In-depth review of NoSQL and SQL performance based on data size

Ralfs Zangis
University College of Northern Denmark
1062012@ucn.dk

Abstract. Today, it is not uncommon to hear terms such as SQL and NoSQL, but it is not always clear whether one is better performing. While it is accepted a practice that relational databases are more used on a single server and non-relational databases excel at clusters of computers, it does not explain which of these technologies better coups with different sizes of datasets, when limited with the use of a single computer. The results of this document will lead to better understanding of the effects of data size on the performance of CRUD (Create, Read, Update and Delete) actions, when looking at the two already mentioned types of databases. The conclusion of the research is that non-relational databases are generally better in dealing with large amounts of data, but they are not outperforming relational databases for all use cases, meaning that NoSQL in its current form will not completely replace SQL databases.

Keywords: SQL, NoSQL, performance, database, data size.

1 Introduction

In the past, the choice for saving relatively huge amounts of information was a relational database, also known as SQL [1], but in recent years, there has been development in the sphere of data storage with a new type of database becoming highly popular, that being non-relational or simply NoSQL [2]. Name of NoSQL comes from Carlo Strozzi and it was later popularized by Johan Oskarsson, when he organized an event to discuss "open source distributed, non-relational databases". Ever since the creation and further popularization of non-relational databases, it is becoming harder to choose the right type of storage method for the project's needs. While, as already mentioned, in past there was essentially a single answer to what database to choose. Nowadays it could be considered irresponsible to at least not consider both technologies for the task at hand. SQL and NoSQL have many differences, starting from how they deal with data and ending with what they are used for [3]. NoSQL is most beneficial when it is used in a cluster of computers and for the most part, it is free to use, unlike many relational databases, but SQL databases have fixed scheme, allowing them to hold more consistent data and be more reliable. The many benefits of NoSQL have made it the ideal type of database for projects, involving huge amounts of data, especially since, as already said, it can be distributed over many computer nodes, increasing the speed of processing while reducing the cost of maintenance and future expansion of existing hardware. However, the purpose of this document is not to compare any of these features, rather focusing on how scalable the technology is on a single computer. The scalability of the databases in this research paper will be tested with an increasingly larger size of data and by performing various actions, that would have been done to information on day to day bases, later documenting the results and making conclusions from the observations. Although, there are many different databases, that could have been compared, during the duration of this document we will explore only most used ones, as it would be out of the scope of this document to consider completely all of them. The analysis will be done using both desk research and practical experiments, adopted for proving the theory. The answer received from this research paper should let the reader better understand the differences between the

two technologies, thus allowing him to make a better-informed decision when choosing the appropriate solution for the data storage problem.

2 Related Work

History of the different types of databases is interesting to know and it gives a better understanding of the current trends for data storage [4]. The two of the most commonly used types of databases, at the time of writing this document, are relational and non-relational. Differences between them are discussed in [5]. Authors also cover the 4 commonly acknowledged types of NoSQL database, while considering their capabilities with respect to their characteristics and features. Moreover, Yishan Li and Sathiamoorthy Manoharan have already compared read, write, delete, and instantiate operations using SQL and key-value stores implementations of NoSQL database [6]. Example showing a simple comparison of the two types of the databases was also carried out by Z. Parker, S. Poe, and S. Vrbsky, where they showed the differences in performance between MongoDB and Microsoft SQL Server Express [7]. However, even though NoSQL databases are no longer considered to be brand-new technology and they have had time to evolve, the non-relational databases, with all their benefits over SQL, have not been able to replace relational databases [8]. This finding is further supported by D. Bartholomew, showing that NoSQL shouldn't be looked at as a competitor to SQL, but rather as a solution to problems where relational databases struggle [9].

3 Method

databases

SQL is used for management of data held in a relational database management system (RDBMS), it is somewhat standardized, but it is not completely portable among different database systems without adjustments, meaning that SQL code made for MySQL will not necessarily work in Oracle Database. There are 4 commonly acknowledged types of NoSQL [10]:

- Key-Value Store: values are stored in a hash table with a unique key
- Document-based Store: like a key-value store, but information saved provides some structure
- Column-based Store: data is stored in cells (similar to the relational database), but it is grouped in columns rather than rows
- Graph-based Store: the information is stored using edges edges and nodes

For this paper, we will compare some of the currently most relevant solutions in NoSQL and SQL. The selected databases are intentionally kept different from one another, based on what they are used for. This is done to give a better understanding of their possible differences in performance. To determine how popular a database is, it was decided to use a website [11], that uses various internet resources such as: google trends, mentions on search engines, job offerings and blogs about technology.

From NoSQL it was determined to use the following databases:

1. MongoDB- a cross-platform, document-oriented database that provides, great performance, high availability, and easy scalability. It uses dynamic schema, meaning that documents in the same collection do not need to have the same set of fields, structure and may even hold different types of data.

2. Redis (Remote Dictionary Server)- a fast, open-source, in-memory key-value data store for use as a database, cache, message broker, and queue. It is a popular choice for caching, session management, gaming, leader boards, real-time analytics, geospatial, ride-hailing, chat/messaging, media streaming, and pub/sub apps.
3. Cassandra- column-based Store that can be decentralized, meaning, that there is no single point of failure as data is automatically replicated in multiple nodes. Some of the largest deployments include Apple, with over 75,000 nodes storing over 10 PB of data and Netflix with 2,500 nodes, 420 TB of data and over 1 trillion requests per day.

From SQL we will investigate:

1. MySQL- one of few open source relational database management system, this makes it attractive to companies, especially in the field of web development, where it is not uncommon to be seen even in use for biggest companies with some of the most complex operations.
2. Microsoft SQL Server- one of the most secure databases, it is commonly used for online transaction processing and data warehousing. Being developed by Microsoft it has also many other advanced features, which user could find useful, especially when used together with other companies' products.

Database	Version
MongoDB	4.0.8
Redis	5.0.4
Cassandra Datastax	3.0.9
MySQL	8.0.15
MSSQL	14.0.1000.169

Table 1 The databases used for research

Table 1 shows what versions of databases were used for testing, as it is possible that speeds can change with future updates of software. It was also considered to use one of the most popular databases- Oracle, but due to low bulk data insert speeds and many different ways of configuring the database it was opted to not use it for this comparison.

Approach

The scalability of the solution was tested by using a single computer, performing simple CRUD (Create, read, update and Delete) actions on the selected databases, while gradually increasing the dataset size and documenting the resulting changes to performance. The comparison of speed change was achieved by setting the performance of database with 100 unique entries as a benchmark and comparing the following results with it. The outcomes later were documented and visualized for better understanding using graphs and diagrams. Before continuing, it is important, that reader knows, that these technologies can largely be optimized for different tasks and needs, so performance may differ depending on the software setup and hardware used. For the duration of the experiments, we are going to look at the performance of databases with default/standard settings. To mitigate outside factors, such as the influence of the chosen library, it was decided to compare each run with how long it took to perform an action on a database consisting of 100 values. This was done since the performance of the library would not change with increased data in the database, leaving the only variable to change to be the database behavior speed itself.

Implementation

All of the work on this project was done using Python [12], this decision was made as it was easy to work with, it had a great community with many useful libraries for the project, and there was a need for relatively little code to get the necessary results. The only drawback of python was its performance when compared to other programming languages, but that would not affect the results of our experiments. The implementation of tests was started by creating a simple code, used for filling databases with automatically generated datasets of various sizes. Data created consisted of 2 columns, those being: “id” later used as a primary key and “value” for storing simple text information. Data was generated using python libraries “numpy” and “pandas” [13], they are commonly being used for various big data projects. The generated data size, in the beginning, was 100 unique entries with each subsequent test data size being increased by 10 times until ending up with the tests for following number of values: 100, 1 thousand, 10 thousand, 100 thousand, 1 million and 10 million. The limit of 10 million values was set since trends were already clearly visible and increasing the data size required more RAM than accessible for the experiment, even if data was inserted in smaller batches, it would have taken a long time to compute. For inserting and testing of the database, it was decided to adoption libraries offered by the database software manufacturer, for use in python (pymongo, redis, cassandra, pyodbc and mysql). However, for relational databases, it was decided to also operate “sqlalchemy”, which is another great library and allows the developer to easily insert data frames generated by “pandas” into the database. To standardize testing across different the databases all tests were made as similar as possible and the connection to the database was established before testing was started. This was done as we are only interested in the actual performance of the operation and not into how long it takes to also establish contact. Furthermore, the already mentioned steps allow to better simulate normal use environment of the database and to give it more realistic results, to what users of the database would be experiencing. To accommodate more consistent results, it was also decided to automate the testing using a library called “timeit”, it allowed the author to specify how many tests to perform and how many times to repeat them, greatly reducing the time spent on redundant tasks and allowing for greater amount of assessments. The number of tests necessary to receive a quite a high degree of accuracy was determined to be 10, the further increase to the number of the tests would only return slight improvement in precision. The results constructed by inquiry were saved into CSV files, so they could be easily used for future calculations and making of graphs.

Hardware

The computer, used for research, had an Intel i5-5300U processor with a base frequency of 2.3GHz, 16 GBs of RAM, 500GB solid-state drive and Windows 10 Pro N (OS Built 16299.492) installed. To keep results received consistent and more reliable all non-essential applications were terminated for the duration of the practical experiments.

Data cleaning

To improve the precision of measurements it was decided to clean the data after being generated by tests. The main concern when improving results was to get rid of outliers [14], who were especially common for the first call to the database and in some places like for “Redis” it was possible for the first call to take up to 1 second when the average of other operations take only less than a millisecond. To get rid of outliers, it was decided to replace the outliers, who deviated from median with 2 or more standard deviations, with the average of the action for the specific database.

NAME	1	2	3	...	MEAN
MONGO CREATE	0.003644	0.001148	0.00103	...	0.000906
MONGO READ	0.00221	0.00186	0.001741	...	0.001164
MONGO UPDATE	0.001337	0.001259	0.001227	...	0.001265
MONGO DELETE	0.001346	0.001301	0.001325	...	0.001281
REDIS CREATE	1.006366	0.000433	0.000443	...	0.000436

Table 2 Snippet of results from running the tests

Table 2 shows an example of data having outliers (indicated by cells being coloured in yellow), they would be replaced with the mean of the row (when calculating mean the outliers were not considered).

4 Results

Findings

The first experiment shows the average execution time of 10 create, read, update and delete actions, performed on a database consisting of 100 values, using already mentioned libraries. This experiment was made to give the reader a better understanding of what times were initially compared, providing a better sense of the following diagrams. See Figure 1 summarizing the results of this experiment.

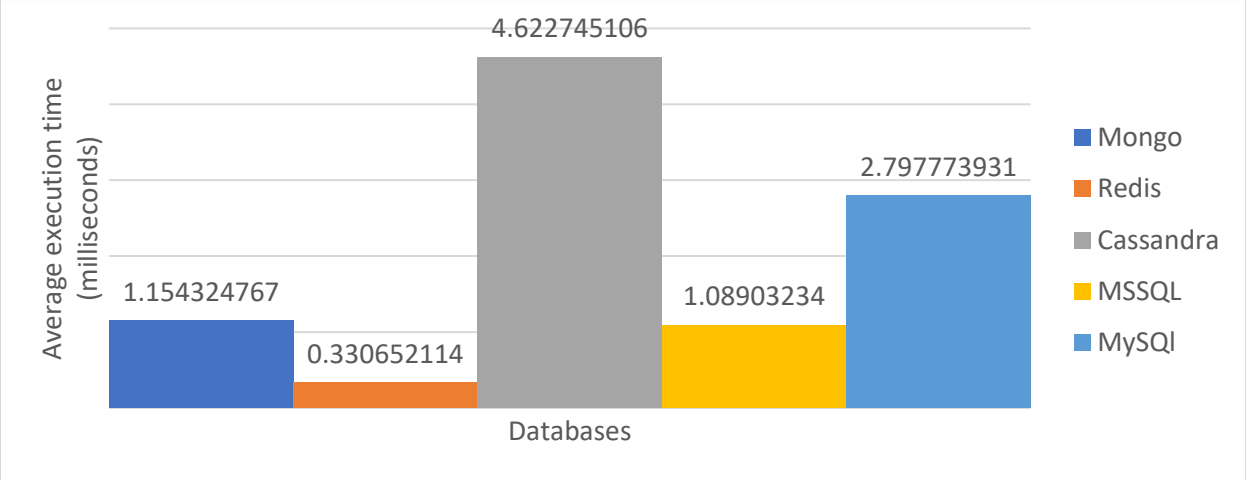

Figure 1 The average execution time of databases consisting of 100 entries

By reviewing Figure 1 we can clearly see that some databases are noticeably slower than others, for example, Cassandra was slower than its pears taking more than 4 milliseconds to perform a CRUD action on average, while Redis took only 0.3 milliseconds.

Following this experiment, it was decided to take a deeper look into how the basic database actions compare to the different data storage methods. The outcomes extracted for CRUD methods are shown in the following figures, positive result indicating that the database has increased in performance, while negative showing that the speed of performing an action has reduced.

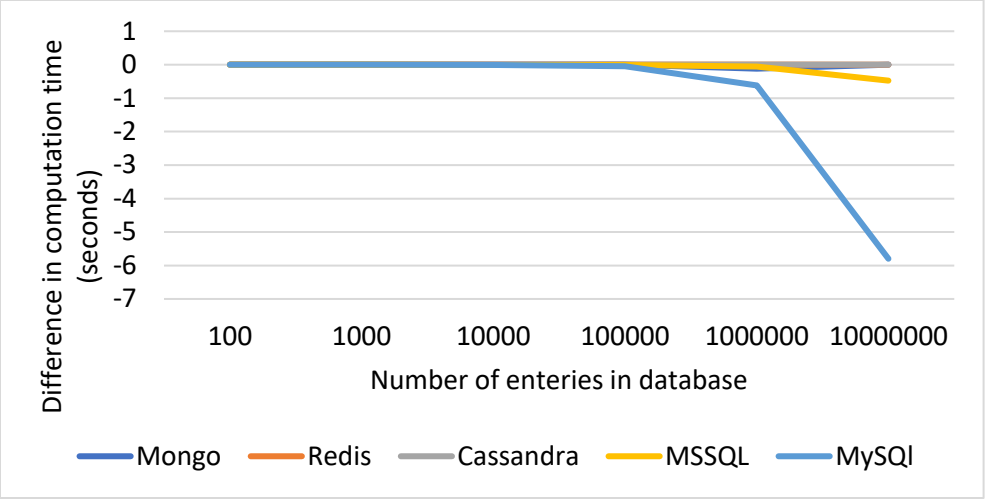

Figure 2 The difference of Read execution time based on element count in the databases

Figure 2 shows how read speeds have changed for the considered databases. By observing the development, we can note that time needed to finish read action for NoSQL has not visibly changed, while SQL databases have been affected. MySQL performance with 10 million rows has been drastically altered, now taking approximately 6 seconds longer to perform a task than for benchmark of 100 values, with MSSQL execution also being prolonged by close to 1 second.

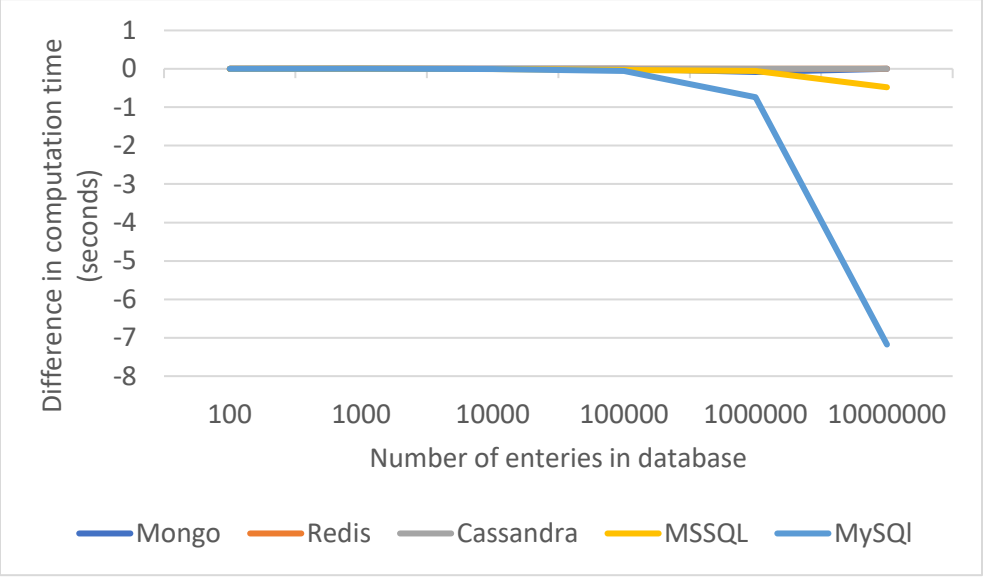

Figure 3 The difference of Update execution time based on element count in the databases

Update action shown in Figure 3 illustrates change to update speeds and it is following similar growth trends as what can be seen in Read action, the only notable difference between the two graphs is how

MySQL performance has worsened.

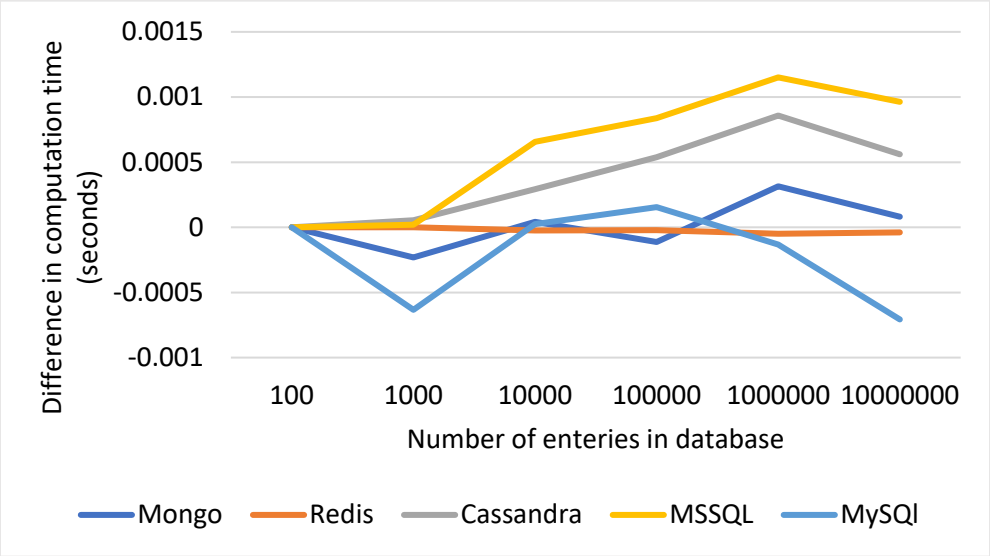

Figure 4 The difference of Create execution time based on element count in the databases

In figure 4 we can notice that there is very small change to single entry create function times, with change not exceeding more than 1 millisecond and it not following any previously established tendency.

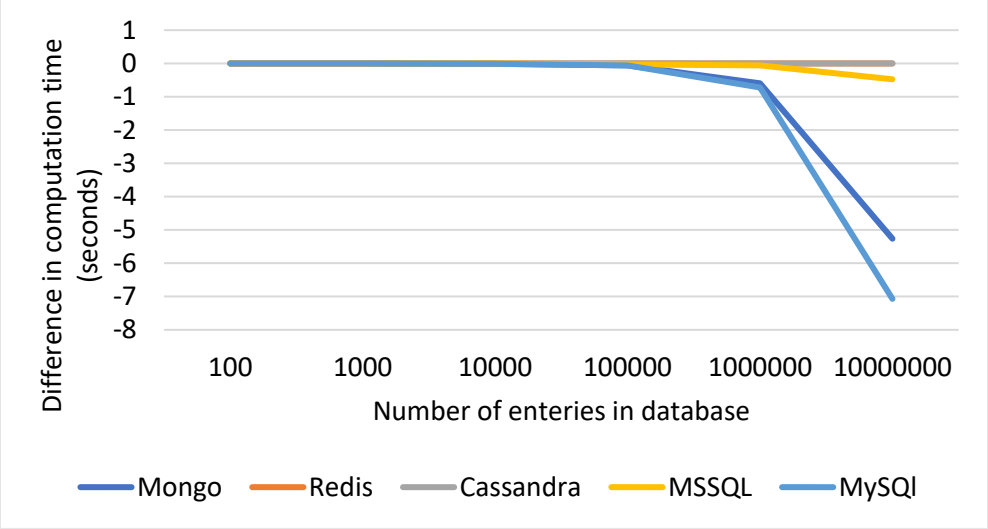

Figure 5 The difference of Delete execution time based on element count in the databases

Delete execution times are shown in Figure 5 and we can note that performance of Redis and Cassandra has not changed, while rest of the compared databases have been affected, with MongoDB and MySQL performing worst for this experiment. MySQL now taking approximately 7 seconds to perform a simple delete task.

Evaluation

From the experiments performed during the making of this paper, we can see that non-relational databases generally deal better with increasing amounts of simple data. The speed of finishing a task not

being affected for NoSQL databases: Redis and Cassandra, with only delete times changing for MongoDB. Their performance sometimes was slightly improving in places with larger amounts of data. While the only activity where the performance of relational databases did not noticeably change was creating. This makes sense as you do not have to find the element in a database to create it, meaning that it should not be affected by data size. This explains why NoSQL is so commonly used for projects requiring management of huge amounts of information.

5 Conclusions and Future Work

Non-relational and relational databases are two of the most popular ways of saving data, with each of them being better suited for specific tasks. During the extent of this document, we have looked at the change in time to perform a standard database operation on increasingly larger amounts of data. Based upon previous research and results of experiments done, we can notice that there is a clearly visible trend, showing that relational databases are for the most part negatively affected by the increased amount of information, while non-relational databases seem to not have experienced any major change to performance. The undertaken analysis explains why NoSQL is so attractive for solutions where a huge amount of information is the main concern. However, for testing, we used only simple data with varying sizes and it is quite likely that SQL would perform better in different scenarios, requiring more complex operations on smaller amounts of information. Having a rigid schema for a database, like provided by SQL, also provides other features, which flexible database simply could not achieve, this means that SQL can never be completely replaced by NoSQL. In the future, it would be a great idea to investigate performance for complex data and different types of data. Furthermore, looking into resource (RAM, CPU and disc space) consumption of each type of database could be advantageous, as that could provide information, that could allow to further optimisation of various solutions, such as IOT for example.

Acknowledgment Thanks to F. E. Nordbjerg and A. Birta for support and help with improving the quality of research.

References

[1] D. D. Chamberlin and R. F. Boyce, “SEQUEL: A Structured English Query Language,” in *ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control*, 1974, pp. 249–264.

[2] S. Weber, “NoSQL Databases,” *HSR Hochschule für Tech. Rapperswil*, pp. 1–8, 2010.

[3] S. Vatika and D. Meenu, “SQL and NoSQL Databases,” *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 2, no. June, pp. 20–27, 2016.

[4] K. L. Berg, T. Seymour, and R. Goel, “History Of Databases,” *Int. J. Manag. Inf. Syst.*, vol. 17, no. 1, pp. 29–36, 2013.

[5] A. Nayak, A. Poriya, and D. Poojary, “Type of NOSQL Databases and its Comparison with Relational Databases,” *Int. J. Appl. Inf. Syst.*, vol. 5, no. 4, pp. 16–19, 2013.

[6] Y. Li and S. Manoharan, “A performance comparison of SQL and NoSQL databases,” in *IEEE Pacific RIM Conference on Communications, Computers, and Signal Processing - Proceedings*, 2013, pp. 15–19.

[7] Z. Parker, S. Poe, and S. V. Vrbsky, “Comparing NoSQL MongoDB to an SQL DB,” in *ACMSE ’13 Proceedings of the 51st ACM Southeast Conference*, 2013, p. 6.

[8] C. Nance, T. Losser, R. Iype, and G. Harmon, “NoSQL vs RDBMS - Why There is Room for Both,” in *Proceedings of the Southern Association for Information Systems Conference (SAIS 2013)*, 2013, pp. 111–116.

[9] D. Bartholomew, “SQL vs. NoSQL,” *Linux J.*, no. July, pp. 54–58, 2010.

[10] E. Mitreva and K. Kaloyanova, “NoSQL Solutions to Handle Big Data,” in *Proc. Doctoral Conference in MIE*, 2015, no. 30 April 2015, pp. 77–85.

[11] solid IT, “DB-Engines Ranking,” 2019. [Online]. Available: https://db-engines.com/en/ranking_trend.

[12] T. E. Oliphant, “Python for scientific computing,” *IEEE*, vol. 9, no. 3, pp. 10–20, 2007.

[13] W. McKinney, “pandas: a Foundational Python Library for Data Analysis and Statistics,” in *PyHPC 2011 : Python for High Performance and Scientific Computing*, 2011, pp. 1–9.

[14] Y. Zhang, N. Meratnia, and P. Havinga, “Outlier detection techniques for wireless sensor networks: A survey,” *IEEE Commun. Surv. Tutorials*, vol. 12, no. 2, pp. 1–12, 2010.

Appendix

The code made for testing the database performance and resulting findings can be found here: https://github.com/bubriks/Review-of-NoSQL-and-SQL