How to improve MongoDB performance with proper queries

Percona Webinar - November 29th
Tim Vaillancourt

PERCONA

`whoami`

```
{
        name: “tim”,
        lastname: “vaillancourt”,
        employer: “percona”,
        techs: [
                “mongodb”,
                “mysql”,
                “cassandra”,
                “redis”,
                “rabbitmq”,
                “solr”,
                “kubernetes”,
                “mesos”
                “kafka”,
                “couch*”,
                “python”,
                “golang”
        ]
}
```

Agenda

- Document Design
- Common query operators
- Indexes
- Evaluating queries performance
- Good Practices
- Common Issues
- Q&A

Document Design

Document Design

- MongoDB is schema less database which means no predefined schema but this doesn't means it will perform well in a really messed schema.
- Avoid complex documents as it may become complex to query.

Document Design - Bad Practice

```
db.people.findOne()
{
        "_id" : ObjectId("5bfd90b18791fb8236b8ab13"),
        "name" : "Joseph",
        "address_id" : ObjectId("5bfd90c58791fb8236b8ab14")
}
db.addresses.findOne()
{
        "_id" : ObjectId("5bfd90c58791fb8236b8ab14"),
        "Street" : "Street Name",
        "number" : 100,
        "zip" : "ASDFG"
}
```

PERCONA

Document Design - Good Practice

```
db.people.find().pretty()
         {
             "_id" : ObjectId("5bfd908a8791fb8236b8ab11"),
             "name" : "Joseph",
             "address" : {
                 "street" : "Street Name",
                 "number" : 100,
                 "zip" : "ASDFG"
             }
         }
```

Document Design

- Do not create deep level of documents as it may become complex to query
- Would be easier to query by the top level fields and also to index. (We will talk about it shortly)

Document Design

- Use efficient data types to save storage + index size + RAM
  - Store string date: ***"2018-11-29T17:24:50.288Z"***
    - as -
    ISODate("2018-11-29T17:24:50.288Z") *(48% smaller)*
  - Store string number: ***“12321139”***
    - as -
    12321139 *(25% smaller)*
  - Store string boolean: **“true”**
    - as -
    true *(53% smaller)*

PERCONA

Common Query Operators

Common Query Operators

- MongoDB uses javascript like query language and for who is just starting with MongoDB it may seem a bit confusing.
- The default mongodb syntax follows: db.collection.command({args}) being:

db = database variable
Collection = the collection name
Command an operation such as find() or remove
{args} argument for the operation such as the where clause.

Common Query Operators

- The following query will return all the documents in a collection that matches the name field.

```
> db.foo.find({name : 'Joseph'})
```

- The following query will return the same result, but with only the *"_id" (default)* and *"name"* fields; this is more efficient.

```
> db.foo.find({name : 'Joseph'},  {name: 1})
```

Common Query Operators

- Additionally, for numeric/date fields it is possible to use: $gt, $gte, $lt, $lte, $in

```
> db.foo.find({code : {$gte : 1, $lte: 10}})
> db.foo.find({code : {$in : [1,2,3,4,5,6,7,8,9,10]}})
```

$lte means less than and equal and the $lt only less than. Doesn't include the value for comparison.

Common Query Operators

- It is possible to check if a field exists as well using the $exists command.

```
> db.foo.find({address : {$exists : true})
```

PERCONA

Common Query Operators

- By default the operator is $and, so, there is no need to use $and just add one expression after another.

```
> db.foo.find({$and : [{name : 'Joseph'}, {address : {$exists : true}}]})
> db.foo.find({name : 'Joseph', address : {$exists : true}})
```

PERCONA

Common Query Operators

- The $or command works very similarly to the $in.
  - Caveat: $or and $in will run more than one query in background.

```
> db.foo.find({$or : [{code : 1}, {code : 2}])
> db.foo.find({code : {$in : [1,2]}})
```

- $or will <u>collection-scan</u> unless all clauses have an index!
- $in generally executes faster when matches are for 1 field only

PERCONA

Common Query Operators

- Regex is also an option to search for partial text:

```
> db.foo.find({name : /seph/})
```

- Regex can be expensive. For lots of text searches, try $text:

```
> db.foo.find( { $text: { $search: "seph" } } )
```

Note: only 1 $text index is possible per-collection

Common Query Operators

- We can use all the operators at once in order to retrieve the data we need.

```
> db.foo.find({name : /seph/, address : {$exists: true}, age : {$gte :
30}})
```

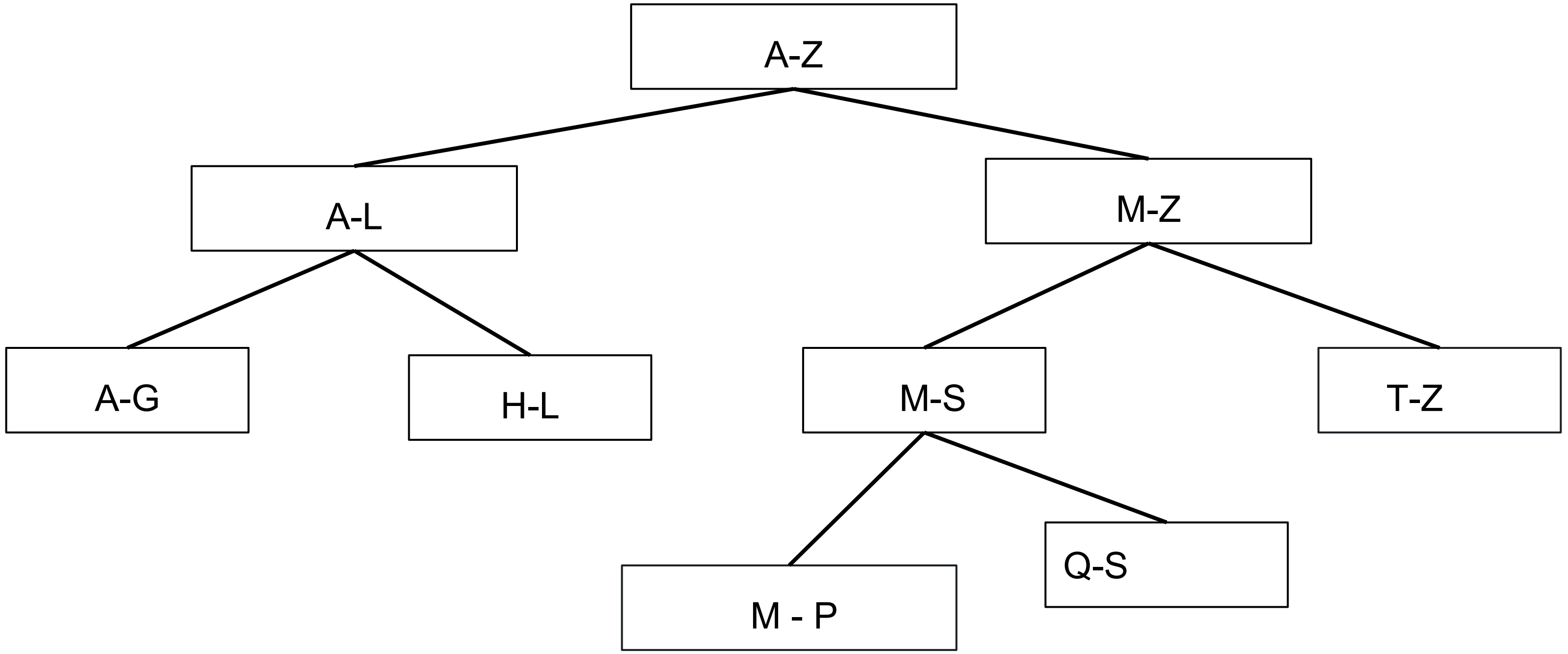
Indexes

How does index work?

- Not only MongoDB but most of the databases uses b-tree style indexes

- Indexes help the database to "know" where a document is and if it exists, instead of reading the entire collection the optimizer can evaluate if the data can be retrieved by the index

How does index work?



How does index work?

- Running a query without indexes will result in a collection scan.
- Collection Scan in, in the other words a process where all the documents in a collection is opened, evaluated against the where condition and then returned or not.

How does index work?

- Documents may be huge and the collection scan is really resource intensive.
- Indexes will be the "shortcut" to the query optimizer to find the docs.

Good Practices

- Do not create a lot of indexes as for each index the overhead to write increases
- Use the more restrictive field as the first field in a index.
- Avoid duplicate indexes
- Match the index direction to your common sort patterns:
    - Ascending (-1)
    - Descending (1)

PERCONA

Good Practices

- Compound Indexes
  - Several fields supported that are read Left -> Right
    - *Index can be partially-read in queries*
  - Left-most fields should not be duplicated!
    - *All Indexes below are duplicates of the first index:*

      {username: 1, status: 1, date: 1, count: -1}

      {username: 1, status: 1, data: 1 }

      {username: 1, status: 1 }

      {username: 1 }

Evaluating queries performance

Evaluating queries performance

- Within small databases (dataset entire in ram) it is very unlikely to have performance issues.
- However in a production environment with high concurrency and with the low response time expected we need to make sure the queries are running with the best execution plan.

Evaluating queries performance

- But.. what is the Query Optimizer?
  - Like other databases the MongoDB query language is declarative. Meaning we ask the database to get the data but we don't teach the database how to do it.
  - The Query optimizer is the responsible to finding the best path to get the data.

Evaluating queries performance

- If there are no indexes in the database the Query Optimizer will always read all the documents to answer a query.

- Each index increase the cost of performing a write, so, creating a lot of indexes will help

PERCONA

How to evaluate index usage:

- We can use .explain() to evaluate the execution plan;

```
db.foo.find({myvalue :
'cfcd208495d565ef66e7dff9f98764da'}).explain(
)
```

Examples

```
"queryPlanner" : {
     "plannerVersion" : 1,
     "namespace" : "percona.foo",
     "indexFilterSet" : false,
     "parsedQuery" : {
          "myfield" : {
               "$eq" : "cfcd208495d565ef66e7dff9f98764da"
          }
     },
     "winningPlan" : {
          "stage" : "COLLSCAN",
          "filter" : {
               "myfield" : {
                    "$eq" : "cfcd208495d565ef66e7dff9f98764da"
```

Evaluating queries performance

- As the query explain returned COLLSCAN we are absolutely sure there is no index in the 'myfield' field and the database needed to read all the documents in order to return the result.

Examples

```
"queryPlanner" : {
"plannerVersion" : 1,
"namespace" : "percona.foo",
     "indexFilterSet" : false,
     "parsedQuery" : {
          "myvalue" : {
               "$eq" : "cfcd208495d565ef66e7dff9f98764da"
          }
     },
     "winningPlan" : {
          "stage" : "FETCH",
          "inputStage" : {
               "stage" : "IXSCAN",
```

Finding Slow Queries

- We are not going to execute every query one by one by hand. MongoDB features slow query log as well as the profiler to help us finding slow queries.

Finding Slow Queries in the Logs

```
command percona.smalcollection
appName: "MongoDB Shell" command: find { find: "foo", filter: { myfield: "aasvere" } }
planSummary: COLLSCAN keysExamined:0 docsExamined:897620 cursorExhausted:1 numYields:7017
nreturned:0 reslen:95 locks:{ Global: { acquireCount: { r: 14036 } }, Database: {
acquireCount: { r: 7018 } }, Collection: { acquireCount: { r: 7018 } } }
protocol:op_command 691ms


2018-11-28T16:15:02.616-0200 I COMMAND  [conn6] command percona.smalcollection appName:
"MongoDB Shell" command: find { find: "foo", filter: { myfield: "LOJgBj9Qnf" } }
planSummary: COLLSCAN keysExamined:0 docsExamined:897620 cursorExhausted:1 numYields:7015
nreturned:1 reslen:383 locks:{ Global: { acquireCount: { r: 14032 } }, Database: {
acquireCount: { r: 7016 } }, Collection: { acquireCount: { r: 7016 } } }
protocol:op_command 546ms
```

Finding Slow Queries in the Logs

```
2018-11-28T16:17:43.388-0200 I COMMAND  [conn7] command percona.smalcollection appName:
"MongoDB Shell" command: find { find: "foo", filter: { myfield: "LOJgBj9Qnf" } }
planSummary: IXSCAN { myfield: 1 } keysExamined:1 docsExamined:1 cursorExhausted:1
numYields:0 nreturned:1 reslen:383 locks:{ Global: { acquireCount: { r: 2 } }, Database: {
acquireCount: { r: 1 } }, Collection: { acquireCount: { r: 1 } } } protocol:op_command 2ms
```

Finding Slow Queries with Profiler

- With profiler it is possible to record any query taking more than x seconds or all the queries.
- Profiler is per database and will create a collection called system.profile.
- Commands to enable profiler:

```
db.setProfilingLevel(1,<time ms>)
db.setProfilingLevel(2)
```

Finding Slow Queries with Profiler

```
db.system.profile.find().sort({$natural :-1}).limit(1).pretty()
{
     "op" : "query",
     "ns" : "percona.foo",
     "query" : {
          "find" : "foo",
          "filter" : {
               "myfield" : "aasvere"
          }
     },
     "keysExamined" : 0,
     "docsExamined" : 0,
     "cursorExhausted" : true,
```

PERCONA

Good Practices

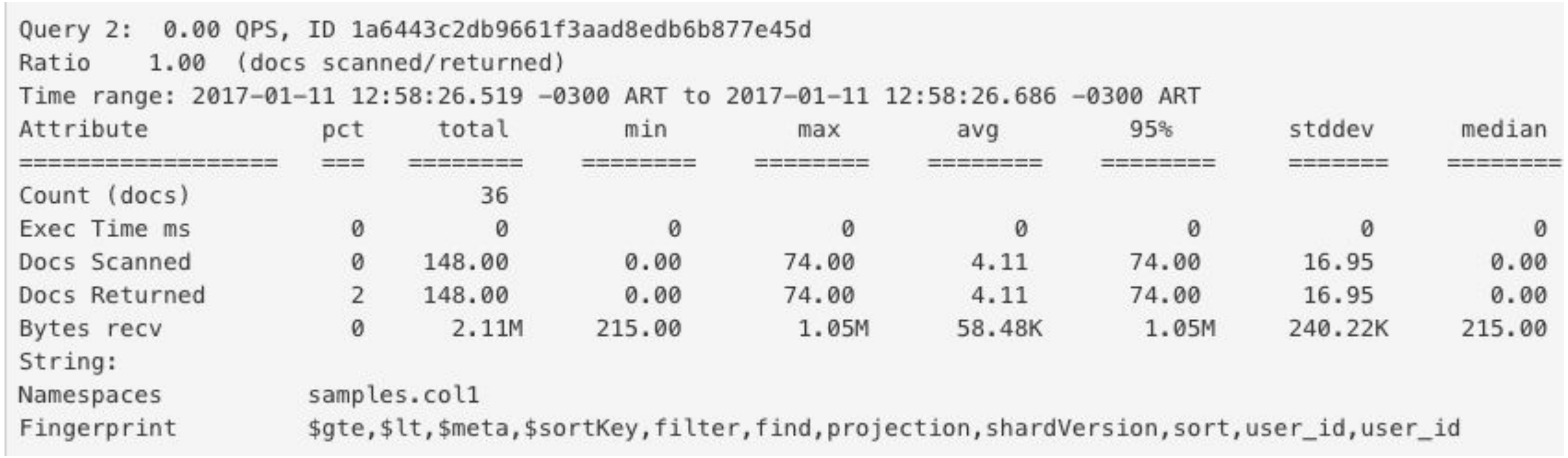
Good Practices

- There are a lot of tools out there to analyze slow queries in MongoDB:
    - PT-QUERY-DIGEST
    - MTOOLS
    - "GREP"

Evaluate queries plan

- pt-query-digest creates a report of the slowest queries in the database.
- It uses the system.profile collection with recorded queries to generate a list of common slow queries

Evaluate queries plan

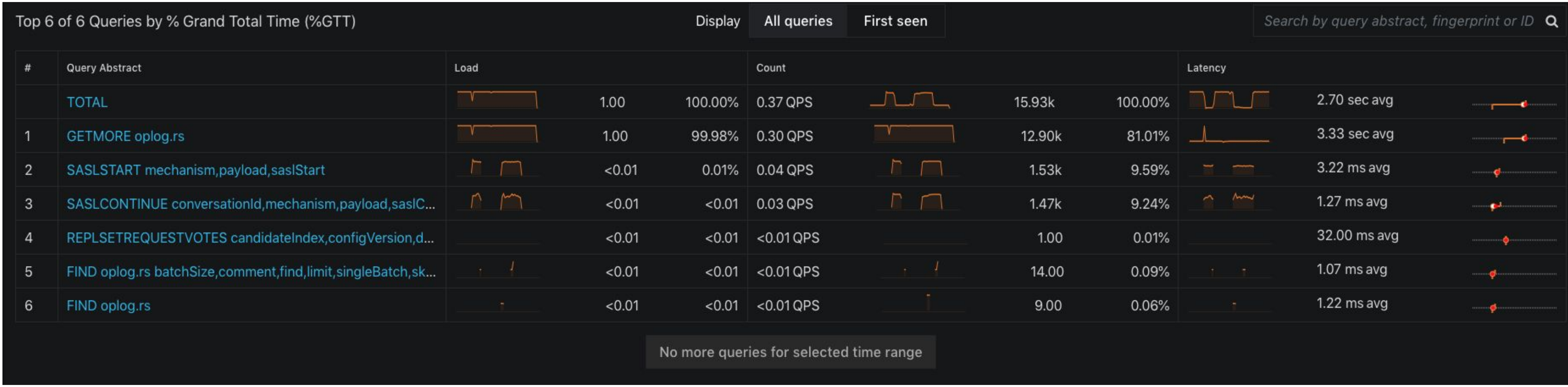


```
Query 2:  0.00 QPS, ID 1a6443c2db9661f3aad8edb6b877e45d
Ratio    1.00  (docs scanned/returned)
Time range: 2017-01-11 12:58:26.519 -0300 ART to 2017-01-11 12:58:26.686 -0300 ART
Attribute                  pct     total        min         max        avg         95%        stddev      median
==================         ===   ========    ========    ========    ========    ========     =======    ========
Count (docs)                           36
Exec Time ms                 0          0           0           0           0           0           0           0
Docs Scanned                 0     148.00        0.00       74.00        4.11       74.00       16.95        0.00
Docs Returned                2     148.00        0.00       74.00        4.11       74.00       16.95        0.00
Bytes recv                   0      2.11M      215.00       1.05M      58.48K       1.05M     240.22K      215.00
String:
Namespaces                samples.col1
Fingerprint               $gte,$lt,$meta,$sortKey,filter,find,projection,shardVersion,sort,user_id,user_id
```

Evaluate queries plan PMM

- If you'd like to see in a graphical way we have PMM with tons of metrics and also the QAN
- This is very useful for a quick overview in the system

Evaluate queries plan



Top 6 of 6 Queries by % Grand Total Time (%GTT)

Display: All queries | First seen

Search by query abstract, fingerprint or ID

#	Query Abstract	Load		Count			Latency
	TOTAL	1.00	100.00%	0.37 QPS	15.93k	100.00%	2.70 sec avg
1	GETMORE oplog.rs	1.00	99.98%	0.30 QPS	12.90k	81.01%	3.33 sec avg
2	SASLSTART mechanism,payload,saslStart	<0.01	0.01%	0.04 QPS	1.53k	9.59%	3.22 ms avg
3	SASLCONTINUE conversationId,mechanism,payload,saslC...	<0.01	<0.01	0.03 QPS	1.47k	9.24%	1.27 ms avg
4	REPLSETREQUESTVOTES candidateIndex,configVersion,d...	<0.01	<0.01	<0.01 QPS	1.00	0.01%	32.00 ms avg
5	FIND oplog.rs batchSize,comment,find,limit,singleBatch,sk...	<0.01	<0.01	<0.01 QPS	14.00	0.09%	1.07 ms avg
6	FIND oplog.rs	<0.01	<0.01	<0.01 QPS	9.00	0.06%	1.22 ms avg

No more queries for selected time range

Evaluate queries plan

- MTOOLS in the other hand can parse logs and show the slowest queries, most time consuming and a lot of different filters/sorts

- More info can be found: http://blog.rueckstiess.com/mtools/mlogfilter.html

Evaluate queries plan

- In house scripts are very useful, there are a couple of scripts in github to help you to evaluate query execution plan and time

- https://github.com/search?q=mongodb+log+parser

PERCONA

Common Issues

Common Issues

- When running aggregation, if using $unwind the new document doesn't have index.
- The space used by $unwind is equal to the size of the document multiplied by the array.
- Using .skip() doesn't improve performance, in fact it does increase the response time.
- $where command doesn't use index.

PERCONA

Common Issues

- Tons of COLLSCAN can bring your server down.
- The oplog.rs collection doesn't have any index. It will always run a COLLSCAN.
- Server side queries (map reduce) doesn't appear in the currentOP() and runs outside of the database scope - May lead to an OOM issue

Common Issues

- Usually performance is better when the working set + indexes fits in RAM.

Questions