**Week 1** – MongoDB Sharding pitfalls

Pitfalls regarding sharding:

**1. Using a monotonically increasing shard key (like ObjectID)**

The most common mistake we see is the selection of a monotonically increasing shard key when using range-based sharding rather than hashed sharding, which is means that the shard key value for new documents only increases. Examples of this would be a timestamp (naturally) or anything that has a time component as its most significant component like ObjectID (first 4 bytes are a time stamp).

The issue with this comes with insert scalability. If you select such a shard key, all inserts (new documents) will go to a single chunk - the highest range chunk, and that will never change. Hence, regardless of how many shards you add, your maximum write capacity will never increase - you will only ever write new documents to a single chunk and that chunk will only ever live on a single shard.

Occasionally, this type of shard key can be the correct choice, but if so then you won’t be able to scale for write capacity.

**Possible Mitigation Strategies**

* Change the shard key - this is problematic with large collections, because the data essentially has to be dumped out and re-imported
* More specifically, use a hash based shard key, which will allow the use of the same field while providing good write scalability.

**2. Trying to Change Value of the Shard Key**

Shard keys are immutable (cannot be changed) for an existing document. This issue usually only crops up when sharding a previously unsharded collection. Prior to sharding, certain updates will be possible that are no longer possible after the collection has been sharded.

Attempting to update the shard key for an existing document will fail with the following error:

cannot modify shard key's value fieldid for collection: foo.foo

**Possible Mitigation Strategies**

* Delete and re-insert the document to alter the shard key rather than attempting to update it in-place. It should be noted that this will not be an atomic operation, so must be done with caution.

**3. Waiting too long to add a new shard (overloaded)**

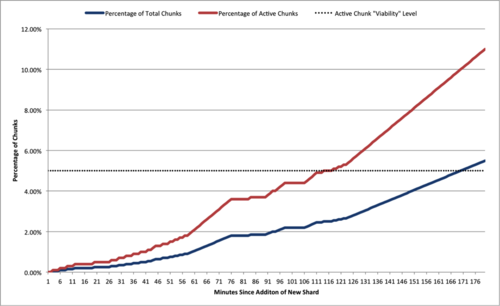
Adding a new shard to a cluster is not free, and it is not instantaneous. It consumes resources and (initially) accepts very little traffic. Essentially, at the start of its existence, a newly added shard costs you capacity instead of adding capacity. The length of time it will stay in this state will depend on the balancer and how long it takes for a significant portion of “busy/active” chunks to move onto the new shard.

It can often be easier to visualize this process, so let’s make up some hypothetical numbers and set the bar relatively low. Our imaginary existing cluster will be a set of 2 shards, with 2000 chunks (500 considered “active”) and to that we need to add a 3rd shard. This 3rd shard will eventually store one third of the active chunks (and total chunks). The question is, when does this shard stop adding overhead overall and instead become an asset?

In reality, this will vary from cluster to cluster and have a lot of dependencies and variables - in other words you need to have good metrics about your cluster, particularly your load bottleneck.

Therefore we will once again use our imaginations and go with a relatively low bar: when 5% of active chunks–that is, those chunks seeing most traffic–have migrated to the new shard, you should expect a net gain in performance. In our imaginary system we have evaluated our load levels, the expected impact of migrations and have determine that once that 5% threshold of active chunks has been migrated to the new shard it can be considered a net gain for the overall system. Once all chunks have been balanced, then the migration overhead disappears, but initially this will be an expected trade off.

This chart shows how long it would take for new shards to reach net positive contribution in your cluster (the dotted line implies net gain):



In this fabricated example, it takes almost 2 hours for the new shard to attain a viable level of active chunks and be considered a net gain for the overall system. Although these numbers are fictional, these numbers are based on setups we have seen in real systems with moderate load.

From there it is relatively easy to imagine this set of migrations taking even longer on an overloaded set of shards, and taking far longer for our newly added shard to cross the threshold and become a net gain. As such it is best to be proactive and add capacity before it becomes a necessity.

**Possible Mitigation Strategies**

* Manual balancing of targeted “hot” chunks (chunk that is being accessed more than others) to move activity to the new shard more quickly
* Add the shard at low traffic time so that there is less competition for resources
* Disable balancing on some collections, prioritise balancing busy collections first

Sources:

<https://www.mongodb.com/blog/post/sharding-pitfalls-part-i>

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