Analysing Replication Lag of MongoDB

Archana P Saipriya L Vishnu S Murali Dinkar Sitaram

PES1201701543 PES1201701565 PES1201701624 dinkar.sitaram@gmail.com

Computer Science, Computer Science, Computer Science, CCBD,

PES University PES University PES University PES University

***Abstract-*** High availability[[1]](#footnote-0) is an essential characteristic of any database as a service. Mongodb achieves the same with the concept of Replica sets that consist of one primary and multiple secondary nodes for a database. Replication offers increased read capacity and tolerance to single node failure.

Replication lag is the time difference between the instant at which an operation was performed on the primary and the instant at which it was replicated on a secondary. A possible delay could result in reading from a secondary that hasn’t been updated. Such an error could have disastrous consequences and needs to be tackled. In this paper, driven by the motive of analysing this lag as a factor of mongodb’s performance, replication lag has been studied by varying parameters. The data obtained from multiple runs were used to predict this lag. We will be observing how write concerns and journaling affect the performance and replication lag of the database.Hence predicting the replication lag using various Machine Learning techniques is also discussed.

# Introduction[[2]](#footnote-1)

The field of WAN data replication covers the problems and solutions for distributing and replicating data across wide-area networks. Concentrating on databases alone, a wide-area database is defined as a collection of multiple, logically interrelated databases distributed and possibly replicated across sites that are connected through a wide-area network.

The characteristic feature is that data are spread across sites that are separated through wide-area links. Unlike links in local-area networks, the quality of communication through wide-area links is relatively poor. Links are subject to latencies of tens to thousands of milliseconds, there are often severe bandwidth restrictions, and connections between sites are much less reliable.

In principle, WAN data replication also covers the distribution and replication of plain files. These issues are traditionally handled by wide-area distributed file systems

Mongodb is a popular document oriented database, known for its high scalability and flexibility. It is a NoSQL database in which data is stored in the form of collections and documents. The concept of a record is implemented as a document in Mongodb, and consists of key:value pairs.

A collection is a group of documents.Documents within a collection can have different fields. Typically, all documents in a collection have a similar or related purpose.

Mongodb owes its popularity to its numerous advantages.Being a NoSQL database, Mongodb allows for dynamic schema and for data to be stored in an unstructured format.Storing records in tables as done in relational databases is not a requirement here, which makes data access simpler, with no complex joins.Performance tuning is also made simpler.

Easy installation, dynamic queries and its support for sharding are some other features of mongodb that make it one of the most widely used NoSQL databases today. Mongodb provides for higher availability of data through Replication. A replica set in mongodb can be thought of as a set of copies of one database. A group of mongod instances host the same data set. All the writes happen onto the primary instance. Operations done on the primary are recorded in an oplog. Each secondary member maintains a copy of the source oplog which is used to replicate each operation , onto the secondary.

Consistency amongst the members of a replica set is crucial to any application’s functioning. If a secondary is accessed before it is updated, it could lead to erroneous results which could have huge consequences.Replication lag, or the delay in updating a secondary member, is a major cause for inconsistency amongst the members of the replica set.

The factors that influence the lag hence form a crucial area of research for database performance.

This paper was hence developed with the motive of analysing the lag and trying to find an approach of predicting it, to be able to tackle its undesired consequences.

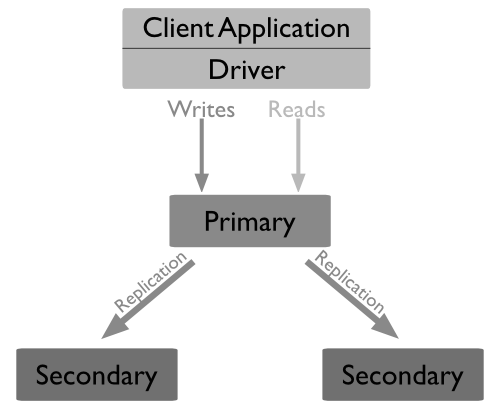
# Related Work

Comparison of other papers that have studied replication lag. Reference the earlier paper by Adarsh and say that here, the difference is that we are studying replication lag in the presence of WAN delays. Also, we are comparing Cassandra and MongoDB. Cassandra has the reputation of having high write throughput – is this true and why/

# Background

# **MongoDB Architecture**

MongoDB uses a document oriented data model and a non-structured query language. Unlike RDBMS (relational database) the basic block of MongoDB is in the form of key-value pairs.It is also unaffected by SQL injection and has dynamic schema. One of the main features of Mongodb is that it supports Replication, shown in Fig.1.

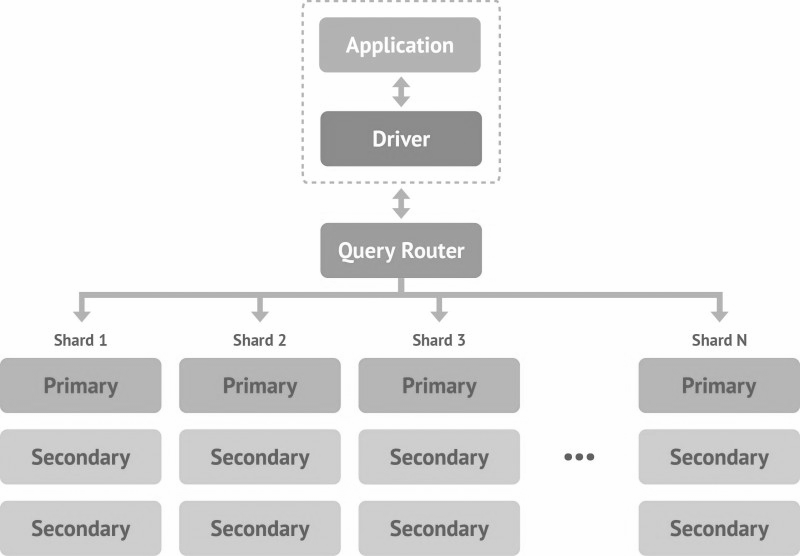
****

*Fig.1: Replication in MongoDB*

Replication is the process of syncing data across multiple servers so that there exists multiple copies of data across different database servers. This makes it easier to recover from the loss of any single database server. It also increases data availability, provides redundancy and read potential as

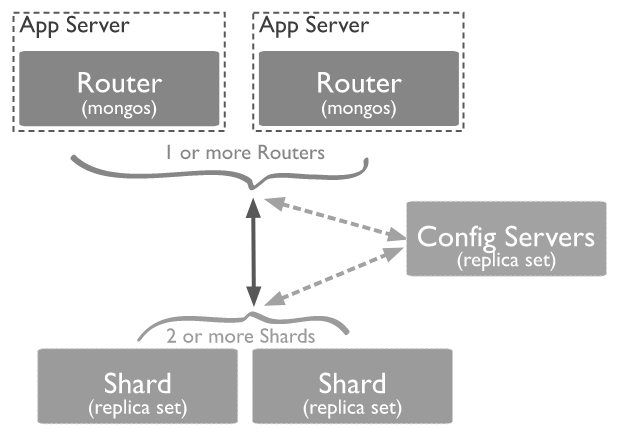
users can send the operations to different servers.

Another aspect of mongoDB is auto data sharding, Fig.2. Sharding is a routine for distributing data across multiple machines. It uses sharding to brace deployments with very large datasets and high throughput operations. Each sharded cluster consists of a shard, mongos and config servers. Each shard consists of a replica set and the mongos provides an interface between user application and the shard cluster. Config servers store the metadata and the configuration settings required by the cluster. Fig 3. depicts the interaction between sharded clusters.



*Fig 2. Sharding in mongoDB*

Data Synchronization is the process of establishing data consistency between two or more databases, with subsequent continuous updates to maintain consistency.



*Fig 3.: Interaction between sharded cluster components*

**Synchronization of unordered data** is an attempt to compute the symmetric difference A+B=(A-B)U(B-A) between two remote sets A and B of b-bit numbers.

The above can be made possible using the following techniques:

1. **Wholesale transfer:** In this case all data is transferred to one host for a local comparison.
2. **Timestamp synchronization:** In this case all changes to the data are marked with timestamps. Synchronization proceeds by transferring all data with a timestamp later than the previous synchronization.
3. **Mathematical synchronization:**In this case data are treated as mathematical objects and synchronization corresponds to a mathematical process.
4. **Synchronization of ordered data** is the process of reducing the edit distance between two sets or strings A and B.

**Synchronisation in MongoDB**

MongoDB uses replica set Data Synchronisation.Each replica set consists of a primary node and multiple secondary nodes which are constantly synced.

1. **Initial Sync:** Initial sync copies all the data from one member of the replica set to another member. MongoDB allows two ways of performing an initial sync:

* By restarting the mongoDb instance with an empty directory and performing initial sync.
* To restart the machine with a copy of the data from another stable replica set.

1. **Replication:** Replication is the process of synchronizing data across multiple servers. Replication makes data redundant and hence makes data more available across multiple servers. Replication protects a database from single point failure.MongoDB achieves replication by the use of replica set, a group of mongod instances that host the same data set. In a replica, the primary node receives all write operations and by default, all the read operations as well. All other instances, such as secondaries, apply operations from the primary so that they have the same data set. Replica sets can have only one primary node each.

**Resynchronization in MongoDb**

A replica becomes unresponsive when its replication process falls so far behind that the primary overwrites oplog entries the member has not yet replicated. Hence resynchronization is necessary.

In order to maintain up-to-date copies of the data, the secondary members replicate data from other members of the data set.

**Replication Lag**

Replication lag is the delay between the time when an operation gets applied on the primary and on the secondary.

Possible causes of replication lag include:

1. **Concurrency**: multiple users running bulk operations on the same database causes significant replication lag
2. **Disk Throughput**:If the file system and disk device on the secondary is unable to flush data to disk as quickly as the primary, then the secondary will start lagging behind the primary.
3. **Write concerns and Journaling**:Bulky load operations and immediate flushing to the disk will require a large number of writes on the primary because of which secondaries may not be able to read the oplog as fast enough to keep up with the changes.
4. **Network latency**: Latency caused due to the internetwork between the servers also causes replication lag.

**Write concerns:**

The w option requests acknowledgment that the write operation has propagated to a specified number of [mongod](https://docs.mongodb.com/manual/reference/program/mongod/#bin.mongod) instances or to [mongod](https://docs.mongodb.com/manual/reference/program/mongod/#bin.mongod) instances with specified tags.

***w=1*** Requests acknowledgment that the write operation has propagated to the standalone [mongod](https://docs.mongodb.com/manual/reference/program/mongod/#bin.mongod) or the primary in a replica set. w: 1 is the default write concern for MongoDB.

***w=0*** Requests no acknowledgment of the write operation. However, w: 0 may return information about socket exceptions and networking errors to the application.

***w greater than 1*** requires acknowledgment from the primary and as many additional data-bearing secondaries to meet the specified write concern.

Synchronization time is the replication lag which can be calculated using db.printreplicationinfo() command on a secondary.

Additionally, we can also view recent and earlier replication lag using MongoDB Monitoring Service(MMS) which produces the results in the form of graphs. A glance at the charts can be used as an alert for excessive lag.

# **Workload**

**Description of YCSB (Yahoo Cloud Service Benchmark)**

YCSB or Yahoo! Cloud Serving Benchmark is a java based open source software that is used to evaluate the performance of various NoSQL databases, developed by Yahoo! labs.

Parameters: There are separate load and run commands for loading the database and running the ycsb programs respectively. The load command takes a ‘workload’ parameter, that indicates which workload is loaded onto the database, and a recordcount parameter. Run command parameters include operation count(number of operations per second), target(target number of operations per second), number of threads(which is single by default but can be altered to increase the amount of load offered to the database).

An output file includes numerous parameters indicating the performance of the database for a particular workload:

1. Latency- time taken for the database to respond to a single request
2. Throughput - number of requests the database responds to per unit of time
3. Runtime, number of operations are also output parameters common to all workloads.
4. Other parameter like update latency, insert latency, and read latency, depending on the workload.

The tool offers six core workloads which vary in the ratio of types of queries, and are as follows[[3]](#footnote-2):

|  |  |
| --- | --- |
| Workload A | Update heavy workload: 50/50% Mix of Reads/Writes |
| Workload B | Read mostly workload: 95/5% Mix of Reads/Writes |
| Workload C | Read-only: 100% reads |
| Workload D | Read the latest workload: More traffic on recent inserts |
| Workload E | Short ranges: Short range based queries. |
| Workload F | Read-modify-write: Read, modify and update existing records |

*Fig.4:Table of YCSB Workloads*

The tests are generally run by keeping the operation count and thread parameters as a constant1, and by varying other parameters like record count, etc. on different workloads to obtain the correspondingly varying parameters like throughput, read/update/insert latency. Graphs can be plotted to highlight the differences between various databases.

For a better picture of system performance over the cloud, these tests are automated and repeated at various points of the day. Performance characteristics can vary significantly throughout the day.

From the perspective of a generic, database-neutral, performance evaluation utility, YCSB is currently the de-facto comparative benchmark for NoSQL stores. It includes support for a wide range of database bindings and is commonly used to compare their performance for a set of desired workloads. Being open source and extensible, support for additional databases is regularly added.

##### III. Experimental Setup[[4]](#footnote-3)

In our research on MongoDB, we have used Yahoo ! Cloud Serving Benchmark(YCSB) for benchmarking our MongoDB database.YCSB populates the database with **recordcount of 5 million**  and **operation count of 3 million.** Our main goal is to find how the replication lag across multiple servers vary with different parameters.

Client and server instances were set up on Google Cloud Platform. The client would make the YCSB runs onto the server which had mongoDB installed.

The three members of the replica set were hosted on three different ports of the same server instance.

YCSB workloads were run from the client instance onto the primary instance with workload definitions as 30% reads and 70% writes. Four runs were made in total, with the following cases:

1. Write concern =1 and Journaling =True
2. Write concern =2 and Journaling =True
3. Write concern =1 and Journaling =False
4. Write concern =2 and Journaling =False

Observations were made on the replication lag by calling the function **db.serverStatus()**  using python scripts,with which cpu utilizations and memory management parameters were also recorded. The commands and parameters have been described in detail in the sections ahead.

The client-side parameter values for the YCSB run were kept constant as follows:

1. threads: 16
2. recordcount: 5 million
3. operation count : 3 million

The operation count was fixed by identifying the threshold opcount for the steady state of the CPU to be achieved.This was found to be around 2.5 million.

Script files were written to collect data using various commands, from the server instances, during each run.

The write concern and journaling options were also changed for each run on the server instance.

IV. Prediction

**Dataset Construction**

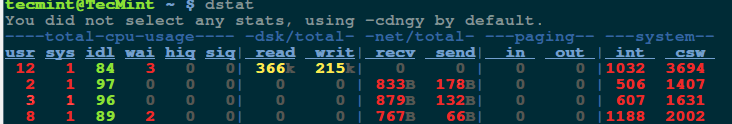
Various parameters were considered from different analysis commands to build an enormous dataset**.**

The commands used to obtain data are as follows:

***dstat***- versatile tool for generating system resource statistics,with details of CPU and disk utilisation. A sample output of the command has been shown in Fig.5.

Fields can be described as follows:

1. CPU stats: cpu usage by a user (usr) processes, system (sys) processes, as well as the number of idle (idl) and waiting (wai) processes, hard interrupt (hiq) and soft interrupt (siq).
2. Disk stats: total number of read (read) and write (writ) operations on disks.
3. Network stats: total amount of bytes received (recv) and sent (send) on network interfaces.
4. Paging stats: number of times information is copied into (in) and moved out (out) of memory.
5. System stats: number of interrupts (int) and context switches (csw).



*Fig 5. : Fields of dstat*

***mongostat-*** The mongostat utility provides a quick overview of the status of a currently running mongod or mongos instance. mongostat is functionally similar to the UNIX/Linux file system utility vmstat, but provides data regarding mongod and mongos instances.**mongostat** returns values that reflect the operations over a 1 second period.

Its output fields give us information about number of queries under each type, number of connections, number of page faults, etc.

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

The fields indicated are as follows:

1. **mongotop.total** -- Provides the total amount of time that this mongod spent operating on this namespace.
2. **mongotop.read** -- Provides the amount of time that this mongod spent performing read operations on this namespace.
3. **mongotop.write** -- Provides the amount of time that this mongod spent performing write operations on this namespace.
4. **mongotop.<timestamp>** -- Provides a time stamp for the returned data.

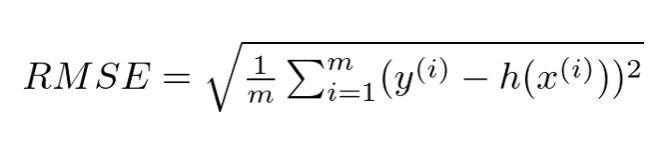
The various parameters obtained were further reduced by identifying strongly correlated features with the help of a correlation matrix as shown in *Fig 6*.

The columns were put together into one CSV file. The mapping of various outputs was done with timestamps.

**Technique used for Prediction:**

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node has two or more branches. Leaf node represents a classification or decision(used for regression). The topmost decision node in a tree which corresponds to the best predictor (most important feature)is called a root node. Decision trees can handle both categorical and numerical data. They are used for classification and regression problems.

*BasicWorking*: Our Node class represents one decision point in our model. Each division within the model has 2 possible outcomes for finding a solution—go to the left or go to the right. That decision point also divides our data into two sets. We create our split such that it has as low standard deviation as possible. We find the split that minimizes the weighted averages of the standard deviations which is equivalent to minimizing Root Mean Squared Error. Hence the Cost function used is Root-Mean Square Error.



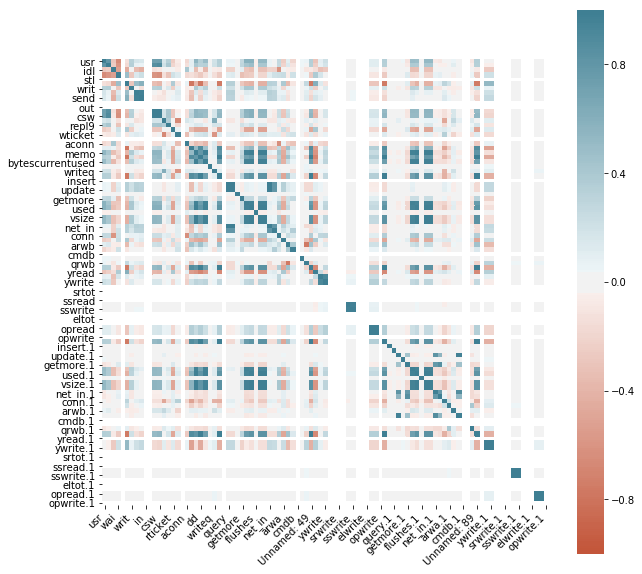
The property idxs stores indexes of the subset of the data that this Node is working with. The decision (prediction) is based on the value that Node holds. To make that prediction we’re just going to take the average of the data of the dependent variable for this Node. The method find-varsplit finds where should we split the data. We try to find a better feature to split on. If no such feature is found (we’re at a leaf node) we do nothing. Then we use the split value found by find-better-split, create the data for the left and right nodes and create each one using a subset of the data. We create our split such that it has as low standard deviation as possible. We find the split that minimizes the weighted averages of the standard deviations which is equivalent to minimizing RMSE. If we find a better split we store the following information: index of the variable, split score and value of the split. The score is a metric that tells us how effective the split was (note that leaf nodes do not have scores, so it will be infinity). The method find-score calculates a weighted average of the data. If the score is lower than the previous we have a better split. Note that the score is initially set to infinity to only leaf nodes and really shallow trees have a score of infinity.At the end we use the predict-row function to predict the values.

V. Results

The covariance matrix obtained has been shown in *Fig .6.*

Columns and their correlations were identified, and the decision tree was built using the independent features.

Average Replication lag was found for each combination of runs on the predicted values by the model with dataset obtained from both secondary instances.Table 3 summarises the results obtained.

 *Fig.6: Correlation matrix of the parameters used*

Figures and Tables

Root MeanSquare Error was found for each combination of runs on the predicted values by the model with dataset obtained from both secondary instances.Table 2 summarises the results obtained.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sl.no | write concern | journaling | instance 27018 | instance27019 |
| 1. | 1 | true | 3.7 secs | 8.5secs |
| 2. | 1 | false | 4.6secs | 3.6secs |
| 3. | 2 | true | 9secs | 19secs |
| 4. | 2 | false | 2.2secs | 6.4secs |

*Table 2 :Root Mean Square Error of the predicted data for each YCSB run*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sl.no | write concern | journaling | instance 27018 | instance27019 |
| 1. | 1 | true | 50 secs | 66.3secs |
| 2. | 1 | false | 26.8secs | 48secs |
| 3. | 2 | true | 82.8secs | 88.5secs |
| 4. | 2 | false | 42secs | 43secs |

*Table 3:Average Replication Lag for each YCSB run*

## **Conclusion**

From the above results we can conclude that as the write

concerns increase ,average replication lag increases and

average replication lag increases if journaling is made true,

then the average replication lag increases more.

Since the Root Mean Square error is very less compared to

the average replication lag, we can infer that the Decision

Tree Regressor gives reasonably good results on the dataset.

VI. Future scope

With this analysis we have considered a subset of parameters to predict the replication lag. However, there are numerous other parameters that could be checked for their influence on the same. Our study can further be extended to identifying all the impactful factors for the replication lag. Different models can be used to predict the lag. An ultimate goal would be to use the prediction to allocate resources to avoid the disastrous consequences of large delays and inconsistencies.

##### VII . Acknowledgement

We would like to extend our gratitude to our mentor, Dr. Dinkar Sitaraman, for his guidance and valuable suggestions throughout our course of the study.

We would also like to thank our ‘Big Data’ course professor, Dr. K V Subramaniam for his helpful way of teaching that gave us a good understanding of Big Data concepts.

This project has helped us learn a lot about the internals of a database as a service.

##### References

<https://docs.mongodb.com/manual/tutorial/deploy-replica-set/>

<https://docs.mongodb.com/manual/reference/write-concern/>

<https://docs.mongodb.com/manual/replication/>

<https://blog.serverdensity.com/mongodb-monitoring-mongostat/>

<https://www.tecmint.com/dstat-monitor-linux-server-performance-process-memory-network/>

1. Say we are studying replication lag in the context of WAN – one of the servers being in a different data center [↑](#footnote-ref-0)
2. Introduction should not talk about MongoDB. Introduction should talk about the problem - replication lag. First, why is replication lag important? Second, we are studying this in two contexts – (i) how does WAN latency affect replication lag, and maybe what is the best way to deal with replication lag in the context of WAN latency (ii) how is replication lag affected by the replication algorithm (Cassandra vs MongoDB). Move all the MongoDB stuff to a background section that is no more than ¾ page. [↑](#footnote-ref-1)
3. depends on application/ motive of analysis [↑](#footnote-ref-2)
4. Insert diagram here [↑](#footnote-ref-3)