MongoDB Analysis with Prometheus and Grafana

Akira Kurogane
Percona
Talk Overview

- The 'math' in MongoDB metrics
  - MongoDB's counters and 'gauges'
  - mongodb_exporter metrics
  - Prometheus equations
- PMM's Grafana dashboards
- How to cook new dashboards
The 'Math' in MongoDB Metrics

Implementation

- Mostly Counters
  Eg. opcounters, bytes transferred
- Gauge values
  Eg. open cursors, WT 'tickets' in use
- Histograms
  Counters vs sub-ranges

Exported as timeseries to a monitoring server.

Mostly any software's

Eventually enters your brain as:

- A graph
- An alert threshold

The graph / threshold value is:

- Rate function on a counter
- Instantaneous value of a gauge
- Derivative functions combining several of the above
"That Can't Be Sane!"

It gets scarier - metric op rate is very high.

100,000+ queries or updates per second

==

> several million counter increments per second

Plus daemon 'housework' threads:

● Storage engine, Replication, Network, Journal, etc.
namespace mongo {

void CollectionIndexUsageTracker::recordIndexAccess(StringData indexName) {
    invariant(!indexName.empty());
    dassert(_indexUsageMap.find(indexName) != _indexUsageMap.end());
    _indexUsageMap[indexName].accesses.fetchAndAdd(1);
}

void CollectionIndexUsageTracker::registerIndex(StringData indexName, const BSONObj& indexKey) {
    invariant(!indexName.empty());
    dassert(_indexUsageMap.find(indexName) == _indexUsageMap.end());
    // Create map entry.
    _indexUsageMap[indexName] = IndexUsageStats(_clockSource->now(), indexKey);
}

CollectionIndexUsageMap CollectionIndexUsageTracker::getUsageStats() const {
    return _indexUsageMap;
}
## Gauges, Histograms

<table>
<thead>
<tr>
<th>Component</th>
<th>Code Example</th>
</tr>
</thead>
</table>
| Counter   | `metricX.fetchAndAdd()`  
            | `WT_STAT_*_INCR(..., metric_x)` |
| Gauge     | `metricX.set(x)`  
            | `WT_STAT_SET(..., metric_x)` |
| Histogram | Counters, put in bucket ranges automatically.  
            | (Only one so far: $collStat's latencyStats) |
Reading the MongoDB Metrics

db.serverStatus()

OpCounter
Network, Connections
WiredTiger (or MMAP)
ReplicationInfo, OplogInfo
Sharding, ShardingStatistics
Transactions, LogicalSession
GlobalLock, LockStats
OpReadConcern, OpWriteConcern
Storage, DataFileSync, DurSSS
...
...

SS does **NOT** include these stats:
- Database
- Collection
- Index

E.g. doc count or avg. size, storageSize, access counts **per** db / coll / index

Iterate **each** DB, collection & index to get those **as well as** calling serverStatus.
Side-Topic: FTDC

= Metrics persisted to disk once per second

Code search: FTDC controller's `addPeriodicCollector`

- `serverStatus` 935 stats
- `replSetGetStatus` 81 stats
- `collStats` on local.oplog.rs 152 stats
- `linux/windows OS metrics` 211 stats
- `connpool` stats (on mongos)

Not purely internal - Try the `{ getDiagnosticData: 1 }` command.

Impact: 'cheap' cache re-read but a lot of BSON/JSON data.

N.b. does NOT include per-database, per-collection and per-index stats
MongoDB --> Prometheus
mongodb_exporter

Passively awaits Prometheus server's call once per x seconds

Returns up to 200+ metrics from:

- serverStatus
- replsetGetStatus
- Optionally
  - dbStats
  - collStats (to become $collStats)
  - $indexStats
Installing / Running

PMM:

pmm-admin  add  mongodb  [NAME]  [OPTIONS]
pmm-admin  [list  |  stop  |  restart  |  remove  |  ...]  [NAME]  [OPTIONS]

Command line:

./mongodb_exporter
-mongodb.uri  "mongodb://user:pwd@localhost:27017/...."
-collect.database
-collect.collection
-collect.indexusage
..
Eg. Prometheus Metrics (serverStatus)

```python
metrics.queryExecutor.scannedObject =
    mongodb_mongod_metrics_query_executor_total{state="scanned_objects"}

wiredTiger.cache["bytes written from cache"] =
    mongodb_mongod_wiredtiger_cache_bytes_total{type="written"}

connections.current =
    mongodb_connections{state="current"}

network.bytesIn =
    mongodb_network_bytes_total{state="in_bytes"}

...
Eg. Prometheus Metrics ($collStats, etc.)

Collection's aggregate indexes size =
  mongodb_mongod_db_coll_indexes_size{db="x",collection="y"}

Index accesses =
  mongodb_mongod_index_usage_count{db="x",collection="y",index="z"}

Read latency =
  mongodb_mongod_op_latencies_latency_total{type="read",db="x",collection="y"}

Write latency =
  mongodb_mongod_op_latencies_latency_total{type="write",db="x",collection="y"}

...
Viewing With Prometheus's Own GUI

Prometheus graph page

In PMM2:
https://<host>/prometheus/graph
(PMM admin role users only)

A web console to
- Discover metric names
- View as graph
- View labelled metrics at a single point of time.
How Long Can You Store the Statistics?

Depends on **disk space** budget.

By my subjective picture of small and big budgets:

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Small budget</th>
<th>Big budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 sec 😞</td>
<td>1 day</td>
<td>1 week</td>
</tr>
<tr>
<td>10 sec</td>
<td>1 week</td>
<td>1 month</td>
</tr>
<tr>
<td>1 min ☹</td>
<td>6 month</td>
<td>Years</td>
</tr>
</tbody>
</table>

N.b. no automatic downscaling of metrics resolution for older time ranges.
Summary So Far...

- MongoDB has a very large number of statistics
- Very high resolution: Typically many metrics updated every `microsecond`
- 100+ metrics exported to Prometheus.
  (Optionally enable per-collection etc. metrics as well.)
- Metric names changed - but similar.
- Counters still counters; Gauges still gauges.
- Labels: `metric_aa_x`, `metric_aa_y` => `m_aa{label="x"}`, `m_aa{label="y"}`
- Now you can access MongoDB metric `history`. 
Displaying the Metrics
"Pass to Graphing GUI, Done. Right?"

Counter metric types, gauge metric types
⇒ both become a y value over a time x-axis in graphs.

"So it's a learning step for the first two or so, but after that it's all the same right?"

No - DBAs need to see a 'natural' picture.
Many different 'natural' concepts (Grrr! Humans! Grrr!)

Also need to join related metrics in Prometheus - tricky.
# Various Shapes of Prometheus Equations

<table>
<thead>
<tr>
<th>Cursors open, WT tickets</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ops, bytes, ..</td>
<td>rate(x)[interval]</td>
</tr>
<tr>
<td>Lag</td>
<td>x{state=&quot;PRIMARY&quot;} - x{state=&quot;SECONDARY&quot;}</td>
</tr>
<tr>
<td>Mem threshold</td>
<td>x{server=hhh} / y{server=hhh}</td>
</tr>
<tr>
<td>Reads on all secondaries</td>
<td>(z{state=&quot;SECONDARY&quot;} - z{state=&quot;SECONDARY&quot;}) + x</td>
</tr>
<tr>
<td>Cluster totals</td>
<td>sum(.....&lt;various&gt; ....) by (cluster)</td>
</tr>
<tr>
<td>Worst latency on a primary</td>
<td>max(x{cluster=ccc} + (z{state=&quot;P.&quot;} - z{state=&quot;P.&quot;}))</td>
</tr>
<tr>
<td>Shard imbalance of x</td>
<td>max(sum(x) by (replset)) - min(sum(x) by (replset))</td>
</tr>
</tbody>
</table>
Prometheus Vector Matching - Tricky

\[\text{metric}_x + \text{metric}_y \quad \text{(or } *, /, -)\]

*similar concept to:*

```sql
SELECT x.value + y.value, x.label_1, x.label_2, ...
FROM metric_x x INNER JOIN metric_y y
  ON x.label_1 = y.label_1
  AND x.label_2 = y.label_2 AND .. (join all labels)
```

Runtime error if vector match 'join' fails is invalid.

Vector-label modifying operators often needed:

- `on(...)`, `unless(...)`, `group_left(...)`, ... `ignoring(...)`, etc.
Many Unique, Complex, Equations

When in Grafana you can edit/explore a graph to see the equation.

Some are simpler, some are more complex. The above is average.
Prometheus as a Graph's Datasource

- Same Prom. equation
- Accepts substitute $variable$ values from the GUI.
- Every timeseries + label combination becomes its own line.
- Unless:
  - Filter to single
  - Aggregate to single
  - Use 'repeat graph on $variable$' option.
PMM's MongoDB Dashboards in PMM
Percona Monitoring and Management

PMM Server containerizes:
- Prometheus
- Grafana
- pmm-managed daemon
- Additional web services (e.g. QAN)
- Backing DBs for the above (e.g. tsdb for Prometheus)

After PMM server started install PMM Client on MongoDB host servers. Then:

```
pmm-admin config --server <ip_address>
pmm-admin add mongodb [OPTIONS]
```
Default MongoDB Dashboards in PMM2

For comparing all MongoDB nodes in the environment at once:

- Services Overview

For a cluster (or subset):

- Cluster Summary: Sharding stats
- Overview: Various stats aggregated up
- Summary: A subset of Overview
- Replset: Elections, oplog lag, oplog volume
- Storage Engines: WiredTiger, MMAP, RocksDB, InMemory
- Compare: Instance side-by-side comparison
Roll Your Own Dashboards
"Democratize Metrics"

Grafana / Raintank 2015:

"Make the tools of observability accessible to everyone in an organization, not just the single Ops person."

See and edit dashboards and graphs through the same Web GUI.

(Edit for "admin" role users, at least.)
Editing PMM's MongoDB Dashboards

PMM's packaged MongoDB dashboards are generic.

Specialize for your environment
- Save a copy as new dashboard and edit as you like.
- Ignore / cut out what you don't need
- Merge with other dashboards’ graphs, etc.

If you edit packaged dashboards in place:
- Overwritten by PMM updates
MongoDB Dashboard Mash-up Demo

(Graph demo in Grafana front-end)