

MongoDB Performance Tuning Techniques

MongoDB Meetup Aug 2014

Kenny Gorman

Chief Architect and Co-Founder, ObjectRocket

@objectrocket @kennygorman



Order Matters

1. Schema Design
2. Workload tuning
3. Instance tuning



Schema Design

- Which patterns matter?
- Test those patterns
- Where is your dev environment?

Workload Tuning

- Profiler
- Explain

Profiler

```
while true:
    bad_statements = find_candidates()
    for statement in bad_statements:
        statement.explain()
```

Profiler

```
$> db.setProfilingLevel(2);  
{ "was" : 0, "slowms" : 100, "ok" : 1 }  
  
$> db.testme.save({"name":"Kenny"});  
  
$> db.system.profile.find().pretty()  
{  
  "ts" : ISODate("2013-02-11T18:45:06.857Z"),  
  "op" : "insert",  
  "ns" : "test.testme",  
  "keyUpdates" : 0,  
  "numYield" : 0,  
  "lockStats" : {...},  
  "millis" : 0,  
  "client" : "127.0.0.1",  
  "user" : "" }
```

Profiler

```
{
  "ts" : ISODate("2012-09-14T16:34:00.010Z"), // date it occurred
  "op" : "query", // the operation type
  "ns" : "game.players", // the db and collection
  "query" : { "total_games" : 1000 }, // query document
  "ntoreturn" : 0, // # docs returned with limit()
  "ntoskip" : 0, // # of docs to skip()
  "nscanned" : 959967, // number of docs scanned
  "keyUpdates" : 0, // updates of secondary indexes
  "numYield" : 1, // # of times yields took place
  "lockStats" : { ... }, // subdoc of lock stats
  "nreturned" : 0, // # docs actually returned
  "responseLength" : 20, // size of doc
  "millis" : 859, // how long it took
  "client" : "127.0.0.1", // client asked for it
  "user" : "" // the user asking for it
}
```

<http://docs.mongodb.org/manual/reference/database-profiler/>

Profiler: Finding Candidates

```
// response time by operation type
db.system.profile.aggregate(
{ $group : {
  _id :"$op",
  count:{$sum:1},
  "max response time":{$max:"$millis"},
  "avg response time":{$avg:"$millis"}
}});
```

```
// slowest by namespace
db.system.profile.aggregate(
{ $group : {
  _id :"$ns",
  count:{$sum:1},
  "max response time":{$max:"$millis"},
  "avg response time":{$avg:"$millis"}
}},
{$sort: {
  "max response time":-1}
});
```

```
// slowest by client
db.system.profile.aggregate(
{$group : {
  _id :"$client",
  count:{$sum:1},
  "max response time":{$max:"$millis"},
  "avg response time":{$avg:"$millis"}
}},
{$sort: {
  "max response time":-1}
});
```

```
// summary moved vs non-moved
db.system.profile.aggregate(
{ $group : {
  _id :"$moved",
  count:{$sum:1},
  "max response time":{$max:"$millis"},
  "avg response time":{$avg:"$millis"}
}});
```


Explain: What to look for

- fastmod
- nscanned != nreturned
- key updates
- moves
- lock waits

Explain

- Take the document from profiler output, explain it
- Look at results, make corrections
- Rinse and repeat

Explain

```
$>db.system.profile.find({"op":"query","ns":"test.testme"}).pretty();
{  "ts" : ISODate("2013-02-11T19:53:16.302Z"),
    "op" : "query",
    "ns" : "test.testme",
    "query" : { "name" : 1 },
    "ntoreturn" : 0,
    "ntoskip" : 0,
    "nscanned" : 32001,           // why scanning so many?
    "keyUpdates" : 0,
    "numYield" : 0,
    "lockStats" : {...},
    "nreturned" : 1,             // just to return 1
    "responseLength" : 56,
    "millis" : 29,               // slow!
    "client" : "127.0.0.1",
    "user" : ""
}
```

Explain

```
$> db.testme.find({ "name": 1 }).explain()
{
  "cursor" : "BasicCursor",                                // Basic
    "isMultiKey" : false,
    "n" : 1,
    "nscannedObjects" : 32001,
    "nscanned" : 32001,
    "nscannedObjectsAllPlans" : 32001,
    "nscannedAllPlans" : 32001,
    "scanAndOrder" : false,
    "indexOnly" : false,
    "nYields" : 0,
    "nChunkSkips" : 0,
    "millis" : 14,
    "indexBounds" : {
                                                                // WTF!
    },
    ...
}
```

Explain

```
$> db.testme.ensureIndex({"name":-1});
$> db.testme.find({"name":1}).explain()
{
  "cursor" : "BtreeCursor name_-1",
  "isMultiKey" : false,
  "n" : 1,
  "nscannedObjects" : 1,
  "nscanned" : 1,
  "nscannedObjectsAllPlans" : 1,
  "nscannedAllPlans" : 1,
  "scanAndOrder" : false,
  "indexOnly" : false,
  "nYields" : 0,
  "nChunkSkips" : 0,
  "millis" : 0,
  "indexBounds" : {
    "name" : [
      [
        1,
        1
      ]
    ]
  },
  ...
}
```

// Btree

// w00t!

Response time analysis

- Response time analysis techniques come from Oracle community circa 2000-2004.
- Response time = service time + queue time (time_to_complete + time_waiting_in_queue)
- Each document in profile collection a couple response time attributes.
 - millis
 - timeAcquiring
 - timeLocked
- The only true measure of response time in MongoDB
- Aids in prioritization of tuning opportunities. Finding the bang for the buck, or the immediate performance problem.

Definitions:

`system.profile.lockStats.timeLockedMicros`

The time in microseconds the operation held a specific lock. For operations that require more than one lock, like those that lock the `localdatabase` to update the `oplog`, then this value may be longer than the total length of the operation (i.e. `millis`.)

`system.profile.lockStats.timeAcquiringMicros`

The time in microseconds the operation spent waiting to acquire a specific lock.

`system.profile.millis`

The time in milliseconds for the server to perform the operation. This time does not include network time nor time to acquire the lock.

Response time analysis

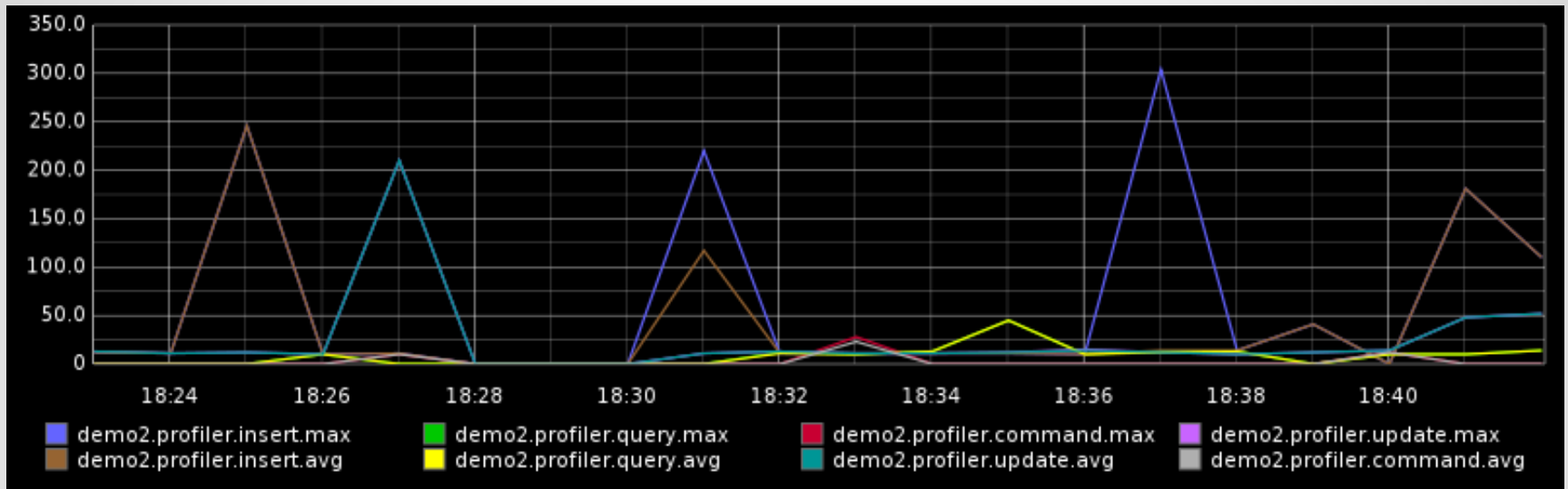
```
$>db.system.profile.aggregate(  
  [  
    { $project : {  
      "op" : "$op",  
      "millis" : "$millis",  
      "timeAcquiringMicrosrMS" : { $divide : [ "$lockStats.timeAcquiringMicros.r", 1000 ] },  
      "timeAcquiringMicroswMS" : { $divide : [ "$lockStats.timeAcquiringMicros.w", 1000 ] },  
      "timeLockedMicrosrMS" : { $divide : [ "$lockStats.timeLockedMicros.r", 1000 ] },  
      "timeLockedMicroswMS" : { $divide : [ "$lockStats.timeLockedMicros.w", 1000 ] } }  
    },  
    { $project : {  
      "op" : "$op",  
      "millis" : "$millis",  
      "total_time" : { $add : [ "$millis", "$timeAcquiringMicrosrMS", "$timeAcquiringMicroswMS" ] },  
      "timeAcquiringMicrosrMS" : "$timeAcquiringMicrosrMS",  
      "timeAcquiringMicroswMS" : "$timeAcquiringMicroswMS",  
      "timeLockedMicrosrMS" : "$timeLockedMicrosrMS",  
      "timeLockedMicroswMS" : "$timeLockedMicroswMS" }  
    },  
    { $group : {  
      _id : "$op",  
      "average response time" : { $avg : "$millis" },  
      "average response time + acquire time" : { $avg : "$total_time" },  
      "average acquire time reads" : { $avg : "$timeAcquiringMicrosrMS" },  
      "average acquire time writes" : { $avg : "$timeAcquiringMicroswMS" },  
      "average lock time reads" : { $avg : "$timeLockedMicrosrMS" },  
      "average lock time writes" : { $avg : "$timeLockedMicroswMS" } }  
    }  
  ]  
);
```


Response time analysis

```
{
  "_id" : "insert",
  "average response time" : 0.07363770250368189,           // time executing
  "average acquire time reads" : 0,
  "average acquire time writes" : 5.623796023564078,       // time waiting
  "average lock time reads" : 0,
  "average lock time writes" : 0.25491826215022123         // time in lock.. woah.
}

{
  "_id" : "update",
  "average response time" : 0.23551171393341552,           // time executing.. moves?
  "average acquire time reads" : 0,
  "average acquire time writes" : 10.261996300863133,       // lots of waiting
  "average lock time reads" : 0,
  "average lock time writes" : 0.3795672009864362          // time in lock.. again!
}
```

Response time analysis



- Python code to get you started with profiler and graphite:

<https://github.com/kgorman/slum>

Instance Tuning

- Mongostat
- Cache management, Hot Set, and Disk I/O.
- Locks and Sharding
- Some thoughts on Hardware

Links

<http://www.kennygorman.com/mongodb-profiler-helpful-queries/>

<https://gist.github.com/kgorman/134896c7414fde8e090b>

<https://github.com/kgorman/slum>

<https://github.com/kgorman/ocean>

<http://docs.mongodb.org/manual/reference/database-profiler/>

<http://docs.mongodb.org/manual/reference/method/cursor.explain/#explain-output-fields-core>

Contact

@kennygorman

@objectrocket

kgorman@objectrocket.com

<https://www.objectrocket.com>

WE ARE HIRING!

<https://www.objectrocket.com/careers/>

