-3645, but does not stop people from consistently hitting this issue. The nature of the issue means that count will report documents in the results that it should not, for example:

* Documents that are being deleted as part of a chunk migrations
* Documents that have been left behind from previous chunk migrations (also known as orphans)
* Documents currently being copied as part of an in-flight chunk migration

A regular query (rather than a count) will have its results filtered by the respective primary and not suffer from the same problem. Hence, if you were to manually count the results from a query client-side you would get an accurate result.

This quirk of sharded environments will eventually be fixed, but for now it will inevitably crop up from time to time in all active sharded clusters used by a large team.

**Possible Mitigation Strategies**

* Do counts on the client side, or use targeted, range based queries (with a primary read preference) to count instead
* Use cleanUpOrphaned and disable the balancer (make sure it has finished current round) when performing counts across the cluster

Source: <https://www.mongodb.com/blog/post/sharding-pitfalls-part-ii-running-a-sharded>