A Comparison Between MongoDB and MySQL Document Store Considering Performance

Erik Andersson and Zacharias Berggren

Erik Andersson and Zacharias Berggren
VT 2017
Examensarbete, 15 hp
Supervisor: Kai-Florian Richter
Examiner: Pedher Johansson
Kandidatprogrammet i datavetenskap, 180 hp

Abstract

Databases are an important part of today's applications where data has to be stored and accessed quickly. One of the important criteria when choosing what database technology to use is performance, where you want the operations to be as fast as possible.

In April 2016 Oracle released a alternative way of working with MySQL as a document store. This creates an opportunity to compare it to MongoDB which is one of the most popular document store databases.

The comparison was performed by testing different operations on the databases and comparing the resulting time it took.

The result showed that MongoDB was faster in every test case for every operation.

Acknowledgements

Thanks to Kai-Florian Richter at the Department of Computing Science at Umeå University for valuable supervision.

Contents

1 Introduction

For quite some time relational databases such as MySQL and PostgreSQL have dominated the database market.[1]

NoSQL databases have existed since the late 1960s [5], but it was not until big companies like Google, Facebook and Amazon picked it up that it really got popular. ”NoSQL” refers to ”non SQL”, ”non relational”, or ”not only SQL”. It offers a different mechanic to store and retrieve data compared to tabular relation databases. Today many companies use NoSQL databases in their systems since it allows for a simple design and structure. [2]

This thesis will compare the two NoSQL databases MongoDB and MySQL Document Store. The primary focus of the comparison is performance as measured by execution time.

This comparison is interesting because MySQL Document Store is a new product on the market that can compete with the already existing NoSQL databases. MySQL is a big and known brand which could help their development of this technology.

1.1 Earlier Research

Since MySQL Document Store was released just a year ago, there have not been a lot of coverage of it. The closest research that exist is comparison of regular MySQL and MongoDB.

Fredriksson and Wester [8] tested MongoDB and MySQL against each other by testing the time needed for inserting and selecting different amounts of data. The insertion tests were performed both with and without something called multi-insert which involves sending all the insertions as a single operation to the database. They found MongoDB to perform better on the insertion tests except when inserting large amounts of images, and large amounts of data in multi-insert. On selecting MongoDB performed better in every single one of their tests.

Győrödi et.al [4] also compared MongoDB and MySQL by measuring execution time on different operations. They named four different basic operations that could be performed on any database; Insert, Select, Update, and Delete. In their tests MongoDB performed better in every case.

1.2 MySQL

MySQL is a relational database that stores data in a tabular structure and is grouped by tables. It uses SQL as language for communication with the database, which allows for a very powerful syntax to create both simple and complex queries to retrieve and structure data as you want. A relational database is, as the name implies, useful when the data you want to store in different tables has a relation between each other.

MySQL comes with with 8 different database engines which can be chosen based on different criteria that one might have. By default, MySQL uses InnoDB if no other engine is defined when creating a new database.

1.3 NoSQL

NoSQL is a grouping of several different technologies that are defined as any database technology that is not SQL such as Document Store or Key-Value Store. Both of the NoSQL technologies that this thesis focused on are Document Store.

1.3.1 Document Store

Document Store works by storing the data as JSON-objects in documents whose structure does not have to be predefined. As opposed relational databases these objects do not have to have a common structure and can also be nestled. The support of nestling data removes the need to join data the same way as it is done in relational databases, and can improve performance drastically. Each document can be compared to a traditional row in a relational database, and each property in the JSON-object as a column.

JSON stands for JavaScript Object Notation and is an open standard data format that uses a attribute-value structure. It is similar to XML both in structure and usage. An example JSON-object looks like this:

```
{
    "_id" : ObjectId("59144d21f69ebb81cac402b8"),
    "name" : "William",
    "age" : 28,
    "middle_name" : ["Henrik", "Kurt"],
    "family" : {
            "mother" : "Sara",
            "dad" : "Jens"
        }
}
```

The attributes is on the left side of the colon, and the values on the right side.

A Document Store is suited for non-relational data where objects can have different properties from each other.

Because of this way of working with data, Document Store databases has grown in popularity. Some large companies like Twitter and Google has adapted it, thus pushing its popularity and development.

1.4 MongoDB

MongoDB is part of the new wave of NoSQL database technologies. It was released in 2009 and is one of the most popular NoSQL databases today that uses Document Store. [1] MongoDB has tables like relational databases, but they are called collections. In the collection the documents are stored as rows, but as opposed to a relational database, each collection does not have any columns.

1.4.1 Example

item
{ "_id": "00000aac2a36e711f16cac9e174d5f20", "name": "Monitor 121467", "size": 48.81092784926295, "type": "Monitor", "width": 2560, "height": 1440, "quantity": 345, "interface": "S–Video", "universal": 1, "refreshRate": 144, "responseTime": 5 }
{ "_id": "00001e162a36e711f16cac9e174d5f20", "frequency": 1008, "memory": 24, "millProp": 0, "name": "GTX 99249", "quantity": 475, "speed": 1513.87, "type": "Graphics Card", "universal": 1, "watt": 439 }
{ "_id": "0000205f2a36e711f16cac9e174d5f20", "cores": 16, "name": "i35879", "quantity": 88, "speed": 3.83171, "tenKProp": 3, "type": "CPU", "universal": 1 }

To find the documents with the name ”GTX 99249”, in this case the middle item, you run:

```
db.item.find({"name": "GTX 99249"}).
```

which returns a JSON-array with the JSON-objects matching the name:

```
[
    {
        "_id": "00001e162a36e711f16cac9e174d5f20",
        "frequency": 1008,
        "memory": 24,
        "millProp": 0,
        "name": "GTX 99249",
        "quantity": 475,
        "speed": 1513.87,
        "type": "Graphics Card",
        "universal": 1,
        "watt": 439
    }
]
```

1.4.2 Shell

MongoDB has a shell called mongo. It is an interactive Javascript interface that can execute queries and perform more administrative operations. To execute a Javascript-file from the shell, you run:

```
mongo <host>:<port>/<database> <myjsfile.js>
```

1.4.3 Storage

MongoDB supports multiple storage engines to choose from depending on your workflow. The default engine is WiredTiger since MongoDB version 3.2.

WiredTiger stores the data in files represented as BSON (Binary JSON)[6]. MongoDB pre-allocates the data files sizes to avoid file system fragmentation.

Each file is a collection and its data. If the file becomes bigger than 2 GB, a new file is created where more data in the collection is stored.

1.5 MySQL Document Store

MySQL Document Store is a recent development by the MySQL developers Oracle and is their attempt at a NoSQL database. MySQL Document Store was first released with MySQL version 5.7.12 [7] and is thus not as mature as MongoDB in that regard. It still runs on InnoDB which is mature software.

To be able to use MySQL as a document store, you are required to install the plugin ”X Plugin” for MySQL. This plugin is used to implement a network protocol to communicate with MySQL, and is shipped with MySQL 5.7.12 and later.

MySQL Document Store keeps it tabular structure, but requires a column of type ”JSON” named ”doc” where it will save the JSON-data. Thanks to this, it allows mixing both document store and normal relational database data in the same table. This could be useful if you both have relational data and unstructured data.

1.5.1 Example

item	
_id	doc
00000aac2...	{ ”_id”: ”00000aac2a36e711f16cac9e174d5f20”, ”name”: ”Monitor 121467”, ”size”: 48.81092784926295, ”type”: ”Monitor”, ”width”: 2560, ”height”: 1440, ”quantity”: 345, ”interface”: ”S–Video”, ”universal”: 1, ”refreshRate”: 144, ”responseTime”: 5 }
00001e162...	{ ”_id”: ”00001e162a36e711f16cac9e174d5f20”, ”frequency”: 1008, ”memory”: 24, ”millProp”: 0, ”name”: ”GTX 99249”, ”quantity”: 475, ”speed”: 1513.87, ”type”: ”Graphics Card”, ”universal”: 1, ”watt”: 439 }
0000205f..	{ ”_id”: ”0000205f2a36e711f16cac9e174d5f20”, ”cores”: 16, ”name”: ”i35879”, ”quantity”: 88, ”speed”: 3.83171, ”tenKProp”: 3, ”type”: ”CPU”, ”universal”: 1 }

To find the first item, in this case a monitor with the name ”Monitor 121467”, you run:

```
db.item.find(”name = 'Monitor 121467'”)
```

which returns a JSON-array with the JSON-objects matching the name:

```
[
    {
        ”_id”: ”00000aac2a36e711f16cac9e174d5f20”,
        ”height”: 1440,
        ”interface”: ”S–Video”,
        ”name”: ”Monitor 121467”,
        ”quantity”: 345,
        ”refreshRate”: 144,
        ”responseTime”: 5,
        ”size”: 48.8109,
        ”type”: ”Monitor”,
        ”universal”: 1,
        ”width”: 2560
    }
]
```

1.5.2 Shell

MySQL Document Store has a shell called mysqlsh. Mysqlsh allows for execution of Javascript using the X DevAPI to work with document based data. It also allows for operations on the traditional relational data with Javascript.

A javascript-file cant be executed from shell with the following line:

```
mysqlsh --uri <user>:<password>@<host>/<database> --file <myjsfile.js>
```

1.5.3 Storage

InnoDB is one of the popular database engines for MySQL and comes with a lot of functionality. Since it is the default engine when no other is defined, MySQL Document Store also uses this engine by default.

By default InnoDB stores each table structure and data in separate files physically on the hard drive. This can also be changed by editing the option *innodb_file_per_table* to false.

- *tablename*.ibd contains all the data in the table
- *tablename*.frm contains the table structure and definitions

2 Method

2.1 Hardware & Software

The tests has been done on a computer running Ubuntu version 16.04 with the specs shown in table 1.

Table 1 Specification of hardware and software.

Name	Value
CPU	AMD FX 8350 (8 cores @ 4 GHz)
RAM	8 Gb
Harddrive	1 TB, 7200 RPM
MySQL Server	Version 5.7.18
MongoDB	Version 3.4.4

2.2 Test

The tests were done by executing JavaScript files that contained different queries to the database. To make sure the tests were not affected by network speed when timing the different operations, the scripts were executed locally on the server where the databases were hosted. All the scripts that performed these operations were also executed directly by the technologies' own client program, to remove any effect that libraries or other third-part application could cause the performance, and made it as fair as possible.

With MySQL Document Store we used `mysqlsh`, and with MongoDB we used `mongo`. Both of these clients execute Javascript-code, which made it easy to use the same scripts for both database technologies.

The measurement of performance between the database technologies was made by looking how fast each technology could finish the execution of four different types of queries:

- Inserting data
- Updating data
- Removing data
- Selecting data

These operations were chosen because they are the four main ones when operating on databases. They also have equivalent operations that were used in previous research.

The execution of the scripts were timed with the unix-command ”time”. Each test was performed 10 times in a row to get an average of of the time it took to execute the script. Timing a test-script looked like this:

```
time mongo localhost:27171/inventory insert1000.js
```

2.2.1 Type of data

Both databases were inventories where information about different computer components were stored. Each document in the collection was a computer component where the attributes describes the component. Therefore each different type had different attributes.

All the values that were saved in the different attribute fields were of the type string or integer. These types were chosen because they are usually the most common data types to store in this kind of database.

2.2.2 Data amount

The insertion tests combined all the different start sizes (0, 10^3, 10^4, 10^5, 10^6) with all the insertion amounts (10^3, 10^4, 10^5, 10^6). The other tests were not run with any empty-database cases because it would not make any sense. The test were also executed on the other operations on all of the data in the database or only one element. For example: fill database with 10^3 and remove all elements, fill database with 10^3 and remove one element.

The tests were executed on the databases when there already were different amounts of data present. This was done to see how it affected the performance. There were five levels of data:

- 0 documents (empty database)
- 10^3 documents
- 10^4 documents
- 10^5 documents
- 10^6 documents

2.2.3 Size

One aspect that is interesting is also how much space the data allocated on the hard drive. This was measured by using the technologies’ own tools.

MySQL stores information about each database in the ”information_scheme” database. By using a SELECT-query, the size of the table could be retrieved by selecting the ”data_length”-column. This column contains the size of the data, including index-size.

In MongoDB you can query the data size by using the ”stats()”-method on a collection. By summarizing the data-size and the index-size, you get the actual data-size.

2.2.4 Insertion

Insertion of data was performed in two ways: single-insert and multi-insert. Multi-insert is used when inserting a lot of data at the same time as a batch, i.e inserting multiple documents at the same time. Single-insert is usually used when inserting one single document.

2.2.5 Updating, Selecting, and Removing

These tests were performed on the previously defined database sizes of 1000 documents and above. Each database size was tested with the database sizes up to and including the size of the database. For example if we were testing on a database with 10^5 documents we would perform the test filtering on 10^3, 10^4, and 10^5 documents. We also made tests filtering on just a single document.

3 Results

The full results of our tests can be seen in appendix A.

3.1 Insert

In multi-insert MongoDB and MySQL Document Store performed quite equally with 10^3 and 10^4 objects. At 10^5 objects the difference between the two increased even more. At 10^6 objects MongoDB had a big increase in the time it took to insert it, but managed to do it in just under 30 seconds. MySQL Document Store could not complete this query, and instead returned error. See figure 1.

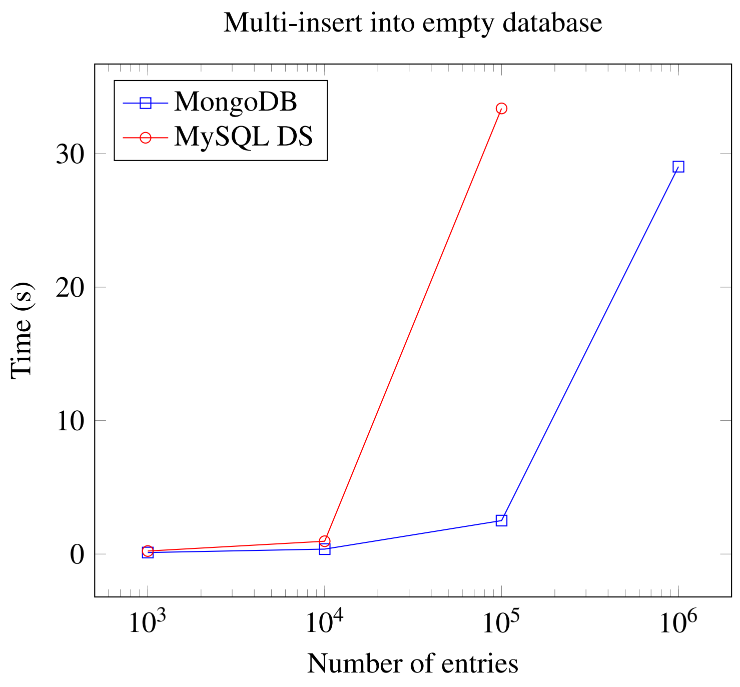

Figure 1: Time for inserting different amounts of data into an empty database.

In single-insert MongoDB could insert single documents much faster than MySQL Document Store. See figure 2.

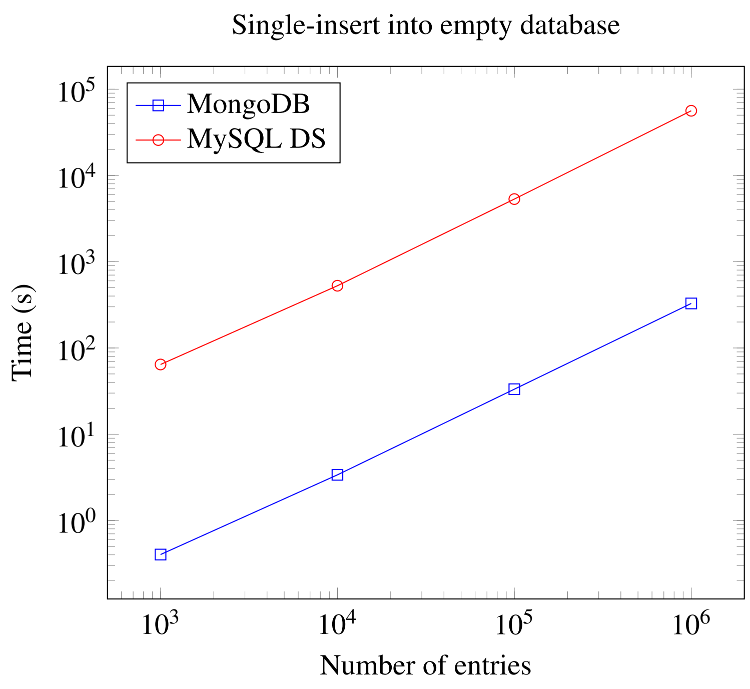

Figure 2: Time for inserting different amounts of data one at a time into an empty database.

In multi-insert with different amounts of data already existing in the database, MongoDB performed equally for every amount of pre-inserted data. See figure 3. In MySQL DS the results for different starting sizes affected the results more, as you can see in figure 4.

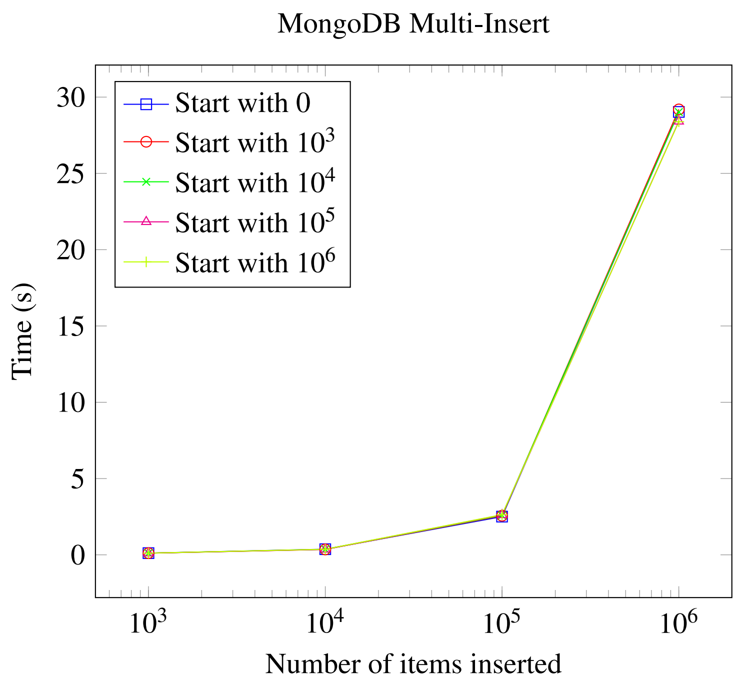

Figure 3: Time for inserting different amounts of data into databases already containing different amounts of data in MongoDB.

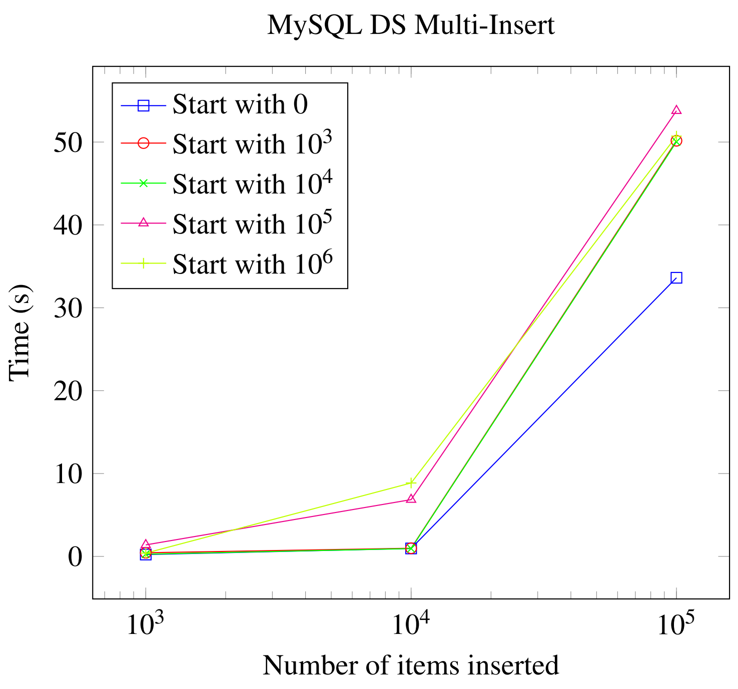

Figure 4: Time for inserting different amounts of data into databases already containing different amounts of data in MySQL DS.

3.2 Update

MongoDB performed better when updating all data in the database. This was specially noticeable when updating 10^5 documents. The tests that involved updating 10^6 documents in MySQL Document Store at once could not be completed. See figure 5.

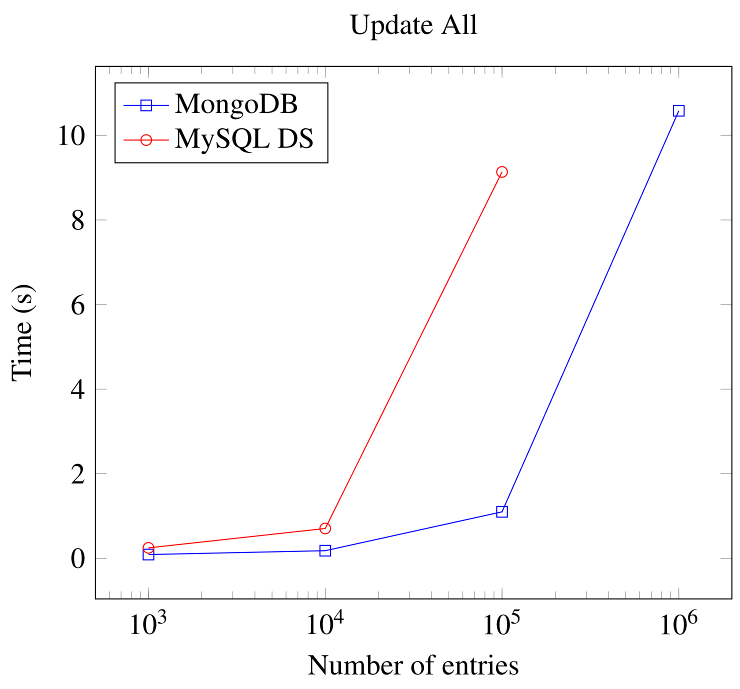

Figure 5: Time for updating every element for different database sizes.

When updating one document, both database technologies iterates through all the documents. MySQL Document Store managed to compete with MongoDB in the first three levels, but at 10^6 the difference got a lot more noticeable. See figure 6.

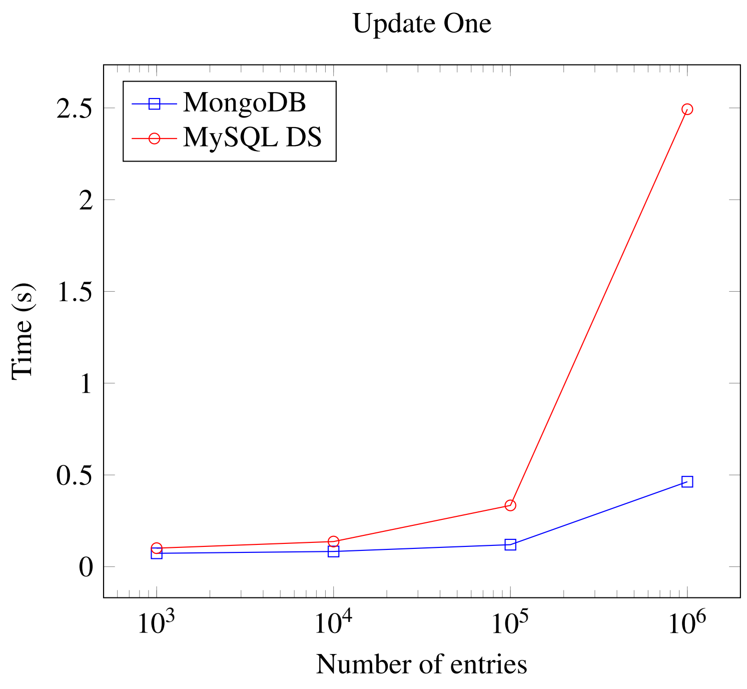

Figure 6: Time for updating one element in databases of different sizes.

When updating multiple documents with different amounts of data in the database, MySQL

Document Store performed poorly. While it could not update 10^6 documents at once, updating 10^5 documents when there were 10^5 documents in the collection was possible. See figure 7.

MongoDB performed better and could complete the test to update 10^6 documents in a database containing 10^6 documents. See figure 8.

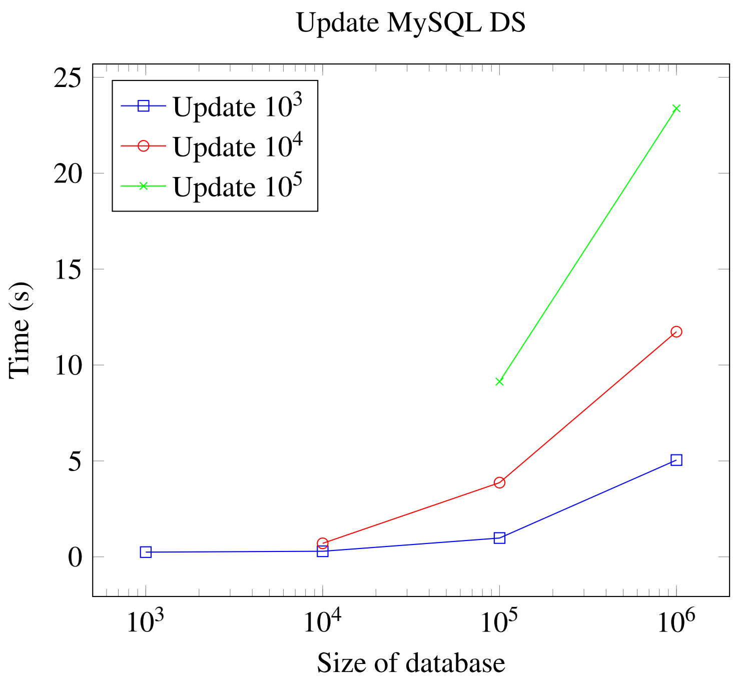

Figure 7: Time for updating different amounts in databases of different sizes in MySQL DS.

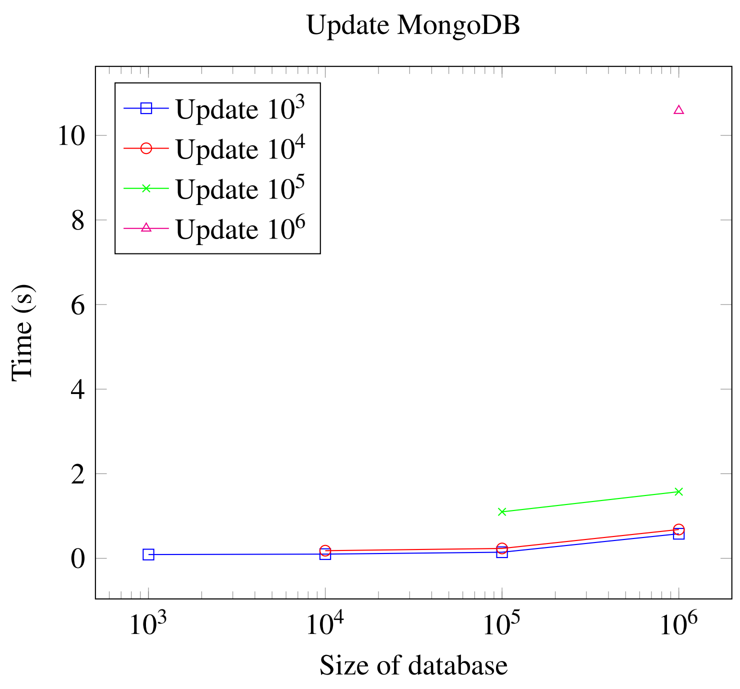

Figure 8: Time for updating different amounts in databases of different sizes in MongoDB.

3.3 Select

Selecting all documents in the database performed almost equally when it contained 10^3 and 10^4 documents. When it contained 10^5 documents MySQL Document Store increased more in time than MongoDB did, and at 10^6 documents the difference was even larger. See figure 9.

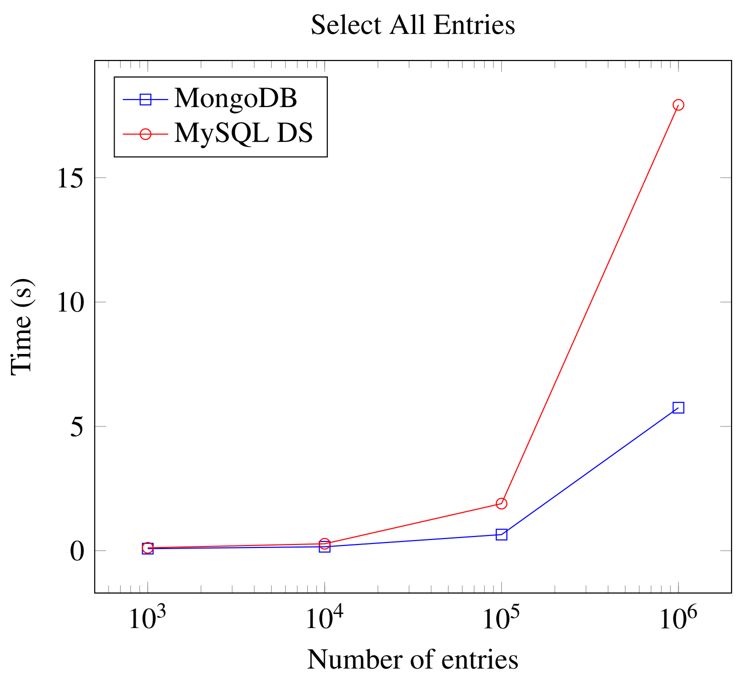

Figure 9: Time for selecting every document in databases of different sizes.

Selecting only one document the graph pattern was pretty identical to when selecting all documents. See figure 10.

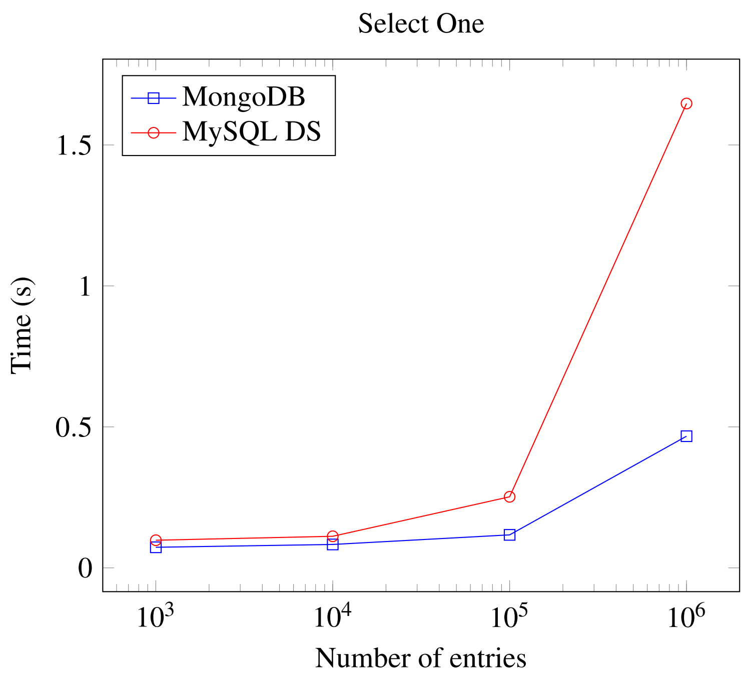

Figure 10: Time for selecting one document in databases of different sizes.

MongoDB beat MySQL Document Store when selecting multiple documents with different amounts of data in the database. Selecting 10^3-10^5 documents MySQL actually managed to keep up with MongoDB, but when selecting 10^6 documents, MySQL Document Store took three times as much time. See figure 12 & 11.

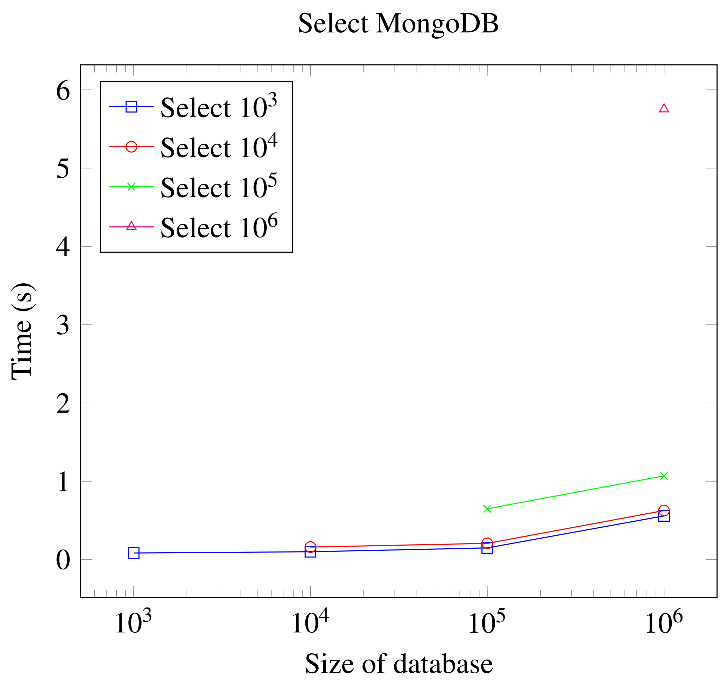

Figure 11: Time for selecting different amounts of data from databases of different sizes in MongoDB.

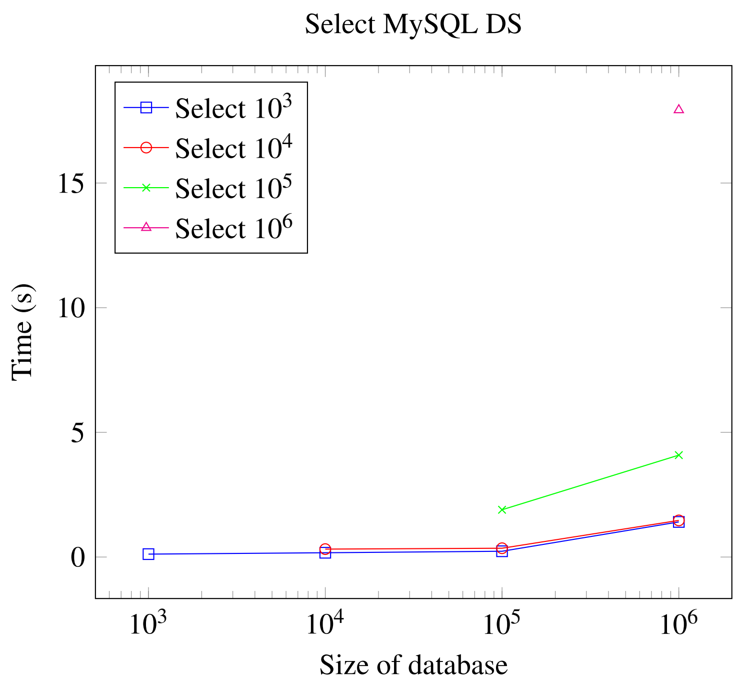

Figure 12: Time for selecting different amounts of data from databases of different sizes in MySQL DS.

3.4 Remove

MySQL Document Store performed pretty close to MongoDB when removing all documents up to 10^5, but at 10^6 it did not manage to complete the execution. See figure 13.

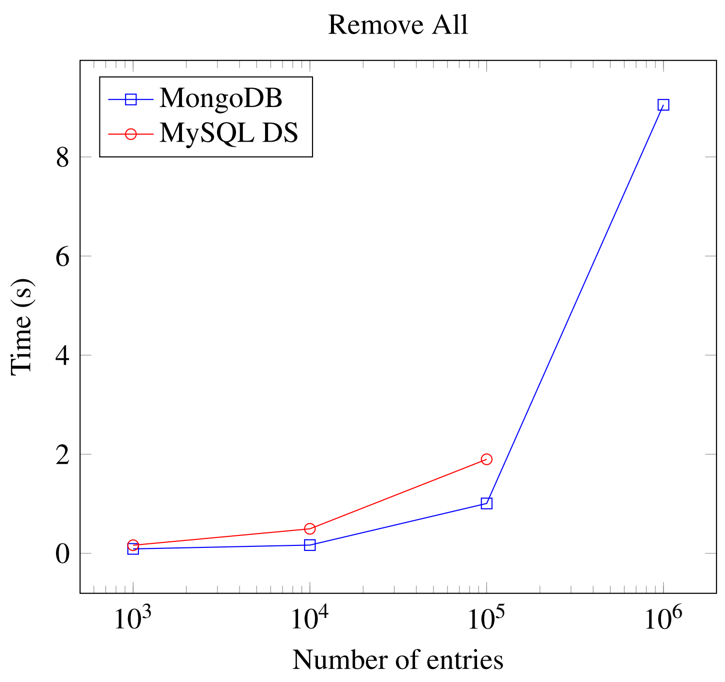

Figure 13: Time for removing all documents in databases of different sizes.

MongoDB had no problem removing one document from the database and kept a steady execution time until 10^5 documents, and at 10^6 size of the database only increases a little in time. MySQL Document Store on the other hand deviated from the MongoDB line from the start, and at 10^6 documents had a greater increase up to almost three seconds. See figure 14.

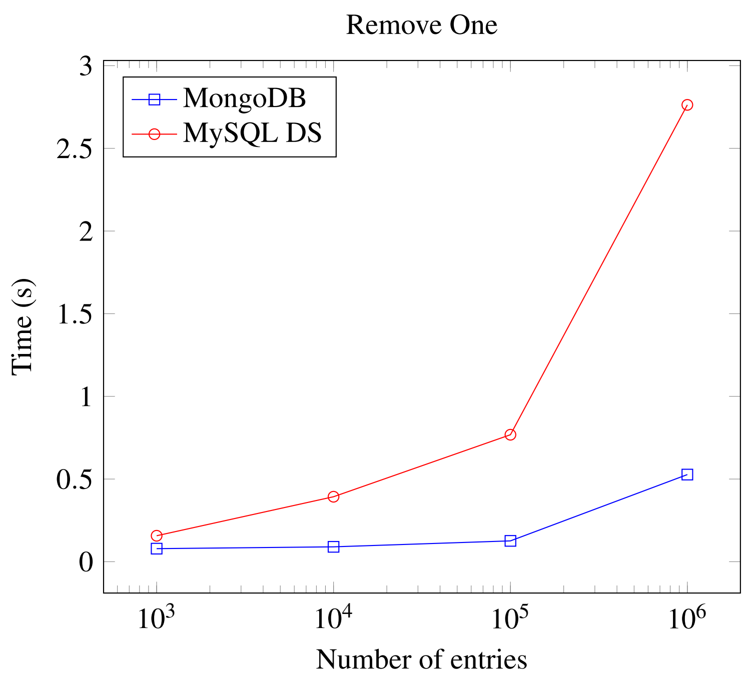

Figure 14: Time for removing a single document from databases of different sizes.

MySQL Document Store performed pretty well in the beginning, but increased drastically when trying to remove more than 10000 documents when the database had 1000000 documents already in it. See figure 16. MongoDB managed this test very well. See figure 15.

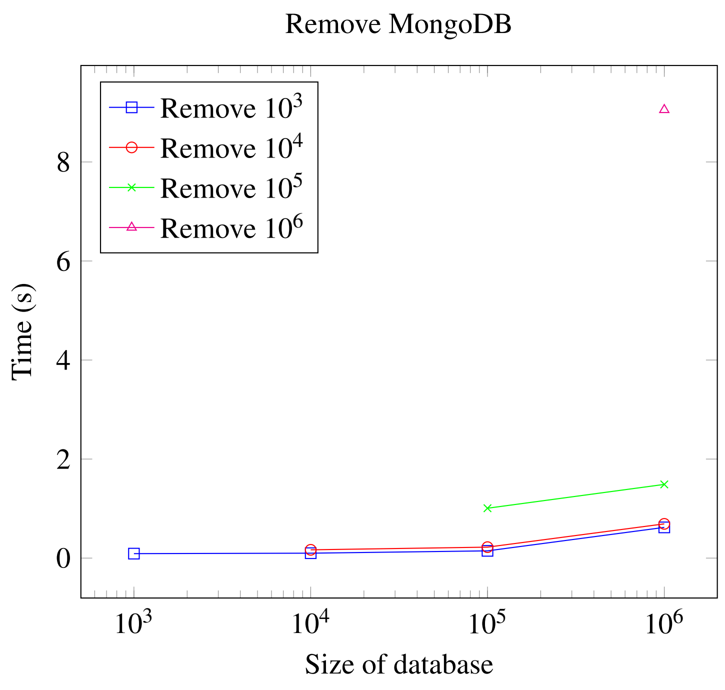

Figure 15: Time for removing different amounts of data from databases of different sizes in MongoDB.

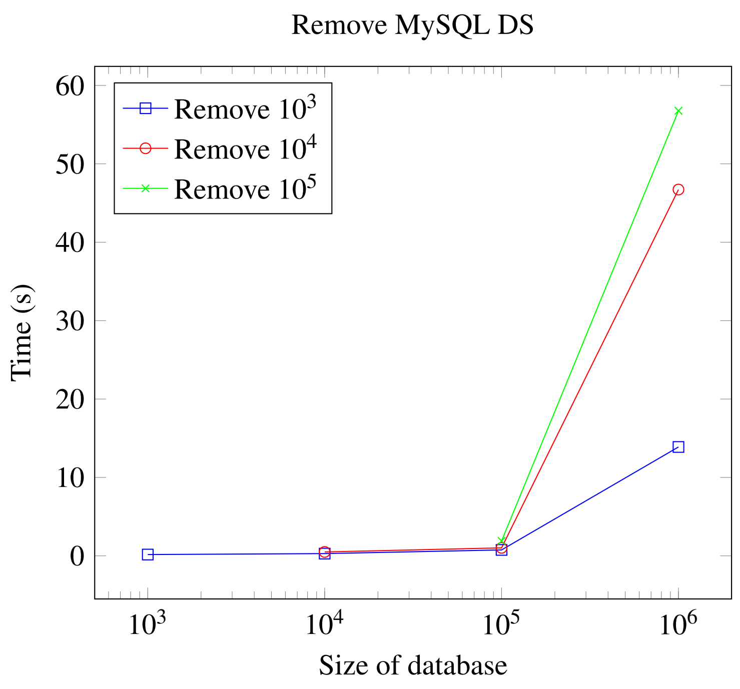

Figure 16: Time for removing different amounts of data from databases of different sizes in MySQL DS.

3.5 Size

MongoDB uses less space than MySQL DS for the same amount of information in these tests, but since these tests are focused on the document store part it doesn't use all the space that MySQL DS allocates.

Table 2 Sizes of the different databases at varying number of documents.

Table size (in bytes)		
Num	MongoDB	MySQL DS
1000	98304	409600
10000	692224	3686400
100000	6524928	50331648
1000000	65212416	444596224

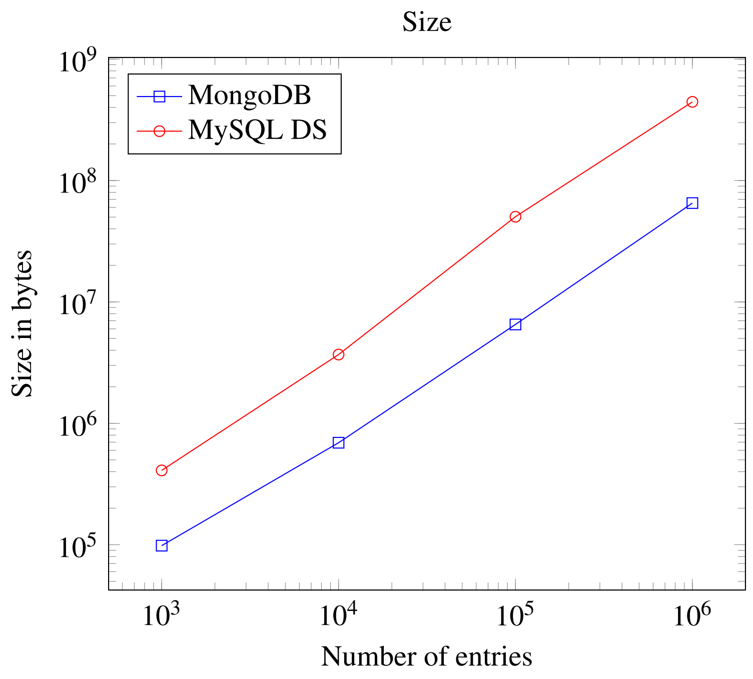

Figure 17: Size of the database when containing different amounts of data.

3.5.1 Hardware utilization

While the tests were performed we also monitored the CPU, RAM and Hard drive usage to see any potential limitation.

The CPU did not have any problem with handling any of the operations, only 1-2 core sometimes had a 100% usage, while the rest of the cores had minimal to no usage.

The hard drive usage was sometimes 100% busy ,though it did not use its maximum writing speed.

The RAM and swap-space usage reached 100% when executing the test in figure 4, inserting 10^6 documents in MySQL Document Store.

4 Discussion

MongoDB performed better but had an interesting increase of time at updating 10^6 documents when the size of the database was 10^6 documents. See figure 8.

MongoDB performed equally when multi-inserting different amounts of data into different amounts of already present data, shown in figure 3. This suggest that the technology scales pretty well, while updating, removing and selecting had a increase in time when executing the operations on 10^5 or more documents with 10^5 documents in the database.

MongoDB allocated less space for the same data, which was not so surprising since InnoDB stores more information about tables, such as table-structure. It also stores the data in files as BSON, while InnoDb stores the data in files in a B+ tree. [3] This could also be a explanation why MongoDB allocates less space. See table 2 and figure 17.

4.1 Limitations

MySQL Document Store failed to perform insert, update or remove on 10^6 documents, see figures 4, 7, 16. The execution would return MySQL Error 2000 and that the execution failed. Error 2000 is defined as ”unknown MySQL error” in the documentation, but by looking in the error-logs we could see the error ”lock_wait_timout_exceded”. This could with high probability be because of the many changes we do to the database in short period of time. We tried resolving this by changing the ”innodb_lock_wait_timeout” property to a higher value, but it was unsuccessful. Because it would return an error whenever the query ran for too long the ten successful attempts that we used for our average on a few of the tests is really the best-case scenario, that is why the lines seem to converge around 50 seconds on inserting 10^5 into a non-empty database in MySQL Document Store.

The performance issues with MySQL Document Store seems to be tied to the X Plugin and how it handles the executions. While we ran the test we noticed some serious issues with the memory-usage while doing multi-insertion where sometimes the RAM and swap space would be used up fully. It could possibly be because of a memory leak, or just how the plugin handles insertions and therefor requires more RAM for such a high amount of data to be inserted at once.

What caused the performance issues could depend on both hardware and software. To be able to get the most out of the database technologies you would also have to configure them for the use-cases. This is something that we did not do, since we wanted to base our results on the default settings.

4.2 Conclusion

In every single test we found MongoDB to perform better than MySQL DS. This was specially noticeable when operating on 10^5 documents or more, where MySQL Document Store could perform multiple times worse than MongoDB.

But since MySQL Document Store was recently released you could expect some performance issues. This will hopefully be addressed by Oracle while the technology matures and more people and organizations start using it. It is still a interesting approach by Oracle to implement this alternative way to work with data in MySQL and it shows that they want to have a foot in the NoSQL market.

References

[1] Historical trend of the popularity ranking of database management systems. https://db-engines.com/en/ranking_trend. Accessed: 2017-06-05.

[2] Our customers. https://www.mongodb.com/who-uses-mongodb. Accessed: 2017-06-06.

[3] Rudolf Bayer and Edward M. McCreight. Organization and maintenance of large ordered indices. *Acta Informatica*, 1(3):173–189, 1972.

[4] Cornelia Győrödi, Robert Győrödi, George Pecherle, and Andrada Olah. A comparative study: MongoDB vs. MySQL. In *Engineering of Modern Electric Systems (EMES), 2015 13th International Conference on*, pages 1–6. IEEE, 2015.

[5] Neal Leavitt. Will nosql databases live up to their promise? *Computer*, 43(2), 2010.

[6] MongoDB. *MongoDB Architecture*. https://www.mongodb.com/mongodb-architecture#data-model.

[7] Oracle. *Changes in MySQL 5.7.12*, 04 2016. https://dev.mysql.com/doc/relnotes/mysql/5.7/en/news-5-7-12.html.

[8] Alfred Wester and Olof Fredriksson. Jämförelse av Mysql och MongoDb, 2012.

A Result table

Type	Num	Existing	MySQL DS	MongoDB
Multi-insert	1000	0	0.226	**0.113**
Multi-insert	10000	0	0,960	**0,370**
Multi-insert	100000	0	33,622	**2,506**
Multi-insert	1000000	0	-	**29,028**
Multi-insert	1000	1000	0,441	**0,110**
Multi-insert	10000	1000	0,985	**0,354**
Multi-insert	100000	1000	50,158	**2,564**
Multi-insert	1000000	1000	-	**29,192**
Multi-insert	1000	10000	0,274	**0,110**
Multi-insert	10000	10000	0,943	**0,361**
Multi-insert	100000	10000	50,026	**2,587**
Multi-insert	1000000	10000	-	**29,020**
Multi-insert	1000	100000	1,389	**0,108**
Multi-insert	10000	100000	6,852	**0,357**
Multi-insert	100000	100000	53,774	**2,616**
Multi-insert	1000000	100000	-	**28,410**
Multi-insert	1000	1000000	0,398	**0,112**
Multi-insert	10000	1000000	8,861	**0,366**
Multi-insert	100000	1000000	50,710	**2,633**
Multi-insert	1000000	1000000	-	**28,386**
Single-insert	1000	0	64,339	**0,403**
Single-insert	10000	0	526,650	**3,382**
Single-insert	100000	0	5322,850	**33,273**
Single-insert	1000000	0	56224,030	**328,246**

Type	Num	Existing	MySQL DS	MongoDB
Multi-Select	1000	1000	0,118	**0,084**
Multi-Select	1000	10000	0,171	**0,100**
Multi-Select	10000	10000	0,317	**0,160**
Multi-Select	1000	100000	0,231	**0,149**
Multi-Select	10000	100000	0,352	**0,207**
Multi-Select	100000	100000	1,895	**0,649**
Multi-Select	1000	1000000	1,407	**0,558**
Multi-Select	10000	1000000	1,468	**0,626**
Multi-Select	100000	1000000	4,087	**1,069**
Multi-Select	1000000	1000000	17,929	**5,752**
Select	1	1000	0,098	**0,074**
Select	1	10000	0,112	**0,080**
Select	1	100000	0,252	**0,117**
Select	1	1000000	1,647	**0,467**

Type	Num	Existing	MySQL DS	MongoDB
Multi-Update	1000	1000	0,247	**0,089**
Multi-Update	1000	10000	0,291	**0,099**
Multi-Update	10000	10000	0,705	**0,180**
Multi-Update	1000	100000	0,981	**0,144**
Multi-Update	10000	100000	3,869	**0,233**
Multi-Update	100000	100000	9,136	**1,099**
Multi-Update	1000	1000000	5,046	**0,580**
Multi-Update	10000	1000000	11,738	**0,681**
Multi-Update	100000	1000000	23,381	**1,575**
Multi-Update	1000000	1000000	-	**10,582**
Update	1	1000	0,101	**0,073**
Update	1	10000	0,137	**0,083**
Update	1	100000	0,334	**0,120**
Update	1	1000000	2,493	**0,463**

Type	Num	Existing	MySQL DS	MongoDB
Multi-Remove	1000	1000	0,164	**0,090**
Multi-Remove	1000	10000	0,286	**0,099**
Multi-Remove	10000	10000	0,495	**0,167**
Multi-Remove	1000	100000	0,754	**0,146**
Multi-Remove	10000	100000	1,018	**0,221**
Multi-Remove	100000	100000	1,898	**1,007**
Multi-Remove	1000	1000000	13,885	**0,620**
Multi-Remove	10000	1000000	46,718	**0,691**
Multi-Remove	100000	1000000	56,768	**1,488**
Multi-Remove	1000000	1000000	-	**9,052**
Remove	1	1000	0,157	**0,079**
Remove	1	10000	0,393	**0,090**
Remove	1	100000	0,768	**0,126**
Remove	1	1000000	2,763	**0,527**

B MongoDB Code

Listing B.1: mongodb.sh - used for sending test scripts to the database

```
#!/usr/bin/env bash
mongo --authenticationDatabase admin -u master -p \
 zachariaserik localhost/inventory $@
```

Listing B.2: runTest.sh - used for automating tests

```
#!/bin/bash
./mongodb.sh empty.js > /dev/null 2> /dev/null || exit
./mongodb.sh fillWebstore.js > /dev/null 2> /dev/null || exit
echo "start␣loop"
for i in {1..10}
do
#       time ./mongodb.sh getComponents.js
        ./mongodb.sh empty.js && ./mongodb.sh fillWebstore.js && \
        time ./mongodb.sh delete.js
#       ./mongodb.sh reset.js && time ./mongodb.sh update.js
#       ./mongodb.sh empty.js && ./mongodb.sh fillWebstore.js && \
#       time ./mongodb.sh insert.js
#       ./mongodb.sh empty.js && ./mongodb.sh fillWebstore.js && \
#       time ./mongodb.sh singleInsert.js
done
```

Listing B.3: empty.js - used for emptying the database

```
var filter = {"universal": 1.0};  //delete all

db.item.remove(filter);
```

Listing B.4: fillWebstore.js - used for filling the database in preparation for other scripts.

```
const TOTAL_RECORDS = 1000;
const NUM_GPU       = TOTAL_RECORDS/5;
const NUM_RAM       = TOTAL_RECORDS/5;
const NUM_CPU       = TOTAL_RECORDS/5;
const NUM_MONITOR   = TOTAL_RECORDS/5;
const NUM_PSU       = TOTAL_RECORDS/5;

var tmp  = [];

for(var i=0; i<NUM_GPU; i++) {
        tmp.push(createGpu(i));
```

```
}
db.item.insert(tmp);
tmp = [];

for(var i=0; i<NUM_RAM; i++) {
        tmp.push(createRam(i));
}
db.item.insert(tmp);
tmp = [];

for(var i=0; i<NUM_CPU; i++) {
        tmp.push(createCpu(i));
}
db.item.insert(tmp);
tmp = [];

for(var i=0; i<NUM_MONITOR; i++) {
        tmp.push(createMonitor(i));
}
db.item.insert(tmp);
tmp = [];

for(var i=0; i<NUM_PSU; i++) {
        tmp.push(createPsu(i));
}
db.item.insert(tmp);

function createGpu(index){
        return {
                name:       "GTX "+index,
                type:       "Graphics Card",
                quantity:   Math.floor(inInterval(1,10000)),
                frequency:  Math.floor(inInterval(902,1784)),
                memory:     fromList([1,2,3,4,6,8,11,12,16,24,32]),
                speed:      inInterval(250,1800),
                watt:       Math.floor(inInterval(15,580)),
                millProp:   Math.floor(index/1000000),
                universal:  1
        }
}

function createRam(index){
        return {
                name:       "Memory "+index,
                type:       "RAM",
                quantity:   Math.floor(inInterval(1,10000)),
                memory:     fromList([1,2,3,4,6,8,12,16,24,32,48,64,96,128]),
                voltage:    inInterval(0.31,2.5),
                hundKProp: Math.floor(index/100000),
                universal: 1
        }
}

function createCpu(index){
        return {
                name:      "i"+index,
                type:      "CPU",
                quantity:  Math.floor(inInterval(1,200)),
                speed:     inInterval(1.1,4.7),
                cores:     Math.floor(inInterval(1,22)),
                tenKProp:  Math.floor(index/10000),
                universal: 1
        }
}

function createMonitor(index){
        var res = fromList([{
                x: 1024,
                y: 768
        },{
                x: 1280,
                y: 1024
        },{
                x: 1360,
                y: 768
        },{
                x: 1366,
                y: 480
        },{
                x: 1366,
                y: 768
        },{
                x: 1400,
                y: 900
        },{
                x: 1440,
                y: 900
        },{
                x: 1600,
                y: 900
        },{
                x: 1600,
                y: 1200
        },{
                x: 1680,
                y: 1050
        },{
                x: 1920,
                y: 1080
        },{
                x: 1920,
                y: 1200
        },{
                x: 2048,
                y: 1536
        },{
                x: 2560,
                y: 1024
        },{
                x: 2560,
                y: 1080
        },{
                x: 2560,
                y: 1440
        },{
                x: 2560,
                y: 1600
        },{
                x: 3440,
                y: 1440
        },{
                x: 3840,
                y: 1600
        },{
                x: 3840,
                y: 2160
        },{
                x: 4096,
                y: 2160
        },{
                x: 5120,
                y: 2880
        },{
                x: 5760,
                y: 2160
        }]);
        return {
                name:           "Monitor␣"+index,
                type:           "Monitor",
                quantity:       Math.floor(inInterval(1,500)),
                width:          res.x,
                height:         res.y,
                refreshRate:    fromList([30,60,85,100,120,144,180,240]),
                responseTime:   Math.floor(inInterval(1,25)),
                interface:    fromList(["BNC","Component","DisplayPort"
                                        "DVI","DVI-A","DVI-D␣Dual-Link"
                                        "DVI-D␣Single-Link",
                                        "DVI-I␣Dual-Link","HDMI",
                                        "Mini-Display␣Port","S-Video",
                                        "VGA"]),
                size:         inInterval(15,55),
                universal:    1
        }
}

function createPsu(index){
        return {
                name:       "PSU␣"+index,
                type:       "PSU",
                quantity:   Math.floor(inInterval(1,1500)),
                watt:       Math.floor(inInterval(180,2000)),
                kProp:      Math.floor(index/1000),
                universal:  1
        }
}

function inInterval(min, max) {
        return (Math.random() * (max - min)) + min;
}
function fromList(list) {
        return list[Math.floor(Math.random() * list.length)];
}
```

Listing B.5: getComponents.js - used for getting different amounts of data

```
//var filter = {"name": "GTX 0"};   //get 1
//var filter = {"universal": 1.0}; //get all
var filter = {"kProp": 1.0};       //get 1000
//var filter = {"tenKProp": 1.0};   //get 10000
//var filter = {"hundKProp": 1.0}; //get 100000
//var filter = {"millProp": 1.0};   //get 1000000

var cursor = db.item.find(filter).toArray();
```

Listing B.6: delete.js - used for removing different amounts of data

```
//var filter = {"name": "GTX 0"};     //delete 1
//var filter = {"universal": 1.0};   //delete all
var filter = {"kProp": 1.0};         //delete 1000
//var filter = {"tenKProp": 1.0};    //delete 10000
//var filter = {"hundKProp": 1.0};  //delete 100000
//var filter = {"millProp": 1.0};    //delete 1000000

db.item.remove(filter);
```

Listing B.7: reset.js - used for resetting the value of the universal property between tests

```
var filter = {};
var operation = {$set: {universal: 1.0}};
db.item.updateMany(filter ,operation);
```

Listing B.8: update.js - used for updating different amounts of data

```
//var filter = {"name": "GTX 0"};     //update 1
//var filter = {"universal": 1.0};   //update all
var filter = {"kProp": 1.0};        //update 1000
//var filter = {"tenKProp": 1.0};    //update 10000
//var filter = {"hundKProp": 1.0};  //update 100000
//var filter = {"millProp": 1.0};    //update 1000000

var operation = {$set: {universal: 2.0}};
db.item.updateMany(filter ,operation);
```

Listing B.9: insert.js - used for inserting different amounts of data

```
const TOTAL_RECORDS = 100000;
const NUM_GPU       = TOTAL_RECORDS/5;
const NUM_RAM       = TOTAL_RECORDS/5;
const NUM_CPU       = TOTAL_RECORDS/5;
const NUM_MONITOR   = TOTAL_RECORDS/5;
const NUM_PSU       = TOTAL_RECORDS/5;

var tmp  = [];

for(var i=0; i<NUM_GPU; i++) {
        tmp.push(createGpu(i));
}

for(var i=0; i<NUM_RAM; i++) {
        tmp.push(createRam(i));
}

for(var i=0; i<NUM_CPU; i++) {
        tmp.push(createCpu(i));
}

for(var i=0; i<NUM_MONITOR; i++) {
        tmp.push(createMonitor(i));
}

for(var i=0; i<NUM_PSU; i++) {
        tmp.push(createPsu(i));
}
db.item.insert(tmp);

function createGpu(index){
        return {
                name:       "GTX␣"+index,
                type:       "Graphics␣Card",
                quantity:  Math.floor(inInterval(1,10000)),
                frequency: Math.floor(inInterval(902,1784)),
                memory:     fromList([1,2,3,4,6,8,11,12,16,24,32]),
                speed:      inInterval(250,1800),
                watt:       Math.floor(inInterval(15,580)),
                millProp:  Math.floor(index/1000000),
                universal: 1
        }
}

function createRam(index){
        return {
                name:       "Memory␣"+index,
                type:       "RAM",
                quantity:  Math.floor(inInterval(1,10000)),
                memory:     fromList([1,2,3,4,6,8,12,16,24,32,
                                        48,64,96,128]),
                voltage:    inInterval(0.31,2.5),
                hundKProp: Math.floor(index/100000),
                universal: 1
        }
}

function createCpu(index){
        return {
                name:       "i"+index,
                type:       "CPU",
                quantity:  Math.floor(inInterval(1,200)),
                speed:      inInterval(1.1,4.7),
                cores:      Math.floor(inInterval(1,22)),
                tenKProp:  Math.floor(index/10000),
                universal: 1
        }
}

function createMonitor(index){
        var res = fromList([{
                x: 1024,
                y: 768
        },{
                x: 1280,
                y: 1024
        },{
                x: 1360,
                y: 768
```

```
            },{
                    x: 1366,
                    y: 480
            },{
                    x: 1366,
                    y: 768
            },{
                    x: 1400,
                    y: 900
            },{
                    x: 1440,
                    y: 900
            },{
                    x: 1600,
                    y: 900
            },{
                    x: 1600,
                    y: 1200
            },{
                    x: 1680,
                    y: 1050
            },{
                    x: 1920,
                    y: 1080
            },{
                    x: 1920,
                    y: 1200
            },{
                    x: 2048,
                    y: 1536
            },{
                    x: 2560,
                    y: 1024
            },{
                    x: 2560,
                    y: 1080
            },{
                    x: 2560,
                    y: 1440
            },{
                    x: 2560,
                    y: 1600
            },{
                    x: 3440,
                    y: 1440
            },{
                    x: 3840,
                    y: 1600
```

```
        },{
                x: 3840,
                y: 2160
        },{
                x: 4096,
                y: 2160
        },{
                x: 5120,
                y: 2880
        },{
                x: 5760,
                y: 2160
        }]);
        return {
                name:          "Monitor␣"+index,
                type:          "Monitor",
                quantity:      Math.floor(inInterval(1,500)),
                width:         res.x,
                height:        res.y,
                refreshRate:   fromList([30,60,85,100,120,144,180,240]),
                responseTime:  Math.floor(inInterval(1,25)),
                interface:     fromList(["BNC","Component",
                               "DisplayPort","DVI","DVI-A",
                               "DVI-D␣Dual-Link",
                               "DVI-D␣Single-Link",
                               "DVI-I␣Dual-Link","HDMI",
                               "Mini-Display␣Port",
                               "S-Video","VGA"]),
                size:          inInterval(15,55),
                universal:     1
        }
}

function createPsu(index){
        return {
                name:       "PSU␣"+index,
                type:       "PSU",
                quantity:   Math.floor(inInterval(1,1500)),
                watt:       Math.floor(inInterval(180,2000)),
                kProp:      Math.floor(index/1000),
                universal:  1
        }
}

function inInterval(min, max) {
        return (Math.random() * (max - min)) + min;
}
function fromList(list) {
        return list[Math.floor(Math.random() * list.length)];
}
```

Listing B.10: singleInsert.js - used for inserting different amounts of data one at a time

```
const TOTAL_RECORDS = 100000;
const NUM_GPU       = TOTAL_RECORDS/5;
const NUM_RAM       = TOTAL_RECORDS/5;
const NUM_CPU       = TOTAL_RECORDS/5;
const NUM_MONITOR   = TOTAL_RECORDS/5;
const NUM_PSU       = TOTAL_RECORDS/5;

for(var i=0; i<NUM_GPU; i++) {
        db.item.insert(createGpu(i));
}

for(var i=0; i<NUM_RAM; i++) {
        db.item.insert(createRam(i));
}

for(var i=0; i<NUM_CPU; i++) {
        db.item.insert(createCpu(i));
}

for(var i=0; i<NUM_MONITOR; i++) {
        db.item.insert(createMonitor(i));
}

for(var i=0; i<NUM_PSU; i++) {
        db.item.insert(createPsu(i));
}

function createGpu(index){
        return {
                name:       "GTX "+index,
                type:       "Graphics Card",
                quantity:   Math.floor(inInterval(1,10000)),
                frequency:  Math.floor(inInterval(902,1784)),
                memory:     fromList([1,2,3,4,6,8,11,12,16,24,32]),
                speed:      inInterval(250,1800),
                watt:       Math.floor(inInterval(15,580)),
                millProp:   Math.floor(index/1000000),
                universal:  1
        }
}

function createRam(index){
        return {
                name:       "Memory "+index,
                type:       "RAM",
                quantity:   Math.floor(inInterval(1,10000)),
                memory:     fromList([1,2,3,4,6,8,12,16,24,32,48,64,96,1
                voltage:    inInterval(0.31,2.5),
                hundKProp:  Math.floor(index/100000),
                universal:  1
        }
}

function createCpu(index){
        return {
                name:       "i"+index,
                type:       "CPU",
                quantity:   Math.floor(inInterval(1,200)),
                speed:      inInterval(1.1,4.7),
                cores:      Math.floor(inInterval(1,22)),
                tenKProp:   Math.floor(index/10000),
                universal:  1
        }
}

function createMonitor(index){
        var res = fromList([{
                x: 1024,
                y: 768
        },{
                x: 1280,
                y: 1024
        },{
                x: 1360,
                y: 768
        },{
                x: 1366,
                y: 480
        },{
                x: 1366,
                y: 768
        },{
                x: 1400,
                y: 900
        },{
                x: 1440,
                y: 900
        },{
                x: 1600,
                y: 900
        },{
                x: 1600,
        y: 1200
},{
        x: 1680,
        y: 1050
},{
        x: 1920,
        y: 1080
},{
        x: 1920,
        y: 1200
},{
        x: 2048,
        y: 1536
},{
        x: 2560,
        y: 1024
},{
        x: 2560,
        y: 1080
},{
        x: 2560,
        y: 1440
},{
        x: 2560,
        y: 1600
},{
        x: 3440,
        y: 1440
},{
        x: 3840,
        y: 1600
},{
        x: 3840,
        y: 2160
},{
        x: 4096,
        y: 2160
},{
        x: 5120,
        y: 2880
},{
        x: 5760,
        y: 2160
}]);
return {
        name:           "Monitor␣"+index,
        type:           "Monitor",
        quantity:       Math.floor(inInterval(1,500)),
                width:          res.x,
                height:         res.y,
                refreshRate:    fromList([30,60,85,100,120,144,180,240]),
                responseTime:   Math.floor(inInterval(1,25)),
                interface:      fromList(["BNC","Component","DisplayPort"
                                "DVI","DVI-A","DVI-D␣Dual-Link",
                                "DVI-D␣Single-Link","DVI-I␣Dual-Link",
                                "HDMI","Mini-Display␣Port","S-Video","VGA
                size:           inInterval(15,55),
                universal:      1
        }
}

function createPsu(index){
        return {
                name:       "PSU␣"+index,
                type:       "PSU",
                quantity:   Math.floor(inInterval(1,1500)),
                watt:       Math.floor(inInterval(180,2000)),
                kProp:      Math.floor(index/1000),
                universal:  1
        }
}

function inInterval(min, max) {
        return (Math.random() * (max - min)) + min;
}
function fromList(list) {
        return list[Math.floor(Math.random() * list.length)];
}
```

C MySQL Document Store Code

Listing C.1: mysql-ds.sh - used for sending test scripts to the database

```
#!/usr/bin/env bash
mysqlsh --uri root:zachariaserik@localhost/inventory --file $@
```

Listing C.2: runTest.sh - used for automating tests

```
#!/bin/bash
#./mysql-ds.sh empty.js > /dev/null 2> /dev/null || exit
#./mysql-ds.sh fillWebstore.js > /dev/null 2> /dev/null || exit
echo "start loop"
for i in {1..10}
do
#    time ./mysql-ds.sh getComponents.js
    ./mysql-ds.sh empty.js && ./mysql-ds.sh fillWebstore.js && \
    time ./mysql-ds.sh delete.js
#    ./mysql-ds.sh reset.js && time ./mysql-ds.sh update.js
#    ./mysql-ds.sh empty.js && ./mysql-ds.sh fillWebstore.js && \
#    time ./mysql-ds.sh insert.js
#    ./mysql-ds.sh empty.js && ./mysql-ds.sh fillWebstore.js && \
#    time ./mysql-ds.sh singleInsert.js
done
```

Listing C.3: empty.js - used for emptying the database

```
db.item.remove().execute();
```

Listing C.4: fillWebstore.js - used for filling the database in preparation for other scripts.

```
const TOTAL_RECORDS = 1000000;
const NUM_GPU       = TOTAL_RECORDS/5;
const NUM_RAM       = TOTAL_RECORDS/5;
const NUM_CPU       = TOTAL_RECORDS/5;
const NUM_MONITOR   = TOTAL_RECORDS/5;
const NUM_PSU       = TOTAL_RECORDS/5;
const NUM_INSERT    = 50000;

var tmp   = [];

for(var i=0; i<NUM_GPU; i++) {
        tmp.push(createGpu(i));
        if(i%NUM_INSERT==0){
                db.item.add(tmp).execute();
                tmp = [];
            }
    }
    db.item.add(tmp).execute();
    tmp = [];

    for(var i=0; i<NUM_RAM; i++) {
            tmp.push(createRam(i));
            if(i%NUM_INSERT==0){
                    db.item.add(tmp).execute();
                    tmp = [];
            }
    }
    db.item.add(tmp).execute();
    tmp = [];

    for(var i=0; i<NUM_CPU; i++) {
            tmp.push(createCpu(i));
            if(i%NUM_INSERT==0){
                    db.item.add(tmp).execute();
                    tmp = [];
            }
    }
    db.item.add(tmp).execute();
    tmp = [];

    for(var i=0; i<NUM_MONITOR; i++) {
            tmp.push(createMonitor(i));
            if(i%NUM_INSERT==0){
                    db.item.add(tmp).execute();
                    tmp = [];
            }
    }
    db.item.add(tmp).execute();
    tmp = [];

    for(var i=0; i<NUM_PSU; i++) {
            tmp.push(createPsu(i));
            if(i%NUM_INSERT==0){
                    db.item.add(tmp).execute();
                    tmp = [];
            }
    }
    db.item.add(tmp).execute();

    function createGpu(index){
            return {
                    name:           "GTX "+index,
                type:         "Graphics␣Card",
                quantity:     Math.floor(inInterval(1,10000)),
                frequency:    Math.floor(inInterval(902,1784)),
                memory:       fromList([1,2,3,4,6,8,11,12,16,24,32]),
                speed:        inInterval(250,1800),
                watt:         Math.floor(inInterval(15,580)),
                millProp:     Math.floor(index/1000000),
                universal:    1
        }
}

function createRam(index){
        return {
                name:         "Memory␣"+index,
                type:         "RAM",
                quantity:     Math.floor(inInterval(1,10000)),
                memory:       fromList([1,2,3,4,6,8,12,16,24,
                                         32,48,64,96,128]),
                voltage:      inInterval(0.31,2.5),
                hundKProp:    Math.floor(index/100000),
                universal:    1
        }
}

function createCpu(index){
        return {
                name:         "i"+index,
                type:         "CPU",
                quantity:     Math.floor(inInterval(1,200)),
                speed:        inInterval(1.1,4.7),
                cores:        Math.floor(inInterval(1,22)),
                tenKProp:     Math.floor(index/10000),
                universal:    1
        }
}

function createMonitor(index){
        var res = fromList([{
                x: 1024,
                y: 768
        },{
                x: 1280,
                y: 1024
        },{
                x: 1360,
                y: 768
        },{
                x: 1366,
                y: 480
        },{
                x: 1366,
                y: 768
        },{
                x: 1400,
                y: 900
        },{
                x: 1440,
                y: 900
        },{
                x: 1600,
                y: 900
        },{
                x: 1600,
                y: 1200
        },{
                x: 1680,
                y: 1050
        },{
                x: 1920,
                y: 1080
        },{
                x: 1920,
                y: 1200
        },{
                x: 2048,
                y: 1536
        },{
                x: 2560,
                y: 1024
        },{
                x: 2560,
                y: 1080
        },{
                x: 2560,
                y: 1440
        },{
                x: 2560,
                y: 1600
        },{
                x: 3440,
                y: 1440
        },{
                x: 3840,
                y: 1600
        },{
                x: 3840,
                y: 2160
        },{
                x: 4096,
                y: 2160
        },{
                x: 5120,
                y: 2880
        },{
                x: 5760,
                y: 2160
        }]);
        return {
                name:           "Monitor␣"+index,
                type:           "Monitor",
                quantity:       Math.floor(inInterval(1,500)),
                width:          res.x,
                height:         res.y,
                refreshRate:    fromList([30,60,85,100,120,144,180,240]),
                responseTime:   Math.floor(inInterval(1,25)),
                interface:      fromList(["BNC","Component",
                                "DisplayPort","DVI","DVI-A",
                                "DVI-D␣Dual-Link","DVI-D␣Single-Link",
                                "DVI-I␣Dual-Link","HDMI",
                                "Mini-Display␣Port","S-Video","VGA"]),
                size:           inInterval(15,55),
                universal:      1
        }
}

function createPsu(index){
        return {
                name:       "PSU␣"+index,
                type:       "PSU",
                quantity:   Math.floor(inInterval(1,1500)),
                watt:       Math.floor(inInterval(180,2000)),
                kProp:      Math.floor(index/1000),
                universal:  1
        }
}

function inInterval(min, max) {
        return (Math.random() * (max - min)) + min;
}
function fromList(list) {
        return list[Math.floor(Math.random() * list.length)];
}
```

Listing C.5: getComponents.js - used for getting different amounts of data

```
//var filter = "universal = 1"  //get all
var filter = "kProp = 0";       //get 1000
//var filter = "tenKProp = 0";  //get 10000
//var filter = "hundKProp = 0"; //get 100000
//var filter = "millProp = 0";  //get 1000000

var items = db.item.find(filter).execute().fetchAll();
```

Listing C.6: delete.js - used for removing different amounts of data

```
var filter = "name = 'GTX 0'"; //Remove 1
//var filter = "universal = 1";  //Remove all
//var filter = "kProp = 0";      //Remove 1000
//var filter = "tenKProp = 0";   //Remove 10000
//var filter = "hundKProp = 0";  //Remove 100000
//var filter = "millprop = 0";   //Remove 1000000

db.item.remove(filter).execute();
```

Listing C.7: reset.js - used for resetting the value of the universal property between tests

```
db.item.modify().set("universal", 1.0).execute();
```

Listing C.8: update.js - used for updating different amounts of data

```
//var filter = "name = 'GTX 0'"; // Update 1
//var filter = "universal = 1";  // Update all
var filter = "kProp = 0";        // Update 1000
//var filter = "tenKProp = 0";   // Update 10000
//var filter = "hundKProp = 0";  // Update 100000
//var filter = "millProp";       // Update 1000000

db.item.modify(filter).set("universal", 2.0).execute();
```

Listing C.9: insert.js - used for inserting different amounts of data

```
const TOTAL_RECORDS = 1000000;
const NUM_GPU       = TOTAL_RECORDS/5;
const NUM_RAM       = TOTAL_RECORDS/5;
const NUM_CPU       = TOTAL_RECORDS/5;
const NUM_MONITOR   = TOTAL_RECORDS/5;
const NUM_PSU       = TOTAL_RECORDS/5;

var tmp  = [];

for(var i=0; i<NUM_GPU; i++) {
        tmp.push(createGpu(i));
}

for(var i=0; i<NUM_RAM; i++) {
        tmp.push(createRam(i));
```

```
}

for(var i=0; i<NUM_CPU; i++) {
        tmp.push(createCpu(i));
}

for(var i=0; i<NUM_MONITOR; i++) {
        tmp.push(createMonitor(i));
}

for(var i=0; i<NUM_PSU; i++) {
        tmp.push(createPsu(i));
}
db.item.add(tmp).execute();

function createGpu(index){
        return {
                name:       "GTX␣"+index,
                type:       "Graphics␣Card",
                quantity:   Math.floor(inInterval(1,10000)),
                frequency:  Math.floor(inInterval(902,1784)),
                memory:     fromList([1,2,3,4,6,8,11,12,16,24,32]),
                speed:      inInterval(250,1800),
                watt:       Math.floor(inInterval(15,580)),
                millProp:   Math.floor(index/1000000),
                universal:  1
        }
}

function createRam(index){
        return {
                name:       "Memory␣"+index,
                type:       "RAM",
                quantity:   Math.floor(inInterval(1,10000)),
                memory:     fromList([1,2,3,4,6,8,12,16,24,
                                        32,48,64,96,128]),
                voltage:    inInterval(0.31,2.5),
                hundKProp:  Math.floor(index/100000),
                universal:  1
        }
}

function createCpu(index){
        return {
                name:       "i"+index,
                type:       "CPU",
                quantity:   Math.floor(inInterval(1,200)),
                speed:      inInterval(1.1,4.7),
                        cores:      Math.floor(inInterval(1,22)),
                        tenKProp:   Math.floor(index/10000),
                        universal:  1
                }
        }

        function createMonitor(index){
                var res = fromList([{
                        x: 1024,
                        y: 768
                },{
                        x: 1280,
                        y: 1024
                },{
                        x: 1360,
                        y: 768
                },{
                        x: 1366,
                        y: 480
                },{
                        x: 1366,
                        y: 768
                },{
                        x: 1400,
                        y: 900
                },{
                        x: 1440,
                        y: 900
                },{
                        x: 1600,
                        y: 900
                },{
                        x: 1600,
                        y: 1200
                },{
                        x: 1680,
                        y: 1050
                },{
                        x: 1920,
                        y: 1080
                },{
                        x: 1920,
                        y: 1200
                },{
                        x: 2048,
                        y: 1536
                },{
                        x: 2560,
                y: 1024
        },{
                x: 2560,
                y: 1080
        },{
                x: 2560,
                y: 1440
        },{
                x: 2560,
                y: 1600
        },{
                x: 3440,
                y: 1440
        },{
                x: 3840,
                y: 1600
        },{
                x: 3840,
                y: 2160
        },{
                x: 4096,
                y: 2160
        },{
                x: 5120,
                y: 2880
        },{
                x: 5760,
                y: 2160
        }]);
        return {
                name:           "Monitor␣"+index,
                type:           "Monitor",
                quantity:       Math.floor(inInterval(1,500)),
                width:          res.x,
                height:         res.y,
                refreshRate:    fromList([30,60,85,100,120,144,180,240]),
                responseTime:   Math.floor(inInterval(1,25)),
                interface:      fromList(["BNC","Component",
                                "DisplayPort","DVI","DVI-A",
                                "DVI-D␣Dual-Link","DVI-D␣Single-Link",
                                "DVI-I␣Dual-Link","HDMI",
                                "Mini-Display␣Port","S-Video","VGA"]),
                size:           inInterval(15,55),
                universal:      1
        }
}

function createPsu(index){
        return {
                name:       "PSU "+index,
                type:       "PSU",
                quantity:   Math.floor(inInterval(1,1500)),
                watt:       Math.floor(inInterval(180,2000)),
                kProp:      Math.floor(index/1000),
                universal:  1
        }
}

function inInterval(min, max) {
        return (Math.random() * (max - min)) + min;
}
function fromList(list) {
        return list[Math.floor(Math.random() * list.length)];
}
```

Listing C.10: singleInsert.js - used for inserting different amounts of data one at a time

```
const TOTAL_RECORDS = 1000000;
const NUM_GPU       = TOTAL_RECORDS/5;
const NUM_RAM       = TOTAL_RECORDS/5;
const NUM_CPU       = TOTAL_RECORDS/5;
const NUM_MONITOR   = TOTAL_RECORDS/5;
const NUM_PSU       = TOTAL_RECORDS/5;

for(var i=0; i<NUM_GPU; i++) {
        db.item.add(createGpu(i)).execute();
}

for(var i=0; i<NUM_RAM; i++) {
        db.item.add(createRam(i)).execute();
}

for(var i=0; i<NUM_CPU; i++) {
        db.item.add(createCpu(i)).execute();
}

for(var i=0; i<NUM_MONITOR; i++) {
        db.item.add(createMonitor(i)).execute();
}

for(var i=0; i<NUM_PSU; i++) {
        db.item.add(createPsu(i)).execute();
}

function createGpu(index){
        return {
                name:       "GTX "+index,
                type:       "Graphics␣Card",
                quantity:   Math.floor(inInterval(1,10000)),
                frequency:  Math.floor(inInterval(902,1784)),
                memory:     fromList([1,2,3,4,6,8,11,12,16,24,32]),
                speed:      inInterval(250,1800),
                watt:       Math.floor(inInterval(15,580)),
                millProp:   Math.floor(index/1000000),
                universal:  1
        }
}

function createRam(index){
        return {
                name:       "Memory␣"+index,
                type:       "RAM",
                quantity:   Math.floor(inInterval(1,10000)),
                memory:     fromList([1,2,3,4,6,8,12,16,24,32,
                                     48,64,96,128]),
                voltage:    inInterval(0.31,2.5),
                hundKProp:  Math.floor(index/100000),
                universal:  1
        }
}

function createCpu(index){
        return {
                name:       "i"+index,
                type:       "CPU",
                quantity:   Math.floor(inInterval(1,200)),
                speed:      inInterval(1.1,4.7),
                cores:      Math.floor(inInterval(1,22)),
                tenKProp:   Math.floor(index/10000),
                universal:  1
        }
}

function createMonitor(index){
        var res = fromList([{
                x: 1024,
                y: 768
        },{
                x: 1280,
                y: 1024
        },{
                x: 1360,
                y: 768
        },{
                x: 1366,
               y:  480
        },{
               x:  1366,
               y:  768
        },{
               x:  1400,
               y:  900
        },{
               x:  1440,
               y:  900
        },{
               x:  1600,
               y:  900
        },{
               x:  1600,
               y:  1200
        },{
               x:  1680,
               y:  1050
        },{
               x:  1920,
               y:  1080
        },{
               x:  1920,
               y:  1200
        },{
               x:  2048,
               y:  1536
        },{
               x:  2560,
               y:  1024
        },{
               x:  2560,
               y:  1080
        },{
               x:  2560,
               y:  1440
        },{
               x:  2560,
               y:  1600
        },{
               x:  3440,
               y:  1440
        },{
               x:  3840,
               y:  1600
        },{
               x:  3840,
                        y: 2160
                },{
                        x: 4096,
                        y: 2160
                },{
                        x: 5120,
                        y: 2880
                },{
                        x: 5760,
                        y: 2160
                }]);
                return {
                        name:           "Monitor␣"+index,
                        type:           "Monitor",
                        quantity:       Math.floor(inInterval(1,500)),
                        width:          res.x,
                        height:         res.y,
                        refreshRate:    fromList([30,60,85,100,120,144,180,240]),
                        responseTime:   Math.floor(inInterval(1,25)),
                        interface:      fromList(["BNC","Component",
                                        "DisplayPort","DVI","DVI−A",
                                        "DVI−D␣Dual−Link","DVI−D␣Single−Link",
                                        "DVI−I␣Dual−Link","HDMI",
                                        "Mini−Display␣Port","S−Video","VGA"]),
                        size:           inInterval(15,55),
                        universal:      1
                }
        }

        function createPsu(index){
                return {
                        name:       "PSU␣"+index,
                        type:       "PSU",
                        quantity:   Math.floor(inInterval(1,1500)),
                        watt:       Math.floor(inInterval(180,2000)),
                        kProp:      Math.floor(index/1000),
                        universal:  1
                }
        }

        function inInterval(min, max) {
                return (Math.random() ∗ (max − min)) + min;
        }
        function fromList(list) {
                return list[Math.floor(Math.random() ∗ list.length)];
        }
```