EPiC Series in Computing

Volume 52, 2018, Pages 289–300

ICT4S2018. 5th International Conference on Information and Communication Technology for Sustainability

Data Storage and Maintenance Challenges: The Case of Advanced Metering Infrastructure Systems

Lucas Pereira[1], Rodolfo Gonçalves[2], Filipe Quintal[13], and Nuno Nunes[14]

[1] M-ITI/LARSYS, Funchal, Portugal
lucas.pereira@m-iti.org
[2] Exictos, Funchal, Portugal
orencio.goncalves@exictos.com
[3] University of Madeira, Funchal, Portugal
filipe.quintal@m-iti.org
[4] Técnico, University of Lisbon, Lisbon, Portugal
nunojnunes@tecnico.ulisboa.pt

Abstract

In today's digital age, massive amounts of data are steadily being generated from various sources, such as smart-phones and social media. Smart-Grids are among the fields that are currently experiencing a burst in the data being generated, in part due to the recent investments in Advanced Metering Infrastructure Systems. In this paper, we present a benchmark between MySQL and MongoDB, when used to store and maintain the data that results from Advanced Metering Infrastructure Systems deployments. Our results show that MongoDB clearly outperforms MySQL for reading operations but at a cost of a much larger database size. As such, when deploying such systems, developers should be aware of this important trade-off that may greatly affect the overall experience.

1 Introduction

In today's digital age, massive amounts of data are steadily being generated from various sources, such as smart-phones, and social media. This large amount of data is known as Big Data, one of the most discussed topics in digital information. It can be described as massive volumes of both structured and unstructured data that is so large that it is difficult to process with traditional database and software techniques [20].

Smart electric energy grids (Smart-Grids), are among the fields that are currently experiencing a burst in the data being generated, in part due to the recent investments towards the deployment of Advanced Metering Infrastructure Systems (AMIS) worldwide [17, 10].

Due to the high volume and the non-homogeneous data structures, scalability and query performance are the two main topics to be aware of. For example, in the case of AMIS, it is important to enable that consumers interact seamlessly with their consumption data, while also allowing advanced data analysis on the utility side [2]. Consequently, in recent years there has been an urge to develop scalable and high performing database technologies [7].

B. Penzenstadler, S. Easterbrook, C. Venters and S.I. Ahmed (eds.), ICT4S2018 (EPiC Series in Computing, vol. 52), pp. 289–300

1.1 SQL, NoSQL and different Database Models

A database is an organized collection of interrelated (persistent) data, organized to model aspects of the real world in a way that supports processes requiring information. Databases can be managed through software applications known as Database Management System (DBMS), computer software applications that interact with the end-user, and the database itself to capture and analyze data [8].

Database models are data models that define the logical structure of a database, and how data shall be stored and manipulated. It is a collection of concepts and rules that describe the database structure, such as data types, constraints and relationships among different pieces of information.

Every database application adopts a database model, defining the logical structure of the data. The data model is the biggest determiner of how a database application will work and handle the information it deals with. There are several different database models offering different logical data structures, from which the relational model (RM), commonly known by SQL, clearly stands out as the most used data model in the last few decades.

Relational model and SQL databases, despite being powerful, flexible and the most known and used solutions, have several issues or features that have never been over-crossed or provided. Consequently, a series of new and different systems called NoSQL (Not only SQL) have emerged with the purpose of overcoming some of the barriers imposed by SQL databases, and immediately gained popularity.

NoSQL aims to offer a much more freely shaped way of working with information, providing more flexibility and ease data management. NoSQL systems are known for the schema-less data approach that, unlike the relational model, can handle data with not very well defined structures, supporting structures that are or can become heterogeneous. It has its pros and cons, considering the important and indispensable nature of data.

Due the scale and agility challenges that modern applications face, NoSQL databases have become the first alternative to relational databases, being scalability, availability, and fault tolerance the key deciding factors in satisfying the user needs. Likewise, another important factor in favour of NoSQL databases is the cheap storage and processing power available today [6, 14]. As such, several NoSQL database technologies were developed in recent years in response to the rising need for large data storage, access frequency and greater performance and processing needs.

1.2 Related Work

SQL and NoSQL database technologies, have been the subject of several discussion in the past few years regarding which is the best technology for data storage (e.g., [4, 13]). Yet, the overall conclusion is that the selection of the appropriate technology should be made taking into account the application specifications, and that there is no "one-size-fits-all" approach.

Ultimately, the general conclusion is that choosing the right technology depends of the use-case. If data is continuously changing or growing fast and you need to be able to scale it quickly and efficiently, maybe NoSQL is the right choice. On the other hand, if a data structure is well defined, and it will not change much frequently and data does not grow that much, then SQL is the best answer.

In order to get empirical evidence for such statements, several benchmarks between SQL and NoSQL databases have been conducted (e.g., [19, 5, 11]). Among others, there works suggest that not all NoSQL databases perform better than SQL databases, and that even within NoSQL databases there is a wide variation in the performance of these operations.

Regarding smart-grid data management, a number of benchmarks have been proposed (e.g., [1, 9, 12]), but none addressing the differences between SQL and NoSQL databases.

In this work, we aim to go beyond the theoretical guarantees about data storage technologies, and traditional benchmarks. To do so, we perform an extensive benchmark between SQL and NoSQL databases with a big focus on the particular case of AMIS data.

This benchmark was performed using the SustData public dataset [16], and the different tests were created taking into consideration not only the database technologies, but also the actual purpose of the data. More specifically, we benchmark MySQL and MongoDB, two of the most well-known technologies on the data storage world, using energy consumption data aggregated at 1-minute intervals.

1.3 Contribution

The main contribution of this paper is an in-depth benchmark of these two database technologies according to the following four dimensions: 1) read and write operations; 2) database size; 3) scalability; and 4) data pre-presentation (i.e., fetching the results before presenting them to the users). We selected these dimensions since they provide a accurate combination of what might support a developer when deciding on which database technologies to use with AMIS data.

Read and write performance will affect the usability of a platform and the end-use performance, since it will make any given task faster or slower.

The *database size* will affect the setup and subsequent maintenance and upgrades on the the hosting machine(s). This dimension will also have a significant impact on the cost of hosting, on the account of the strong correlation between disk capacity and cost.

The *scalability* dimension relates with the previous ones, since both the read/write performance and database size will evolve as the database grows, and this is an important factor when designing any cloud platform. Finally, the *data pre-presentation* dimension directly affects the work of the developer, and how the results from a query can be quickly presented to the end-user.

1.4 Paper Outline

This paper is organized as follows: First, we describe the methodology used in this benchmark. When then present and discuss the obtained results. Finally, we discuss the limitations of this work, and outline future work possibilities.

2 Benchmark Methodology

In this section, we thoroughly describe the methodology that was followed in this paper. First we describe the setup used for the benchmark, then we define the data used in the tests, the last three subsections explain the practical implementations used for the benchmarks, the indexing, queries and finally the algorithms used to execute the sequence of queries used in for the benchmark.

2.1 Benchmark Environment Setup

In order to test both technologies under the same conditions, we created two virtual machines (VM) on top of an iMac physical machine. The iMac was a 2009 machine, with the specifications presented in table 1.

	Physical	Virtual Machines
Operating System	OS X Yosemite	Ubuntu 14 (64 bits)
Disk	1 TB Sata HDD	500 GB
RAM	10 GB 1067MHz DDR3	4096 MB
CPU	3.06GHz Core 2 Duo	2 Processors

Table 1: Technical characteristics of the benchmark physical and virtual machines

As for the VM, we used the Oracle VM VirtualBox[1] virtualization software.The two virtual machines were created with the following technical specifications:

We then had to install additional software in each machine, in order to create and execute the benchmarks algorithms. In the "MySQL Machine" we installed XAMPP[2]. This is a very popular, free and open-source cross-platform web-server stack, consisting mainly of the Apache HTTP Server, MySQL database, and PHP interpreters. As for the "MongoDB Machine", we resorted to separate installations of the MongoDB database engine, and the Node.JS[3] javascript runtime.

2.2 Testing Data

The SustData [16] dataset contains about five years of electric energy consumption data. This includes, over 35 million individual records from 50 monitored homes covering electricity consumption logs and demographic information, as well as the energy generation information. The dataset also contains three years worth of environmental data (e.g., temperature and cloud coverage).

In this work we used the energy consumption logs, using data from three different houses. More concretely, we used 15 different sets of data, each containing a different number of records. Figure 1 shows the different volumes of data in each test set. The size of the different sets of data were selected such that it was possible to simulate progressive write and read operations with different amounts of data, until a total of 10 million records is introduced in the database.

2.2.1 Data Representation and Storage

MySQL (and other Relational Database Management Systems) uses a normalized data structure approach for data storage. Nevertheless, since the main type of operations in AMI systems are to read data, our major concern was to optimize the read operations. Therefore, we decided not to normalize the data. Instead we created a schema that supports all the different data structures in a single table (i.e., a materialized view [4]). The structure of the table is shown in figure 2 (a). The different sets of data were then loaded using the SQL code snippet presented in figure 3 (a).

We are fully aware that this approach is not the most common and correct for relational databases since this will result in worst performance for data storage, sizing and scalability.

[1] VirtualBox, https://www.virtualbox.org/
[2] XAMPP, https://www.apachefriends.org/index.html
[3] Node.js, https://nodejs.org/en/
[4] Materialized Views, http://www.fromdual.com/mysql-materialized-views

Yet, this approach has the upper hand on read/write performance since its not necessary to introduce time consuming JOIN statements in the SQL read queries, and all the data is inserted through a single query. In a normalized fashion, to store this kind of data we would need two additional tables. One to store all the different variables (i.e., Current -I-, Voltage -V-, Active Power -P-, Reactive Power -Q- and Power Factor -PF-), and second table holding a *many-to-many relationship* between the house identifiers and the variables table, the different values and the measurement timestamp.

MongoDB is a schema-less JSON-style data storage technology that supports multiple documents with different structures in a single collection. Therefore, having a single collection with only non-null values for all the dataset is not a problem. In our scenario, we used the *mongoimport* tool with the $--$ *ignoreblanks* option set to true (see code snippet in figure 2 (a)) to import all the data, producing documents like the one shown in figure 3 (b).

2.3 Data Queries

We selected five queries that are normally among the most common tasks that need to be performed in an AMI system for feedback purposes. All the selected queries perform aggregation operations in order to produce electric energy usage statistics. The five queries are summarized in table 2.

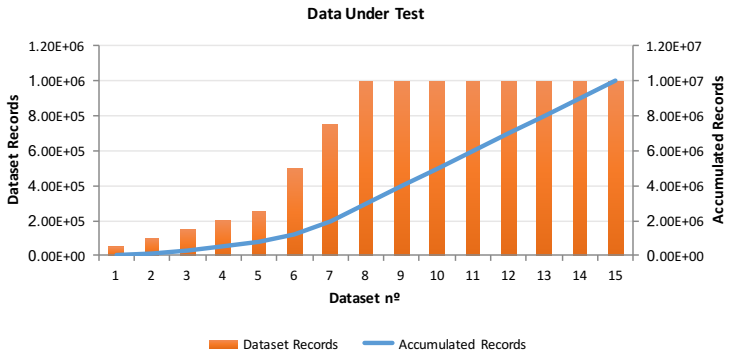


Figure 1: Data under test. The column series represents the individual sets of data, and the line series represents the accumulated data in the database as the different sets are inserted.

Table 2: List of data manipulation queries used in the benchmark

Id	Description
1	Calculate the average active power by hour for a specific house across the entire dataset.
2	Calculate average apparent power (i.e., current x voltage) by hour for a specific house across the entire dataset.
3	Calculate the average power per hour for a specific date and house.
4	Select all the power averages for the three households during a specific week of the year.
5	Sum power averages per hour during a specific month for a specific house.

id	iid	tmstp	deploy
1	1	24/09/11 23:59	2
Imin	Imax	Iavg	Vmin
2,15	2,17	2,23	237
Vmax	Vavg	Pmin	Pmax
240	238,4	507,2	542,4
Pavg	Qmin	Qmax	Qavg
531,6	NULL	NULL	NULL
PFmin	PFmax	PFavg	miss_flag
0,993	1	0,999	0

(a) MySQL

```
{
    "_id" : ObjectId("554932591a3d425d24eead51"),
    "iid" : 1,
    "tmstp" : ISODate("2011-09-14T23:39:46Z"),
    "deploy" : 2,
    "Imin" : 2.14,
    "Imax" : 2.27,
    "Iavg" : 2.23067,
    "Vmin" : 237,
    "Vmax" : 240,
    "Vavg" : 238.424,
    "Pmin" : 507.18,
    "Pmax" : 542.427,
    "Pavg" : 531.607,
    "PFmin" : 0.993068,
    "PFmax" : 1,
    "PFavg" : 0.999552,
    "miss_flag" : 0
}
```

(b) MongoDB

Figure 2: Data representation in the two database engines.

```
LOAD DATA INFILE '".$file_path."'
INTO TABLE power_sample
FIELDS TERMINATED BY ','
(   home_id, timestamp, deploy, Imin, Imax, Iavg,
    Vmin, Vmax, Vavg, Pmin, Pmax, Pavg, Qmin, Qmax,
    Qavg, PFmin, PFmax, PFavg, miss_flag)
```

(a) MySQL

```
mongoimport
    --host localhost
    --port 27017
    --collection power_sample
    --db '+config.db.database+'
    --type csv
    --headerline
    --file '+config.fs.storage_path+file)+'
    --ignoreBlanks
```

(b) MongoDB

Figure 3: Code blocks for inserting data in the two database engines.

2.4 Query Implementation

In MySQL we used the default GROUP BY clause for data aggregation. This clause offers several aggregation functions such as average, sum and count. As for the MongoDB, we used the aggregation pipeline[5], a framework for performing aggregation tasks, modelled on the concept of data processing pipelines.

2.5 Benchmark Logic

To proceed with the actual benchmark, we implemented the same benchmarking algorithm using the two database technologies and server-side technologies. More concretely, MySQL with PHP and MongoDB with Node.JS. The logic of the algorithm is shown in figure 4.

This algorithm is responsible for the two main tasks in our benchmark, namely: i) inserting the different sets of data, and measuring the insertion times, and ii) executing the different queries and measuring the respective execution times.

The five data manipulation queries were implemented and executed against data from three different houses. Each query was executed 10 times in order to minimize the external effects such as processor or memory overhead. Ultimately, this resulted in a total of 150 log records for each individual set of data, producing a total of 2250 records per database technology.

[5]MongoDB Aggregation, https://docs.mongodb.com/manual/core/aggregation-pipeline/

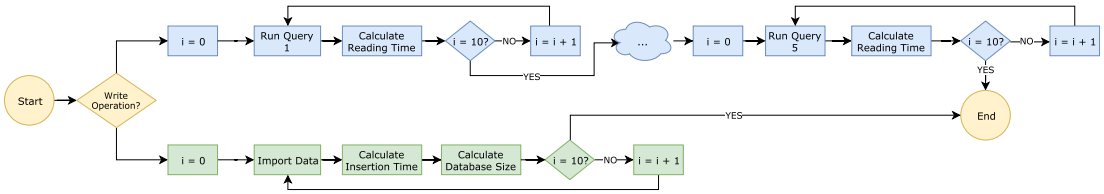


Figure 4: Flowchart of the benchmarking logic use in this paper.

3 Benchmark Results

Once the all the queries were concluded, the collected data was catalogued according to two different groups. Group I contains the insertion queries, and is analysed in terms of the database size and insertion times. Group II contains the data manipulation queries, which are analysed in terms of the time it takes to execute each query.

3.1 Group 1 - Data Insertion

The results from the inserting records with both database technologies were organized by number of records in the database (50k, 100k, 150k, 200k, 250k, 500k, 750k, 1M, 2M, 3M, 4M, 5M, 6M, 7M, 8M, 9M and 10M records). The table size was saved using the MegaByte (MB) size of each database, and the insertion time was recorded in seconds.

Figure 5 (a) presents the insertion times for the different sets of data in MySQL and MongoDB. It is perceptible through the chart that MySQL has faster insertion times than MongoDB. This observation was confirmed through a Mann–Whitney U test where we compared the insertion time of each technology per number of records in the database.

The Mann–Whitney U test was selected since the data was not normally distributed (confirmed through a Shapiro-Wilk test). The statistical test indicated that the insertion time was in fact significantly higher for MongoDB (median=37.32) than for MySQL (median=9.22), U=208, p=0.029.

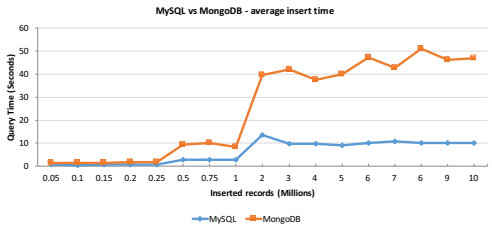
It is also clear from Figure 5 (b), that MySQL can store the same number of records using considerable less space. Similarly to the previous analysis the table size between technologies was also compared with Mann–Whitney U test.

When considering the full sample, this test was not significant (p=0.052), however the low p value suggests that the difference between table sizes is close to significant. We repeated the analysis omitting the result for smaller table size (50k records) and, as expected, the Mann–Whitney U test was significant. This confirms that just after about 50k records, MongoDB will require significantly more space to store the same information (median=1182.56) than MySQL (median=420.96), U=64, p=0.044.

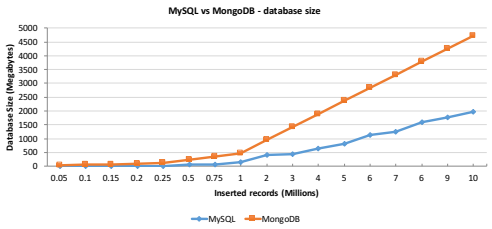
3.2 Group 2 - Data Manipulation

In figure 6 (a) we present the average performance (measured in seconds) of each query in both technologies, taking the average of the three different houses. In this chart, MongoDB data is represented in the series with the light blue colour palate.

It is visible from the chart that from a certain number of records onward, the difference between both technologies greatly increases. To determine the database size (number of records) in which the differences between technologies is significant, 17 Mann-Whitney U tests were

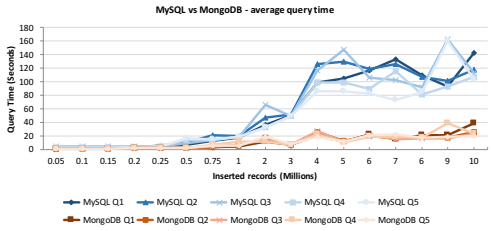


(a) Insertion time

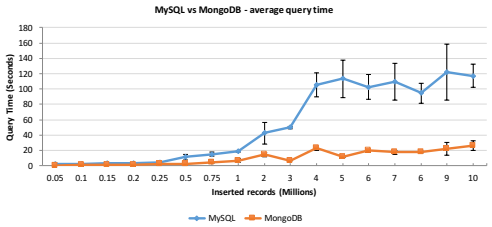


(b) Database size

Figure 5: Charts showing the read performance as more records are added.



(a) Individual Queries



(b) Average

Figure 6: Charts showing how the insertion time and database size changes as more records are added.

performed (one for each database size), considering each of the 5 queries (see table 2). This test returned significant differences for all the database sizes above 50k.

Finally, figure 6 (b) shows the average performance for all the different queries in both technologies. As it can be observed, it is clear that in all the cases, MongoDB is able to query the datasets much faster than MySQL.

4 Discussion and Conclusions

With the benchmark results in place, we are able to make our evaluation over write and read operations in MySQL and MongoDB.

4.1 Write Performance

With regard to data insertion, MySQL consumed 1964 MB to store the total of 10 million records and had an average insertion time of 6.15 seconds for all the different sets of data. On the other hand, MongoDB consumed 4730 MB to store all the 10 million records and took in average 25.26 seconds to insert all of the records. This means that MongoDB takes 240% more disk space and is 410% slower when compared to MySQL.

We should however note that the obtained results for storage size and time could be enhanced if we deleted all the data indexes. In fact, indexing data improves read operations, but compromise writes because every time a record is inserted all the associated indexes must be

updated. Likewise, the decision of not normalizing the data in MySQL costs a non-scalable solution, but it definitely helped achieve better results both for write and read operations.

MongoDB stores data in a $key : value$ format in order to enable the storage of multiple documents with different structures while ignoring empty (i.e., null) fields. This produces more scalable data structures, but consumes more disk space and increases the amount of data to write since every key must be inserted along with its values.

It is also important to refer that the *mongoimport* tool is known for having worse performance than the MySQL LOAD DATA INFILE operation. This happens because MongoDB data is stored in BSON format, as such, most of the effort is devoted to data serialization, since neither JSON or CSV are native MongoDB formats.

This said, we can conclude that MySQL has the best performance for data insertion, since MongoDB consumed about 2.5 times more disk space, and took about 4.1 times longer to conclude the write operations. Yet, it is important to remember that the non-normalized data approach in MySQL is not an optimal solution, since it leads to a lot of effort every time a change in the database is required. A situation that is likely to happen, depending on the type of application.

Our statistical tests revealed significant differences for insert times between technologies, independently of the number of records, as well as statistically significant differences in the size of the database size between both technologies for all the databases expect the smaller one (with 50k records). 50k data-points is not an exaggerated amount of records, especially if we consider studies with high frequency data, or studies with a more distributed sample, for example 10 houses sending one energy consumption point per minute produces more than 50k records in less than 4 days. Therefore we argue that even small studies/interventions should take into account the database technology, or the database size and insert time can become unmanageable.

4.2 Read Performance

With respect to read operations, MySQL produced a global average of about 53 seconds to query the different sets of data. On the other hand, MongoDB presented an overall average of about 10 seconds for the same operations, which is about five times faster than MySQL.

Looking at the databases with different number of records, the statistical analysis confirmed the overall observation of MongoDB being generally faster than MySQL. More concretely, pairwise comparisons revealed significant differences for all the databases sizes except the smaller 50 thousand records size.

The main reason that justifies this difference in performance is the fact that MongoDB stores embedded data into the same document/collection. This way, the data is written in sequential disk positions, which accelerates and reduces the number of round trips to one, since the information can be read all at once. Consequently, because the first disk access is the one that consumes more time (1ms essentially), this detail is important to consider.

4.3 Scalability

Another concern is data scalability, since there is a high chance of having to handle more non-homogeneous data structures in the future. The non-normalized approach for MySQL helped achieved best performance for read/write operations, however this does not offer scalability due to the MySQL relational model. For this specific requirement, MongoDB offers the best options, due to its schema-less feature; any kind of documents can be stored along the same collection without the need for any change in the database structure.

4.4 Fetching

An important task that we did not consider in this benchmark was the time necessary to fetch the query results. At the beginning of the benchmark this task was performed, but the PHP maximum allowed memory size was easily exceeded when fetching MySQL results. Consequently, it was not possible to produce a fair benchmark on this type of operation.

In Node.JS + MongoDB algorithm we did not have this problem. When a MongoDB query is performed through the Node.JS driver, a cursor is returned. Afterwards we can loop the cursor, iterating all the query results. This operation has a singular particularity: the results are loaded in batches until all the results are fetched. This not only reduces the amount of data to be loaded (avoiding the memory exhaustion), but also turns the data access faster since there are smaller chunks of data to return at each time.

One possible alternative to solve this kind of problems in MySQL is to use OFFSET and LIMIT operators or, for instance, the $mysql\_unbuffered\_query$ function. This function enables the query execution without automatically fetching and buffering the resulting rows, in contrast to what $mysql\_query()$ does. But, besides being deprecated since PHP 5.5.0, it has some drawbacks since we cannot use the $mysql\_num\_rows()$ and $mysql\_data\_seek()$ functions on a result-set until all the rows are fetched. Furthermore, we also have to fetch all the resulting rows from an unbuffered SQL query before we can perform a new SQL query within the same database connection.

4.5 Summary

We can summarize our work by looking at the four dimensions we presented in section 3, and how they come together when developing systems for the smart grid. Yet, it is important to state that the best system will very likely combine SQL and NoSQL technologies.

Our analysis disclosed that MongoDB in average outperformed MySQL in read queries, which means that MongoDB could be use to store data that needs to be accessed repeatedly and quickly, for example data to be used by client applications as it will improve the system usability.

On the other hand, data with a fixed structure and to accessed for offline analysis could be stored in SQL which will consume less space and also provide the structure that could be needed that analysis. On the other hand, if dealing with data with heterogeneous structures, MongoDB is possibly the best option, since its heterogeneous nature allows the integration of data from different sources. In this scenario, MongoDB can also be used as a data broker, which could then feed data into relational databases to enable situations where having fixed data structures is important.

Regarding the scalability of the developed platform, it is clear that both technologies have their strengths and weaknesses. For example, MongoDB allows data structure scalability since it does not rely on a fixed structure. On the other hand, MySQL makes a more efficient use of space, which reduces the need for scaling up the physical infrastructure. Hence, a hybrid approach also appears to be the most promising approach regarding a system scalability. For example, the use MongoDB for a datastore and quick reads from the underlying APIs, and MySQL for data staging.

5 Limitations and Future Work

Despite we have achieved our original objectives, there are a number of limitations that should be addressed in future work.

Although we have created symmetric environments for both technologies in order to test them in the exact same conditions using virtual machines, there was no guarantee that the test execution of one technology would not have impact on the other's performance. This happens because both virtual machines are sharing the same physical resources. Furthermore, the Virtual Machines were created on a 8 year world iMac, which inevitably leads to lower performance values. Still, we should stress that our goal was not to quantify the time required by each technology, but instead provide a comprehensive benchmark of both technologies.

As we can see from the results, some of them present some disturbance and consequently, not very linear results. Such disturbance can be justified with the execution of the tests on both technologies in simultaneous. Yet, we cannot claim this with 100% confidence. An alternative would be to create identical Virtual Machines in a cloud-server, or perform the tests on two physical machines with the exact same characteristics. Unfortunately, at the time this work was being conducted, none of these two alternatives was possible.

Furthermore, despite the observed disturbances, given the significant differences in the obtained results we are confident that the final conclusions would be very similar to the ones presented here.

In future work we should consider the possibility of replicating this study with different database models such as object oriented or graph based database, and different database engines within the same technologies. Likewise, in future work it would be important to quantify the amount of energy that is required by the different database technologies, as this is a major concern among the research community [18, 15, 3].

Downloads The source-code and the data that resulted from this benchmark is freely and publicly available at `http://aveiro.m-iti.org/feel/`.

References

[1] M. Arenas-Martinez, S. Herrero-Lopez, A. Sanchez, J. R. Williams, P. Roth, P. Hofmann, and A. Zeier. A comparative study of data storage and processing architectures for the smart grid. In *2010 First IEEE International Conference on Smart Grid Communications*, pages 285–290, 2010.

[2] Zeyar Aung. Database Systems for the Smart Grid. In *Smart Grids*, Green Energy and Technology, pages 151–168. Springer, London, 2013.

[3] Béchir Bani, Foutse Khomh, and Yann-Gaël Guéhéneuc. A study of the energy consumption of databases and cloud patterns. In *Service-Oriented Computing*, Lecture Notes in Computer Science, pages 606–614. Springer, Cham, 2016.

[4] Daniel Bartholomew. SQL vs. NoSQL. *Linux J.*, 2010(195), July 2010.

[5] Alexandru Boicea, Florin Radulescu, and Laura Ioana Agapin. MongoDB vs Oracle – Database Comparison. In *2012 Third International Conference on Emerging Intelligent Data and Web Technologies*, pages 330–335, Bucharest, Romania, 2012. IEEE.

[6] Planet Cassandra. NoSQL Databases Defined and Explained | DataStax Academy: Free Cassandra Tutorials and Training.

[7] Rick Cattell. Scalable SQL and NoSQL Data Stores. *SIGMOD Rec.*, 39(4):12–27, May 2011.

[8] Digital Ocean. Understanding SQL And NoSQL Databases And Different Database Models, February 2014.

[9] D. Ilić, S. Karnouskos, and M. Wilhelm. A comparative analysis of smart metering data aggregation performance. In *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*, pages 434–439, July 2013.

[10] Javier Leiva, Alfonso Palacios, and José A. Aguado. Smart metering trends, implications and necessities: A policy review. *Renewable and Sustainable Energy Reviews*, 55:227–233, March 2016.

[11] Y. Li and S. Manoharan. A performance comparison of SQL and NoSQL databases. In *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, pages 15–19, August 2013.

[12] Xiufeng Liu, Lukasz Golab, Wojciech Golab, Ihab F. Ilyas, and Shichao Jin. Smart Meter Data Analytics: Systems, Algorithms, and Benchmarking. *ACM Trans. Database Syst.*, 42(1):2:1–2:39, November 2016.

[13] Eileen McNulty. SQL vs. NoSQL- What You Need to Know, July 2014.

[14] MongoDB. NoSQL Databases Explained.

[15] Raik Niemann, Nikolaos Korfiatis, Roberto Zicari, and Richard Göbel. Does query performance optimization lead to energy efficiency? A comparative analysis of energy efficiency of database operations under different workload scenarios. *arXiv:1303.4869 [cs]*, March 2013. arXiv: 1303.4869.

[16] Lucas Pereira, Filipe Quintal, Rodolfo Gonçalves, and Nuno J. Nunes. SustData: A Public Dataset for ICT4s Electric Energy Research. In *Proceedings of ICT for Sustainability 2014*, Stockholm, Sweeden, August 2014. Atlantis Press.

[17] James Sprinz. Global Trends in Smart Metering. *Metering and Smart Energy International*, 1(5):16–17, 2016.

[18] Dimitris Tsirogiannis, Stavros Harizopoulos, and Mehul A. Shah. Analyzing the energy efficiency of a database server. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 231–242. ACM, 2010.

[19] J. S. van der Veen, B. van der Waaij, and R. J. Meijer. Sensor Data Storage Performance: SQL or NoSQL, Physical or Virtual. In *2012 IEEE Fifth International Conference on Cloud Computing*, pages 431–438, June 2012.

[20] Pete Warden. *Big Data Glossary*. O'Reilly, 2011.