



Finding Slow Operations

DF200 - Optimizing Storage and Retrieval

Finding slow operations

- The most common source of bad performance is a missing index
- This is definitely not the only cause - bad client code is a close second.
- There are two ways to find inefficient Queries
 - The database log
 - Profiling a database

Profiling requires a feature that is not available on Atlas Shared Instances.



Slow operations are usually caused by a missing or incorrect index.
Atlas and Cloud Manager now use the database log for profiling rather than using native profiling.

The getLog Command

The command shown here gets the last 1024 log entries.

Returns a single document - logs are in the **log** array as strings.

In MongoDB 4.4+, these strings are JSON.

Previously they were just text.

```
> var loglines = db.adminCommand({getLog:"global"})
...
> printjson(loglines)
{
  "totalLinesWritten" : 2625918,
  "log" : [ ... ],
  "ok" : 1,
  "$clusterTime" : {
    "clusterTime" : Timestamp(1594293362, 5),
    "signature" : {
      "hash" : BinData(0,"3+Mc0j...2cUERBHnc3Qs="),
      "keyId" : NumberLong("6835033877094858756")
    }
  },
  "operationTime" : Timestamp(1594293362, 5)
}
```

The MongoDB log file can show slow queries

From MongoDB 4.4, the log is JSON, so it can be written to a searchable collection.

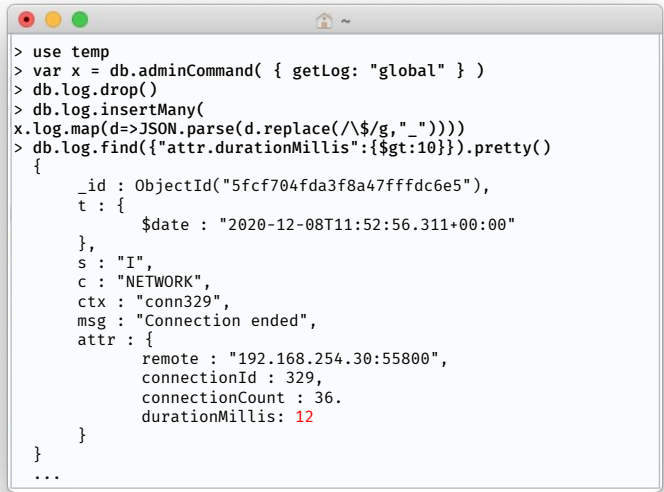
Note: The `getLog` command does not return log lines on Atlas Shared Tiers currently.

Copying the log to a collection

From MongoDB 4.4, all log output is now in JSON format.

You can easily copy the output of `getLog` to a temporary collection.

And then search or aggregate the data.



```
> use temp
> var x = db.adminCommand( { getLog: "global" } )
> db.log.drop()
> db.log.insertMany(
x.log.map(d=>JSON.parse(d.replace(/\$/g, "_"))))
> db.log.find({"attr.durationMillis":{$gt:10}}).pretty()
{
  _id : ObjectId("5fcf704fda3f8a47fffdc6e5"),
  t : {
    $date : "2020-12-08T11:52:56.311+00:00"
  },
  s : "I",
  c : "NETWORK",
  ctx : "conn329",
  msg : "Connection ended",
  attr : {
    remote : "192.168.254.30:55800",
    connectionId : 329,
    connectionCount : 36,
    durationMillis: 12
  }
}
```

We may not always see the results in the beginning of our deployment because the slow operations only appear in the log if the duration exceeds the slow operation threshold. In the versions prior to 4.4, you can put it in a collection, but then only query using regular expressions would be possible as these would be just lines of text.

Note: The `getLog` command does not return log lines on Atlas Shared Tiers currently.

```
use temp
var x = db.adminCommand( { getLog: global } )
db.log.drop()

db.log.insertMany(x.log.map(l=>({l})))
db.log.find({l:/COLLSCAN/}).pretty()
```

The setProfilingLevel command

```
db.setProfilingLevel(  
  level, slowms  
)
```

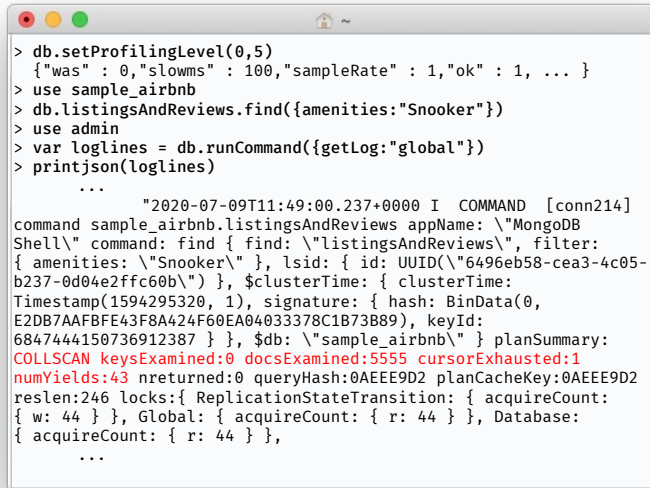
level: 0, 1, or 2

0 - The Profiler is off and does not collect any data.

1 - Record slow queries in a collection for analysis

2 - Record all queries in a collection for analysis

slowms - the time threshold in milliseconds for slow operations



```
> db.setProfilingLevel(0,5)
{"was" : 0,"slowms" : 100,"sampleRate" : 1,"ok" : 1, ... }
> use sample_airbnb
> db.listingsAndReviews.find({amenities:"Snooker"})
> use admin
> var loglines = db.runCommand({getLog:"global"})
> printjson(loglines)
...
"2020-07-09T11:49:00.237+0000 I COMMAND [conn214]
command sample_airbnb.listingsAndReviews appName: \"MongoDB
Shell\" command: find { find: \"listingsAndReviews\", filter:
{ amenities: \"Snooker\" }, lsid: { id: UUID(\"6496eb58-cea3-4c05-
b237-0d04e2ffc60b\") }, $clusterTime: { clusterTime:
Timestamp(1594295320, 1), signature: { hash: BinData(0,
E2DB7AABFBE43F8A424F60EA04033378C1B73B89), keyId:
6847444150736912387 } }, $db: \"sample_airbnb\" } planSummary:
COLLSCAN keysExamined:0 docsExamined:5555 cursorExhausted:1
numYields:43 nreturned:0 queryHash:0AEEE9D2 planCacheKey:0AEEE9D2
reslen:246 locks:{ ReplicationStateTransition: { acquireCount:
{ w: 44 } }, Global: { acquireCount: { r: 44 } }, Database:
{ acquireCount: { r: 44 } },
...
```

The db profiler can also be used to find slow queries - slowms sets the threshold for 'slow' sampleRate is what proportion 0-1 (1 is 100%) of slow queries to log

Note: The setProfilingLevel() command is not allowed on Atlas Shared Tiers.

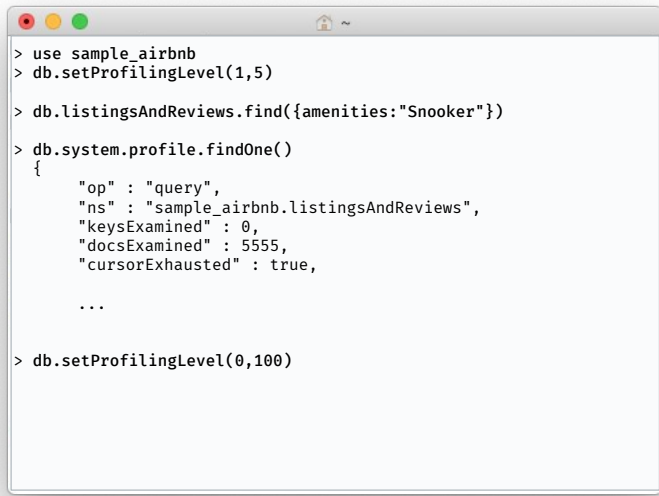
Recording slow operations

Can be used to capture longer term but means extra writes.

**TURN THIS OFF AFTER USE
AS IT CAN SLOW
PRODUCTION!**

Recorded in a capped collection - so only 10MB max.

Visualization and aggregations are possible.



```
> use sample_airbnb
> db.setProfilingLevel(1,5)

> db.listingsAndReviews.find({amenities:"Snooker"})

> db.system.profile.findOne()
{
  "op" : "query",
  "ns" : "sample_airbnb.listingsAndReviews",
  "keysExamined" : 0,
  "docsExamined" : 5555,
  "cursorExhausted" : true,
  ...
}

> db.setProfilingLevel(0,100)
```

Important: The db profiler should be turned off after used as it will degrade performance

Note: The setProfilingLevel() command would not be allowed on Atlas Shared Tiers.

Causes of slow operations

- Missing indexes leading to lots of Disk I/O
- Writes to cache outstripping disk write capability
 - Waiting for cache to flush
- Locking
 - You can see in logs what locks things take
 - Most things don't block others - but a few admin things do
- Excessive CPU usage
 - Constantly logging in
 - Document lock contention.
 - Inappropriately large arrays
 - Running code in the database
- Nested Joins exploding complexity.



There are other causes of slow operations to investigate