Monitoring Mongo Database Server

eG Innovations Product Documentation

www.eginnovations.com

Total Performance Visibility

Table of Contents

Table of Figures

Chapter 1: Introduction

MongoDB is an open source document store NoSQL DBMS designed to make it easier for organizations to develop and run applications that address performance, availability and scalability, and support a variety of data types. MongoDB's document data model lets developers easily store and combine data of any structure, without sacrificing data access or indexing functionality. This enables database administrators to dynamically modify the schema with no downtime.

Because of its ease of use, MongoDB is used as the backend for many mission-critical applications. Since the stability of these applications hinges on the availability and the operational efficiency of MongoDB, administrators need to keep a close watch on the health of the MongoDB server.

eG Enterprise provides a specialized monitoring model for a MongoDB server instance, with the help of which administrators can keep a constant check on the availability and overall performance of that instance.

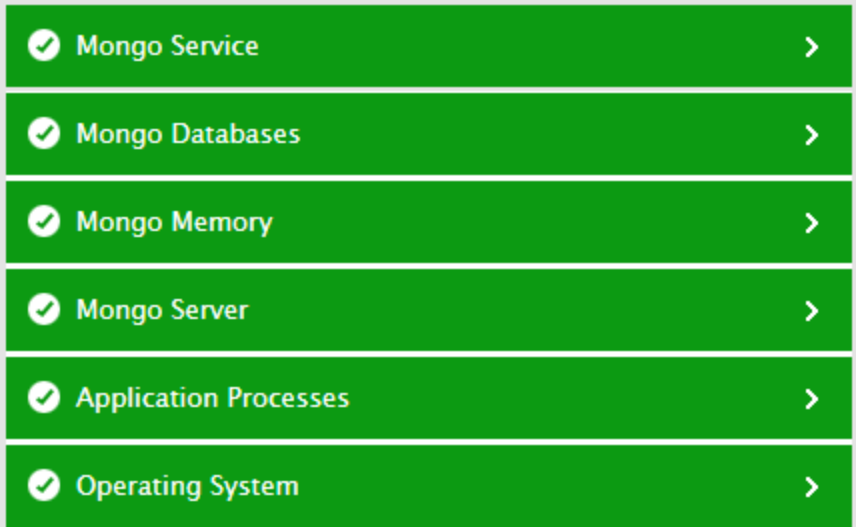

Figure 1.1: The layer model of a MongoDB server instance

Each layer of Figure 1.1 above is mapped to tests that provide deep insights into how the MongoDB server instance manages connections to it, the space at its disposal, locks, indexes, transactions, user sessions, and in the process provides effective pointers to how the server can be fine-tuned to perform better.

The metrics reported by these tests help answer the following performance questions:

- Is the MongoDB server up and running? If so, how well is it responding to requests?
- Is connection to the server available? Is there a delay in connecting to the server?
- Are there adequate free connections to the server?
- How frequently do assertions occur? What type of assertions are common?
- Are writes been flushed to disk quickly?
- Is the cache sized right?
- Are all databases available? Which database is presently unavailable?
- Is any database locked for an unusually long time? If so, which collection(s) in that database was for the maximum time?
- Is any database slow in processing queries? If so, which one is it?
- Are there too many open cursors on the server?
- Is any database running short of database space? If so, which one is it?
- Is any database growing alarmingly in size? Which database is it? Which collection in this database is is contributing to this growth?
- Is the compression ratio of any database poor?
- Are too many requests waiting in queue for a lock?
- Are requests waiting too long for locks? What type of locks do they often wait for?
- Are many queries not been serviced by indexes?
- Is journaling taking too long than usual? If so, where does the bottleneck lie - when writing to the journal? when committing the writes? or when writing to the datafiles?
- Were many errors and warnings captured by the log files recently? What type of errors or warnings are these?
- Is the MongoDB server instance utilizing memory excessively?
- Were many page faults detected?
- Are requests to the server consuming bandwidth excessively?
- Has a primary node in a replica set failed over to the secondary node?
- Is any member node in a replica set not running currently?
- Is the replication time lag abnormally high between the primary and secondary?

- Is the server overloaded with requests? What type of operations are overloading the server?
- Are tickets used optimally or is the server running out of free tickets?
- Are transaction rollbacks kept at a minimum?
- Are checkpoints created regularly?

The chapters that follow elaborately discusses the tests, metrics, and how they are collected.

Chapter 2: How Does eG Enterprise Monitor MongoDB?

eG Enterprise can monitor MongoDB in an agent-based or an agentless manner. In case of the agentless approach, the remote agent used to monitor the MongoDB server should be deployed on a remote Windows host in the environment.

Regardless of the approach (agent-based or agentless), the eG agent connects to a database on the MongoDB server instance and runs built-in MongoDB API commands to pull the metrics of interest from across databases. To enable this connection, each test that the eG agent runs on the MongoDB server instance should be configured with the name of a MongoDB database.

In highly secure environments, access control may be enabled on the target MongoDB deployments. Such deployments enforces authentication, which requires users to identify themselves while accessing the MongoDB. eG Enterprise can monitor the MongoDB databases that are access control enabled too. For this administrators need to do the following:

- create a custom role
- assign a user to that role
- Finally, enable access control i.e., authenticate the user.

2.1 How to monitor access control enabled MongoDB database?

To monitor the access control enabled MongoDB database, it is essential to create a custom role in the admin database of the MongoDB instance and a user.

- To create a custom role for e.g., *egmonrole*, issue the following command on the MongoDB instance:

```
db.createRole({role: "egmonRole",privileges: [{ resource: { anyResource: true},
actions: [ "serverStatus" ,"dbStats","top", "listDatabases", "getCmdLineOpts",
"collStats", "hostInfo", "listCollections", "find", "replSetGetConfig",
"replSetGetStatus"] }],roles: [] })
```

- Once the role is created, you can create a user for e.g., *eguser* and assign the newly created role i.e., *egmonrole* to that user by issuing the following command:

```
db.createUser({ user: "eguser", pwd: "egurkha", roles: [ "egmonRole" ]} )
```

Once you have created the user, you need to specify the credentials of the user while configuring the tests for the target MongoDB instance being monitored.

Chapter 3: How to Monitor MongoDB Using eG Enterprise?

The broad steps for monitoring MongoDB using eG Enterprise are as follows:

- Managing the MongoDB server
- Configuring the tests

These steps have been discussed in this topic.

3.1 Managing the MongoDB Server

The MongoDB server cannot be automatically discovered by eG Enterprise. This implies that you will have to manually add the server into the eG Enterprise system to manage it. Follow the steps below to achieve the same:

1. Follow the Components -> Add/Modify menu sequence in the **Infrastructure** tile of the **Admin** menu.
2. Next, select *Mongo Database* from the **Component type** drop-down and then click the **Add New Component** button.
3. When Figure 3.1 appears, provide the **Host IP/Name** of the MongoDB server that you want to manage.

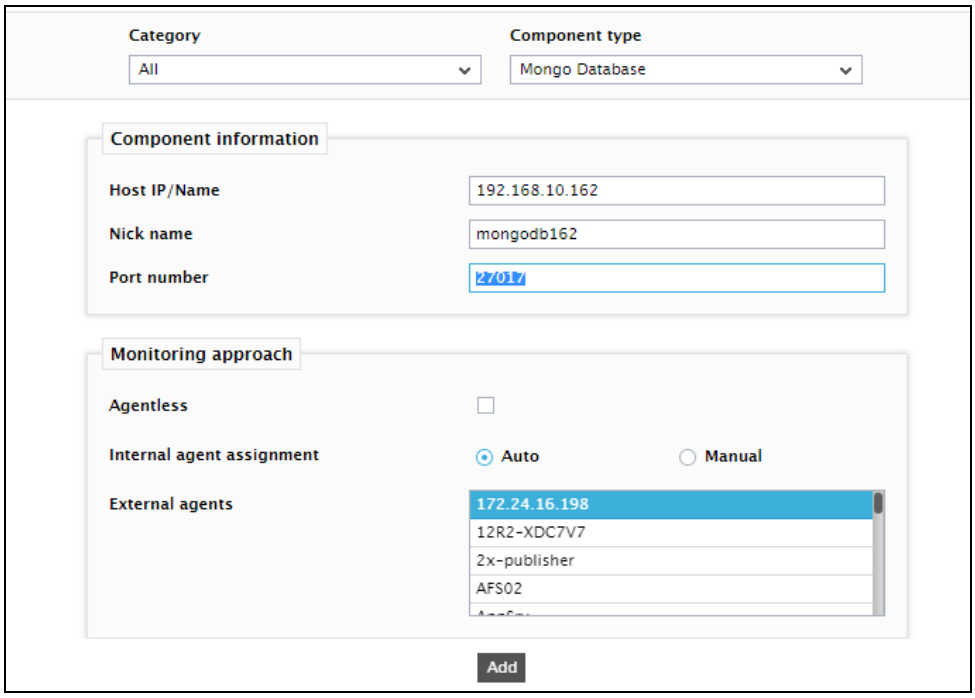

Figure 3.1: Managing a MongoDB server in an agent-based manner

4. Then, provide a **Nick name** for the server.
5. The **Port number** will be set as 27017 by default. If the MongoDB server is listening on a different port in your environment, then override this default setting.
6. In case you are monitoring a MongoDB server in an agent-based manner, just pick an external agent from the **External agents** list box and click the **Add** button to add the component for monitoring.
7. On the other hand, if you are monitoring a MongoDB server in an agentless manner, then do the following:
   - Select the **Agentless** check box.
   - Pick the **OS** on which the MongoDB server is running.
   - Set the **Mode** to **Other**.
   - Select the **Remote agent** that will be monitoring the MongoDB server. **Note that the Remote agent you choose should run on a Windows host.**
   - Choose an external agent for the server by picking an option from the **External agents** list box.
   - Finally, click the **Add** button to add the MongoDB server for monitoring.

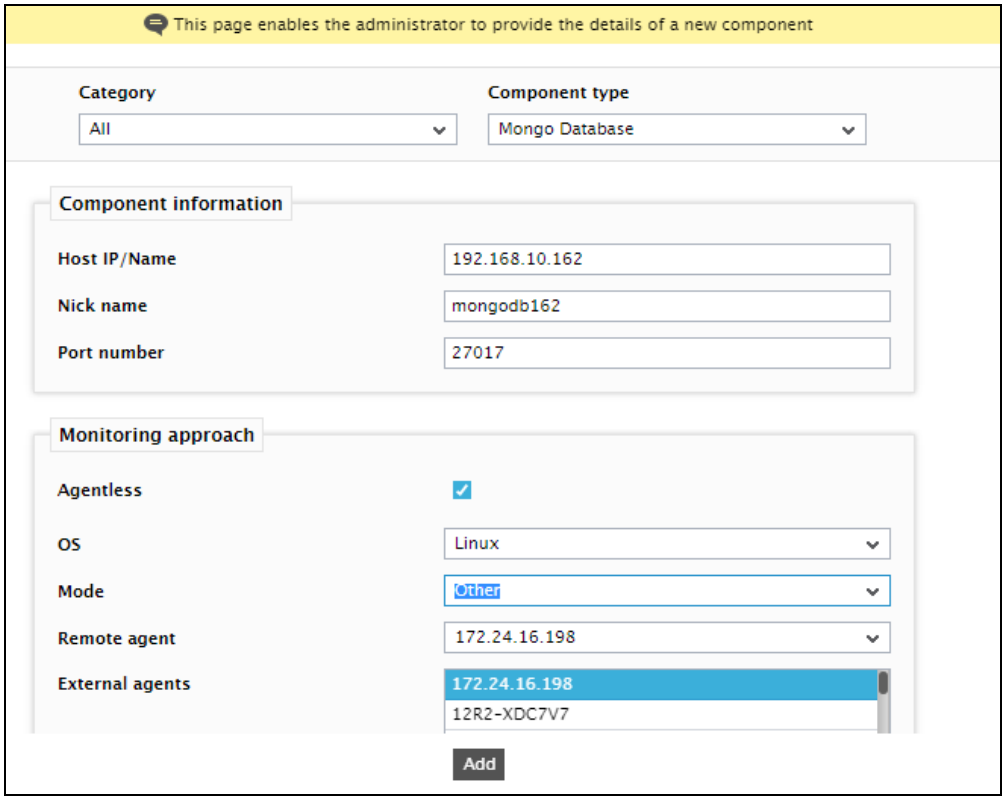

Figure 3.2: Managing a MongoDB server in an agentless manner

8. Finally, click the **Signout** button at the right, top corner of the eG admin interface to sign out.

3.2 Configuring Tests

When you try to sign out of the eG admin interface, a **LIST OF UNCONFIGURED TESTS** page will appear, revealing the list of tests mapped to the MongoDB server that require manual configuration:

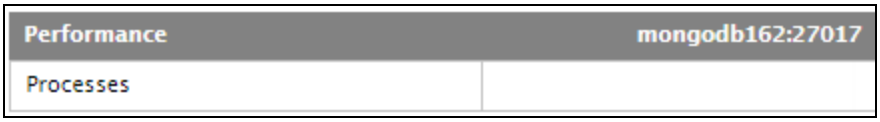

Figure 3.3: The Processes test requiring manual configuration

Figure 1 indicates that the Processes test mapped to the MongoDB server needs to be configured manually. To know how to configure the **Processes** test, refer to the *Monitoring Unix and Windows Servers* document.

After configuring the Processes test, sign out of the eG administrative interface. Then, login to the eG monitoring console to view the state of and metrics reported by the specialized monitoring model that eG Enterprise offers for the MongoDB server.

Chapter 4: Monitoring the MongoDB Server Using eG Enterprise

This chapter takes you inside the top 4 layers of the MongoDB server monitoring model, and discusses the tests and measures mapped to each layer.

4.1 The Mongo Server Layer

This layer monitors critical MongoDB server parameters such as its uptime, the network traffic to and from the server, the overall load on the server, and how well it handles the load. Additionally, the layer also monitors the server's index usage, its replication throughput, and its alert logs.

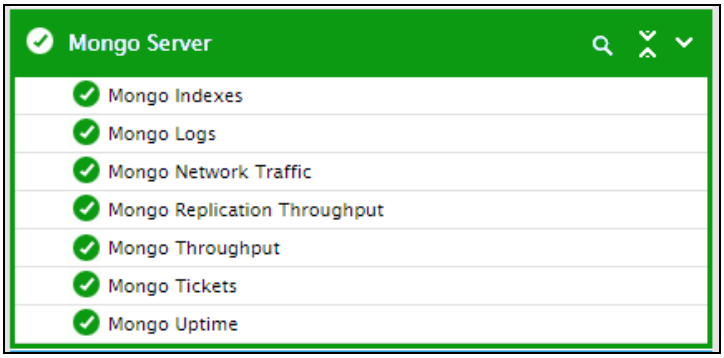

Figure 4.1: The tests mapped to the Mongo Server layer

4.1.1 Mongo Indexes Test

Indexes support the efficient execution of queries in MongoDB. If an appropriate index exists for a query, MongoDB can use the index to limit the number of documents it must inspect. Without indexes, MongoDB must perform a collection scan, i.e. scan every document in a collection, to select those documents that match the query statement. Collection scans are time-consuming, resource-intensive operations, which if not contained, can degrade performance of the database server and the dependent applications. To avoid these scans, each query to the Mongo server should be indexed. This is where the **Mongo Indexes** test helps.

This test monitors the queries to a target Mongo database server and reports the count and percentage of queries that are not serviced by indexes - i.e., the count and ratio of queries that are serviced by collection scans. This way, the test brings index-related inadequacies to the surface, and

prompts administrators to improve query and over- all database performance by taking the necessary corrective action.

Note:

This test runs only on Mongo database servers below v3.0 only.

Target of the test : A MongoDB server

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for the Mongo database server being monitored.

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed.
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens.
Database Name	The test connects to a specific Mongo database to run API commands and pull metrics of interest. Specify the name of this database here. The default value of this parameter is *admin*.
Username and Password	The eG agent has to be configured with the credentials of a user who has the required privileges to monitor the target MongoDB instance, if the MongoDB instance is access control enabled. To know how to create such a user, refer to Section **2.1** . If the target MongoDB instance is not access control enabled, then, specify *none* against the Username and Password parameters.
Confirm Password	Confirm the password by retyping it here.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Accesses	Indicates the rate at which queries/operations accessed the indexes.	Accesses/Sec	This is a good indicator of the workload of the database server.
Hits	Indicates the rate at which	Hits/Sec	A high value is desired for this

Measurement	Description	Measurement Unit	Interpretation
	queries ran using indexes in memory.		measure.
Misses	Indicates the rate at which queries attempted to access an index that was not in memory.	Misses/Sec	A low value is desired for this measure.
Resets	Indicates the rate at which the index counter was reset.	Resets/Sec	
Miss ratio	Indicates the percentage of time queries attempted to access indexes that are not in memory.	Percent	If the value of this measure is over 50%, then you need to look at your queries to see if they are making optimal use of the indexes you have created. You should consider adding new indexes and seeing if your queries run faster as a result.

4.1.2 Mongo Logs Test

The MongoDB logs are a vast source of information related to errors and warnings that a MongoDB server encounters. Administrators use these log files not only to spot problem conditions, but also to troubleshoot them. By periodically scanning the MongoDB error log for errors/warnings, the **Mongo Logs** test promptly notifies administrators as soon as a new error, warning, or fatal error is logged in the file. Detailed diagnostics of the test clearly indicate what error it is and when it occurred, so as to aid troubleshooting efforts.

As messages keep getting logged in the log files, these log files grow large in size. If log file growth is left unchecked, it can consume all available space in the database. Administrators can effectively tracklog file growth and initiate measures to control it using the **Mongo Logs** test. The test reports the current size of the log files and their growth rate, and proactively alerts administrators if the rate of growth is abnormal.

Target of the test : A MongoDB server

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for the MongoDB server monitored.

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed.
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens.
Database Name	The test connects to a specific Mongo database to run API commands and pull metrics of interest. Specify the name of this database here. The default value of this parameter is *admin*.
Username and Password	The eG agent has to be configured with the credentials of a user who has the required privileges to monitor the target MongoDB instance, if the MongoDB instance is access control enabled. To know how to create such a user, refer to Section **2.1** . If the target MongoDB instance is not access control enabled, then, specify *none* against the Username and Password parameters.
Confirm Password	Confirm the password by retyping it here.
Detailed Diagnosis	To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the **On** option. To disable the capability, click on the **Off** option. The option to selectively enable/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled: • The eG manager license should allow the detailed diagnosis capability • Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Errors	Indicates the number of error messages logged in the log files during the last	Number	Ideally, the value of this measure should be 0.

Measurement	Description	Measurement Unit	Interpretation
	measurement period.		If a non-zero value is reported, you can use the detailed diagnosis of this measure to know what error occurred and when.
Fatal errors	Indicates the number of fatal errors captured by the log files during the last measurement period.	Number	Ideally, the value of this measure should be 0. If a non-zero value is reported, you can use the detailed diagnosis of this measure to know what error occurred and when.
Warning messages	Indicates the number of warning messages logged in the log files during the last measurement period.	Number	Ideally, the value of this measure should be 0. If a non-zero value is reported, you can use the detailed diagnosis of this measure to know what error occurred and when.
File size	Indicates the current size of the log file.	MB	
Growth rate	Indicates the rate at which the log file is growing.	MB/Sec	A high value for this measure or a consistent increase in its value indicates that the Mongo log is rapidly growing and may end up occupying too much space on the volume.

4.1.3 Mongo Network Traffic Test

The workload of a MongoDB server is best revealed by the count of requests to it. The impact of this workload on the network throughput of a Mongo server must be assessed to understand whether/not enough network resources are available to handle the current and future workload. This is exactly what the **Mongo Network Traffic** test does! This test monitors the requests to a Mongo database server and reports the rate at which data is received and transmitted over the network by that server. In the process, the test indicates whether the server is rightly sized in terms of network resources or not.

Target of the test : A MongoDB server

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for the database server being monitored.

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed.
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens.
Database Name	The test connects to a specific Mongo database to run API commands and pull metrics of interest. Specify the name of this database here. The default value of this parameter is admin.
Username and Password	The eG agent has to be configured with the credentials of a user who has the required privileges to monitor the target MongoDB instance, if the MongoDB instance is access control enabled. To know how to create such a user, refer to Section **2.1**. If the target MongoDB instance is not access control enabled, then, specify *none* against the Username and Password parameters.
Confirm Password	Confirm the password by retyping it here.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Requests	Indicates the rate at which requests are processed by this server.	Requests/Sec	
Data transmitted	Indicates the rate at which data is transmitted by the server.	MB/Sec	A high value is indicative of high bandwidth usage by the Mongo DB server.
Data received	Indicates the rate at which data is received by the server.	MB/Sec	

4.1.4 Mongo Replication Throughput Test

Replication provides redundancy and increases data availability. With multiple copies of data on different database servers, replication provides a level of fault tolerance against the loss of a single database server.

A replica set is a group of mongod instances that maintain the same data set. A replica set contains several data bearing nodes and optionally one arbiter node. Of the data bearing nodes, one and only one member is deemed the primary node, while the other nodes are deemed secondary nodes. The primary node receives all write operations. A replica set can have only one primary capable of confirming writes with { w: "majority" } write concern. The primary records all changes to its data sets in its operation log, i.e. oplog. The oplog is a limited-size collection stored on primary nodes that keeps track of all the write operations. Secondary members replicate this log and apply the operations to their data sets.

If the secondary is unable to apply the changes as fast as they are written to the primary's oplog, then changes will be lost if the primary crashes. Similarly, if the oplog is not sized right, then it will not be able to hold many changes, thus causing significant data loss in the event of a primary failure. This is why, administrators should constantly measure the level of activity on a replica set, check whether the oplog is sized according to this workload, and also ensure that there is little-to-no time lag in data replication between the primary and secondaries of the replica set. This can be achieved using the **Mongo Replication Throughput** test.

This test tracks the operations performed on the replica set and in the process, reveals the load on the replica set. The test also tracks the usage of the oplog and alerts administrators if the oplog is about to run out of space for recording changes. Additionally, the test also keeps an eye out for long gaps between when a change is recorded in the primary's oplog and when it is actually applied on the secondary , and promptly notifies administrators of the same. This way, the test brings inconsistencies in data replication to the immediate attention of administrators and averts the data loss that might occur if the primary crashes.

Target of the test : A MongoDB server

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for the Mongo database server being monitored.

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed.
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens
Database Name	The test connects to a specific Mongo database to run API commands and pull metrics of interest. Specify the name of this database here. The default

Parameter	Description
	value of this parameter is *admin*.
Username and Password	The eG agent has to be configured with the credentials of a user who has the required privileges to monitor the target MongoDB instance, if the MongoDB instance is access control enabled. To know how to create such a user, refer to Section **2.1** . If the target MongoDB instance is not access control enabled, then, specify *none* against the Username and Password parameters.
Confirm Password	Confirm the password by retyping it here.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Insert operations	Indicates the rate at which replicated insert operations were performed on the target server.	Inserts/Sec	A consistent increase in the value of these measures could indicate a high level of activity on the replica set.
Query operations	Indicates the rate at which replicated query operations are performed on the target server.	Queries/Sec	
Update operations	Indicates the rate at which replicated update operations are performed on the target server.	Updates/Sec	
Delete operations	Indicates the rate at which replicated delete operations are performed on the target server.	Deletes/Sec	
Get more operations	Indicates the rate at which replicated get more operations are performed on the target server.	Getmores/Sec	The value of this measure can be high even if the query count is low. Secondary nodes send getMore operations as part of the replication process.
Command	Indicates the rate at which	Commands/Sec	

Measurement	Description	Measurement Unit	Interpretation
operations	replicated commands are issued to the target server.		
Replication lag	Indicates how far a secondary is behind a primary.	Secs	Ideally, the value of this measure should be 0. If it is very high, then the integrity of your data set might be compromised in case of failover (secondary member taking over as the new primary because the current primary is unavailable). A high value also implies that write operations are not immediately propagated to secondaries; in this case, related changes might be lost if the primary fails. A high replication lag can be due to: • Networking issue between the primary and secondary making nodes unreachable • A secondary node applying data slower than the primary • Insufficient write capacity in which case you should add more shards; • Slow operations on the primary node blocking replication; • Heavy write operations on the primary node or an under-provisioned secondary. You can prevent the latter by scaling up the secondary to match the

Measurement	Description	Measurement Unit	Interpretation
			primary capacity.
Oplog window	Indicates the interval of time between the oldest and the latest entries in the oplog.	Secs	If a secondary is down longer than this oplog window, it will be able to catch up unless it completely resyncs all data from the primary. The amount of time it takes to fill the oplog varies: during heavy traffic times, it will shrink since the oplog will receive more operations per second. If the oplog window for a primary node is getting too short you should consider increasing the size of your oplog.
Replication head room	Indicates the time difference between the primary's oplog window and the replication lag of the secondary .	Secs	If the replication headroom is rapidly shrinking and is about to become negative, that means that the replication lag is getting higher than the oplog window. In that case, write operations recorded in the oplog will be overwritten before secondary nodes have time to replicate them. MongoDB will constantly have to resync the entire data set on this secondary which takes much longer than just fetching new changes from the oplog. Properly monitoring and alerting on *Replication lag* and *Oplog window* should allow you to prevent this.
Total oplog size	Indicates the amount of space allocated to the oplog.	MB	
Used oplog size	Indicates the amount of space currently used by operations stored in the oplog.	MB	If this value grows closer to the Total oplog size, it is a cause for concern. This is because it implies that the oplog may soon not have enough space to record any more changes. To avoid this, you may want to consider

Measurement	Description	Measurement Unit	Interpretation
			resizing your oplog. Before that, check the level of activity on the replica set and figure out what would be the ideal size setting for the oplog, so that its able to capture all the changes that occur on the replica set. Before mongod creates an oplog, you can specify its size with the **oplogSizeMB** option. If you can predict your replica set's workload to resemble one of the following patterns, then you might want to create an oplog that is larger than the default. Conversely, if your application predominantly performs reads with a minimal amount of write operations, a smaller oplog may be sufficient. The following workloads might require a larger oplog size. • The oplog must translate multi-updates into individual operations in order to maintain idempotency. This can use a great deal of oplog space without a corresponding increase in data size or disk use. • If you delete roughly the same amount of data as you insert, the database will not grow significantly in disk use, but the size of the operation log can be quite large.

4.1.5 Mongo Throughput Test

Periodically, administrators should measure the workload on the Mongo database server and evaluate the server's ability to handle the load, so that they can figure out whether the server's sized

commensurate to its load or not. Administrators can perform this load analysis using the **Mongo Throughput** test.

This test tracks the load on the Mongo database server and reports what type of operations are contributing to the workload of the server. The test additionally reports the number of clients performing read and write operations on the server, so that you can figure out the count of clients that are generating the load on the MongoDB server. Capacity planning and clustering decisions can be taken based on insights provided by this test.

Target of the test : A MongoDB server

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for the Mongo database server being monitored.

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens.
Database Name	The test connects to a specific Mongo database to run API commands and pull metrics of interest. Specify the name of this database here. The default value of this parameter is *admin*.
Username and Password	The eG agent has to be configured with the credentials of a user who has the required privileges to monitor the target MongoDB instance, if the MongoDB instance is access control enabled. To know how to create such a user, refer to Section **2.1** . If the target MongoDB instance is not access control enabled, then, specify *none* against the Username and Password parameters.
Confirm Password	Confirm the password by retyping it here.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Insert operations	Indicates the rate at which insert operations are	Inserts/Sec	

Measurement	Description	Measurement Unit	Interpretation
	performed on the target server.		
Query operations	Indicates the rate at which query operations are performed on the target server.	Queries/Sec	
Update operations	Indicates the rate at which update operations are performed on the target server.	Updates/Sec	
Delete operations	Indicates the rate at which delete operations are performed on the target server.	Deletes/Sec	
Get more operations	Indicates the rate at which get more operations are performed on the target server.	Getmores/Sec	
Command operations	Indicates the rate at which command operations are performed on the target server.	Commands/Sec	
Read requests	Indicates the rate at which read requests are received by the target server.	Requests/Sec	A consistent increase in the value of these measures could indicate a potential overload.
Write requests	Indicates the rate at which write requests are received by the target server.	Requests/Sec	
Read operation queued clients	Indicates the number of the active client connections performing read operations.	Number	The value of these measures indicate how many clients are generating the load on the MongoDB server.
Write operation queued clients	Indicates the number of the active client	Number	

Measurement	Description	Measurement Unit	Interpretation
	connections performing write operations.		

4.1.6 Mongo Tickets Test

Tickets are an internal representation for thread management. WiredTiger uses tickets to control the number of read/write operations simultaneously processed by the storage engine. The default value is 128 and works well for most cases. This count can be fine-tuned depending upon the workload of MongoDB.

If sufficient tickets are not available for processing read/write requests, it can reduce concurrency and cause request processing to slow down on the MongoDB server, thus scarring user experience with the server. If this is to be avoided, then administrators should periodically check ticket usage and detect a ticket crunch well before it impacts database performance or UX. This can be achieved using the **Mongo Tickets** test. This test monitors how the MongoDB server uses the read and write tickets allotted to it and warns administrators of a probable contention for processing power. This way, the test enables administrators to optimize its ticket configuration, so that concurrency is not compromised and critical processing resources are also conserved.

Target of the test : A MongoDB server

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for the MongoDB server monitored.

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed.
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens.
Database Name	The test connects to a specific Mongo database to run API commands and pull metrics of interest. Specify the name of this database here. The default value of this parameter is *admin*.
Username and Password	The eG agent has to be configured with the credentials of a user who has the required privileges to monitor the target MongoDB instance, if the MongoDB instance is access control enabled. To know how to

Parameter	Description
	create such a user, refer to Section **2.1** . If the target MongoDB instance is not access control enabled, then, specify *none* against the Username and Password parameters.
Confirm Password	Confirm the password by retyping it here.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Total read tickets	Indicates the maximum number of read tickets the target server can use.	Number	
Used read tickets	Indicates the number of read tickets currently used.	Number	If the value of this measure is equal to or close to the value of *Total read tickets* measure, it is a cause for concern.
Available read tickets	Indicates the number of read tickets currently unused.	Number	If the value of this measure is equal to or close to 0, it is a cause for concern.
Read ticket usage	Indicates the percentage of read tickets currently in use.	Percent	A value close to 100% indicates that the server is about to run out of read tickets. This can cause subsequent read requests to be queued, waiting for tickets. Consequently, read request processing will slow down. One of the common causes for this is long-running read operations. Such operations typically hold on to read tickets without releasing them for long periods of time, thus reducing concurrency. Terminating such read operations can help free tickets. Alternatively, you can increase the maximum number of read tickets the server can use by modifying the *wiredTigerConcurrentReadTransactions* setting. However be careful when increasing it: if the number of simultaneous operations gets too

Measurement	Description	Measurement Unit	Interpretation
			high, you might run out of system resources (CPU in particular). Scaling horizontally by adding more shards can help to support high throughputs.
Total write tickets	Indicates the maximum number of write tickets the target server can use.	Number	
Used write tickets	Indicates the number of write tickets currently used.	Number	If the value of this measure is equal to or close to the value of *Total write tickets* measure, it is a cause for concern.
Available write tickets	Indicates the number of write tickets currently unused.	Number	If the value of this measure is equal to or close to 0, it is a cause for concern.
Write ticket usage	Indicates the percentage of write tickets currently in use.	Percent	A value close to 100% indicates that the server is about to run out of write tickets. This can cause subsequent write requests to be queued, waiting for tickets. Consequently, writerequest processing will slow down. One of the common causes for this is long-running write operations. Such operations typically hold on to write tickets without releasing them for long periods of time, thus reducing concurrency. Terminating such write operations can help free tickets. Alternatively, you can increase the maximum number of write tickets the server can use by modifying the *wiredTigerConcurrentWriteTransactions* setting. However be careful when increasing it: if the number of simultaneous operations gets too high, you might run out of system resources (CPU in particular). Scaling horizontally by adding more shards can help to support high throughputs.

4.1.7 Mongo Uptime Test

In most production environments, it is essential to monitor the uptime of critical database instances in the infrastructure. By tracking the uptime of each of the database instances, administrators can determine what percentage of time a database instance has been up. Comparing this value with service level targets, administrators can determine the most trouble- prone areas of the infrastructure.

In some environments, administrators may schedule periodic reboots of their database instance. By knowing that a specific database instance has been up for an unusually long time, an administrator may come to know that the scheduled reboot task is not working on a database instance.

This **Mongo Uptime** test monitors the uptime of the target Mongo database instance.

Target of the test : A MongoDB server

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for each database on the server being monitored.

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed.
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens.
Database Name	The test connects to a specific Mongo database to run API commands and pull metrics of interest. Specify the name of this database here. The default value of this parameter is *admin*.
Username and Password	The eG agent has to be configured with the credentials of a user who has the required privileges to monitor the target MongoDB instance, if the MongoDB instance is access control enabled. To know how to create such a user, refer to Section **2.1**. If the target MongoDB instance is not access control enabled, then, specify *none* against the Username and Password parameters.
Confirm Password	Confirm the password by retyping it here.
ReportManagerTime	By default, this flag is set to **Yes**, indicating that, by default, the detailed diagnosis of this test, if enabled, will report the shutdown and reboot times of the device in the manager's time zone. If this flag is set to **No**, then the shutdown and reboot times are

Parameter	Description
	shown in the time zone of the system where the agent is running(i.e., the system being managed for agent-based monitoring, and the system on which the remote agent is running - for agentless monitoring).
DD Frequency	Refers to the frequency with which detailed diagnosis measures are to be generated for this test. The default is *1:1*. This indicates that, by default, detailed measures will be generated every time this test runs, and also every time the test detects a problem. You can modify this frequency, if you so desire. Also, if you intend to disable the detailed diagnosis capability for this test, you can do so by specifying *none* against DD Frequency.
Detailed Diagnosis	To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the **On** option. To disable the capability, click on the **Off** option. The option to selectively enable/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled: • The eG manager license should allow the detailed diagnosis capability • Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Has server been restarted?	Indicates whether the database instance has been rebooted during the last measurement period or not.		If the value of this measure is *Yes*, it means that the database instance was rebooted during the last measurement period. By checking the time periods when this metric changes from *No* to *Yes*, an administrator can determine the times when this database instance was rebooted. The Detailed Diagnosis of this measure, if enabled, lists the *TIME, SHUTDOWN DATE, RESTART DATE, SHUTDOWN DURATION, and IS MAINTENANCE*.

Measurement	Description	Measurement Unit	Interpretation
Uptime since the last measurement	Indicates the time period that the database instance has been up since the last time this test ran.	Secs	If the database instance has not been rebooted during the last measurement period and the agent has been running continuously, this value will be equal to the measurement period. If the database instance was rebooted during the last measurement period, this value will be less than the measurement period of the test. For example, if the measurement period is 300 secs, and if the database instance was rebooted 120 secs back, this metric will report a value of 120 seconds. The accuracy of this metric is dependent on the measurement period – the smaller the measurement period, greater the accuracy.
Uptime	Indicates the total time that the database instance has been up since its last reboot.		This measure displays the number of years, months, days, hours, minutes and seconds since the last reboot. Administrators may wish to be alerted if the database instance has been running without a reboot for a very long period. Setting a threshold for this metric allows administrators to determine such conditions.

4.2 The Mongo Memory Layer

Using the tests mapped to this layer, administrators can proactively detect:

- A potential memory contention on the server
- Locking irregularities
- Poor cache usage
- Delays in flushing writes to disk

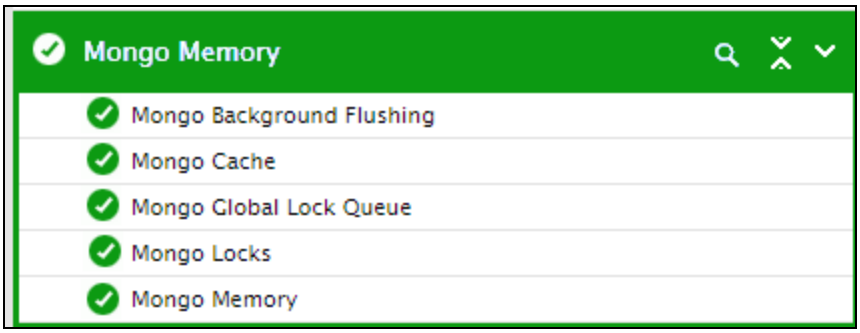

Figure 4.2: The tests mapped to the Mongo Memory layer

4.2.1 Mongo Background Flushing Test

By default, MongoDB instances using the MMAPv1 storage engine write in-memory changes to disk every 60 seconds. If Journaling is enabled, then the MongoDB server writes changes to the on-disk journal first. These changes will then flush from journal to the disk. If the server crashes before all the changes are flushed to the disk, then journaling ensures that the changes can still be recovered from the journal. However, where Journaling is not enabled, if changes in memory are not flushed to the disk quickly, then the sudden terminaton of the MongoDB server can result in loss of critical data. If this is to be avoided, then administrators should be able to proactively detect any potential slowness in background flushing and promptly initiate measures to pre-empt the data loss than an server crash can cause. The **Mongo Background Flushing** test helps with this.

This test tracks flushes to disk and reports the average time taken by the monitored server to flush writes in memory to disk. In the process, the test proactively alerts administrators to slowness in disk writes. The test also reports the duration of the last disk write, thus enabling administrators to figure out when the slowness could have occurred - did it creep in recently? or has it been persistent?

Note:

This test reports metrics only for those MongoDB instances that use the MMAPv1 storage engine.

Target of the test : A MongoDB server

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for the Mongo database server being monitored

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens
Database Name	The test connects to a specific Mongo database to run API commands and pull metrics of interest. Specify the name of this database here. The default value of this parameter is *admin*.
Username and Password	The eG agent has to be configured with the credentials of a user who has the required privileges to monitor the target MongoDB instance, if the MongoDB instance is access control enabled. To know how to create such a user, refer to Section **2.1** . If the target MongoDB instance is not access control enabled, then, specify *none* against the Username and Password parameters.
Confirm Password	Confirm the password by retyping it here.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Flushes	Indicates the number of times the monitored server flushed writes to disk.	Number	
Flushes rate	Indicates the rate at which writes were flushed to disk.	Flushes/Sec	A high value is desired, as it indicates that changes are written to disk frequently. Frequent disk writes will minimize the data loss that may occur if the server abnormally exits.
Average flush time	Indicates the average time taken by the server to flush writes to disk.	Millsecs	If the value of this measure is over 1000 milliseconds, it is a cause for concern, as it indicates that writing to disk is taking a long time. The most common causes for high flush time are: • Disk not having enough IOPS to handle the load;

Measurement	Description	Measurement Unit	Interpretation
			• Massive write activities happen all at once; Normally a flush should not take more than 1000 ms, if it does, it is likely the amount of data flushing to disk is too large for the disk to handle. With journaling enabled, which is standard on production MongoDB service, write operations go to journal file on disk and take away valuable disk I/O needed for flushing and page fault. To resolve this issue, you can do one or all of the following: • Upgrade to disk with higher IOPS, e.g. SSD / Flash Array. • Separate journal file and data file onto separate drives, to free up disk I/O taken by journal file. • Spikes in background flush occur when a large amount of dirty pages needed to flush to disk. Tune the application to spot for errors, or disburse writes to a longer time span.
Last flush time	Indicates the time taken by the last disk write.	Millisecs	If this value is close to the *Average flush time,* it could indicate that the slowness occurred during the last flush. Investigating the last flush can provide pointers to why it took more time than normal.

4.2.2 Mongo Cache Test

With WiredTiger, MongoDB utilizes both the WiredTiger internal cache and the filesystem cache.

Starting in 3.4, the WiredTiger internal cache, by default, will use the larger of either:

- 50% of RAM minus 1 GB, or
- 256 MB

This internal cache should be sized such that the working set of your application fits into it. If the internal cache is poorly sized or if the working set outgrows the cache, then, the cache will be unable to hold additional data, thereby increasing expensive disk reads.

Likewise, if changes to cached data are not written to the disk fast enough, it can cause the cache size to grow, leaving little room for additional data; once again, direct disk reads become inevitable, degrading database performance.

Additionally, if stale data is not evicted from the cache in a timely manner, it can increase cache size and consequently disk reads.

On the other hand, if the internal cache is set too high, then very little RAM will be left outside of this cache for aggregations, sorting, connection management, and the like. If there is insufficient RAM for these operations, then MongoDB can get killed by the OS Out of memory (OOM) killer! Also, over-sizing the internal cache will considerably reduce the free memory that will otherwise be available to the filesystem cache. This can also adversely impact performance!

This is why, administrators should continuously monitor the RAM usage of the WiredTiger internal cache, proactively detect excessive RAM usage by the cache, and accurately isolate its root-cause. The **Mongo Cache** test helps with this.

This test reports the maximum cache size and how much of this size is presently occupied by cached data; this reveals, whether/not the cache has enough RAM to hold additional data. Inconsistencies in cache sizing can be detected in the process and their impact on performance analyzed. Writes from cache to disk and cache evictions are also monitored, so that administrators can quickly detect bottlenecks in these processes and initiate measures to fine-tune these processes to curb cache growth.

Target of the test : A MongoDB server

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for the MongoDB server monitored.

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed.
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens.
Database Name	The test connects to a specific Mongo database to run API commands and pull metrics of interest. Specify the name of this database here. The default value of this parameter is *admin*.
Username and Password	The eG agent has to be configured with the credentials of a user who has the required privileges to monitor the target MongoDB instance, if the MongoDB instance is access control enabled. To know how to create such a user, refer to Section **2.1** If the target MongoDB instance is not access control enabled, then, specify *none* against the Username and Password parameters.
Confirm Password	Confirm the password by retyping it here.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Cache used ratio	Indicates the percentage of the maximum cache size that is used by the cache.	Percent	A value close to 100% is a cause for concern, as it indicates that the cache is consuming RAM excessively. You may want to consider resizing the cache to avoid direct disk reads. To adjust the size of the WiredTiger internal cache, use the e storage.wiredTiger.engineConfig.cacheSizeGB parameter. Avoid increasing the WiredTiger internal cache size above its default value, as this may erode the memory resources required by the filesystem cache and other critical MongoDB operations.
Used cache size	Indicates the amount of RAM used by the cached data.	MB	Ideally, this value should be well below the Maximum cache size. A steady and significant rise in this value could mean that cached data

Measurement	Description	Measurement Unit	Interpretation
			is not been written to the disks frequently and/or least-used data is not been evicted properly. You may want to fine-tune these operations and then check to see if it reduces cache memory usage.
Maximum cache size	Indicates the maximum size of the cache that WiredTiger will use for all data.	MB	
Dirty cache ratio	Indicates what percentage of the Maximum cache size is used by dirty data.	Percent	Dirty data designates data in the cache that has been modified but not yet applied (flushed) to disk. A steady and significant growth in this percentage represents a bottleneck, because it means that cached data is not being written to the disk fast enough. When writing to disk, WiredTiger writes all the data in a snapshot to disk in a consistent way across all data files. The now-durable data acts as a checkpoint in the data files. The checkpoint ensures that the data files are consistent up to and including the last checkpoint; i.e. checkpoints can act as recovery points. Using WiredTiger, even without journaling, MongoDB can recover from the last checkpoint; however, to recover changes made after the last checkpoint, run with journaling. By default, MongoDB sets checkpoints to occur in WiredTiger on user data at an interval of 60 seconds or when 2 GB of journal data has been written, whichever occurs first. This means that the amount of dirty data is expected to grow until the next checkpoint. Scaling out by adding more shard will help you

Measurement	Description	Measurement Unit	Interpretation
			reduce the amount of dirty data.
Dirty cache size	Indicates the amount of dirty data in the cache.	MB	A consistent increase in this value is a cause for concern.
Data evicted from cache	Indicates the amount of data evicted from cache.	MB	
Data read into cache	Indicates the amount of data read from cache.	MB	
Data written from cache	Indicates the amount of data written from cache into disk.	MB	
Pages evicted from cache	Indicates the number of pages evicted from cache.	Number	A high value is desired for this measure. If the Cache used ratio is very high and the value of this measure is very low, it can only mean that data is not evicted frequently enough to control cache growth. You may have to fine-tune eviction to ensure that the cache does not grow uncontrollably. Typically, when a MongoDB server approaches its maximum cache size, WiredTiger begins eviction to stop memory use from growing too large, approximating a least-recently-used algorithm. WiredTiger provides several configuration options for tuning how pages are evicted from the cache. The eviction_trigger configuration value is the occupied percentage of the total cache size that causes eviction to start. By default, WiredTiger begins evicting pages when the cache is 95% full. An application concerned about a latency spike as the cache becomes full might want to begin eviction earlier. The eviction_target configuration value is the

Measurement	Description	Measurement Unit	Interpretation
			overall target for eviction, expressed as a percentage of total cache size; that is, once eviction begins, it will proceed until the target percentage of bytes in the cache is reached. Note the eviction_target configuration value is ignored until eviction is triggered. The eviction_dirty_target configuration value is the overall dirty byte target for eviction, expressed as a percentage of total cache size; that is, once eviction begins, it will proceed until the target percentage of dirty bytes in the cache is reached. Note the eviction_dirty_target configuration value is ignored until eviction is triggered. By default, WiredTiger cache eviction is handled by a single, separate thread. In a large, busy cache, a single thread will be insufficient (especially when the eviction thread must wait for I/O). The eviction=(threads_min) and eviction=(threads_max) configuration values can be used to configure the minimum and maximum number of additional threads WiredTiger will create to keep up with the application eviction load.
Pages read into cache	Indicates the number of pages read from cache.	Number	A high value for this measure indicates effective usage of the cache.
Pages written from cache	Indicates the number of pages written from cache into disk.	Number	Typically, at configured intervals, data modified in the cache is flushed into disk, so that data in cache and disk are in sync. Flushing also frees up memory in the cache and controls its abnormal growth. This is why, ideally, the value of this measure should be high.

4.2.3 Mongo Global Lock Queue Test

MongoDB uses multi-granular reader-writer locking. Operations are allowed to lock at the global, database or collection level. Individual storage engines are allowed to implement their own

concurrency control below the collection level (e.g., at the document-level in WiredTiger).

A global 'instance-wide' lock is required when some global operations - typically, short lived operations involving multiple databases - are performed. The following MongoDB operations lock multiple databases, and may hence require a global lock:

- db.copyDatabase() must lock the entire mongod instance at once.
- db.repairDatabase() obtains a global write lock and will block other operations until it finishes.
- Journaling, which is an internal operation, locks all databases for short intervals. All databases share a single journal.
- User authentication requires a read lock on the admin database for deployments using 2.6 user credentials. For deployments using the 2.4 schema for user credentials, authentication locks the admin database as well as the database the user is accessing.
- All writes to a replica set's primary lock both the database receiving the writes and then the local database for a short time. The lock for the local database allows the mongod to write to the primary's oplog and accounts for a small portion of the total time of the operation.

MongoDB uses a readers-writer [ed: also known as "multi-reader" or "shared exclusive"] lock that allows concurrent reads access, but gives exclusive access to a single write operation. When a read lock exists, many read operations may use this lock. However, when a write lock exists, a single write operation holds the lock exclusively, and no other read or write operations may share the lock. If read or write requests are received by a database instance on which a write lock pre-exists, such requests will be placed in a queue, where they will wait until the write lock is released. While some level of queuing is to be expected, an abnormal number of enqueued requests is a cause for concern, as it only means that application requests are being continuously blocked; if the problem persists, it can cause application performance to suffer greatly. It is therefore imperative that requests waiting in queues for locks are tracked and consistent increase in this request count brought to the attention of administrators. This is what the Mongo Global Lock Queue test does.

This test monitors the current global lock queue of a Mongo database server and reports the total count of requests in the queue. Additionally, the test also reports how many of these enqueued requests are read requests and how many are write requests. This way, the test indicates whether/not locks are released and granted to new requests in a timely manner. Locking inefficiencies are thus revealed. In addition, the test also indicates what type of requests the server is blocking more - read requests? or write requests? From this, administrators can figure out what type of lock currently exists on the server being monitored - read lock? or write lock?

Target of the test : A MongoDB server

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for the Mongo database server being monitored.

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed.
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens.
Database Name	The test connects to a specific Mongo database to run API commands and pull metrics of interest. Specify the name of this database here. The default value of this parameter is *admin*.
Username and Password	The eG agent has to be configured with the credentials of a user who has the required privileges to monitor the target MongoDB instance, if the MongoDB instance is access control enabled. To know how to create such a user, refer to Section **2.1** . If the target MongoDB instance is not access control enabled, then, specify *none* against the Username and Password parameters.
Confirm Password	Confirm the password by retyping it here.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Total lock in current queue	Indicates the number of requests currently in queue.	Number	A consistently small queue, particularly of shorter operations, should cause no concern. However, if the value of this measure increases consistently, it could mean that new locks are not being granted to the Mongo database server being monitored. This can happen if an operation has been holding a write lock for a long time. Further investigation will reveal which operation this is and why it is not releasing the lock.
Read lock in current queue	Indicates the number of read requests currently in queue.	Number	If the Total locks in queue measure is exhibiting abnormal trends, observing

Measurement	Description	Measurement Unit	Interpretation
			the behavior of these two measures will provide useful pointers to what could be causing the high queue length. For instance, if the value of the Read locks in current queue measure and the Write lock in current queue measure is increasing steadily over time, it could mean that an operation is probably holding a write lock for a long time. If there are more Read locks in current queue, but very less Write locks in current queue, it could mean that write requests are being prioritized over read requests and are being granted locks.
Write lock in current queue	Indicates the number of write requests currently in queue.	Number	

4.2.4 Mongo Locks Test

MongoDB allows multiple clients to read and write the same data. In order to ensure consist ency, it uses locking to prevent multiple clients from modifying the same piece of data simultaneously.

MongoDB uses reader-writer locks that allow concurrent readers shared access to a resource, such as a database or collection, but in MMAPv1, give exclusive access to a single write operation.

In addition to a shared (S) locking mode for reads and an exclusive (X) locking mode for write operations, intent shared (IS) and intent exclusive (IX) modes indicate an intent to read or write a resource using a finer granularity lock. When locking at a certain granularity, all higher levels are locked using an intent lock.

For example, when locking a collection for writing (using mode X), both the corresponding database lock and the global lock must be locked in intent exclusive (IX) mode. A single database can simultaneously be locked in IS and IX mode, but an exclusive (X) lock cannot coexist with any other modes, and a shared (S) lock can only coexist with intent shared (IS) locks.

Locks are fair, with reads and writes being queued in order. However, to optimize throughput, when one request is granted, all other compatible requests will be granted at the same time, potentially

releasing them before a conflicting request. For example, consider a case in which an X lock was just released, and in which the conflict queue contains the following items:

IS → IS → X → X → S → IS

In strict first-in, first-out (FIFO) ordering, only the first two IS modes would be granted. Instead MongoDB will actually grant all IS and S modes, and once they all drain, it will grant X, even if new IS or S requests have been queued in the meantime.

Sometimes, in an attempt to release locks for compatible requests, MongoDB may end up increasing the lock wait times for certain incompatible requests. Some other times, certain long-running operations can cause the length of the queue to increase along with the waiting time for locks. Long lock wait times can adversely impact application performance. This is why, administrators will have to keep an eye on the locking activity on a MongoDB server, determine whether/not requests are waiting too long for locks, and also identify the types of locks (S, X, IS, IX, etc.) these requests are waiting for. This will enable administrators to quickly diagnose why certain lock types are taking longer to be released, and resolve the bottleneck. The **Mongo Locks** test helps administrators achieve this.

This test auto-discovers the different lock types/lock modes that exist currently on the target server. For each lock type/mode, the test reports the rate at which locks of that type were acquired, the rate at which requests waited for locks of that type, and the average wait time for the locks. In the process, the test promptly alerts administrators to prolonged waiting time for locks and also pinpoints the exact lock types for which requests are waiting too long. With this information, administrators can easily troubleshoot long wait times.

Target of the test : A MongoDB server

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for each lock type / lock mode.

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed.
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens.
Database Name	The test connects to a specific Mongo database to run API commands and pull metrics of interest. Specify the name of this database here. The default

Parameter	Description
	value of this parameter is *admin*.
Username and Password	The eG agent has to be configured with the credentials of a user who has the required privileges to monitor the target MongoDB instance, if the MongoDB instance is access control enabled. To know how to create such a user, refer to Section **2.1** . If the target MongoDB instance is not access control enabled, then, specify *none* against the Username and Password parameters.
Confirm Password	Confirm the password by retyping it here.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Locks	Indicates the rate at which a lock of this type was acquired.	Locks/Sec	
Lock waits	Indicates the rate at which requests were waiting to acquire a lock of this type.	Waits/Sec	A consistent increase in the value for this measure could indicate that locks are not being released in a timely manner.
Lock wait time	Indicates the time period for which requests were waiting to acquire a lock of this type, during the last measurement period.	Seconds	
Avg lock wait time	Indicates the average time for which requests were waiting to acquire a lock of this type.	Seconds	A high average lock wait time may mean sessions are having to wait for a long time to acquire locks on objects. A high value may indicate one of the following: • Too many transactions happening • Locked resources not being released properly • Locks are being held by long-running operations

4.2.5 Mongo Memory Test

MongoDB maps data files to memory for managing and interacting with all data. Memory mapping assigns files to a block of virtual memory with a direct byte-for-byte correlation. MongoDB memory maps data files to memory as it accesses documents. Unaccessed data is not mapped to memory. This provides MongoDB with an extremely fast and simple method for accessing and manipulating data.

This implies that if MongoDB is not sized with adequate virtual memory, then data accesses are bound to slow down, thus adversely impacting user experience with the server. To avoid this, administrators should periodically run the Mongo Memory test. This test monitors how the target MongoDB server utilizes the virtual memory, proactively detects potential memory contentions, and promptly alerts administrators to it, so that they can initiate measures to avert the contention and the consequent slow down.

Target of the test : A MongoDB server

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for the Mongo database server being monitored.

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed.
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens
Database Name	The test connects to a specific Mongo database to run API commands and pull metrics of interest. Specify the name of this database here. The default value of this parameter is *admin*.
Username and Password	The eG agent has to be configured with the credentials of a user who has the required privileges to monitor the target MongoDB instance, if the MongoDB instance is access control enabled. To know how to create such a user, refer to Section **2.1** . If the target MongoDB instance is not access control enabled, then, specify *none* against the Username and Password parameters.
Confirm Password	Confirm the password by retyping it here.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Total memory	If the MMAPv1 storage engine is in use, then this indicates the amount of virtual memory mapped by the database. Because MongoDB uses memory-mapped files, this value is likely to be roughly equivalent to the total size of your database or databases. This value does not include memory allocated to journal. If the WiredTiger storage engine is in use, then the value of this measure displays the quantity of virtual memory provided to the mongod process. This value includes memory allocated to the journal.	GB	
Available memory	Indicates the amount of memory unused.	GB	**Note that this measure will be reported only if the target server uses the MMAPv1 storage engine.** The value of this measure is the difference between the value of the *Total memory* and *Used memory* measures.
Used memory	Indicates the amount of RAM currently used by the target server.	GB	During normal use, this value tends to grow. In dedicated database servers, this number tends to approach the total amount of system memory.
Memory usage	Indicates the percentage of memory used.	Percent	This value is computed using the following formula:

Measurement	Description	Measurement Unit	Interpretation
			*(Used memory / Total memory) * 100* A value close to 100% indicates that almost all the memory is been utilized. No free memory or very little free memory will be available for database operations in this case, causing database performance to severely deteriorate. You may want to increase the memory allocation to the database instance to pre-empt this.
Allocated memory with journal	Indicates the amount of mapped memory, including the memory used for journaling.	GB	**Note that this measure will be reported only if the target server uses the MMAPv1 storage engine and journaling is enabled.** The value of this measure will always be twice the value of the *Total memory* measure.
Heap memory used	Indicates the amount of Java heap memory used by the MongoDB server being monitored.	GB	A low value is desired for this measure. **Note that this measure will be reported only if the target MongoDB server runs on Unix/Linux.**
Page faults	Indicates the total number of page faults that require disk operations	Number	Page faults refer to operations that require the database server to access data which isn't available in active memory. The page faults counter may increase dramatically during moments of poor performance and may correlate with limited memory environments and larger data sets. Limited and sporadic page faults do not necessarily indicate an issue. **Note that this measure will be**

Measurement	Description	Measurement Unit	Interpretation
			reported only if the target MongoDB server runs on Unix/Linux.

4.3 The Mongo Databases Layer

With the help of the tests mapped to this layer, the space utilization of the databases on the MongoDB server can be monitored and space constraint (if any) proactively detected. You can identify the databases that are growing alarmingly in size, and can also inpoint the collections in those databases that are contributing to its abnormal growth. You can also spot the databases that are taking too much time to perform specific operations and can even identify the collections in those databases where those operations are bottlenecked.

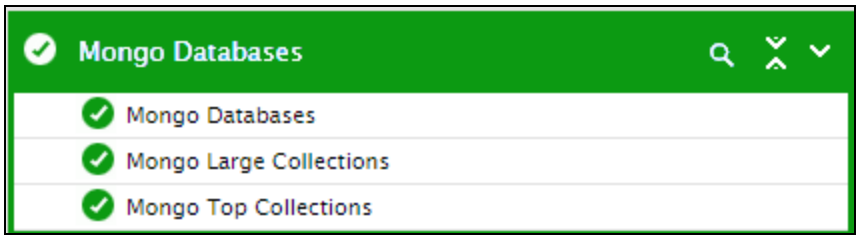

Figure 4.3: The tests mapped to the Mongo Databases layer

4.3.1 Mongo Databases Test

Storage space contentions on a database can result in the loss of critical data. To avoid this, administrators should keep a close watch on the space usage of each database, proactively detect a space crunch, and promptly resolve it. This is where the **Mongo Databases** test helps. This test auto-discovers all the databases on the MongoDB server and monitors the availability and space usage of each database. Alerts are promptly sent out if any database is found to be unavailable or is running out of space. In the event of a space crunch in a database, you can also use this test to figure out what is causing the space drain - documents? index files? or namespace files?

Target of the test : A MongoDB server

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for each database on the server being monitored.

Configurable parameters for the test

Parameter	Description
Test Period	How often should the test be executed.
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens. By default, this is *27107*.
Database Name	The test connects to a specific Mongo database to run API commands and pull metrics of interest. Specify the name of this database here. The default value of this parameter is *admin*.
Username and Password	The eG agent has to be configured with the credentials of a user who has the required privileges to monitor the target MongoDB instance, if the MongoDB instance is access control enabled. To know how to create such a user, refer to Section **2.1** . If the target MongoDB instance is not access control enabled, then, specify *none* against the Username and Password parameters.
Confirm Password	Confirm the password by retyping it here.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Is database available?	Indicates whether/not this database is available.		This measure reports the value *Yes* if the database is available and the value *No* if it is not. The numeric values that correspond to the aforesaid measure values are as follows: **Measure Value** / **Numeric Value**: Yes / 1; No / 0 **Note:** Typically, this measure reports the

Measurement	Description	Measurement Unit	Interpretation
			Measure Values listed above to indicate database availability. However, in the graph of this measure, the same is indicated using the numeric equivalents only.
Total database size	Indicates the total size of this database. **Note:** This measure is applicable only if the monitored MongoDB server uses the MMAPv1 storage engine.	GB	
Storage size	Indicates the total amount of space allocated to collections in this database for document storage.	GB	Every MongoDB database instance consists of a namespace file, journal files and data files. Data files store BSON documents, indexes, and MongoDB-generated metadata in structures called extents. Each data file is made up of multiple extents. Extents are logical containers within data files used to store documents and indexes. Data and indexes are each contained in their own sets of extents. The value of this measure is equal to the size of all the data extents in a database. Storage size does not decrease as you remove or shrink documents. This value may be smaller than the value of the *Total Data size* measure for a database that is using the WiredTiger storage engine with compression enabled. For a database that is using the MMAPv1 storage engine on the other hand, this value may be larger than the value of the *Total Data size*

Measurement	Description	Measurement Unit	Interpretation
			measure because it includes yet-unused space (in data extents) and space vacated by deleted or moved documents within extents.
Index size	Indicates the total size of all indexes created on this database.	GB	If the *Space utilization* measure reports a value close to 100%, you may want to compare the *Storage size, Index size, Namespace file size, or Data size* measures, to know where space is consumed the maximum.
Used space	Indicates how much space allocated to this database is currently in use. **Note:** This measure is applicable only if the monitored MongoDB server uses the MMAPv1 storage engine.	GB	
Free space	Indicates how much space allocated to this database is still available for use. **Note:** This measure is applicable only if the monitored MongoDB server uses the MMAPv1 storage engine.	GB	
Space utilization	Indicates the percentage of allocated space that is used by this database. **Note:** This measure is applicable only if the monitored	Percent	If the *Space utilization* measure reports a value close to 100%, its a cause for concern, as it implies that your database is rapidly running out of space. Under such circumstances, you may want to consider allocating more space to this database.

Measurement	Description	Measurement Unit	Interpretation
	MongoDB server uses the MMAPv1 storage engine.		
Average object size	Indicates the average size of each document in this database.	GB	If the *Space utilization* of a database is increasing consistently along with the value of the *Data size* measure, then, you may want to check the value of this measure. If the value of this measure is also very high, then you can conclude that the space usage is owing to the large size of documents in the database.
Namespace file size	Indicates the total size of all namespace files in this database. **Note:** This measure is applicable only if the monitored MongoDB server uses the MMAPv1 storage engine.	GB	If the *Space utilization* measure reports a value close to 100%, you may want to compare the *Storage size, Index size, Namespace file size, or Data size* measures, to know where space is consumed the maximum.
Total extents	Indicates the total number of extents across all collections in this database.	Number	Extents are logical containers within data files used to store documents and indexes. Data and indexes are each contained in their own sets of extents.
Total indexes	Indicates the total number of indexes across all collections in this database.	Number	
Data size	Indicates the total size of all the documents and padding stored in the database.	GB	The value of this measure will decrease when you delete documents, but will not decrease when documents shrink because the space used by the original document has already been allocated (to that particular document) and cannot be used by other documents. Alternatively if a user

Measurement	Description	Measurement Unit	Interpretation
			updates a document with more data, the value of this measure will remain the same as long as the new document fits within its originally padded pre-allocated space.
Growth rate	Indicates the rate at which the size of this database is growing.	GB	The value of this measure is calculated using the following formula: *(Data size + Index size) of the current measurement period (-) (Data size + Index size) of the previous measurement period* A consistent increase in the value of this measure is a cause for concern, as it indicates a steady growth in database size.
Compression ratio	Indicates the rate at which data in storage is compressed.	Percent	If the *Growth rate* is very high and the *Compression ratio* is very low, you may want to consider increasing the compression ratio to conserve storage space and optimize growth rate.

4.3.2 Mongo Large Collections Test

MongoDB stores documents in collections. Collections are analogous to tables in relational databases. Collections are leading consumers of database space. Larger the collection, larger will be the database size.

If the Mongo Databases test reveals that a particular database is growing in size rapidly, you can use the Mongo Large Collections test to precisely pinpoint the collection in that database, which is contributing to the abnormal database growth. This test auto-discovers the databases on a target MongoDB server and reports the number of collections that exceed a configured size. Detailed diagnostics of this test lists the large collections.

Target of the test : A MongoDB server

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for each database in the server being monitored.

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed.
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens.
Database Name	The test connects to a specific Mongo database to run API commands and pull metrics of interest. Specify the name of this database here. The default value of this parameter is *admin*.
Username and Password	The eG agent has to be configured with the credentials of a user who has the required privileges to monitor the target MongoDB instance, if the MongoDB instance is access control enabled. To know how to create such a user, refer to Section **2.1** . If the target MongoDB instance is not access control enabled, then, specify *none* against the Username and Password parameters.
Confirm Password	Confirm the password by retyping it here.
Collection Size in GB	This test counts all those collections that are of a size larger than the value specified here as large collections. By default, this is set to 10 GB. This means that by default, those collections that are over 10 GB in size will be counted as a large collection by this test. You can change this value, if required.
Detailed Diagnosis	To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the **On** option. To disable the capability, click on the **Off** option. The option to selectively enable/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled: • The eG manager license should allow the detailed diagnosis capability • Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Large collections	Indicates the number of collections in this database that are of a size larger than the value configured against **Collection Size in GB** parameter.	Number	Use the detailed diagnosis of this test to know which are the large collections.
Largest collection size	Indicates the size of the largest collection.	GB	

4.3.3 Mongo Top Collections Test

Administrators often need to track the level of activity on the Mongo databases, so that they can identify the busy databases and the type of activity that is keeping them busy. The time spent on each activity should also be determined, so that activities taking an abnormally long time can be isolated and the reasons investigated. Administrators also need help in understanding how well each database handles concurrency. Are locks held for long or released quickly? Which type of locks are held for too long?, are some of the questions on concurrency for which administrators often need quick and accurate answers. The **Mongo Top Collections** test provides useful insights into database activity, thereby alleviating many of the administrative pains related to the same. For each database on the server, the test reports the rate at which operations such as querying, inserting, updating, deleting, etc., are performed on that database. This points to busy databases and what is keeping them busy! Additionally, the test reveals where the database spent most of its time - query execution? updation? insertion? command execution? Detailed diagnostics of the test also point you to the exact collection in a database that prolonged each of these operations, thus enabling administrators to swoop down on problem collections. Furthermore, the test focuses on the locking activity in each database. The rate, type, and duration of locks held are reported, and administrators proactively alerted to locking-related irregularities. In the event of an alert, you can use the detailed diagnostics of the test to identify which collections hold the maximum locks and which ones have held locks for the maximum time.

Target of the test : A MongoDB server

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for each database in the server being monitored.

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed.
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens.
Database Name	The test connects to a specific Mongo database to run API commands and pull metrics of interest. Specify the name of this database here. The default value of this parameter is *admin*.
Username and Password	The eG agent has to be configured with the credentials of a user who has the required privileges to monitor the target MongoDB instance, if the MongoDB instance is access control enabled. To know how to create such a user, refer to Section **2.1** . If the target MongoDB instance is not access control enabled, then, specify *none* against the Username and Password parameters.
Confirm Password	Confirm the password by retyping it here.
Detailed Diagnosis	To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the **On** option. To disable the capability, click on the **Off** option. The option to selectively enable/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled: • The eG manager license should allow the detailed diagnosis capability • Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Collections	Indicates the number of collections in this database.	Number	

Measurement	Description	Measurement Unit	Interpretation
Modified collections	Indicates the number of collections in this database that were modified since the last measurement period.	Number	
Read locks	Indicates the rate at which read locks were held on this database during the last measurement period.	Locks/Sec	A read lock is typically granted when multiple clients are attempting to issue a query to the same collection in a database or get more data from the same cursor. If the value of this measure is consistently increasing for a database, it hints at a probable read lock contention on that database. Subsequent read requests to that database will hence be blocked. Lock contention may also lead to high CPU usage on the database server. At this juncture therefore, you may want to know which collection in that database is locked and for how long, so that collections holding the maximum locks can be identified. The detailed diagnosis of this measure provides this information. The collections that are locked, the count of read locks on each collection, and the duration of the read locks is reported as part of the detailed diagnosis. From this, you can quickly identify the collection that is holding an unusually large number of read locks. The abnormal locking activity on that collection can then be investigated and the reasons for the same diagnosed, so that read locks are released and server performance

Measurement	Description	Measurement Unit	Interpretation
			is enhanced.
Average read lock time	Indicates the average duration for which read locks were held by collections on this database.	Millisecs	If the value of this measure is consistently increasing for a database, it implies that one/more collections in that database are probably locked for an unduly long time. This can prevent subsequent requests from accessing those collections. Lock contention may also lead to high CPU usage on the database server. At this juncture therefore, you may want to know which collection in that database is locked and for how long, so that collections holding locks for the maximum time can be identified. The detailed diagnosis of this measure provides this information. The collections that are locked, the count of read locks on each collection, and the duration of the read locks is reported as part of the detailed diagnosis. From this, you can quickly identify the collection that has been holding locks for the longest time. The abnormal locking activity on that collection can then be investigated and the reasons for the same diagnosed, so that read locks are released and server performance is enhanced.
Write locks	Indicates the number of write locks on this database during the last measurement period.	Number	Typically, a write lock is granted on a collection if multiple clients are attempting to access that collection for inserting, modifying, or removing data.

Measurement	Description	Measurement Unit	Interpretation
			If the value of this measure is consistently increasing for a database, it implies that many collections in that database are probably locked or one/more collections are holding many locks. This can prevent subsequent requests from writing data into those collections. Lock contention may also lead to high CPU usage on the database server. At this juncture therefore, you may want to know which collections in that database are locked and how many locks are held by each collection, so that collections holding the maximum number of locks can be identified. The detailed diagnosis of this measure provides this information. The collections that are locked, the count of write locks on each collection, and the duration of the write locks is reported as part of the detailed diagnosis. From this, you can quickly identify the collection that is holding an unusually large number of write locks. The abnormal locking activity on that collection can then be investigated and the reasons for the same diagnosed, so that write locks are released and server performance is enhanced.
Average write lock time	Indicates the average duration for which write locks were held by collections on this database.	Millisecs	If the value of this measure is consistently increasing for a database, it implies that one/more collections in that database are probably locked for an unduly long

Measurement	Description	Measurement Unit	Interpretation
			time. This can prevent subsequent requests from accessing those collections. Lock contention may also lead to high CPU usage on the database server. At this juncture therefore, you may want to know which collection in that database is locked and for how long, so that collections holding locks for the maximum time can be identified. The detailed diagnosis of this measure provides this information. The collections that are locked, the count of write locks on each collection, and the duration of the write locks is reported as part of the detailed diagnosis. From this, you can quickly identify the collection that has been locked for an unduly long time. The abnormal locking activity on that collection can then be investigated and the reasons for the same diagnosed, so that write locks are released and server performance is enhanced.
Query operations	Indicates the rate at which query operations are performed on this database.	Queries/Sec	
Insert operations	Indicates the rate at which data insert operations are performed on this database.	Inserts/Sec	
Update operations	Indicates the rate at which update operations are performed on this	Updates/Sec	

Measurement	Description	Measurement Unit	Interpretation
	database.		
Delete operations	Indicates the rate at which delete operations are performed on this database.	Deletes/Sec	
Get more operations	Indicates the rate at which get more operations are performed on this database.	Get more/Sec	A get more command is typically used in conjunction with commands that return a cursor, e.g. find and aggregate, to return subsequent batches of documents currently pointed to by the cursor.
Command operations	Indicates the rate at which data insert operations are performed on this database.	Inserts/Sec	Certain administrative commands can exclusively lock the database for extended periods of time. For instance, *db.collection.createIndex()*, when issued without setting background to *true*, *reIndex*, *compact*, *db.repairDatabase()*, *db.createCollection()*, when creating a very large (i.e. many gigabytes) capped collection,*db.collection.validate()*, and *db.copyDatabase()*. Some administrative commands lock the database but only hold the lock for a very short time - eg., db.collection.dropIndex(), db.getLastError(), db.addUser(), etc. Some other commands can even lock multiple databases. These include, db.copyDatabase(), db.repairDatabase(), etc.
Total locks	Indicates the rate at which locks (both read and write) are held by this database.	Locks/Sec	If the value of this measure is increasing consistently for a database, then, check the values of the *Read locks* and *Write locks*

Measurement	Description	Measurement Unit	Interpretation
			measures for that database. This will indicate the type of locks that are held more by the database. You can then use the detailed diagnosis of the corresponding measure to figure out which collection is holding the locks and for how long.
Average lock time	Indicates the average duration for which this database held locks (both read and write).	Millisecs	If the value of this measure is increasing consistently for a database, then, check the values of the *Average read lock time* and *Average write lock time* measures for that database. This will indicate the type of locks that are held for the maximum time. You can then use the detailed diagnosis of the corresponding measure to figure out which collection is holding the locks for the maximum time.
Maximum query time	Indicates the maximum time taken by query operations on this database.	Millisecs	If the value of this measure is abnormally high for a database, use the detailed diagnosis of the measure to identify the collection in that database that executed queries slowly. The detailed diagnosis displays the top-10 collections in terms of query execution time. Besides the names of collections and the total time each took to execute queries, the detailed diagnosis also displays the number of query operations performed by each collection and the average query time per collection (i.e., *Execution rate*). This information will enable you to figure out what could have caused query execution to be slow on a collection - is it because of too many

Measurement	Description	Measurement Unit	Interpretation
			queries to that collection? or is it owing to a few long running queries on that collection?
Maximum get more time	Indicates the maximum time taken by get more operations on this database.	Millisecs	If the value of this measure is abnormally high for a database, use the detailed diagnosis of the measure to identify the collection in that database that executed get more commands slowly. The detailed diagnosis displays the top-10 collections in terms of get more command execution time. Besides the names of collections and the total time each took to execute get more commands, the detailed diagnosis also displays the number of get more operations performed by each collection and the average get more time per collection (i.e., *Execution rate*). This information will enable you to figure out what could have caused get more command execution to be slow on a collection - is it because of a command overload on that collection? or is it because a few commands took a significantly longer time to execute the commands?
Maximum insert time	Indicates the maximum time taken by insert operations on this database.	Millisecs	If the value of this measure is abnormally high, use the detailed diagnosis of the measure to identify the collection that took the maximum time for performing inserts. The detailed diagnosis of this measure displays the top-10 collections in terms of insert time. Besides the names of collections and the total time each took to insert data, the detailed

Measurement	Description	Measurement Unit	Interpretation
			diagnosis also displays the number of insert operations performed by each collection and the average insert time per collection (i.e., *Execution rate*). This information will enable you to figure out what could have caused query execution to be slow on a collection - is it because of too many inserts on that collection? or is it owing to a few long running inserts on that collection?
Maximum update time	Indicates the maximum time taken by update operations on this database.	Millisecs	If the value of this measure is abnormally high for a database, use the detailed diagnosis of the measure to identify the exact collection in that database, on which updates were slowest. The detailed diagnosis displays the top-10 collections in terms of update time. Besides the names of collections and the total time each took to perform update operations, the detailed diagnosis also displays the number of update operations performed by each collection and the average update ime per collection (i.e., *Execution rate*). This information will enable you to figure out what could have caused updates to be slow on a collection - is it because of too many updates to that collection? or is it because a few updates took too long a time?
Maximum delete time	Indicates the maximum time taken by delete operations on this database.	Millisecs	If the value of this measure is abnormally high for a database, use the detailed diagnosis of the measure to identify the collection that performed deletes most slowly. The

Measurement	Description	Measurement Unit	Interpretation
			detailed diagnosis displays the top-10 collections in terms of deletion time. Besides the names of collections and the total time each took to delete data, the detailed diagnosis also displays the number of delete operations performed by each collection and the average delete time per collection (i.e., *Execution rate*). This information will enable you to figure out what could have caused deletion to be slow on a collection - is it because of too many delete requests to that collection? or is it because a few delete operations took too long a time?
Maximum command time	Indicates the maximum time taken by this database to execute commands.	Millisecs	If the value of this measure is abnormally high for a database, use the detailed diagnosis of the measure to identify the collection that was the slowest in executing commands. The detailed diagnosis displays the top-10 collections in terms of command time. Besides the names of collections and the total time each took to perform command operations, the detailed diagnosis also displays the number of command operations performed by each collection and the average command time per collection (i.e., *Execution rate*). This information will enable you to figure out what could have caused a collection to execute commands slowly - is it because of a command overload? or is it because of a few long running commands?
Maximum query time rate	Indicates the maximum time this database took	Millisecs/execution	If the value of this measure is

Measurement	Description	Measurement Unit	Interpretation
	to perform a single query operation.		abnormally high for a database, use the detailed diagnosis of the measure to zero-in on the exact collection in that database that took the maximum time to perform a single query operation. Typically, the detailed diagnosis of this measure displays the top-5 collections in terms of *Execution rate* - i.e., the average time taken by a collection to execute query operations. Besides the names of collections and the *Execution rate*, the detailed diagnosis also displays the number of query operations performed by each collection and the total query time per collection. If the *Execution rate* is equal to or close to the total *Query time* of a collection, you can conclude that one or very few query operations are taking too long to execute on that collection.
Maximum get more time rate	Indicates the maximum time this database took to perform a get more operation.	Millisecs/execution	If the value of this measure is abnormally high for a database, use the detailed diagnosis of the measure to zero-in on the exact collection in that database that took the maximum time to perform a single get more operation. Typically, the detailed diagnosis of this measure displays the top-5 collections in terms of *Execution rate* - i.e., the average time taken by a collection to execute get more operations. Besides the names of collections and the *Execution rate*, the detailed diagnosis also displays the number of get more operations performed by each collection and the total get more

Measurement	Description	Measurement Unit	Interpretation
			time per collection. If the *Execution rate* is equal to or close to the total *Get more* time of a collection, you can conclude that one or very few get more operations are taking too long to execute on that collection.
Maximum insert time rate	Indicates the maximum time this database took to perform a single insert operation.	Millisecs/execution	If the value of this measure is abnormally high for a database, use the detailed diagnosis of the measure to zero-in on the exact collection in that database that took the maximum time to perform a single insert operation. Typically, the detailed diagnosis of this measure displays the top-5 collections in terms of *Execution rate* - i.e., the average time taken by a collection to execute insert operations. Besides the names of collections and the *Execution rate*, the detailed diagnosis also displays the number of insert operations performed by each collection and the total insert time per collection. If the *Execution rate* is equal to or close to the total Insert time of a collection, you can conclude that one or very few insert operations are taking too long to execute on that collection.
Maximum update time rate	Indicates the maximum time this database took to perform a single update operation.	Millisecs/execution	If the value of this measure is abnormally high for a database, use the detailed diagnosis of the measure to zero-in on the exact collection in that database that took the maximum time to perform a single update operation. Typically, the detailed diagnosis of this measure displays the top-5 collections in terms of

Measurement	Description	Measurement Unit	Interpretation
			Execution rate - i.e., the average time taken by a collection to execute update operations. Besides the names of collections and the *Execution rate*, the detailed diagnosis also displays the number of update operations performed by each collection and the total update time per collection. If the *Execution rate* is equal to or close to the total *Update time* of a collection, you can conclude that one or very few update operations are taking too long to execute on that collection.
Maximum delete time rate	Indicates the maximum time this database took to perform a single delete operation.	Millisecs/execution	If the value of this measure is abnormally high for a database, use the detailed diagnosis of the measure to zero-in on the exact collection in that database that took the maximum time to perform a single delete operation. Typically, the detailed diagnosis of this measure displays the top-5 collections in terms of *Execution rate* - i.e., the average time taken by a collection to execute delete operations. Besides the names of collections and the *Execution rate*, the detailed diagnosis also displays the number of delete operations performed by each collection and the total delete time per collection. If the *Execution rate* is equal to or close to the total delete time of a collection, you can conclude that one or very few delete operations are taking too long to execute on that collection.
Maximum command time rate	Indicates the maximum time this database took to perform a single	Millisecs/execution	If the value of this measure is abnormally high for a database, use

Measurement	Description	Measurement Unit	Interpretation
	command operation.		the detailed diagnosis of the measure to zero-in on the exact collection in that database that took the maximum time to perform a single command operation. Typically, the detailed diagnosis of this measure displays the top-5 collections in terms of *Execution rate* - i.e., the average time taken by a collection to execute command operations. Besides the names of collections and the *Execution rate*, the detailed diagnosis also displays the number of command operations performed by each collection and the total command time per collection. If the *Execution rate* is equal to or close to the total *Command time* of a collection, you can conclude that one or very few command operations are taking too long to execute on that collection.

4.4 The Mongo Service Layer

Health of critical services of the MongoDB server -eg., journaling, replication, etc. - are monitored and reported by this layer. Additionally, the tests mapped to this layer monitor the connection usage of the server, its availability and connectivity, and also the transactions on the server.

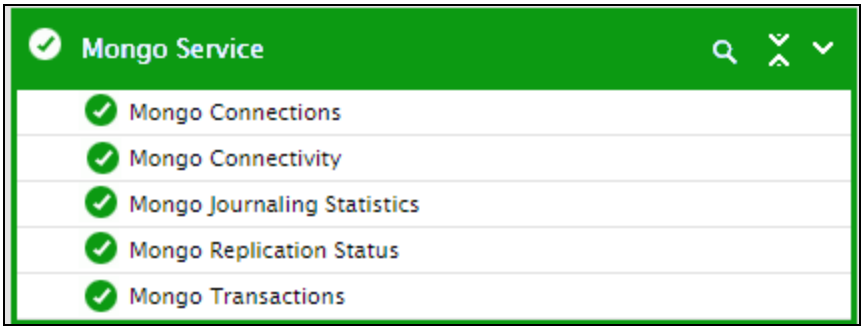

Figure 4.4: The tests mapped to the Mongo Service layer

4.4.1 Mongo Connections Test

Monitoring client connections to a MongoDB server typically provide insights into the workload of the server and whether/not the server is sized well-enough to handle the load. This is exactly the type of visibility the Mongo Connections test provides. This test tracks client connections to the target server and reports their usage. With the help of the metrics reported by this test, administrators can assess the load on the server and can determine whether/not the server has sufficient free connections to handle subsequent connection requests. Based on these insights, administrators can plan the future connection capacity of the server.

Target of the test : A MongoDB server

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for the Mongo database server being monitored

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens
Database Name	The test connects to a specific Mongo database to run API commands and pull metrics of interest. Specify the name of this database here. The default value of this parameter is *admin*.
Username and Password	The eG agent has to be configured with the credentials of a user who has the required privileges to monitor the target MongoDB instance, if the MongoDB instance is access control enabled. To know how to create such a user, refer to Section **2.1** . If the target MongoDB instance is not access control enabled, then, specify *none* against the Username and Password parameters.
Confirm Password	Confirm the password by retyping it here.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Total connections	Indicates the total number	Number	This is the sum of the values of the

Measurement	Description	Measurement Unit	Interpretation
	of incoming connections to the server.		*Open connections* and *Free connections* measure.
Open connections	Indicates the count of incoming connections from clients to the server.	Number	This is a good measure of the current load on the server. The value will include all incoming connections including any shell connections or connections from other servers, such as replica set members or mongos instances.
Free connections	Indicates the number of unused connections.	Number	The value 0 or a very low value for this measure is a cause for concern, as it indicates that there are no (or very few) free connections to handle subsequent connection requests. You may want to increase the maximum number of connections the server can handle, so that connection requests are not rejected.
Connection usage	Indicates the percentage of total connections that is in use.	Percent	A value close to 100% is indicative of over-utilization of connections. This in turn means that there are there are no (or very few) free connections to handle subsequent connection requests. You may want to increase the maximum number of connections the server can handle, so that connection requests are not rejected. The value of this measure is calculated using the formula: *(Open connections / Total connections)*100*

4.4.2 Mongo Connectivity Test

Availability of the MongoDB server and its responsiveness to client requests is crucial to the high uptime and optimum performance of the critical business applications that use that server. If the MongoDB server is inaccessible or does not respond to requests quickly, the performance of the

business applications/services that rely on the server will suffer. To avoid service downtime/slowness, the database administrator should promptly detect the unavailability or poor responsiveness of the MongoDB server and also accurately pinpoint the root-cause of the anomaly - is it because the MongoDB server is down? is it because of a broken or latent network connection to the server? is it because the collection that the user is trying to access is unavailable? or is document retrieval from the collection taking a long time? The Mongo Connectivity test helps with this.

This test periodically checks whether/not the MongoDB server is up and running, verifies network connectivity to the server, and also measures the connection time. In addition, the test also emulates a MongoDB API call to a specified collection on the server for retrieving documents from it; in the process, the test reports collection availability and the time taken to read the documents. This way, the test alerts administrators to the non-availability and poor responsiveness of the server, and also points them to where exactly the bottleneck is.

Target of the test : A MongoDB server

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for the server being monitored

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens
Database Name	The test connects to a specific Mongo database to run API commands and pull metrics of interest. Specify the name of this database here. The default value of this parameter is *admin*.
Username and Password	The eG agent has to be configured with the credentials of a user who has the required privileges to monitor the target MongoDB instance, if the MongoDB instance is access control enabled. To know how to create such a user, refer to Section **2.1** . If the target MongoDB instance is not access control enabled, then, specify *none* against the Username and Password parameters.
Confirm Password	Confirm the password by retyping it here.
Collection Name	A collection is the equivalent of an RDBMS table, which exists within a

Parameter	Description
	single database. A collection contains MongoDB documents. This test emulates a MongoDBI API call to a collection, and attempts to read the documents contained within. In the process, the test captures the collection availability and the time taken by the document read/retrieval operation. To enable this emulation, specify the name of the collection that you want the test to access, in the COLLECTION NAME text box. Your specification should be of the following format: <Namespace>.<collection name>. The default value of this parameter is *system.indexes*.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Server availability	Indicates whether/not the MongoDB server is up and running.	Percent	The availability is 100% when the server is responding to a request and 0% when it is not. Availability problems may be caused by a misconfiguration / malfunctioning of the database server, or because the server has not been started. Besides the above, this measure will report that the server is unavailable even if a connection to the database instance is unavailable, or if an attempt to retrieve documents from the configured collection fails. In this case, you can check the values of the Connection availability and Document availability measures to know what is exactly causing the database instance to not respond to requests - is it owing to a connection unavailability? or is it due to the collection unavailability?
Server response time	Indicates the time taken by the server to respond to a user request.	Secs	A sudden increase in response time is indicative of a bottleneck at the database server.
Connection availability	Indicates whether the database connection is available or not.	Percent	If this measure reports the value 100 , it indicates that the database

Measurement	Description	Measurement Unit	Interpretation
			connection is available. The value 0 on the other hand indicates that the database connection is unavailable. A connection to the database may be unavailable if the database is down or if the database is listening on a port other than the one configured for it in the eG manager or owing to a poor network link. If the Server availability measure reports the value 0, then, you can check the value of this measure to determine whether/not it is due to the unavailability of a connection to the server.
Connection response time	Indicates the time taken by the database connection.	Secs	A high value could indicate a connection bottleneck. Whenever the Server response time of the measure soars, you may want to check the value of this measure to determine whether a connection latency is causing the poor responsiveness of the server.
Collection availability	Indicates whether/not the configured **Collection Name** is available.	Percent	If this measure reports the value 100, it indicates that the **Collection Name** you have configured for this test is available. The value 0 on the other hand indicates that the collection is unavailable.
Collection response time	Indicates the time taken to read documents from the configured **Collection Name**.	Secs	A high value could indicate an I/O bottleneck on the server. If the Server response time measure is unusually high, you may want to check the value of this measure to figure out if a bottleneck when reading from a collection is impacting responsiveness.
Number of records	Indicates the number of	Number	

Measurement	Description	Measurement Unit	Interpretation
	documents fetched from the configured **Collection Name**		

4.4.3 Mongo Cursors Test

A cursor is a pointer to the result set of a query. Clients can iterate through a cursor to retrieve results. By default, cursors timeout after 10 minutes of inactivity. Cursors that remain open for long periods of time can drain valuable memory resources; so can those cursors that have been configured to not timeout at all. To avoid the unnecessary memory drain, the **Mongo Cursors** test instantly alerts administrators to an abnormally large number of open cursors and notimeout cursors.

Target of the test : A MongoDB server

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for the MongoDB server monitored.

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed.
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens.
Database Name	The test connects to a specific Mongo database to run API commands and pull metrics of interest. Specify the name of this database here. The default value of this parameter is *admin*.
Username and Password	The eG agent has to be configured with the credentials of a user who has the required privileges to monitor the target MongoDB instance, if the MongoDB instance is access control enabled. To know how to create such a user, refer to Section **2.1** . If the target MongoDB instance is not access control enabled, then, specify *none* against the Username and Password parameters.
Confirm Password	Confirm the password by retyping it here.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Open cursors	Indicates the number of cursors that are currently opened by MongoDB for clients.	Number	A gradual increase in the number of open cursors without a corresponding growth of traffic is often symptomatic of poorly indexed queries. It can also be the result of long running queries due to large result sets. You should take a look to see how you could optimize them.
Timeout cursor	Indicates the number of cursors that timed out.	Number	The value of this measure is incremented when a client connection has died without having gracefully closed the cursor. This cursor remains open on the server, consuming memory. By default MongoDB reaps these cursors after 10 minutes of inactivity. You should check if you have a large amount of memory being consumed from non-active cursors. A high number of timed out cursors can be related to application issues.
Timeout disabled cursor	Indicates the number of open cursors with timeout disabled.	Number	The value 0 is desired for this measure. This is because, cursors with no timeout can prevent resources to be freed as it should and slow down internal system processes; hence, they should be avoided. A non-zero value for this measure is hence a cause for concern. The *DBQuery.Option.noTimeout* flag (until v3.0) or the *cursor.noCursorTimeout()* method can be used to prevent the server to timeout cursors after a period of inactivity (idle cursors).

4.4.4 Mongo Journaling Statistics Test

With MMAPv1, when a write operation occurs, MongoDB updates the in-memory view. With journaling enabled, MongoDB writes the in-memory changes first to on-disk journal files. If MongoDB should terminate or encounter an error before committing the changes to the data files,

MongoDB can use the journal files to apply the write operation to the data files and maintain a consistent state.

With journaling, MongoDB's storage layer has two internal views of the data set: the private view, used to write to the journal files, and the shared view, used to write to the data files:

1. MongoDB first applies write operations to the private view.
2. MongoDB then applies the changes in the private view to the on-disk journal files in the journal directory roughly every 100 milliseconds. MongoDB records the write operations to the on-disk journal files in batches called *group commits*. Grouping the commits help minimize the performance impact of journaling since these commits must block all writers during the commit. Writes to the journal are atomic, ensuring the consistency of the on-disk journal files.
3. Upon a journal commit, MongoDB applies the changes from the journal to the shared view.
4. Finally, MongoDB applies the changes in the shared view to the data files. More precisely, at default intervals of 60 seconds, MongoDB asks the operating system to flush the shared view to the data files.

If the mongod instance were to crash without having applied the writes to the data files, the journal could replay the writes to the shared view for eventual write to the data files.

However, if MongoDB takes too long to write changes to the journal, then, in the event that a mongod instance crashes, not all the changes will be available in the journal to enable a complete recovery. In such situations therefore, data loss is inevitable. This is why, it is imperative that administrators detect potential delays in journaling, isolate its root-cause, and resolve the bottleneck before it impacts data recovery. This is where the **Mongo Journaling Statistics** test helps!

This test monitors writes to the journals and data files, reports the time taken for these writes, and proactively alerts administrators to delays in the journaling process. Using this test, administrators can also figure out at what step of the process the delay may have occurred - when writing to the journal? when committing the writes? or when writing to the data files? This way, the test enables administrators to accurately diagnose the root-cause of journaling slowness, so that they can fix it before it results in data loss.

Note:

This test reports metrics only for those MongoDB instances that use the MMAPv1 storage engine.

Target of the test : A MongoDB server

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for the Mongo database server being monitored.

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed.
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens
Database Name	The test connects to a specific Mongo database to run API commands and pull metrics of interest. Specify the name of this database here. The default value of this parameter is *admin*.
Username and Password	The eG agent has to be configured with the credentials of a user who has the required privileges to monitor the target MongoDB instance, if the MongoDB instance is access control enabled. To know how to create such a user, refer to Section **2.1**. If the target MongoDB instance is not access control enabled, then, specify *none* against the Username and Password parameters.
Confirm Password	Confirm the password by retyping it here.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Number of commits	Indicates the number of transactions written to the journal during the last time a group commit was performed on the journal.	Number	
Data writes to journal	Indicates the amount of data written to the journal during the last time a group commit was performed on the journal.	MB	
Preparing time to write to the journal	Indicates the amount of time spent preparing to write to the journal.	Millisecs	Smaller values indicate better journal performance.
Average write time to journal	The amount of time, in milliseconds, spent	Millisecs	Ideally, the value of this measure should be low. A consistent increase in this

Measurement	Description	Measurement Unit	Interpretation
	actually writing to the journal.		value is a cause for concern, as it indicates a bottleneck when writing to the journal. File system speeds and device interfaces increase the value of this measure.
Compression ratio of written data to journal	Indicates the compression ratio of the data written to the journal	Percent	A high value is desired for this measure.
Data writes to data file	Indicates the amount of data written from journal to data files during the last group commit.	MB	
Average write time to data file	Indicates the amount of time spent writing to data files after journaling.	Millsecs	Ideally, the value of this measure should be low. A consistent increase in this value is a cause for concern, as it indicates a bottleneck when writing to the data files. File system speeds and device interfaces increase the value of this measure.
Early commits	Indicates the number of times MongoDB requested a commit before the scheduled journal group commit interval.	Number	For the MMAPv1 storage engine, you can set the group commit interval using the *–journalCommitInterval* command line option. The allowed range is 2 to 300 milliseconds. Lower values increase the durability of the journal at the expense of disk performance. Use the value of this measure ensure that your journal group commit interval is not too long for your deployment.
Average time spent for a commit	Indicates the amount of time spent for commits.	Millisecs	
Number of commits during write lock	Indicates the count of the commits that occurred while a write lock was held.	Number	Commits in a write lock indicate a MongoDB node under a heavy write load and call for further diagnosis.
Average commits	Indicates the amount of	Millisecs	

Measurement	Description	Measurement Unit	Interpretation
time during write lock	time spent for commits that occurred while a write lock was held.		

4.4.5 Mongo Replication Status Test

A replica set is a group of mongod instances that maintain the same data set. A replica set contains several data bearing nodes and optionally one arbiter node. Of the data bearing nodes, one and only one member is deemed the primary node, while the other nodes are deemed secondary nodes. The primary node receives all write operations. A replica set can have only one primary capable of confirming writes with { w: "majority" } write concern. The primary records all changes to its data sets in its operation log, i.e. oplog. Secondary members replicate this log and apply the operations to their data sets.

When a primary does not communicate with the other members of the set for more than 10 seconds, an eligible secondary will hold an election to elect itself the new primary. The first secondary to hold an election and receive a majority of the members' votes becomes primary. When such a switch happens, it is only natural that administrators want to be notified of it, as they may then need to troubleshoot the failure of the primary and bring it back up.

Also, for the failover to be successful, the current primary should be able to access at least the majority of members in the replica set. If not, then the primary will step down and become a secondary, rendering the replica set unable to accept any further writes. To avoid this, administrators should be able to instantly detect the inaccessibility or unavailability of any member in the replica set, and quickly restore it to normalcy.

The Mongo Replication Status test enables administrators to achieve these goals!

Using this test, administrators can keep track of the status of each member node and be promptly alerted if that node stops running or switches to an abnormal state. Furthermore, the test keeps tabs on heartbeats received from each member node, pinpoints the member node from which heartbeats were not received for a long time, and thus proactively alerts administrators to the potential non-availability of a node. This way, administrators will be able to ensure that quorum is maintained in the replica set and the primary node is able to communicate with each member node in the replica set. The test also notifies administrators if the primary of the replica set has switched. Detailed diagnostics reveals when the switch occurred and what is the current primary.

Target of the test : A MongoDB server

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for each member node in a replica set.

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed.
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens.
Database Name	The test connects to a specific Mongo database to run API commands and pull metrics of interest. Specify the name of this database here. The default value of this parameter is *admin*.
Username and Password	The eG agent has to be configured with the credentials of a user who has the required privileges to monitor the target MongoDB instance, if the MongoDB instance is access control enabled. To know how to create such a user, refer to Section **2.1** . If the target MongoDB instance is not access control enabled, then, specify *none* against the Username and Password parameters.
Confirm Password	Confirm the password by retyping it here.
Detailed Diagnosis	To make diagnosis more efficient and accurate, the eG Enterprise suite embeds an optional detailed diagnostic capability. With this capability, the eG agents can be configured to run detailed, more elaborate tests as and when specific problems are detected. To enable the detailed diagnosis capability of this test for a particular server, choose the **On** option. To disable the capability, click on the **Off** option. The option to selectively enable/disable the detailed diagnosis capability will be available only if the following conditions are fulfilled: • The eG manager license should allow the detailed diagnosis capability • Both the normal and abnormal frequencies configured for the detailed diagnosis measures should not be 0.

Measurements made by the test

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td>Is node running?</td><td>Indicates whether this node is currently running or not.</td><td></td><td>The values that this measure can report and their corresponding numeric values are listed in the table below:
<table><tr><th>Measure Value</th><th>Numeric Value</th></tr><tr><td>Yes</td><td>1</td></tr><tr><td>No</td><td>0</td></tr></table>
Note:
By default, the measure reports only the Measure Values provided in the table above to indicate whether/not a node is running. In the graph of this measure however, the same is indicated using the numeric equivalents only.
If any replicated node is down, then it will not be responding to requests from any available node in the replica set.</td></tr>
<tr><td>Current status</td><td>Indicates the current status of this node.</td><td></td><td>The values that this measure can report and their corresponding numeric values are listed in the table below:
<table><tr><th>Measure Value</th><th>Description</th><th>Numeric Value</th></tr><tr><td>STARTUP</td><td>Not yet an active member of any set. All members start up in this state. The mongod parses the</td><td>0</td></tr></table></td></tr>
</table>

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td></td><td></td><td>
<table>
<tr><th>Measure Value</th><th>Description</th><th>Numeric Value</th></tr>
<tr><td></td><td>replica set configuration document while in STARTUP.</td><td></td></tr>
<tr><td>PRIMARY</td><td>The member in state primary is the only member that can accept write operations. Eligible to vote.</td><td>1</td></tr>
<tr><td>SECONDARY</td><td>A member in state secondary is replicating the data store. Eligible to vote.</td><td>2</td></tr>
<tr><td>RECOVERING</td><td>Members either perform startup self-checks, or transition from completing a rollback or resync. Eligible to vote.</td><td>3</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td></td><td></td><td>
<table>
<tr><th>Measure Value</th><th>Description</th><th>Numeric Value</th></tr>
<tr><td>STARTUP2</td><td>The member has joined the set and is running an initial sync.</td><td>5</td></tr>
<tr><td>UNKNOWN</td><td>The member’s state, as seen from another member of the set, is not yet known.</td><td>6</td></tr>
<tr><td>ARBITER</td><td>Arbiters do not replicate data and exist solely to participate in elections.

An arbiter does not have a copy of data set and cannot become a primary. Replica sets may have arbiters to add a vote in elections for primary. Arbiters</td><td>7</td></tr>
</table>
</td></tr>
</table>

Measurement	Description	Measurement Unit	Interpretation

Measure Value	Description	Numeric Value
	always have exactly 1 election vote, and thus allow replica sets to have an uneven number of voting members without the overhead of an additional member that replicates data.	
DOWN	The member, as seen from another member of the set, is unreachable.	8
ROLLBACK	This member is actively performing a rollback. A rollback reverts write operations on a former primary when the member rejoins its replica set	9

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td></td><td></td><td>
<table>
<tr><th>Measure Value</th><th>Description</th><th>Numeric Value</th></tr>
<tr><td></td><td>after a failover.
Data is not available for reads.</td><td></td></tr>
<tr><td>REMOVE</td><td>This member was once in a replica set but was subsequently removed.</td><td>10</td></tr>
</table>
Note:
By default, the measure reports only the Measure Values provided in the table above to indicate the current status of a node. In the graph of this measure however, the same is indicated using the numeric equivalents only.
</td></tr>
<tr><td>Is primary switched?</td><td>Indicates whether/not a failover occurred from primary to secondary.</td><td></td><td>
The values that this measure can report and their corresponding numeric values are listed in the table below:
<table>
<tr><th>Measure Value</th><th>Numeric Value</th></tr>
<tr><td>Yes</td><td>1</td></tr>
<tr><td>No</td><td>0</td></tr>
</table>
Note:
By default, the measure reports only the Measure Values provided in the table above to indicate whether/not a failover has occurred. In the graph of this measure however, the same is indicated using the
</td></tr>
</table>

<table>
<tr><th>Measurement</th><th>Description</th><th>Measurement Unit</th><th>Interpretation</th></tr>
<tr><td></td><td></td><td></td><td>numeric equivalents only.

The detailed diagnosis of this measure reveals the current primary, the previous primary, and when the failover occurred. With the help of these details, administrators will be able to rapidly identify which primary failed, and which secondary node has now been elected as the primary.</td></tr>
<tr><td>Uptime</td><td>Indicates the total uptime of this member node.</td><td>Minutes</td><td>A very low uptime could imply that the member node restarted recently.</td></tr>
<tr><td>Last heartbeat time</td><td>Indicates the time that has elapsed since the last heartbeat was received from this node.</td><td>Secs</td><td>If the value of this measure increases consistently, it could imply a prolonged non-availability of the member node.</td></tr>
<tr><td>Is primary?</td><td>Indicates whether/not this node is the primary node of the replica set.</td><td></td><td>The values that this measure can report and their corresponding numeric values are listed in the table below:

<table><tr><th>Measure Value</th><th>Numeric Value</th></tr><tr><td>Yes</td><td>1</td></tr><tr><td>No</td><td>0</td></tr></table>
Note:

By default, the measure reports only the Measure Values provided in the table above to indicate whether/not a node is the primary node. In the graph of this measure however, the same is indicated using the numeric equivalents only.</td></tr>
</table>

4.4.6 Mongo Transactions Test

Rollbacks are expensive operations and will have to be avoided at all costs. Using the Mongo Transactions test, administrators can be instantly alerted if more than a permissible number of rollbacks are happening on a target MongoDB server. In addition, the test monitors transaction

checkpoints. A checkpoint acts as a recovery point for an operation. When a transaction attempts to modify the data in a data file, MongoDB presents an in-memory snapshot/copy of that data and allows changes to be made to that data. While writing the data in the snapshots to the disk, MongoDB creates restoration points between the snapshots, called checkpoints. If these checkpoints are not created frequently, it only means that data is not written to the disk regularly; this increases the risk of data loss during recovery. To avoid this, administrators can use the **Mongo Transactions** test to monitor the time taken for creating the most recent checkpoint and thus, quickly detect delays in check point creation or writing to disk.

Target of the test : A MongoDB server

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for the Mongo database server being monitored.

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed.
Host	The host for which the test is to be configured.
Port	The port number at which the specified host listens.
Database Name	The test connects to a specific Mongo database to run API commands and pull metrics of interest. Specify the name of this database here. The default value of this parameter is *admin*.
Username and Password	The eG agent has to be configured with the credentials of a user who has the required privileges to monitor the target MongoDB instance, if the MongoDB instance is access control enabled. To know how to create such a user, refer to Section **2.1** . If the target MongoDB instance is not access control enabled, then, specify *none* against the Username and Password parameters.
Confirm Password	Confirm the password by retyping it here.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Initiated	Indicates the rate at which transactions are initiated.	Transactions/Sec	

Measurement	Description	Measurement Unit	Interpretation
Committed	Indicates the rate at which transactions are committed.	Transactions/Sec	
Rolledback	Indicates the rate at which transactions are rolledback.	Transactions/Sec	Ideally, the value of this measure should be very low.
Rollback rate	Indicates the percentage of initiated transactions that are rolled back.	Percent	A value close to 100% is a serious cause for concern, as it implies that almost all transactions have been rolledback. Rollbacks consume processing overheads excessively and reduce database performance.
Recent checkpoint duration	Indicates the amount of time taken to create the most recent checkpoint.	Millisecs	An increase in this value under steady write load may indicate saturation on the I/O subsystem.

4.4.7 Mongo Asserts Test

Assertions are errors that may occur on a MongoDB server from time to time. Administrators need to promptly capture these errors and understand their nature, so that they can rapidly resolve them and prevent any negative impact on database performance. This is where the Mongo Asserts test helps. This test captures and reports the rate at which each type of assertion occurs on a monitored MongoDB server. Using these metrics, administrators can instantly capture the asserts/errors, quickly determine from the assert type why the error might have occurred, and then, rapidly initiate the appropriate corrective measures.

Target of the test : A MongoDB server

Agent deploying the test : An internal/remote agent

Outputs of the test : One set of results for the Mongo database server being monitored.

Configurable parameters for the test

Parameter	Description
Test period	How often should the test be executed.
Host	The host for which the test is to be configured.

Parameter	Description
Port	The port number at which the specified host listens
Database Name	The test connects to a specific Mongo database to run API commands and pull metrics of interest. Specify the name of this database here. The default value of this parameter is *admin*.
Username and Password	The eG agent has to be configured with the credentials of a user who has the required privileges to monitor the target MongoDB instance, if the MongoDB instance is access control enabled. To know how to create such a user, refer to Section **2.1** . If the target MongoDB instance is not access control enabled, then, specify *none* against the Username and Password parameters.
Confirm Password	Confirm the password by retyping it here.

Measurements made by the test

Measurement	Description	Measurement Unit	Interpretation
Regular asserts	Defines the rate at which regular asserts occurred on the monitored MongoDB server.	Asserts/Sec	Regular asserts are per-operation variants (eg., "unexpected failure while reading a BSON document"). Ideally, the value of this measure should be 0. If a non-zero value is reported, check the log file for more information.
Warning asserts	Indicates the rate at which warning asserts occurred on the monitored MongoDB server.	Asserts/Sec	Warning asserts are not as serious as errors. They just indicate conditions that might be worth checking like too low ulimit or readahead. Ideally, the value of this measure should be 0. If a non-zero value is reported, check the log file for more information.
Message asserts	Indicates the rate at which message asserts occurred on the monitored MongoDB server.	Asserts/Sec	Message asserts indicate internal server exceptions. Ideally, the value of this measure should be 0. If a non-zero value is

Measurement	Description	Measurement Unit	Interpretation
			reported, check the log file for more information.
User asserts	Indicates the rate at which user asserts occurred on the monitored MongoDB server.	Asserts/Sec	User asserts are triggered as the result of user operations or commands generating an error like a full disk space, a duplicate key exception, or write errors (e.g. insert not properly formatted, or no access right). These errors are returned to the client so most of them won't be logged into the mongod logs. However you should investigate potential problems with your application or deployment. Both regular and user asserts will result in the corresponding operation failing.
Rollover asserts	Indicates the rate at which rollover alerts occurred on the target MongoDB server.	Asserts/Sec	The number of times that the rollover counters have rolled over since the last time the MongoDB process started. The counters will rollover to zero after 2^{30} assertions. Use this value to provide context to the other values reported by the test.

5

Chapter 5: Conclusion

This document has described in detail the monitoring paradigm used and the measurement capabilities of eG Enterprise with respect to Mongo Database server. For details of how to administer and use eG Enterprise, refer to the user manuals.

We will be adding new measurement capabilities into the future versions of eG Enterprise. If you can identify new capabilities that you would like us to incorporate in the eG Enterprise suite of products, please contact support@eginnovations.com. We look forward to your support and cooperation. Any feedback regarding this manual or any other aspects of eG Enterprise can be forwarded to feedback@eginnovations.com.

About eG Innovations

eG Innovations provides intelligent performance management solutions that automate and dramatically accelerate the discovery, diagnosis, and resolution of IT performance issues in on-premises, cloud and hybrid environments. Where traditional monitoring tools often fail to provide insight into the performance drivers of business services and user experience, eG Innovations provides total performance visibility across every layer and every tier of the IT infrastructure that supports the business service chain. From desktops to applications, from servers to network and storage, from virtualization to cloud, eG Innovations helps companies proactively discover, instantly diagnose, and rapidly resolve even the most challenging performance and user experience issues.

eG Innovations is dedicated to helping businesses across the globe transform IT service delivery into a competitive advantage and a center for productivity, growth and profit. Many of the world's largest businesses use eG Enterprise to enhance IT service performance, increase operational efficiency, ensure IT effectiveness and deliver on the ROI promise of transformational IT investments across physical, virtual and cloud environments.

To learn more visit www.eginnovations.com.

Contact Us

For support queries, email support@eginnovations.com.

To contact eG Innovations sales team, email sales@eginnovations.com.