Performing MongoDB Massive Write Operations Efficiently

Gabriel Ciciliani - Pythian

May 29th, 2019

Pythian
AGENDA

● Massive write operations
● Scenarios
● Useful facts
● Different solutions
● Additional features
● Conclusions
● Q&A
When do we perform massive write operations?

- We need to **archive** old data for query or space efficiency..
- We need to **fix** an error introduced by an application..
- We need to update/create a key because of a **new feature**..
- We need to **get rid of** entire databases in multi tenant environments..
Massive write operations: challenges

What’s wrong with running a single `update(multi)` or `remove()` statement:

- Excessive locking
- Could require large collection scans
  - Evicting frequently used pages from the cache
  - Causing IO spikes
- Hard to keep track of progress and recover from failures
- Lag on secondaries
General strategy:

Split the operation in smaller, controllable tasks
## All collections have an indexed _id key that is unique and not null

## There is an optimization for sort() + limit(1) where only 1 key and 1 document is read

```javascript
db.restaurants.find({cuisine:"American"},{ _id:1}).sort({cuisine:-1}).limit(1).explain(true)
db.restaurants.find({cuisine:"American"},{ _id:-1}).sort({cuisine:-1}).limit(1).explain(true)
```

```
... "totalKeysExamined" : 1, "totalDocsExamined" : 1, ...
```
Useful facts

## limit() will effectively limit the amount of keys and documents examined after the specified number of documents matching the filter condition were found.

## When count() is executed against an indexed condition, it will only read the index

```javascript
db.restaurants.explain(true).find({cuisine:"American"},{_id:1}).count()
...
"totalKeysExamined" : 6184,
"totalDocsExamined" : 0,
...
```
Scenarios

We will analyze the following situations:

1) Update and delete documents when *there is* an index for the condition

2) Update and delete documents when *there is no* index for the condition

3) Dropping databases

4) Dropping collections
1) Update and delete documents when there is an index for the condition
Solution approach

1) Update and delete documents when **there is** an index for the condition

   a) **Update cases:**
      
      i) **If the search condition is affected by the update operation** -> iterate with `limit()` until there are no more documents to update

      ii) **If the search condition is *not* affected by the update** -> split the total amount of documents in chunks and run the update for each one

   b) **Delete:**
      
      i) Iterate with `limit()` until there are no more documents to delete
2) Update and delete documents when there is no index for the condition.
2) Update and delete documents when there is no index for the condition

a) Update and delete:

i) Split the entire collection in chunks and run the operation for each one
3) Dropping databases
Solution approach

Database Drop

- `db.dropDatabase()` requires a global write lock, blocking other operations until it has completed.

- `db.collection.drop()` only requires a database level lock

- Drop all collections first, then drop the database during off-peak hours
Database drop PoC

- **Instance type:** t2.large
- **Storage:** 80Gb gp2 EBS volume
- **MongoDB version:** 4.0.6
- **Database size:** 53Gb

<table>
<thead>
<tr>
<th></th>
<th>Without dropping collections</th>
<th>Dropping collections first</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>db.dropDatabase()</code> real time</td>
<td>1.248 secs</td>
<td>0.056 secs</td>
</tr>
</tbody>
</table>
4) Dropping collections
Solution approach

Collection Drop

- `db.collection.drop()` requires a database-level write lock, blocking other operations until it has completed.

- `db.collection.remove()` only requires a collection-level lock

- Drop all documents first, then drop the collection
Additional features

- Add a mechanism to stop/restart the process
- Keep track of the process progress
- Dry run option
Add a mechanism to stop and restart the process

- Use an auxiliary collection and use a specific document as a flag
- Check the status of that document between iterations
- Register last _id processed and completed collections so you can resume the process
Keep track of the process progress

- Use an auxiliary collection and update it upon every iteration, specifying number and timestamp
- If possible, compute the total documents to be modified / removed and the amount of iterations required
Dry run option

- It is always good to test the code by disabling the destructive operation (update, remove, drop, etc) to make sure the script is about to remove what we are expecting.

- This can be implemented by simply adding a boolean parameter and by checking its value before dropping, updating or pruning a collection.
Conclusions

- Split the overall operation in smaller, controllable tasks
- Make sure the most efficient approach is used, depending on the type of operation and indexes available
- Use delays to reduce pressure on the system
- Calculate overall iterations required and log progress using an auxiliary collection
- Use an auxiliary document as emergency stop
Code samples

You will find all code reviewed during this session at:

https://github.com/gabocic/mongodb/tree/pl19tx/pl19tx
Q&A

ONE DOES NOT SIMPLY
GIVE A PRESENTATION WITHOUT Q&A

IT'S NOT A BAD Q&A
IF THERE IS NO A
Thanks!

We are always growing!
Come and see me after the session or reach out at ciciliani@pythian.com