01 Rendimiento

mongostat

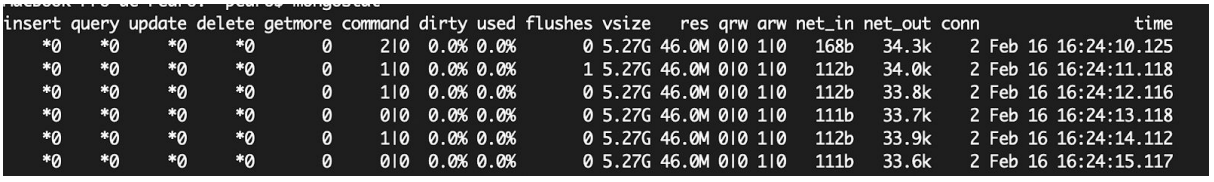
The mongostat utility provides a quick overview of the status of a currently running mongod or mongos instance. mongostat is functionally similar to the UNIX/Linux file system utility vmstat, but provides data regarding mongod and mongos instances.

Run mongostat from the system command line, not the mongo shell.

In order to connect to a mongod that enforces authorization with the --auth option, specify the --username and --password options, and the connecting user must have the serverStatus privilege action on the cluster resources.

The built-in role clusterMonitor provides this privilege as well as other privileges.

mongostat returns values that reflect the operations over a 1 second period. When mongostat <sleeptime> has a value greater than 1, mongostat averages the statistics to reflect average operations per second.



```
insert query update delete getmore command dirty used flushes vsize   res qrw arw net_in net_out conn                time
    *0    *0     *0     *0       0     2|0  0.0% 0.0%       0 5.27G 46.0M 0|0 1|0   168b   34.3k    2 Feb 16 16:24:10.125
    *0    *0     *0     *0       0     1|0  0.0% 0.0%       1 5.27G 46.0M 0|0 1|0   112b   34.0k    2 Feb 16 16:24:11.118
    *0    *0     *0     *0       0     1|0  0.0% 0.0%       0 5.27G 46.0M 0|0 1|0   112b   33.8k    2 Feb 16 16:24:12.116
    *0    *0     *0     *0       0     0|0  0.0% 0.0%       0 5.27G 46.0M 0|0 1|0   111b   33.7k    2 Feb 16 16:24:13.118
    *0    *0     *0     *0       0     1|0  0.0% 0.0%       0 5.27G 46.0M 0|0 1|0   112b   33.9k    2 Feb 16 16:24:14.112
    *0    *0     *0     *0       0     0|0  0.0% 0.0%       0 5.27G 46.0M 0|0 1|0   111b   33.6k    2 Feb 16 16:24:15.117
```

mongotop

mongotop provides a method to track the amount of time a MongoDB instance mongod spends reading and writing data. mongotop provides statistics on a per-collection level. By default, mongotop returns values every second.

Run mongotop from the system command line, not the mongo shell.

mongotop <segundos>

```
MacBook-Pro-de-Pedro:~ pedro$
MacBook-Pro-de-Pedro:~ pedro$ mongotop 5
2020-02-16T16:26:32.132+0100    connected to: mongodb://localhost/

                    ns    total    read    write    2020-02-16T16:26:37+01:00
    admin.system.roles      0ms     0ms      0ms
  admin.system.version      0ms     0ms      0ms
config.system.sessions      0ms     0ms      0ms
   config.transactions      0ms     0ms      0ms
        local.oplog.rs      0ms     0ms      0ms
  local.system.replset      0ms     0ms      0ms

                    ns    total    read    write    2020-02-16T16:26:42+01:00
    admin.system.roles      0ms     0ms      0ms
  admin.system.version      0ms     0ms      0ms
config.system.sessions      0ms     0ms      0ms
   config.transactions      0ms     0ms      0ms
        local.oplog.rs      0ms     0ms      0ms
  local.system.replset      0ms     0ms      0ms
```

Locking Performance

MongoDB uses a locking system to ensure data set consistency. If certain operations are long-running or a queue forms, performance will degrade as requests and operations wait for the lock.

Lock-related slowdowns can be intermittent. To see if the lock has been affecting your performance, refer to the locks section and the globalLock section of the serverStatus output.

Dividing locks.timeAcquiringMicros by locks.acquireWaitCount can give an approximate average wait time for a particular lock mode.

locks.deadlockCount provide the number of times the lock acquisitions encountered deadlocks.

If globalLock.currentQueue.total is consistently high, then there is a chance that a large number of requests are waiting for a lock. This indicates a possible concurrency issue that may be affecting performance.

If globalLock.totalTime is high relative to uptime, the database has existed in a lock state for a significant amount of time.

Long queries can result from ineffective use of indexes; non-optimal schema design; poor query structure; system architecture issues; or insufficient RAM resulting in disk reads.

Práctica

```
> servStatus = db.runCommand( { serverStatus: 1 } )

> servStatus.globalLock
{
        "totalTime" : NumberLong(433019000),
        "currentQueue" : {
                "total" : 0,
                "readers" : 0,
                "writers" : 0
        },
        "activeClients" : {
                "total" : 0,
                "readers" : 0,
                "writers" : 0
        }
}
```

Number of Connections

In some cases, the number of connections between the applications and the database can overwhelm the ability of the server to handle requests. The following fields in the serverStatus document can provide insight:

connections is a container for the following two fields:
connections.current the total number of current clients connected to the database instance.
connections.available the total number of unused connections available for new clients.
If there are numerous concurrent application requests, the database may have trouble keeping up with demand. If this is the case, then you will need to increase the capacity of your deployment.

For read-heavy applications, increase the size of your replica set and distribute read operations to secondary members.

For write-heavy applications, deploy sharding and add one or more shards to a sharded cluster to distribute load among mongod instances.

Spikes in the number of connections can also be the result of application or driver errors. All of the officially supported MongoDB drivers implement connection pooling, which allows clients to use and reuse connections more efficiently. An extremely high number of connections, particularly without corresponding workload, is often indicative of a driver or other configuration error.

Unless constrained by system-wide limits, the maximum number of incoming connections supported by MongoDB is configured with the maxIncomingConnections setting. On Unix-based systems, system-wide limits can be modified using the ulimit command, or by editing your system's /etc/sysctl file. See UNIX ulimit Settings for more information.

Práctica

```
> servStatus = db.runCommand( { serverStatus: 1 } )

> servStatus.connections
{ "current" : 1, "available" : 203, "totalCreated" : 1, "active" : 1 }
```

Database Profiling

The Database Profiler collects detailed information about operations run against a mongod instance. The profiler's output can help to identify inefficient queries and operations.

You can enable and configure profiling for individual databases or for all databases on a mongod instance. Profiler settings affect only a single mongod instance and will not propagate across a replica set or sharded cluster.

The following profiling levels are available:

Level	Description
0	The profiler is off and does not collect any data. This is the default profiler level.
1	The profiler collects data for operations that take longer than the value of slowms (100 ms default).
2	The profiler collects data for all operations.

Enable and Configure Database Profiling

You can enable database profiling for mongod instances .

This section uses the mongo shell helper db.setProfilingLevel() helper to enable profiling. For instructions using the driver, see your driver documentation.

When you enable profiling for a mongod instance, you set the profiling level to a value greater than 0. The profiler records data in the system.profile collection. MongoDB creates the system.profile collection in a database after you enable profiling for that database.

To enable profiling and set the profiling level, pass the profiling level to the db.setProfilingLevel() helper.

db.setProfilingLevel(<nivel>, { slowms: <milisegundos> })

Práctica

```
> use maraton
> db.setProfilingLevel(2)
{ "was" : 0, "slowms" : 100, "sampleRate" : 1, "ok" : 1 }
```

The database profiler writes data in the system.profile collection, which is a capped collection. To view the profiler's output, use normal MongoDB queries on the system.profile collection.

For any single operation, the documents created by the database profiler will include a subset of the following fields. The precise selection of fields in these documents depends on the type of operation.

```
> db.system.profile.find().pretty()
```