**Possible Mitigations**

**Possible Mitigation Strategies for Using a monotonically increasing shard key (like ObjectID)**

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

**Possible Mitigation Strategies for Trying to Change Value of the Shard Key**

1. 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.
2. Now you have a better understanding of how to choose and change your shard key if needed. In our next post, we will go through some potential obstacles you will face when scaling your sharded environment.
3. If you want more insight on scaling techniques for MongoDB, view the slides and video from our recent webinar on how to achieve scale with MongoDB, which reviews three different ways to achieve scale with MongoDB.

**Possible Mitigation Strategies for Waiting too long to add a new shard (overloaded)**

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

**Possible Mitigation Strategies for Under-provisioning Config Servers**

1. Ensure the config servers are load tested, slightly over-provisioned (the first config server in particular)
2. If using virtual machines or cloud based instances, investigate increasing available resources
3. 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.

**Possible Mitigation Strategies for Using the count() command on sharded collections**

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

**Possible Mitigation Strategies for Chunk balancing != data balancing != traffic balancing**

1. If data and traffic balance are important, select an appropriate shard key
2. Move chunks manually to address the imbalances - swap “hot” chunks for “cool” chunks, empty chunks for larger chunks

**Possible Mitigation Strategies for Waiting too long to shard a collection (collection too large)**

1. Since the limit is dictated by the chunk size and the data size, and assuming there is not much to be done about the data size, then the remaining variable is the chunk size. This is adjustable (default is 64MiB) and can be raised in order to let a large collection split initially and then reduced once that has been completed.
2. The required chunk size increase will depend on the actual data size. However, this is relatively easy to work out - simply divide your data size by 256GB and then multiply that figure by 64MiB (and round up if it is not a nice even number). As an example, let’s consider a 4TiB collection:
3. 4TiB divided by 256GiB = 16 64MiB x 16 = 1024MiB
4. Hence, set the max chunk size to 1024MiB, then perform the initial sharding of the collection, and then finally reduce the chunk size back to 64MiB using the same procedure. .