NoSQL in practice: a write-heavy enterprise application

João Ricardo Lourenço
CISUC
Coimbra, Portugal
jorl17.8@gmail.com

Veronika Abramova
CISUC
Coimbra, Portugal
veronika@student.dei.pt

Bruno Cabral
CISUC
Coimbra, Portugal
bcabral@dei.uc.pt

Jorge Bernardino
CISUC,ISEC
Coimbra, Portugal
jorge@isec.pt

Paulo Carreiro
Critical Software
Coimbra, Portugal
pmcarreiro@criticalsoftware.com

Marco Vieira
CISUC
Coimbra, Portugal
mvieira@dei.uc.pt

***Abstract*—The continuous information growth in current organizations has created a need for adaptation and innovation in the field of data storage. Alternative technologies such as NoSQL have been heralded as the solution to the ever-growing data requirements of the corporate world, but these claims have not been backed by many real world studies. Current benchmarks evaluate database performance by executing specific queries over mostly synthetic data. These artificial scenarios, then, prevent us from easily drawing conclusions for the real world and appropriately characterize the performance of databases in a real system. To counter this, we used a real world enterprise system with real corporate data to evaluate the performance characteristics of popular NoSQL databases and compare them to SQL counterparts. In particular, we present one of the first write-heavy evaluations using enterprise software and big data. We tested Cassandra, MongoDB, Couchbase Server and MS SQL Server and compared their performance while handling demanding and large write requests from a real company with an electrical measurement enterprise system.**

***Keywords*-NoSQL, Big Data, Enterprise, Write-Heavy, MongoDB, Couchbase, Cassandra, SQL Server**

I. Introduction

"Big Data" has recently taken more prominence in the industry, with very high amounts of data in need of constant, quick, and always-available processing [1]. Traditional relational systems are known for their ACID (Atomicity, Consistency, Isolation, Durability) properties and consistency guarantees, but this design choice may often limit their availability and scalability [2]. To counter this scenario, NoSQL systems have been developed. They sacrifice some of the ACID properties, namely consistency, favoring availability [3], resulting in BASE (Basically Available Soft-state services with Eventual-consistency) systems. Indeed, these systems are Basically Available, have a Soft state during which consistency is not yet assured, but then are Eventually consistent [4]. Trading off consistency for availability has resulted in many different NoSQL systems which have different architectures and different use-case scenarios [5]. They have been evaluated multiple times in recent years, but there seems to be a lack of real-world studies with enterprise data or systems. Rather, the NoSQL literature is mostly based on artificial benchmarks such as the Yahoo! Cloud Serving Benchmark (YCSB) [6].

We were given a real enterprise system with a write-heavy workload and big datasets, and asked to evaluate the feasibility of replacing part of its storage backend with a NoSQL based one. This allowed us to test the performance of NoSQL systems in the real world, as opposed to the aforementioned artificial benchmarks, with respect to throughput and storage requirements. As we will show in the rest of this paper, although the workloads of the enterprise system are not representative of big data itself, the limits of the system were being reached, with bottlenecks in the database operation becoming clear. To solve this problem, techniques for handling big data were experimented, in particular NoSQL technology. It is the data model associated with the system that justifies our experiments and research into big data technologies.

In this paper we present the problem, our findings and the conclusions we have drawn from our experiments. Several popular NoSQL systems were tested (MongoDB [7], Cassandra [8], Couchbase Server [9]) and compared with the relational model (MS SQL Server) to assert if they presented any performance gains in a real enterprise scenario. To the best of our knowledge, our work is the first to shed light on a real world write-heavy enterprise scenario, not relying on an artificial system and data. Our results may serve as a bridge to fill in the literature gap between theory and practice, artificial and real, by depicting the impact of NoSQL on a real world enterprise application that handles large amounts of data. Henceforth, our main contribution consists of showing how NoSQL systems – a typical big data solution – behave in a write-heavy enterprise scenario, which made use of relational technology, with regards to performance.

The remaining of this paper is structured as follows. Section II presents a state of the art on NoSQL systems which supports our choice of tested databases, together with related work. Section III introduces the enterprise system we worked with, the problems at hand, the current architecture and our proposed architecture. Section IV explains the experiments we ran and their setup. Section V presents our results and their discussion. Finally, in Section VI we draw the conclusions of our work and future work opportunities, also focusing on the lessons learned from our experiments

and suggesting a set of defining characteristics that NoSQL systems need to better support our enterprise scenario.

II. State of the art

With a large amount of available NoSQL solutions, it is important to be able to distinguish the fittest database solution for a particular system with respect to its distinctive characteristics. All the continuous development and evolution of non-relational technology, over the past years, has contributed to the constant interest in evaluating NoSQL databases. Moreover, until now, all the available studies were highly focused on the performance testing using standard benchmarks [10]. Although those evaluations provide a basic knowledge of the database behavior, there is no performance guarantee while working in a real enterprise environment, where data and interaction are much more random, unpredictable and hard to model [10].

In recent NoSQL evaluations, the authors focus on different possibilities of adapting NoSQL solutions and using those databases along with other domain-specific technologies, such as clinical decision support systems [11]–[13]. In these studies, the authors evaluated different possibilities of integrating NoSQL databases in existing systems where flexibility or big data handling capabilities were needed. They concluded that, according to each system characteristic, NoSQL, in fact, could be a good possibility for data management. In particular, its flexibility and scalability were seen as fitting for the needs of each of these works. One of the drawbacks of NoSQL databases is the learning curve. Another recent trend has been the development of different approaches in terms of data queries. While most developers and DBAs are comfortable and accustomed with SQL, the querying and management of non-relational databases requires more time to learn [14]. More than that, NoSQL technology is known for the non-existence of a querying standard, with different databases having different querying languages [14], [15]. Therefore, [16] and [17] describe querying in NoSQL databases and a possibility of a modeling system that is capable of executing queries regardless of the database. Finally, other authors propose efficient data mining and Big Data processing [18], [19]. Their frameworks are capable of providing better data querying and analysis. In this paper we evaluate the writing performance of NoSQL databases in a real working system. This evaluation will generate new insights on how NoSQL databases perform on systems with such a specific set of requirements.

Further reviewing the literature, one can gather that although there have been many evaluations, with a specific focus on synthetic data, there are few evaluations focused on write-heavy datasets (we note, however, there are some read-heavy evaluations, such as [20], [21] and [22]). Regarding write-heavy workloads, the authors of [23] focus on many scenarios, one of them being "mostly writes", but use the artificial YCSB benchmark. In much the same way, in [24], a write intensive analysis is performed for various NoSQL Databases, but, again, the YCSB framework is used. In, [25], an in-depth analysis of various factors with varying datasets, one of them write-heavy, is made. However, the authors also use artificial frameworks for their tests. When YCSB was first introduced, the authors published results with the benchmark [6], analysing read, write and other characteristics. The results among these papers, for the same databases, tend to vary – a typical example of how NoSQL is constantly evolving. Thus, while there is indeed data on the write-heavy performance of NoSQL, it is mostly artificial, disregarding the enterprise applicability and its context. In that sense, similarly to our work, Zhong et al. [10] present three enterprise scenarios where NoSQL is used. However, unlike our work, their scenarios do not represent mature enterprise systems, and are very much far apart from a write-heavy scenario. Their work, and future developments, might provide valuable contributions to the enterprise evaluation of NoSQL, but it neglects write-heavy scenarios.

III. A real enterprise distributed system

The enterprise system we worked with deals with storing several measurements coming from electrical components, obtained at high sample rates. Data is made up of tuples containing mostly simple floating point values, a timestamp and some additional metadata. This system, as it is currently deployed is hindered by a performance bottleneck during large batches of write-only operations, which are issued to a central database.

By itself, the system uses a distributed architecture. The data is generated by specific equipment and is stored in an accessible way – be it XML or a private database. Another set of independent processes eventually gather the data, do some very simple processing on it and then write it to the central server. Since there are so many records being generated per minute in some of these systems (as many as 3 million values in every sampling timeslice), very high loads of data can be gathered at the same time and quickly sent to be written to the SQL server, resulting in large batches of write operations. These are the operations which cause the bottleneck and performance hit in the system.

The experiments consisted, then, of modifying the database backend of the system to reduce the bottleneck, as well as reduce disk space usage, making use of NoSQL systems. A more detailed view of the currently implemented solution and our proposed changes is presented in the next sections.

A. Original System – Relational

The original system uses a centralized MS SQL Server database where all the gathered data is stored. Furthermore, additional data and metadata needed for other application functionality is stored in other tables in the same database.

Data is gathered from all the necessary sources and is written, in parallel, using batch operations, resulting in hundreds of thousand write operations per second. The data itself is very lightly preprocessed from the original source and, indeed, is merely dumped into a table in this database. While considering a relational system for a large number of inserts it is important to disable indexing and some integrity constraints, something which we took into account when we were performing the experiment.

This approach favours the use of well established relational technology with all ACID properties. However, as we have shown, it is prone to bottlenecks and performance problems, suggesting that an alternative might provide an appropriate trade-off between ACID compliance and performance. In this sense, we proposed a system using NoSQL that is now detailed.

B. Proposed System – NoSQL

We sought to minimize the changes to the current system, and only touch areas that really had to be changed. Furthermore, since the system's bottleneck was in a specific table with many write operations, and not in the rest of the system itself, we decided that the original SQL database should be kept in order to store metadata and other application control data. However, we would take the tables where most of the data was being written to and place them in a NoSQL system. Thus, the data causing the bottlenecks, and the code that handled it, had to be moved and adapted to NoSQL. Given that we were looking for ideal picks for a write-heavy scenario, and that NoSQL's strength lies in large part in its high horizontal scalability [26], it was also decided that we would run a cluster with sharded/partitioned NoSQL databases in order to maximize write throughput. One of the basic NoSQL considerations is the possibility to achieve both performance and throughput by parallelizing the execution of requests. Therefore we exploited this in the setup and, ensuring the network could handle all the data, and that there were no limitations from the source, the system should be able to handle the demand and overall performance would increase.

The original system was already finished and in deployment, which meant that our database choice was conditioned by the system architecture. In this sense, we were limited to databases which would run in a Windows environment and that had .NET bindings. We would also not consider any non-qualified and non-robust code since it could compromise the entire evaluation. Therefore, we chose some of the most popular databases with high support and documentation. To this end, we tested Cassandra, MongoDB and Couchbase Server. One of the drawbacks in satisfying this database criteria was that we were not able to choose any Key-Value Store database – the final chosen solutions were two document stores (MongoDB, Couchbase) and one wide column store (Cassandra). These three choices are among the most popular NoSQL solutions [23], [27], [28] and make up for what we believe is a reasonable sample of the best NoSQL technology.

IV. Experimental Procedure

The goal of the experiment was that of measuring the performance of the databases under a real write-heavy scenario. This was the most important quality attribute of the system, so throughput was measured to assess it.

A. Experimental Setup

A homogeneous cluster using four machines with similar hardware was chosen to host the databases. This choice was cost-effective, adequate to the budget that we were allowed, and also mimicked the real-world scenarios that the system is used in. Two additional machines were used to perform the write-heavy queries, simulating real data sources. Figure 1 depicts the experimental setup, and table I summarizes the machine characteristics.

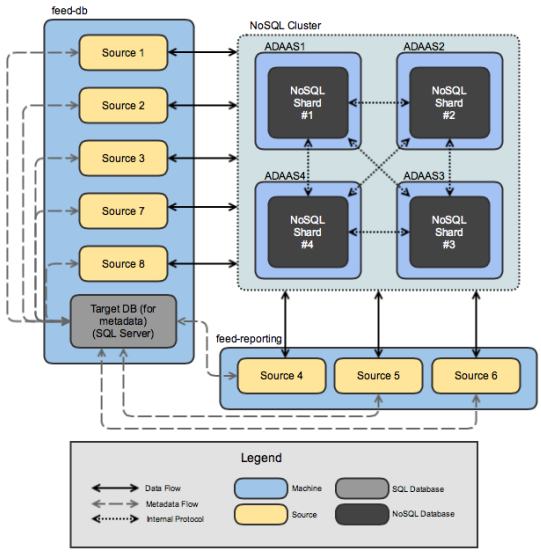

Figure 1. Experimental setup.

In order to ensure that there was no bottleneck in the experimental setup, several tests were made. In particular, we tested network throughput and stability by utilizing all the available bandwidth with traffic generated by an application between the machines, analyzing the throughput and ensuring it did not have spikes. The data sources were also tested for potential bottlenecks by running read-intensive workloads in the source machines and comparing the read throughput to the write throughput of the cluster. Upon comparison, read throughput was higher than write throughput, eliminating the possibility of a bottleneck. When measuring the write throughput with local batch inserts on the clusters and comparing it to network bandwidth, we confirmed that the latter was higher than the former,

Table I
SUMMARY OF THE MACHINE SPECIFICATIONS AND NUMBER OF SOURCES/LOAD GENERATING PROCESSES THEY RAN

CPU + OS	RAM	HDD	# Machines	# Sources
Intel Core i3 3.10GHz Windows Server 2012 R2 64bit (build 9600)	4GB	WD5000AAKX 7200 RPM 16MB Cache SATA 6.0GB/s	4	-
Intel Core i7 2.9GHz Windows Server 2008 Std. 64bit (build 6001)	8GB	N/A	1	5
Intel Core 2Duo 2.53GHz Windows Server 2008 Std. 32bit (build 6001)	4GB	ST9320325AS 5400 RPM 8MB Cache SATA 3.0GB/s	1	3

eliminating possible network bottlenecks. Finally, to make sure that any bottleneck would only be in the database itself, we ran increasingly write-intensive workloads on the database until performance peaked, showing that an increase in workload had a corresponding increase in throughput, without exceeding network bandwidth or disk write throughput.

Since the last two machines (responsible for providing the load to the first four machines) had different characteristics, the number of load generating processes they could run in parallel was also different. We ran initial tests in "ad-hoc" fashion to find the best number for each of these machines, culminating with the total of 8 sources, 5 in one and 3 in another one. These tests involved varying the number of data sources in each machine and evaluating throughput performance. The total of 8 sources, divided in groups of 5 and 3, made the system usable and didn't hinder its performance (using lower or higher values than eight would lead to system performance decrease for our initial tests). Lastly, it is of relevance noting that the machines in the cluster did not have a Solid State Drive (SSD) storage medium.

B. Experimental Methodology

The experiments were performed by running the enterprise system. In each run of the experiment, the source machines had a number of source processes executing write operations on the cluster, as in the real-world scenario (in essence, this simulated the large batches of data being generated on the fly by electrical equipment). All of these sources inserted the same number of records concurrently, putting stress on the databases. Henceforth, the total amount of records being inserted per workload was the sum of all records inserted by these sources. When the insertions were finished, the total insertion time was then measured so that throughput could be calculated. A record structure is presented in Figure 2.

Field	1	2	3	4	5	6	7	8	9	Total
Type	String	int	int	int	Date	String	int	double	Date	
Size (bytes)	36	4	4	4	8	50	4	8	8	126

Figure 2. Record structure of the enterprise system we tested.

Upon measuring the total insertion time, the system was stopped and the databases reset to their initial state, allowing for repetitions of the test or for another test with a different workload to be performed.

During the adaptation of the code for the several NoSQL solutions, we tried to use database best practices, for instance using batch inserts wherever possible. However, we noticed that current NoSQL databases don't seem to have a wide support for large batch insert operations, something which needs to be improved if they are to ever be used with this kind of dataset. While the operations themselves are supported, performance gains are lower than expected, documentation was scarce and code maturity seemed low overall. Nevertheless, they did provide speedup to the operations.

C. Workloads and Database Configurations

The testing procedure described in the previous section was used for a single run. In the experiment, multiple runs were executed to attain statistical significance. In particular, 10 tests were used and the results were averaged. This value was chosen as the best trade-off between the time to run all tests and the significance of the results – with 10 tests, the confidence interval for the throughput had an amplitude of roughly 5% of the total throughput, with some exceptions for higher throughput values, where the confidence interval amplitude peaked at approximately 17% (see Figure 3 where these results can be seen in the form of error bars). If a higher number of tests had been used, there would be little to gain in statistical significance, as the confidence interval amplitude was already quite low. In contrast, a lower number of tests would increase the amplitude and hinder the test significance.

In spite of using 10 tests for averaging the result, the effective number of tests run for each experiment was 13. The first three tests were discarded to simulate a real world environment and, thus, remove cache side-effects (resulting from a cold-boot), meaning that the warm-up time was given by the first three executions.

Regarding the workloads, each of the thirteen tests was executed for 10.000 records per source (i.e., in total, 80.000 records). After that, this procedure was repeated for 20.000 records, 30.000, etc, until 330.000 per source. This way the impact of varying database load on throughput was measured, as intended. We chose 10.000 records based on real-world data and scenarios, as recommended by the system developers. The increment was also 10.000 because

it was concluded, from initial testing, that a lower value did not produce significant throughput change and was, thus, not of much importance to analyse. Furthermore, throughout our tests there seemed to be no need to adjust this uniform increment or focus on any particular domain of records per source, as the throughput showed no irregular behavior and, instead, seemed to be modeled by a linear relationship with the number of records. The 330.000 limit was a consequence of our setup limitations – it was the highest load we could cope with for most tested databases.

NoSQL databases are known for their configurability and tuning [29]. To maximize their performance and adapt them to this scenario, we adjusted their configuration settings.

We used Cassandra version 2.0.9, adjusting all timeouts to 90 seconds, with the exception of the range timeout, whose value was unchanged. This value allowed all tests to finish successfully, since the original workloads would trigger a timeout in the database. Row caching was disabled to maximize write throughput. The concurrent reads and writes settings were adjusted as suggested by the documentation, using the $8 \times number\_of\_cores$ rule of thumb. The memtable flush writer threads setting was set to 1, given that we had no SSDs and only had one data directory. The remaining settings were left at their default values for the results presented in this paper. However, it should be noted that additional tests were made by varying other parameters such as the RPC server type and other memtable settings. Since these changes produced no measurable difference in performance, their default values were used.

MongoDB version 2.6 Standard was tested with the default options and the highest write concern settings. A setup with two config servers and one query router was chosen because it provided the minimum amount of mongod instances that had to be executed, allowing focus to shift to the sharding itself. For sharding to take place, the hashing of the id was used, as suggested by the MongoDB guidelines. This ensured a random distribution of data across nodes.

Couchbase Server was tested with version 3.0.1-1444 and configured to minimize the use of compaction, in order to delay performance degradation when performing IO. The number of writer threads was also increased in comparison to the number of reader threads. However, these configuration settings had little effect in performance. Instead, to achieve a 3-fold performance increase, when compared to our initial testing, the batch insertion method was adapted by using multi-threaded REST requests. The number of threads and records per thread was chosen to maximize performance, after testing several values to see which had th best performance. The resulting settings were 25 threads of 1000 records each (pe source).

Regarding the relational engine, SQL Server 2014 Standard edition was used. The server was used with full ACID guarantees, as is the case in real deployments, but with all indexing facilities turned off to maximize write throughput. This server was only executed on one node, as we shall discuss in the next section (Section V).

All databases were assigned 2GB of the available memory. This was the most memory that could be assigned without damaging normal system operation. Furthermore, since write-performance was the focus of the experiment, all indexing capabilities were disabled wherever possible.

V. Experimental Results

We now present the results of the experiment, as well as our analysis. The limitations of the experiment are also presented and discussed.

A. Results and Analysis

The results (see Figures 3 and 4) show that the relational/original solution provided a much higher throughput than the NoSQL solution with sharding. The throughput of MS SQL Server, whilst running on a single node, is up to five times that of the best NoSQL solution. Indeed, Cassandra performs the best out of the three tested NoSQL systems by a fairly large margin (see Figure 4). These results were predictable, since the other two tested databases are document stores, fine-tuned for read operations [29]. Cassandra, on the other hand, is write-optimized [6], [29]. Nevertheless, in spite of outperforming its NoSQL counterparts by a factor 4, it is itself outperformed by the original system running MS SQL server on a single node. Although one might be tempted to think that the tested workloads were not adequate for the NoSQL systems being tested, we note that MongoDB, for instance, was unable to handle the larger workloads, indicating that the choice of workload, for our experimental setup, was indeed adequate.

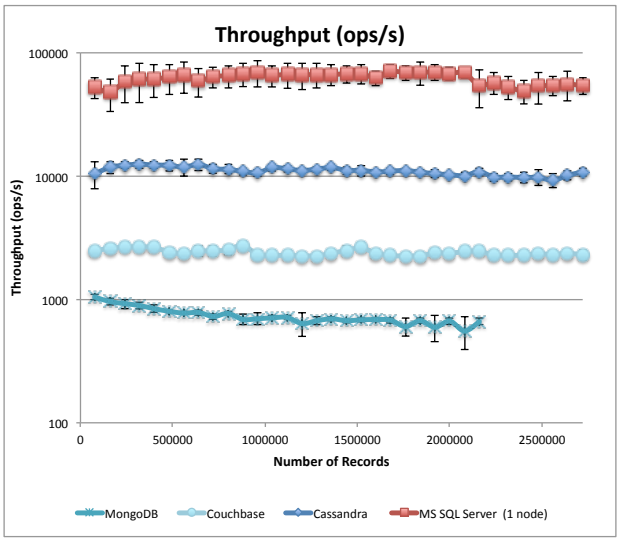

Figure 3. Throughput results for all tested databases. Error bars show standard deviation. There are missing values for MongoDB because it could not finish the test and handle higher workloads.

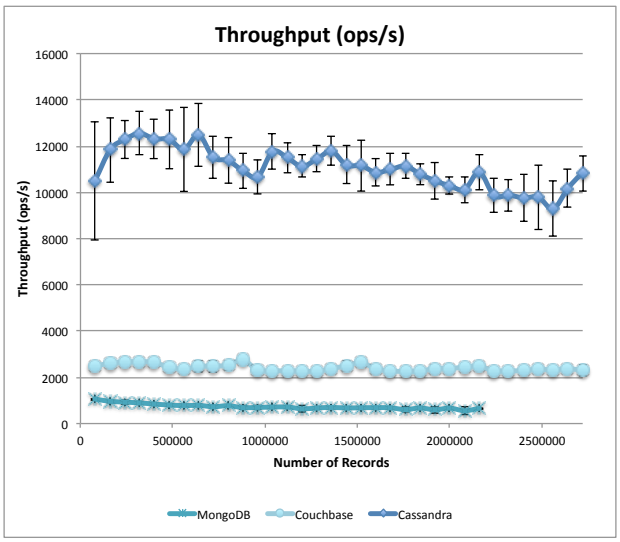

Figure 4. Throughput results only for NoSQL databases. Error bars show standard deviation. There are missing values for MongoDB because it could not finish the test and handle higher workloads.

Cassandra's throughput is more irregular than that of the other NoSQL solutions, showing a decreasing trend with increasing workloads. While the other NoSQL databases also show this trend (although to a lesser extent than Cassandra), MS SQL Server does not seem to suffer as much from performance degradation with increasing workloads, although such degradation can still be seen.

MongoDB, as mentioned, was unable to cope with the highest workloads, which can be seen in Figures 3 and 4. When a certain high workload was reached, it began displaying errors due to internal problems because of the large overload of data. This result, although somewhat surprising (in the sense that a database crash is not expected behavior), can be attributed to the exhaustion of all machine resources – MongoDB ran out of resources, whereas other databases did not. Of the three NoSQL databases, it showed the lowest performance, with a very low throughput when compared to SQL Server. Even taking into account that this database is document-based and, thus, read-oriented, this result was somewhat surprising.

Couchbase has the most stable throughput among all the databases tested, although it is much lower than Cassandra and SQL Servers'. As we have already seen, this is expected due to the fact that it is a document-based database. However, contrary to MongoDB, it can cope with the high workloads.

The "flat-line" results shown in Figures 3 and 4 could be interpreted as some bottleneck in the experiment other than a database bottleneck. We dismiss this idea for two reasons: Firstly, because, as explained in Section IV, extensive tests were executed to assure there was no other possible bottleneck; secondly, because the results make sense if we take into account that our initial workload is already stressing the system – this was intended, because it reflects the real-world enterprise system, where lower workloads do not exist and would thus be meaningless to test.

We conclude that a cluster with sharding enabled, made up of a small number of machines (in our case, 4) running any of these popular NoSQL solutions cannot outperform the existing SQL Server based solution with ACID guarantees. It might be the case that there is a high overhead due to internal protocol communications and that disabling sharding would actually improve performance, instead of doing the reverse, but this is matter for future work.

B. Result Limitations

While our results intend to forward the study of NoSQL in the real-world, they are not without their limitations. Most NoSQL databases are optimized for Solid State Drives (SSDs) and might perform poorly on systems which do not have them [30], [31]. In this particular scenario, our cluster did not have SSDs, meaning it wasn't one for which these NoSQL systems were fine-tuned. The real-world application we are studying is not deployed over SSDs, so to keep our experiments as close to reality as possible, we decided to apply the same restriction on our experimental setup.

Similarly, it is a well-known fact that NoSQL systems tend to be more optimized for UNIX-like operating systems, in particular Linux [30], [31]. Due to our application requirements, we had to use a Windows system. We believe that this might impact the generality of our results but provides important information regarding performance of NoSQL systems on this platform. It would no doubt be interesting to rerun our experiment on other operating systems and perform a comparative performance analysis.

Another key factor in this study regards the relatively small size of our cluster. As we have discussed previously, for such a small number of nodes, the protocol overhead might have hindered the performance of the NoSQL databases[1]. There seems to be a trade-off inherent to the architecture of the NoSQL database, in the nature of its internal protocol and storage, between the performance gain in adding nodes to a cluster, and the protocol overhead resulting from that. These claims are in line with the findings and reasoning of [32] and [33], but need further research. It is possible that a larger cluster size could have rendered better results if the performance gains of more nodes outweighed the protocol overhead, but this experiment alone is not sufficient to study this, leaving room for future work.

Still on the topic of the cluster size, one should also question what the outcome of running MS SQL Server in a

[1]Further tests which we do not present in this paper hinted that running these NoSQL systems on a single-node system increased their throughput, strengthening this idea that there was a protocol overhead, or some other kind of overhead related to sharding.

clustered environment would be, and how our experiment does not inspect this scenario. It may be the case that the whole reason that NoSQL showed poor results when compared with relational technologies is simply due to the aforementioned protocol overhead. By testing SQL Server in cluster mode, it would at least be able to achieve a "fair ground comparison", although we highlight that we did not test this due to the nature of the problem we had at hands – the experiment only compared these NoSQL databases with MS SQL Server to assert the performance benefits of using NoSQL solutions in the enterprise system. In that sense, then, the single-node SQL Server was the baseline test, which would ideally improve by using NoSQL technology.

In spite of the testing setup being based in a real world scenario, contrasting with artificial benchmarks such as YCSB, the specific nature of our records might have an influence in the tests. The records at hand were very simple with no particular schema associated other than a set of integers, doubles and dates. In other scenarios, NoSQL might provide more performance, for instance by exploiting the nature of web-based data [5], which is the basis for MongoDB and Couchbase's document oriented model [5]. The lack of a fixed schema, common to most NoSQL databases, could give performance benefits, and this characteristic is not taken into account into this work.

Lastly, due to the limitations in the resources we had for our experiments, the number of concurrent applications sending records was fixed at 8.

VI. Conclusions and Future Work

NoSQL is constantly evolving, rendering past evaluations obsolete in short time spans. This work not only fills in a gap in what concerns the link between theory and practice, but it also serves as a new up-to-date evaluation of NoSQL systems which, due to their constant evolution, quickly render past evaluations obsolete. In particular, it is, to our knowledge, the first real world test of a write-heavy enterprise application, especially one involving big data.

Our results show that a relational database engine, running on a single node, can outperform popular NoSQL solutions running in a cluster of 4 nodes with sharding by a factor of five when dealing with a write-heavy scenario. Whether this result stems from protocol overhead in the cluster or due to the more developed nature and age-tested technology of relational databases is a question for future work. If, indeed, protocol overhead is at the root of these results, then increasing the cluster size might also increase performance without adding so much overhead, whereas reducing the cluster size might reduce protocol overhead and also provide better results. When faced with a scenario such as the one in our enterprise application, however, these results would recommend to keep the SQL Server and not opt by a NoSQL cluster.

From our work, we might extract some valuable lessons regarding NoSQL and write-heavy datasets. Indeed, if one is looking for a good NoSQL solution for their write-heavy dataset, it becomes clear that a column-family database such as Cassandra will provide the best results, in spite of sacrificing some ACID properties. The protocol overhead involved in the communication inside the cluster, which we believe is one of defining factors of our results, should also be minimized and fine-tuned for large batches of write requests, but we note that this must be the subject of future work. Furthermore, NoSQL databases should have a wider support for large batch insert operations if they are to ever be used with this kind of dataset.

As future work, we propose to complement the original experiment by using Solid State Drives and machines with more resources, as well test SQL Server itself in a clustered environment. Further testing regarding cluster size might also advance the state of the art on NoSQL's practical applications. The impact of the record size is a factor not yet thoroughly reviewed in literature, and has high practical applications [34], thus being a good candidate for future work. The number of sources is an interesting parameter that should be studied with other experimental setups. It is our belief that increasing the number of concurrent applications will only decrease the throughput by saturating the network connection and the databases' request handling mechanisms. In much the same way, decreasing the number of concurrent applications should increase the throughput, a direct consequence of decreasing the number of concurrent records inserted per second. Nevertheless, we leave this experiment for future work. The issue of protocol overhead has been mentioned before and should be subject of future work – is there some cluster setup where protocol overhead outweighs the performance speedup? Lastly, it would be interesting to compare the results of our real-world benchmark with those of YCSB using the same configuration.

Acknowledgment

This research would not have been made possible without support and funding of the FEED – Free Energy Data and iCIS – Intelligent Computing in the Internet Services (CENTRO-07 - ST24 – FEDER – 002003) projects, to which we are extremely grateful.

References

[1] C. P. Chen and C.-Y. Zhang, "Data-intensive applications, challenges, techniques and technologies: A survey on big data," *Information Sciences*, vol. 275, pp. 314–347, 2014.

[2] N. Leavitt, "Will nosql databases live up to their promise?" *Computer*, vol. 43, no. 2, pp. 12–14, 2010.

[3] A. Moniruzzaman and S. A. Hossain, "Nosql database: New era of databases for big data analytics-classification, characteristics and comparison," *arXiv preprint arXiv:1307.0191*, 2013.

[4] E. A. Brewer, "Towards robust distributed systems," in *PODC*, vol. 7, 2000.

[5] R. Hecht and S. Jablonski, "Nosql evaluation," in *International Conference on Cloud and Service Computing*, 2011, pp. 336–41.

[6] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, 2010, pp. 143–154.

[7] (2015) Mongodb. [Online]. Available: http://www.mongodb.org

[8] (2015) The apache cassandra project. [Online]. Available: http://cassandra.apache.org

[9] (2015) Couchbase server. [Online]. Available: http://www.couchbase.com

[10] T. Zhong, K. Doshi, X. Tang, T. Lou, Z. Lu, and H. Li, "Big data workloads drawn from real-time analytics scenarios across three deployed solutions," in *Advancing Big Data Benchmarks*. Springer, 2014, pp. 97–104.

[11] Z. Chen, S. Yang, S. Tan, L. He, H. Yin, and G. Zhang, "A new fragment re-allocation strategy for nosql database systems," *Frontiers of Computer Science*, pp. 1–17.

[12] P. Moore, T. Qassem, and F. Xhafa, "'nosql'and electronic patient record systems: Opportunities and challenges," in *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2014 Ninth International Conference on*. IEEE, 2014, pp. 300–307.

[13] M. Mazurek, "Applying nosql databases for operationalizing clinical data mining models," in *Beyond Databases, Architectures, and Structures*. Springer, 2014, pp. 527–536.

[14] M. Stonebraker, "Sql databases v. nosql databases," *Communications of the ACM*, vol. 53, no. 4, pp. 10–11, 2010.

[15] ——, "Stonebraker on nosql and enterprises." *Commun. ACM*, vol. 54, no. 8, pp. 10–11, 2011.

[16] M. Bach and A. Werner, "Standardization of nosql database languages," in *Beyond Databases, Architectures, and Structures*. Springer, 2014, pp. 50–60.

[17] R. Sellami, S. Bhiri, and B. Defude, "Odbapi: a unified rest api for relational and nosql data stores," in *Big Data (BigData Congress), 2014 IEEE International Congress on*. IEEE, 2014, pp. 653–660.

[18] R. Lomotey and R. Deters, "Terms mining in document-based nosql: Response to unstructured data," in *Big Data (BigData Congress), 2014 IEEE International Congress on*, June 2014, pp. 661–668.

[19] R. K. Lomotey and R. Deters, "Terms mining in document-based nosql: Response to unstructured data," in *Big Data (BigData Congress), 2014 IEEE International Congress on*. IEEE, 2014, pp. 661–668.

[20] T. Dory, B. Mejías, P. Van Roy, and N.-L. Tran, "Comparative elasticity and scalability measurements of cloud databases," in *Proc of the 2nd ACM symposium on cloud computing (SoCC)*, vol. 11, 2011.

[21] J. S. van der Veen, B. van der Waaij, and R. J. Meijer, "Sensor data storage performance: Sql or nosql, physical or virtual," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. IEEE, 2012, pp. 431–438.

[22] L. A. B. Silva, L. Beroud, C. Costa, and J. L. Oliveira, "Medical imaging archiving: A comparison between several nosql solutions," in *Biomedical and Health Informatics (BHI), 2014 IEEE-EMBS International Conference on*. IEEE, 2014, pp. 65–68.

[23] Datastax, "Benchmarking top nosql databases:a performance comparison for architects and it managers." White Paper.

[24] B. G. Tudorica and C. Bucur, "A comparison between several nosql databases with comments and notes," in *Roedunet International Conference (RoEduNet), 2011 10th*. IEEE, 2011, pp. 1–5.

[25] A. Floratou, N. Teletia, D. J. DeWitt, J. M. Patel, and D. Zhang, "Can the elephants handle the nosql onslaught?" *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 1712–1723, 2012.

[26] J. Pokorny, "Nosql databases: A step to database scalability in web environment," pp. 278–283, 2011. [Online]. Available: http://doi.acm.org/10.1145/2095536.2095583

[27] C. Strauch, U.-L. S. Sites, and W. Kriha, "Nosql databases," *Lecture Notes, Stuttgart Media University*, 2011.

[28] D. Hammes, H. Medero, and H. Mitchell, "Comparison of nosql and sql databases in the cloud," *Southern Association for Information Systems (SAIS) Proceedings. Paper*, vol. 12, 2014.

[29] S. K. Gajendran, "A survey on nosql databases," *University of Illinois*, 2012.

[30] P. Menon, T. Rabl, M. Sadoghi, and H.-A. Jacobsen, "Cassandra: An ssd boosted key-value store," in *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*. IEEE, 2014, pp. 1162–1167.

[31] J. Baron and S. Kotecha, "Storage options in the aws cloud," *Amazon Web Services, Washington DC, Tech. Rep*, 2013.

[32] N. Kaviani, E. Wohlstadter, and R. Lea, "Cross-tier application and data partitioning of web applications for hybrid cloud deployment," in *Middleware 2013*. Springer, 2013, pp. 226–246.

[33] R. P. Martin, A. M. Vahdat, D. E. Culler, and T. E. Anderson, *Effects of communication latency, overhead, and bandwidth in a cluster architecture*. ACM, 1997, vol. 25, no. 2.

[34] C. Ordonez and P. Cereghini, "Sqlem: Fast clustering in sql using the em algorithm," in *ACM SIGMOD Record*, vol. 29, no. 2. ACM, 2000, pp. 559–570.