Table of contents

Introduction

Deployment

  Deploy MongoDB in replica set mode

  Read preferences

  Memory size

    Working set

    SWAP configuration

  Storage size

MongoDB performance improvement

  Indexes

    Default collections indexes

    Improved indexes

Horizontal scaling

Time to Live (TTL)

  Advantages and disadvantages of using TTL

  Recommendations

    Retention period

    Data usage and backups

Log rotation

  Possibilities of log rotation in MongoDB

Archive, back up, and restore data

  Options

    Special aspects of backup/restore processes

    Restore the backup data

Upgrade MongoDB

  MongoDB 2.4.x to 2.6.x upgrade steps

  Important points

Troubleshooting

  Most frequent issues

    Low performance of getting data from database

    Failure to rechoose a new primary node

    Failure to automatically reconnect to the MongoDB server

Troubleshooting steps

  Discover the issue

  Perform maintenance on replica set members

  Monitoring tools
Introduction

The main aim of this document is to describe best practices of using MongoDB as a back end for the Ceilometer project. Ceilometer is a Telemetry service for the OpenStack environment. It is used for metering, monitoring, and alarming. For all these cases, Ceilometer collects a great amount of data that needs to be stored in the database.

MongoDB is an open-source NoSQL document-oriented database that provides high availability, high performance, and automatic scaling. It has a number of features and supports various types of objects. Moreover, it does not require much effort for support. MongoDB is one of the databases that can be used as a back end for Ceilometer, but it is only one that supports all operations and propose high performance.

“MongoDB stores data in the form of documents, which are JSON-like field and value pairs. Documents are analogous to structures in programming languages that associate keys with values (for example, dictionaries, hashes, maps, and associative arrays). Formally, MongoDB documents are BSON documents. BSON is a binary representation of JSON with additional type information. In the documents, the value of a field can be any of the BSON data types, including other documents, arrays, and arrays of documents.” (From official MongoDB documentation. For details, see MongoDB CRUD Introduction.)

This document contains highly recommended tips and tricks that will help you to achieve good performance of using MongoDB as a back end for Ceilometer.

Deployment

This section describes deployment recommendations.

Deploy MongoDB in replica set mode

For detailed information on how to deploy the database in replica set mode, refer to MongoDB documentation: Replica set deployment. Pay attention that the replica set cluster should always have an odd number of nodes (at least >= 3). This requirement is highly recommended by the MongoDB team. To achieve odd number of MongoDB nodes if it is chosen to use even MongoDB servers, replica set should be added with the Arbiter server. For details, see Replica Set Arbiter.

Read preferences

The first step to improve performance is to change ReadPreferences for the database from default primary to primaryPreferred. This helps to avoid blocking for reading operations. If
something happens with the primary node, it starts using the secondary node. In all other cases, read and write operations are performed within the primary node. This provides consistency of the data. For details, see Read Preference Reference.

Memory size

To estimate the memory size that is required by MongoDB, consider two main points:

- For a good MongoDB performance, all indexes should be in RAM. Otherwise, it will lead to a great increasing of disk I/O operations and huge degradation of database performance.
- Keep the MongoDB working set in RAM. The working set is the data that is frequently used by MongoDB.

Working set

In MongoDB 2.x, use the `db.serverStatus({ workingSet: 1 })` command to make a raw estimation for the working set. In the `workingSet` output section, see the value of the `pagesInMemory` parameter. Each page in memory is around 4 KB.

In MongoDB 3.x, the `workingSet` section has been removed from the `serverStatus` output. For a correct estimation, `workingSet` should be monitored with ordinary load in your cloud for a certain period on staging environment.

Other estimations can be based on the data you expect to deal with. For example, you are interested in the data for the last week and prepare reports every week. In this case, to estimate the `workingSet`, you can take the size of the data that is collected in one week.

Using journaling (turned on by default), MongoDB has two internal views of data set: the `private view`, used to write to the journal files, and the `shared view`, used to write to the data files. Each of the views contains full MongoDB working set. That is, there should be enough memory to fit the working set multiplied by two.

SWAP configuration

By default, MongoDB is deployed on controller nodes where many RAM consumers are located. In such deployment (MongoDB on controllers) without swap, the system can not be reliant in situations with high memory constraints or when multiple services are using the same memory. You can configure the system with swap space.

Storage size

By default, Ceilometer gathers the maximal possible data. The required storage size depends on two parameters:

- The number of gathered metrics
- The polling interval
You can configure both parameters in the /etc/ceilometer/pipeline.conf file. You can specify not only the common number of metrics and general polling interval, but also a special polling interval for one or all metrics if needed.

A good practice is to investigate what metrics you are really interested in and collect only them with frequency not higher than is needed for your purpose. For more information about metrics, see Measurements.

To calculate the storage size, use the following formula:

\[
\text{Storage size} = \left[ \frac{\text{number of instances} \times \text{number of metrics}}{\text{polling interval 1}} + \left( \frac{\text{number of instances} \times \text{number of metrics}}{\text{polling interval 2}} + \ldots \right) \right] \times \text{retention period} \times \text{average metric size}
\]

where:

**Number of instances** is the number of resources.

**Number of metrics** is the number of metrics to be collected.

**Polling interval** is the polling interval in seconds.

**Retention period** is the retention period in seconds.

**Average metric size** is the average metric size (2018 bytes).

If some metrics should be collected with a different polling interval, add additional decimals in parentheses with appropriate parameters for number of metrics and number of instances for a new polling interval (polling interval 2 in the formula above).

As a result, you will get the storage size that is required for the polling part of the data being collected.

For example, consider the following conditions:

1. 1000 virtual instances in a cloud
   - 20 metrics
   - 60 seconds polling interval
2. 100 resources of some other type (for example, SNMP)
   - 9 metrics
   - 600 seconds polling interval
3. The retention period is 2592000 seconds (30 days)
4. The average metric size is 2018 Bytes

For these conditions, the calculation is as follows:

\[
\text{Storage size} = \left[ \frac{1000 \times 20}{60} + \left( \frac{100 \times 9}{600} \right) \times 2592000 \times 2018 \right]
\]
MongoDB performance improvement

Indexes

Default collections indexes

By default, Ceilometer creates several indexes in each collection. It is one single-field index based on the timestamp field (if TTL is used for collection with a timestamp, the index contains an `expireAfterSeconds` parameter) and one or several compound indexes which consist of several fields that are expected to be frequently used in mostly all queries. The compound index gives the highest performance of the search process in MongoDB. The compound index also has limitations. The compound index works only for queries that contain an index “prefix” field. “Prefix” is the first field in the list that was used for compound index creation. For meter collection, it is the `resource_id` field, for resource collection it is `user_id` field, and for event collection it is `event_type` field. Therefore, if you do not have a “prefix” field in a query, the index will not be used by the database search engine.

Improved indexes

Indexes crucially improve read performance of MongoDB. For this case, indexes should have perfect coverage of all queries that are going to be used. Therefore, for good performance with the MongoDB back end, it is necessary to have the understanding of the data variety that will be collected from the environment and to have a set of main queries that will be used. The best practice is to create a compound index with the most distributed “prefix” field. In such case, the number of indexes and documents that will be used in the search process are decreased.

For example, consider an environment with 100 users, 20 projects, and 1000 resources. The most distributed field in meter collection is `resource_id`. And for the query with all three fields, the most performant index is based on the list `[[resource_id: 1, user_id: 1, project_id: 1, timestamp: -1]]`. The timestamp is on the last position because if there is a range query for any field, that field should be done on the last search stage when there is the smallest amount of found documents.

Horizontal scaling

Sharding in MongoDB is the only way to get horizontal scaling that can highly increase performance of using the MongoDB as a back end for Ceilometer. For more information about Sharding, refer to its official documentation.

There are several reasons to decide that sharding is needed:

- The server has not enough disk space to store all required data
- RAM on the server is less than the database’s working set plus indexes
- IOPs on the server disk is less than required
- Occasionally, CPU and Network can be a reason to choose sharding

If you choose sharding, pay attention that performance increase critically depends on choosing shard keys. A shard key must meet the following requirements:

- Have enough cardinality
- Guarantee write distribution
- Guarantee query isolation

The Ceilometer project provides statistic meters analysis. Therefore, a good shard key for meter collection should be compound and should consist of counter_name and a timestamp. Such keys provide all three requirements.

**Time to Live (TTL)**

Time to Live (TTL) is the native MongoDB feature. It is implemented as an index for a field with type datetime that makes it possible to remove data with expired datetime value. Consider the following:

- An active TTL index does not guarantee immediate deletion of the expired data. It happens because the deleting task runs every 60 seconds and has lower priority comparing with other operations.
- On Replica Set, TTL deletes documents on primary node first, and secondary nodes duplicate deletion operations from the primary node. This means that if read preferences allow making read operation from secondary nodes, some data can be deleted with additional delay.

For more information about TTL in MongoDB, see [TTL Indexes](#).

Ceilometer can configure a TTL index for meter and event collections. The meter collection TTL spreads on resource collection. That is, a resource will be deleted if no samples were received from it during the TTL.

You can provide appropriate TTL parameters to meter collection and event collection by setting the `metering_time_to_live` and `event_time_to_live` parameters in the `[database]` group of the `ceilometer.conf` file.

**Advantages and disadvantages of using TTL**

Consider the advantages and disadvantages of using the TTL feature.

**Advantages**
• Helps to avoid the disk space overload and the necessity to manually remove the expired data.
• The hardware resources requirements for the database do not increase with the cloud lifetime.

Disadvantages
• By using the TTL index, the expired data will be deleted and cannot be used if needed. Note that you can make a backup of collections or databases even if the data is not needed. Backup files can easily be deleted.
• The deletion task is a periodic process, and the expired data will not be deleted immediately.
• A low priority of TTL deletion process can lead to a long delay of the deletion task in case of having high I/O load by MongoDB database.

Recommendations

Retention period
By default, the metering_time_to_live and event_time_to_live parameters are set to 604800 seconds, which is seven days. Change this to an appropriate value depending on the data collection purpose.

For the purpose of autoscaling, when the data is being gathered for the health check of the cloud to scale it in the correct direction, the retention period of seven days is enough.

Another case is billing, when the collected data is used for creating reports of used resources. For this purpose, the TTL should be set to a value not less than the report period.

Data usage and backups
The base rule to achieve high performance of MongoDB is to hold only the frequently used data in MongoDB. All other data that can be required in some special cases should be archived and stored separately.
For example, consider the billing case. Billing reports are created every month. After creating a report, it is highly recommended making a backup for the expired period to have an opportunity to restore the data if needed. The TTL can be set for a report period (for example, one month) plus 5-10% of that period to cover unexpected complications or fails. For more information about backups, see the Backup section.
Log rotation

Possibilities of log rotation in MongoDB

In MongoDB, logging can be configured in several ways. MongoDB can send log information to a standard output, to a log file, and to the host's syslog system. In Mirantis OpenStack, syslog system is used (available since MongoDB 2.2). The log option can be configured in the MongoDB configuration file.

In the configuration file, choose a method for logging:

- In the `systemLog` section, the `destination` parameter (`<string>`) chooses a way for logging. In Mirantis OpenStack, it is set to `syslog` by default.
- For `syslog`, the `destination` parameter `path` should not be set. Otherwise, it will cause an error.
- The `verbosity` parameter (`<int>`) is set to 0 by default. It can be increased up to 5 to raise the amount of reported information.

Log rotation is provided for logging to a file either to syslog. For details, see [Rotate Log Files](#). It is recommended that you use system syslog for log rotation in MongoDB.

Archive, back up, and restore data

Options

To make a backup of MongoDB data, you can use several different strategies. This section briefly describes these strategies. For more details, refer to [MongoDB documentation](#).

Back up by copying data files

This method is not specific to MongoDB. Files with data can be copied by making a snapshot of the files where MongoDB stores its data if the volume with MongoDB supports making snapshots.

Backup software

This method requires using one of the following tools:

- [MongoDB Cloud Manager](#)
- [MongoDB Ops Manager](#)

Back up a database with mongodump

The `mongodump` is a tool for MongoDB backup. It can be used to back up a certain database or even a certain collection. To back up all databases, you need to have a back-up role for a MongoDB user. It is recommended that you use `mongodump` to back up data for a period that is used for report creation and remove older data with the TTL feature. While dumping, MongoDB

© 2005–2017 All Rights Reserved

www.mirantis.com
creates two files, BSON and JSON, for each collection that is being backed up. The first file contains the database data, and the second one contains the collection metadata. For more information, see Backup a database with mongodump.

Special aspects of backup/restore processes

All indexes from collection are being backed up with the data. That is, all TTL indexes are stored in metadata in JSON files. And if TTL indexes are restored with the data, the TTL feature is applied immediately for all the expired data, and that can be all the restored data.

Note: To avoid deleting data from restoring collections, remove the TTL index from collection metadata in the appropriate JSON file before running mongorestore.

Restore the backup data

To restore the back-up data, perform the following steps:

1. Deploy a new single MongoDB server to work with the restored data.
2. Check metadata for all collections that are going to be restored and remove all TTL indexes if any exist.
3. Run mongorestore for the data that needs to be checked on new MongoDB server.

Upgrade MongoDB

Upgrading MongoDB 2.4.x to 2.6.x provides a number of performance improvements. The major changes are as follows:

- Aggregation enhancements to improve performance of Ceilometer statistic requests
- Text search integration
- Insert and update improvements
- A new write operation protocol

For detailed information, see Major Changes.

MongoDB 2.4.x to 2.6.x upgrade steps

For a detailed instruction on upgrading MongoDB from 2.4 to 2.6, see Upgrade MongoDB to 2.6.
In case of upgrading MongoDB to 2.6 as the Ceilometer back end, see Upgrade a Replica Set to 2.6.
Note: The upgrade can be performed in “cycling” style (one by one for secondary nodes and after that for the primary one). Such approach will minimize the downtime by upgrading one member while others are available.

For the description of the update procedure for every mongod process, see Upgrade Procedure.

Important points

Consider the following important points:

- Use \texttt{rs.stepDown} in mongo shell to make primary node become secondary instead of directly shutting it down.
- Improve the mongo configuration file for YAML format that is used since MongoDB 2.6 to simplify database maintenance.

Troubleshooting

This section describes the most frequent issues occurring with MongoDB as a back end for Ceilometer. It also includes troubleshooting steps.

Most frequent issues

Low performance of getting data from database

This issue occurs on a large amount of data stored in MongoDB. The following can cause the issue:

- The query is too general, not detailed and without indexed fields.
  The solution is to use only detailed query with indexed fields.
  \textit{Example of a bad query: “Give me all samples and count <source>s”}.
  \textit{Example of a good query: “Give me all samples with timestamp greater than <start_date> and less than <end_date> and count <source>s”}.
  It is even better to mention more parameters. For example, \texttt{resource_id}, \texttt{user_id}, \texttt{project_id}, and so on.

- Small RAM on MongoDB node, when indexes do not fit RAM.
  This can happen if not enough attention is paid to the database size and RAM estimations. As mentioned above, MongoDB requires taking all indexes and working set in RAM. This issue happens more often when MongoDB is deployed on controller nodes.
  You need to find out the Ceilometer database indexes size and estimate the working set size. The MongoDB working set size must fit the node RAM. If you have MongoDB
on controller nodes, the estimated memory requirements should fit the free RAM that is left after all other services have met the demand.

Failure to rechoose a new primary node

Network issues can cause this issue. It can happen when the replica set nodes cannot connect one to another. Therefore, this issue is not related to MongoDB.

Failure to automatically reconnect to the MongoDB server

This issue, as the previous one, is related to network issues and cannot be solved from the MongoDB side.

Troubleshooting steps

Discover the issue

This section provides recommendations according to the Low performance getting data from database issue.

1. Take a look on atop to clarify what is going on with memory on mongo nodes. As mentioned above, the index size and working set should fit the available memory on mongo nodes. To estimate the indexes size, use command in mongo shell. For a detailed description of the command output, see dbStats.

```
> db.stats()
{
   "db" : "ceilometer",
   "collections" : 9,
   "objects" : 30671815,
   "avgObjSize" : 2022.8199832321627,
   "dataSize" : 62043560304,
   "storageSize" : 63391002224,
   "numExtents" : 91,
   "indexes" : 13,
   "indexSize" : 6961447024,
   "fileSize" : 83644907520,
   "nsSizeMB" : 16,
   "extentFreeList" : {
      "num" : 19,
      "totalSize" : 9630208000
   },
   "dataFileVersion" : {
      "major" : 4,
      "minor" : 21
   },
   "ok" : 1
}
```

Pay attention to the indexSize parameter. And take a look at pagesInMemory or overSeconds parameter in the workingSet section of
db.serverStatus({workingSet: 1}) (the workingSet output was removed since MongoDB 3.x).

Other output parameters can also be useful. For a complete description, see Server Status Output.

Each page should be multiplied by around 4 KB, which gives the size of workingSet held in RAM. And the overSeconds parameter shows how long the pages are being kept in memory. If this parameter is small, the data in memory is constantly read from disk. Usually, both parameters increase or decrease simultaneously.

Additional RAM is required for all open connections, each of them requires 1 MB of memory. Open connections can also be seen from db.serverStatus() output in the connections section current field.

2. To check network issues, use the rs.status() shell command that shows which nodes are in replica set and the status of each node. For a detailed output description, see replSetGetStatus.

3. Occasionally, while performing a health check operation in MongoDB, a very slow query can be made without receiving any response on it from Ceilometer API. It does not mean that MongoDB stopped operating request, but it means that API timeout is exceeded. By default, each long operation continues up to 10 minutes. Therefore, if in this case you make a new query, you will get a timeout exception.

To stop a slow operation, find out its opid using the db.currentOp() command.

The secs_running field shows the duration of operation execution (6 seconds in the example above), query shows that it is an aggregation operation, opid shows the
operation ID which is required to stop this operation. To stop an operation, use the shell
\texttt{db.killOp(<opid>)} command.

4. Check if the TTL feature works properly.
   A. Check collection indexes using \texttt{db.<collection>.getIndexes()} and verify
      that the index based on date has the \texttt{expireAfterSeconds} parameter.
   B. Verify that the documents are being deleted using the \texttt{db.serverStatus()}
      shell command.
   C. In the output, see the “\texttt{deletedDocuments}” parameter in the “\texttt{ttl}” section.
      This parameter shows the number of documents that were deleted.

**Perform maintenance on replica set members**

MongoDB allows performing manipulations on the secondary node, for example, create or drop
indexes, while other part of replica set is working as usual. This procedure is useful when some
changes should be done on replica set, but the database must not be blocked for write/read
operations.

**Note:** Perform the procedure on secondary nodes first. Prior to performing the procedure on the
primary node, make the primary node become secondary using \texttt{rs.stepDown}.

To perform maintenance on replica set members:
   1. Restart the mongod instance as a standalone.
   2. Perform an appropriate task on the standalone instance.
   3. Restart the standalone instance as a member of replica set.

For more information, see [Perform maintenance on replica set members](#).

**Monitoring tools**

MongoDB provides several monitoring tools that can be very helpful for mongo health checking.

The \texttt{mongostat} utility provides the status of running \texttt{mongod, mongos} processes.

**Output example:**

```
Insert query update delete getmore command flushes mapped vizel res faults qr|ar|rw netIn netOut conn time
|6|0|0|0|0|0|0|78.10|156.80|10.46|6|0|0|0|0|79b|10k|4|18:58:33
|6|0|0|0|0|0|0|78.10|156.80|10.46|6|0|0|0|0|79b|10k|4|18:58:32
|6|0|0|0|0|0|0|78.10|156.80|10.46|6|0|0|0|0|79b|10k|4|18:58:33
|6|0|0|0|0|0|0|78.10|156.80|10.46|6|0|0|0|0|79b|10k|4|18:58:34
```

This utility provides the number of CRUD operations, number of evaluating commands,
database mapped, virtualized, reserved sizes, current active connections, and so on. For a
detailed description, see [Mongostat](#).

The \texttt{mongotop} utility provides the amount of time MongoDB spends to read and write data.
For more information, see [Mongotop](#).

**Output example:**
As you can see, the meter collection is loaded with reading and for the timestamp 2016-02-02T19:04:44+02:00 it took 1381 ms.