MongoDB profiler deep dive

MongoAustin 2013
Kenny Gorman
Founder, ObjectRocket

@objectrocket @kennygorman

What is the profiler?

- Captures metadata about what operations ran on the system
- Saves data into capped collection
- Designed for basic performance analysis
- In the spirit of < instrument everything >
- Very interesting advanced analysis possible
  - Aggregation
  - Historical/Time-series analysis
  - Operational monitoring

Using the profiler

- Turn it on, leave it.
- Development cycle
- Production debugging
- Overall performance management
- Find candidates, pull out query, use explain()
- Rinse and Repeat

Simple example

```
$> 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" : "" }
```

example: https://gist.github.com/kgorman/4756589

Annotated

```
{
 "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
}
```

example: https://gist.github.com/kgorman/4957922

Important Profiler Attributes

ts:	timestamp of the operation
ns:	namespace of the db and collection accessed
op:	the operation type
nreturned:	the number of documents being returned
nscanned:	the number of document scanned to return the result
moved:	if the operation caused a move
millis:	the total time in milliseconds that the operation took
keyupdates:	the number of indexes that required update

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

What to look for

- fastMod
  - Good! Fastest possible update. In-place atomic operator ($inc,$set)
- nretunred vs nscanned
  - If nscanned != nscanned, you may have opportunity to tune. Indexing.
- key updates
  - Secondary indexes. Minimize them
  - ~10% reduction in performance for each secondary index
- moved
  - Documents grow > padding factor
  - You can't fix it other than to pad yourself manually
  - db.collection.stats() shows padding
  - 2.3.1+ usePowerOf2Sizes
  - https://jira.mongodb.org/browse/SERVER-1810
- nreturned; high number of them
  - cardinality
  - Just pure I/O

What doesn't it show?

- IndexOnly
  - Very fast, use explain() instead of profiler
- I/O
  - Page accesses
  - Page calls
  - Random I/O
  - Data density/locality
  - https://jira.mongodb.org/browse/SERVER-3546 (track I/O)
- Plans
  - Only explain() shows the full plan(s)

Profiler Analysis - FCS

```
$>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,
      "keyUpdates" : 0,
      "numYield" : 0,
      "lockStats" : {...},
      "nreturned" : 1,
      "responseLength" : 56,
      "millis" : 29,
      "client" : "127.0.0.1",
      "user" : ""
}
```

ObjectRocket

Profiler Analysis - FCS

```
$>db.system.profile.find({"op":"query","ns":"test.testme"}).pretty();
{     "ts" : ISODate("2013-02-11T20:00:52.015Z"),
      "op" : "query",
      "ns" : "test.testme",
      "query" : { "name" : 1 },
      "ntoreturn" : 0,
      "ntoskip" : 0,
      "nscanned" : 1,
      "keyUpdates" : 0,
      "numYield" : 0,
      "lockStats" : {...},
      "nreturned" : 1,
      "responseLength" : 56,
      "millis" : 0,
      "client" : "127.0.0.1",
      "user" : ""
}
```

ObjectRocket

Profiler Analysis - Moved

```
$> db.system.profile.find({"op":"update"}).pretty();
{     "ts" : ISODate("2013-02-11T20:50:36.882Z"),
      "op" : "update",
      "ns" : "test.testme",
      "query" : {
           "name" : 1
      },
      "updateobj" : { "$set" : { "desc" : ... }},
      "nscanned" : 1,
      "moved" : true,
      "nmoved" : 1,
      "nupdated" : 1,
      ...
      "millis" : 2,
}
```

Profiler Analysis - helpful queries

```
show profile                                                   // last few entries

db.system.profile.find({}).sort({$natural:-1})                 // sort by natural order (time in)

db.system.profile.find({"millis":{$gt:20}})                    // anything > 20ms

db.system.profile.find({"ns":"test.foo"}).sort({"millis":-1})  // single coll order by response time

db.system.profile.find({"moved":true})                         // anything thats moved

db.system.profile.find({"nscanned":{$gt:10000}})               // large scans

db.system.profile.find({"nreturned":{$gt:1}})                  // anything doing range or full scans
```

example: https://gist.github.com/kgorman/c5774670feb7436f4d69

Going Deeper with Profiler Analytics

- In prod environment profiler has lots of data
- Prioritize tuning opportunities
- Prioritize performance issues
- Aggregation, summarization required
    - Enter Aggregation Framework

Aggregation Framework - Example

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

{
"result" : [
      { "_id" : "command", "count" : 1, "max response time" : 0, "avg response time" : 0 },
      { "_id" : "query", "count" : 12, "max response time" : 571, "avg response time" : 5 },
      { "_id" : "update", "count" : 842, "max response time" : 111, "avg response time" : 40 },
      { "_id" : "insert", "count" : 1633, "max response time" : 2, "avg response time" : 1 }
],
      "ok" : 1
}
```

ObjectRocket

Aggregation Framework - Example

```
// 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"}
 }});
```

example: https://gist.github.com/kgorman/995a3aa5b35e92e5ab57

Response time analysis

- Response time = service time + queue time
- Each document in profile collection has a 'millis' attribute.
- The only true measure of response time in MongoDB

Response time analysis

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

{
 "result" :
   [
     { "_id" : "update", "count" : 1, "max response time" : 2, "avg response time" : 2 },
     { "_id" : "command", "count" : 1, "max response time" : 0, "avg response time" : 0 },
     { "_id" : "query", "count" : 268, "max response time" : 40, "avg response time" : 0 },
     { "_id" : "insert", "count" : 1002, "max response time" : 137, "avg response time" : 0 }
   ],
   "ok" : 1
}
```

ObjectRocket

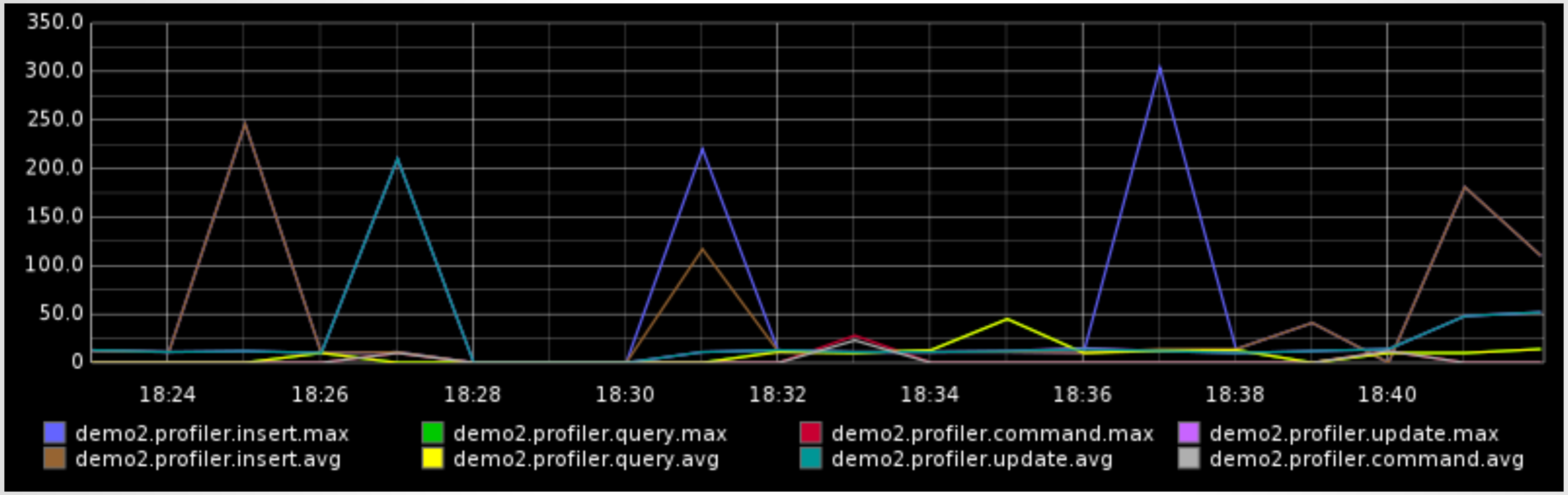
Why is this useful?

- When rolling new code
- Customer activity patterns
- Any time based patterns
- Capacity planning

Going nuts with profiler and time-series

- Turn on profiling
- Pull delta response time data from db.system.profile in aggregate in a loop
- Push to your favorite graphing/time-series program
- https://github.com/kgorman/slum

Going nuts with profiler and time-series



ObjectRocket

Contact

@kennygorman
@objectrocket
kgorman@objectrocket.com

https://www.objectrocket.com