# Determining a consistent experimental setup for benchmarking and optimizing databases

Moisés Silva-Muñoz
moises.silva.munoz@ulb.be
IRIDIA-CoDE, Université Libre de Bruxelles (ULB)
Brussels, Belgium

Gonzalo Calderon
gcalderon@cedint.upm.es
CeDInt-UPM, Universidad Politecnica de Madrid (UPM)
Madrid, Spain

Alberto Franzin
afranzin@ulb.ac.be
IRIDIA-CoDE, Université Libre de Bruxelles (ULB)
Brussels, Belgium

Hugues Bersini
bersini@ulb.ac.be
IRIDIA-CoDE, Université Libre de Bruxelles (ULB)
Brussels, Belgium

## ABSTRACT

The evaluation of the performance of an IT system is a fundamental operation in its benchmarking and optimization. However, despite the general consensus on the importance of this task, little guidance is usually provided to practitioners who need to benchmark their IT system. In particular, many works in the area of database optimization do not provide an adequate amount of information on the setup used in their experiments and analyses. In this work we report an experimental procedure that, through a sequence of experiments, analyzes the impact of various choices in the design of a database benchmark, leading to the individuation of an experimental setup that balances the consistency of the results with the time needed to obtain them. We show that the minimal experimental setup we obtain is representative also of heavier scenarios, which make it possible for the results of optimization tasks to scale.

## CCS CONCEPTS

• **Information systems** → **Database performance evaluation**;
• **Software and its engineering** → *Empirical software validation.*

## KEYWORDS

Databases, benchmarking, optimization, experimental setup, automatic configuration

**ACM Reference Format:**
Moisés Silva-Muñoz, Gonzalo Calderon, Alberto Franzin, and Hugues Bersini. 2021. Determining a consistent experimental setup for benchmarking and optimizing databases. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3449726.3463180


*GECCO '21 Companion, July 10–14, 2021, Lille, France*

ACM ISBN 978-1-4503-8351-6/21/07…$15.00
https://doi.org/10.1145/3449726.3463180

## 1 INTRODUCTION

IT systems are at the heart of many industrial applications, and their performance is a central element in the good operation of a company. Benchmarking is therefore a key task to ensure the functioning of the IT system. It is, however, a complex task where several hardware and software factors come into play, and that requires lengthy evaluations, performed either on the real system or on its simulation, to account for the intrinsic stochasticity of each operation. Databases are a perfect example, due to the ubiquitous use of data in many real-world applications that has enabled new computing paradigms and new possibilities in many sectors from industry to transportation to medicine [10, 18, 21, 22, 26].

The performance of a database depends not only on its internal mechanisms, but also on several factors such as the amount and type of data stored and accessed, the amount and type of operations performed, and the hardware infrastructure [4, 5, 8]. For example, caching is used to speed up operations on frequently accessed records, so an evaluation process that does not take this effect into account is at risk of misrepresenting the actual performance. Several other elements contribute to the performance of a database, such as the optimization of its design, or of the queries to be executed [24, 30, 46]. The parameter configuration is another important factor in the performance of a database [7, 29]. For example, Internet of Things applications continuously receive lots of data and thus frequently perform write operations [17]; conversely, the database underlying a web application performs a higher amount of read operations. If the same database is employed, these two applications will require a different configuration.

One common method to measure the performance of a database is to run some benchmarking tool for an extended amount of time or performing a high given number of operations, in order to factor out the stochasticity in the evaluation and obtain reliable measurements [11, 32, 44]. On the other hand, selecting the best database configuration for a certain task requires hundreds or thousands of evaluations, and therefore we strive for an evaluation to be as fast as possible [25, 27]. These two goals are, of course, conflicting, and a trade-off has to be found between speed of evaluation and consistency of the measurement. As we will see in Section 4.1.6, an improper experimental setup renders the optimization process of a database extremely unreliable and, therefore, useless. Unfortunately, despite a vast literature on database optimization (e.g.

[31, 32, 41, 47], few works properly detail the experimental setup used in their experiments. This makes therefore impossible not only to properly reproduce the results, but also to evaluate the real efficacy of the method proposed.

In this work we describe an empirical procedure to design an experimental setup that balances speed and consistency of the evaluations. In particular, through a series of experiments we determine the amount of data and operations necessary to optimize running time and consistency, and evaluate the impact of the hardware configuration on the performance of the database. We then demonstrate the importance of the experimental setup in obtaining reliable results, showing how it allows to obtain good results in terms of both performance and scalability. We also demonstrate the issues that arise when using an inadequate experimental setup, which makes the evaluations faster but unreliable. In our experiments we consider two popular NoSQL databases, Cassandra and Elasticsearch, and we use YCSB as benchmarking tool [9, 11, 19]. To optimize the parameter configuration we use the irace configurator [28].

In the next section we review the some works about database benchmarking and optimization. In Section 3 we describe the software packages used in our experiments. In Section 4 we report our experiments to determine the setup, and evaluate it. In Section 5 we present our conclusions and outline future research directions.

## 2 RELATED WORKS

There is consensus among researchers on the need to define benchmarks for databases and big data applications [5]. However, in practice this is not easily defined. This task is particularly complex for NoSQL databases because this definition includes systems that can be divided into four categories, according to their different data storage designs [4, 23].

Therefore, comparing the performance of these systems is very complex due to their heterogeneity [12]. In fact, many existing tools have been designed for specific databases, such as Rally [15] or Cassandra Stress [3], or for a particular category of NoSQL databases (XDGBench [13], YCSB++ [33]). However, in recent years generalist approaches such as YCSB [11] or TPC [40] have been introduced, allowing to easily test the performance of various databases, and establishing themselves as the de facto reference solution [2, 35]. All these tools, however, do not provide practical guidelines on how to properly use them in practice, because it depends also on the specific application for which the database will be used.

Benchmarking a database is useful not only to measure its performance or to select the database more suitable for a certain task, but it is also a fundamental operation when optimizing a database [47]. Surprisingly, however, this step is often overlooked in works in this line of research, and little details are usually provided. For example, in [45] it is only mentioned that the data samples are collected every ten minutes, while in [32] the sampling time is five minutes. The experimental section of [44] reports the general infrastructure and the time of each evaluation, but not the specific load. While we can hypothesize that the authors of these works consider a setup representative of the typical load of their applications, in both cases there is no indication on what is the actual load considered. Other works such as [47] do not describe their evaluation setup. Even in works where the setup is described and reproducible, such as [41], its significance is neither analyzed nor discussed.

## 3 MATERIAL AND METHOD

### 3.1 Cassandra

The Cassandra database was originally designed by combining some features of Amazon's Dynamo and Google's Bigtable in order to handle large amounts of unstructured data. This distributed database has a wide-column data store model, using columns as the basic unit of data and structuring the data following the concept of column families [9, 42]. The creation and updating of the database schema as well as the access to the data is possible through a SQL-like language provided by Cassandra called Cassandra Query Language (CQL).

The main reasons for choosing Cassandra for our work are as follows: first, Cassandra is currently one of the most popular NoSQL databases which allows us to directly compare our results with other automatic parameter tuning methods as well as showing us the great number of domains in which Cassandra and our methodology could be used. Second, Cassandra has a large community of users and a very complete documentation, which facilitates the generalization and replication of the methodology presented in this work.

We consider five parameters that impact the performance, as identified in [32]: number of concurrent writes (integer), file cache size (integer), memtable cleanup rate (real-valued), concurrent compactor strategy (categorical), compaction strategy (integer). Additional experiments include instead the full list of 23 parameters, as described in [36].

### 3.2 Elasticsearch

Elasticsearch is a distributed and open-source search engine built on Apache Lucene [6, 19]. The information is stored in Elasticsearch in collections of structured or unstructured JSON documents called indices. A document is a set of fields organized in key-value pairs. Elasticsearch is schema-free, which means that the documents are indexed without the need of defining a structure in advance.

Elasticsearch provides a full query Domain Specific Language (DSL) to perform `create`, `read`, `update` and `delete` (CRUD) operations, searches and aggregations [16]. The data indexed in Elasticsearch is also exposed through a RESTful API supported by most programming languages. Elasticsearch is a popular choice for text search engines, due to its efficiency in performing complex full-text searches in real-time.

Elasticsearch provides several configuration parameters to improve performance [14]. Here we consider a subset of five integer parameters that have an impact on indexing and search [38]: index buffer size, minimal index buffer size, cache size, number of replicas, and number of shards.

### 3.3 YCSB

One of the most popular benchmarks is the Yahoo! Cloud Serving Benchmark (YCSB), which allows to evaluate the performance of both relational and NoSQL database management systems [1, 2, 11]. With YCSB it is possible to define the characteristics of the data to be stored in the database, for example specifying the length and

type of each record, and to load a dataset into a database, defining the distribution of the data during the insertion.

This benchmark also allows to define workloads, that is, sets of operations to be performed during an evaluation, specifying the amount and type of operations to be executed. The operations available to YCSB are (i) `read`: get the value contained in a register, (ii) `update`: overwrite the value of a register already present in the database, (iii) `scan`: read all or a subset are read of records in a table, (iv) `insert`: add a new record to the database, and (v) `read-modify-write`: the value of a record is altered by executing three successive operations as an atomic operation. YCSB also has a set of six default workloads. In this work we consider the default YCSB Workload A, which has a 50/50 read and update ratio.

### 3.4 Experimental procedure

We test each database using YCSB Workload A, under varying loads of data and operations. This step is necessary to trade-off speed and consistency. The metric we consider in our tests is the throughput, the rate of transactions processed by the database per second, one of the most common metrics to evaluate a database system. Having established the size of each experiment, we estimate the impact of different configurations, in terms of number of machines and concurrent threads used, which mimic the use of the database by different users. We also test the impact of destroying and reloading the database in between experiments, since not all the database alterations (e.g. in the configuration) cannot be performed while the database is running, and to test the impact of the startup phase (cold experiments, [34]).

We use Cassandra version 3.6, Elasticsearch version 7.9.0, YCSB version 0.17 and irace version 3.4. All the experiments are performed on Google Cloud machines using `n1-standard-8` machines, with 8 virtual CPUs, 30GB of RAM and a 20GB persistent disk. The database records for Cassandra are ten `varchar` fields of 100 bytes each. The Elasticsearch records are Java strings of 100 characters that Elasticsearch internally converts to a text field. Each evaluation is repeated ten times separately for statistical evaluation.

## 4 EXPERIMENTAL RESULTS

### 4.1 Experimental setup for Cassandra

In this section we expand the analysis performed in determining the experimental setup for the configuration of Cassandra using irace we performed in [36]. Part of the experiments reported here were first presented in that work.

*4.1.1 Number of operations.* What is the minimum number of operations required to run representative experiments? To answer this question we test YCSB workload A with Cassandra's default settings, leaving the amount of data fixed at 100K database rows[1] and varying the number of operations. With this we want to find the minimum number of operations that would be representative of larger experiments. Figure 1 shows the results of experiments with 10K, 50K, 100K, 500K and 1M operations. We can see that the results in terms of throughput (left plot) for 100K, 500K and 1M operations are very similar in terms of performance and consistency. On the other hand, in terms of time (right plot) the results for up to 100K operations are similar to the smallest experiments. This indicates that runs with 100K operations are as good as larger runs in terms of performance and just as fast as smaller runs. Therefore, we choose 100K operations for our experiments.

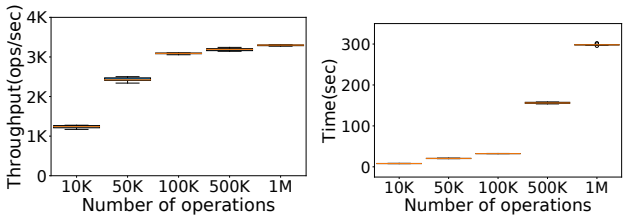


**Figure 1: Throughput (left plot) and time (right plot) obtained with 100K rows of data changing the number of operations between 10K, 50K, 100K, 500K and 1M.**

*4.1.2 Number of rows in the database.* In this experiment, conducted with the same workload and the same default settings of the previous one, we set the number of operations to 100K and we test different amounts of data in the database. In Figure 2 we show the results obtained. In terms of throughput (left plot), the experiments with 10K, 50K and 100K rows obtain similar results, with 100K rows yielding slightly more consistent results than the other ones. In terms of running time (right plot) we see that there is not much difference when executing the experiments with 10K, 50K or 100K. Hence, we choose 100K rows amount of data for our experiments.

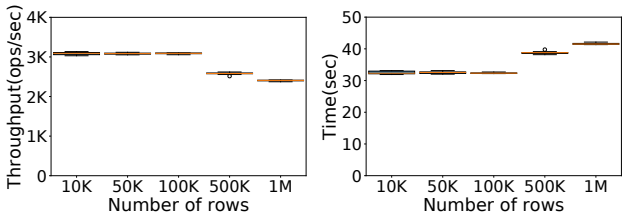


**Figure 2: Throughput (left plot) and time (right plot) obtained with 100K operations, changing the number of rows in the database between 10K, 50K, 100K, 500K and 1M.**

*4.1.3 Number of machines.* Another important factor to consider when evaluating the performance of a distributed database is the number of machines. We evaluate the performance on 1, 2, 4, 8 and 16 machines, using the default configuration and YCSB workload A with 100K rows and operations.

The results shown in Figure 3 indicate a decrease in throughput as the number of machines increases. This may seem surprising, considering that Cassandra is a distributed database, but our observations are consistent with those reported by other authors, and are caused by Cassandra's replication model [20, 39, 43]. The use of YCSB workload A can also partly explain the results, and a more writing-heavy workload for writing would probably perform better on a larger number of machines [20].

[1]This value was determined representative in preliminary experiments. Additional experiments with 1M rows confirmed the validity of our choice [37].

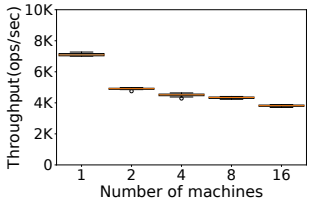


**Figure 3: Throughput obtained by running experiments of 100K operations and rows, changing the number of machines between 1, 2, 4, 8 and 16.**

In the remainder of this section we use one machine for our experiments. In a real world application, however, other considerations such as redundancy and robustness should be taken into account, and a higher number of machines could be employed to ensure a certain level of service.

*4.1.4 Reload the database each time?* How data is loaded between runs is another factor that can influence database performance. In fact, the startup phase can take a non-negligible amount of time, impacting the overall evaluation. Furthermore, when considering applications such as configuration, not all the modifications can be applied without restarting the entire database. We therefore test nine different Cassandra parameter configurations, and run all of them in different order. The results in Figure 4 show that destroying and reloading the database between each run (a situation also known as *cold evaluation*) results in greater consistency and better throughput than keeping the data in the database between runs.

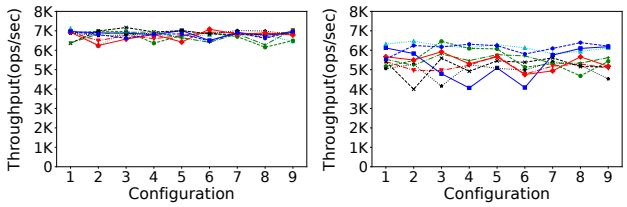


**Figure 4: Throughput obtained by running a series of nine different configurations nine times, changing the order of the configurations each run, destroying the database after each experiment (left plot) and without destroying the database after each experiment (right plot).**

*4.1.5 Multithreading.* Since both Cassandra and YCSB are systems that allow the use of client threads in their executions, we test also the performance of Cassandra in this scenario, observing how the execution of multiple client threads could influence the performance and consistency of the results.

In Figure 5 we report the results of the experiments comparing the performance of Cassandra with 1, 2, 4, 8 and 16 threads of YCSB clients for workload A with 100K operations and rows. We choose to use 4 client threads because of the good performance in terms of consistency and speed.

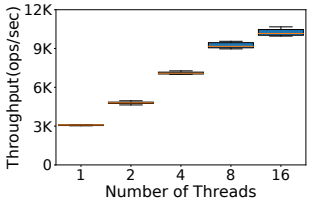


**Figure 5: Throughput obtained by running experiments of 100K operations and rows, with 1, 2, 4, 8 and 16 threads.**

*4.1.6 Results.* We have therefore identified a setup with 100K rows of data, 100K operations, one machine, four threads and cold experiments. We show the usefulness of this experimental setup by using it to configure Cassandra with irace, using 500, 1000 and 2000 experiments as tuning budget, both including (boxplots D-×, where × is the tuning budget used in the experiment) and ignoring (boxplots ND-×) the default configuration, and considering five selected parameters and the whole set of 23 parameters that impact the performance (Figure 6). The results are reported with two different test sets: $TS1_c$ corresponds to the same setup we identified for the tuning, while $TS2_c$ simulates a situation of heavier workload, with a database of 1M rows and operations, to evaluate the scalability enabled by the setup. All the results are reported in terms of speedup with respect to the default configuration; higher boxplots indicate therefore a better performance. For more details on our tuning process we refer to [36].

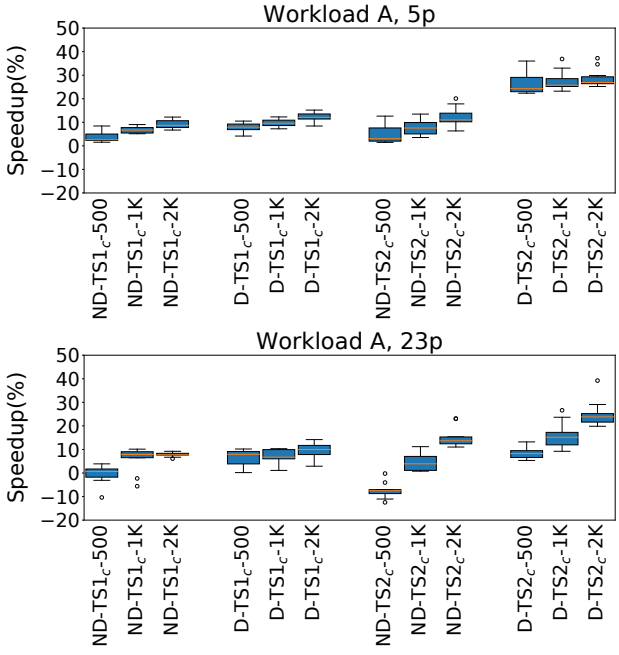


**Figure 6: Speedup with respect to the default configuration obtained in 100K experiments tuning 5 (top) and 23 parameters (bottom) tested on $TS1_c$ and $TS2_c$.**

The results indicate that in almost all the cases we considered we were able to obtain good results. We also see how using our setup is robust to scaling, both including and excluding the default configuration and in both parameter settings. Also, as expected, a higher tuning budget consistently results in better performance.

As a comparison we, report the results obtained when tuning using a lighter setup, labeled $TS0_c$, with 10K rows of data and 10K operations. The results are reported in Figure 7 for five and 23 parameters, in terms of speedup over the default configuration. The results now include the tests on $TS0_c$, $TS1_c$ and $TS2_c$. Now the results exhibit a high variability, and a higher tuning budget does not necessarily yield a better performing configuration, even for the same $TS0_c$, in particular when considering 23 parameters. This indicates that the throughput measured during the tuning phase on this lighter setup is not a reliable value and, therefore, that the experimental setup used is not adequate.

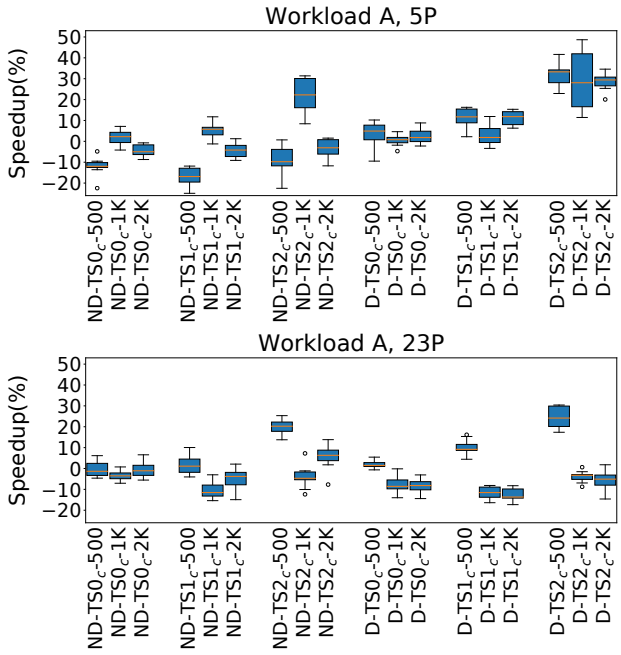


**Figure 7: Speedup with respect to the default configuration obtained in 10K experiments tuning 5 (top) and 23 parameters (bottom) tested on $TS0_c$, $TS1_c$ and $TS2_c$.**

## 4.2 Experimental setup for Elasticsearch

*4.2.1 Number of operations.* To determine an appropriate amount of operations for the Elasticsearch setup, we start by considering 10K rows of data. This value was again determined with preliminary experiments. We use again YCSB workload A with the default configuration of Elasticsearch, testing five different amounts of operations from 10K to 1M.

The results are reported in Figures 8 and 9 for 10K and 20K rows respectively, in terms of throughput and time. In both cases the throughput is more or less constant for the different amount of data, while the difference in terms of time is huge. A single evaluation with one million operations can in fact take more than 100 minutes, even with this little amount of data, making it unsuitable in practice for most database optimization tasks.

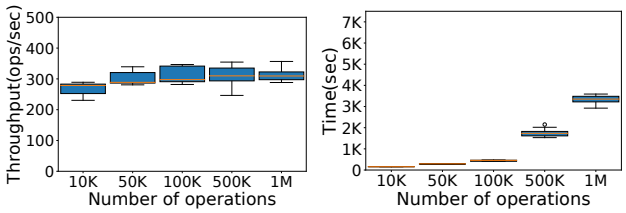


**Figure 8: Throughput (left plot) and time (right plot) obtained with 10K rows of data changing the number of operations between 10K, 50K, 100K, 500K and 1M.**

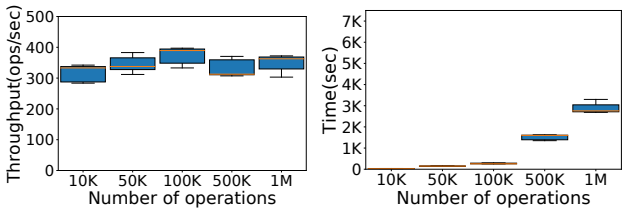


**Figure 9: Throughput (left plot) and time (right plot) obtained with 20K rows of data changing the number of operations between 10K, 50K, 100K, 500K and 1M.**

*4.2.2 Number of rows in the database.* In this experiment, performed with workload A and the default configuration of Elasticsearch, we test the impact of 10K, 50K, 100K, 500K and 1M rows in the database, using 10K operations, whose results are reported in Figure 10 in terms of throughput (left plot) and time (right plot). We include also results with 100K operations, reported in Figure 11.

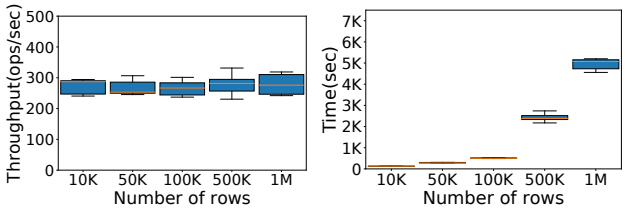


**Figure 10: Throughput (left plot) and time (right plot) obtained by running experiments of 10K operations, changing the number of rows in the database between 10K, 50K, 100K, 500K and 1M.**

In both cases, the results are similar in terms of both throughput and consistency for the different amounts of data rows. The time necessary for each evaluation is, instead, very different, especially for the largest datasets. On the other hand, while the time employed for 100K operations is higher than the time employed with 10K operations for the same amount of rows, the throughput is very similar. The choice to use 10K rows and operations seems therefore a valid one, because it is representative also of heavier loads.

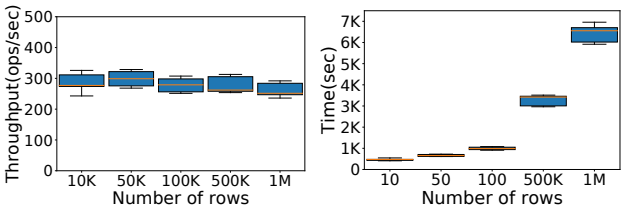


**Figure 11: Throughput (left plot) and time (right plot) obtained by running experiments of 100K operations, changing the number of rows in the database between 10K, 50K, 100K, 500K and 1M.**

*4.2.3 Number of machines.* We again evaluate the performance on 1, 2, 4, 8 and 16 machines, using the default configuration and YCSB workload A with 10K rows and operations. The results obtained are shown in Figure 12 where we see that, again, the results are very similar in terms of throughput and consistency. In the remainder of this section we use two machines for our experiments.

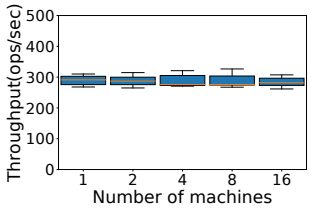


**Figure 12: Throughput obtained by running experiments of 10K operations and rows, changing the number of machines between 1, 2, 4, 8 and 16.**

*4.2.4 Reload the database each time?* Like for Cassandra we test what is the impact of cold experiments on the performance of Elasticsearch. We observe the results obtained with nine Elasticsearch configurations in different order nine times. We compare the difference between experiments where the database is destroyed between each execution and experiments in which the data is only loaded at the beginning of the experiments. The results are reported in Figure 13 for 10K and 100K rows of data and operations. The results are opposite to those observed with Cassandra, in that the performance is more consistent, albeit slightly lower, when the data is only loaded once.

This can be explained with some special features of Elasticsearch. When a document is indexed in Elasticsearch, the information is replicated in multiple shards. There are two types of shards: primaries and replicas. When Elasticsearch receives a request, the shards with the information return the response to the coordinating node. This node collects the data and presents it to the user. The response is stored in the cache memory of each shard to return the result quickly. This fragment cache functionality is what we can see in our results. If the data is not deleted after each experiment, it is cached and the throughput is constant. However, if the data is deleted after an experiment, each shard has to elaborate a response, making more operations. In our results we can see that this chunk

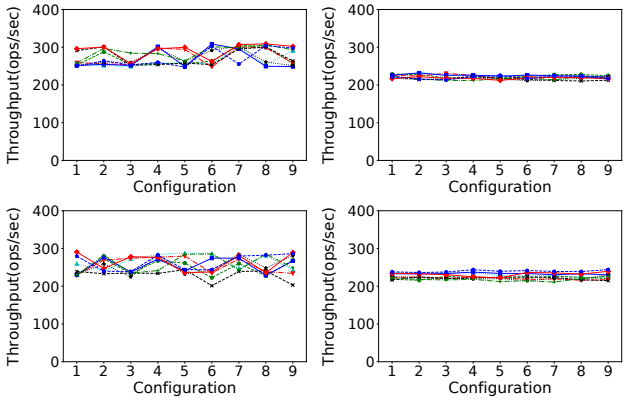


**Figure 13: Throughput obtained with 10K operations and rows (top row) and 100K operations and rows (bottom row) by running a series of nine different configurations nine times, changing the order of the configurations each run, destroying the database after each experiment (left plots) and without destroying the database after each experiment (right plots).**

cache functionality works the same for both small experiments with 10K operations and rows and for larger experiments with 100K operations and rows.

It is interesting to see that the results obtained by destroying the database between each experiment are very similar despite the difference in the number of operations and rows. Similarly, the results for the experiments where the database was not destroyed between each run were very similar to each other in terms of performance and consistency. We can see this by comparing the plots to the right of Figure 13 where the results are shown for 10K and 100K operations and rows respectively.

Despite the difference of results with respect to the Cassandra experiments, the outcome of this analysis is again to perform cold evaluations. In fact, these results are the consequence of a characteristic of the Elasticsearch operation, which does not allow to obtain independent results between executions. Therefore, destroying and reloading the database between each run is the only option to be able to run experiments independent on each other.

*4.2.5 Multithreading.* Elasticsearch provides two multithreading policies that have an impact on `update` operations. When a registry is modified in Elasticsearch, it generates a version number to check the consistency of the data before altering a record. During the updating process, if different threads have different version numbers the update is aborted. By default, Elasticsearch implements the "Optimistic Lock" configuration that allows that several threads handle the same registry. This could result in data consistency problems and cancel update processes. The other configuration is the "Pessimistic Lock", where the thread blocks the registry before updating it. It guarantees that Elasticsearch does not have concurrency issues but the update process is slower because internally it has to ensure the consistency of the data.

In Figure 14 we compare the throughput obtained using the two strategies for up to 16 threads. The "Pessimistic Lock" setting

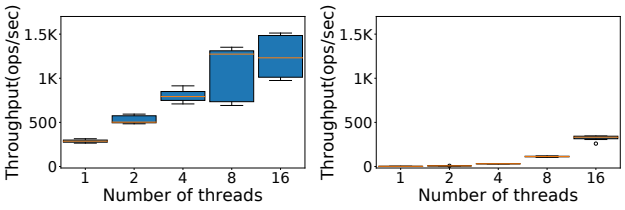


**Figure 14: Throughput obtained by running experiments of 10K operations and rows, changing the multithreading setting between "Optimistic Lock" (left plot) and "Pessimistic Lock" (right plot) and changing the number of threads between 1, 2, 4, 8 and 16.**

is much slower with respect to the "Optimistic Lock" one, which in turn exhibits a high variability in the results as the number of threads increases. Furthermore, with the increase of the number of threads the error probability when executing `update` operations surges as well, in both cases. To balance speed of execution, consistency of the observations and to avoid `update` errors, we choose to use the "Optimistic Lock" strategy with one single thread.

*4.2.6 Results.* Elasticsearch is a database whose characteristics are very different from those of Cassandra, and this is reflected in the experiments reported. The setup we identified is thus composed by 10K rows of data and 10K operations, which offer comparable throughput and consistency with respect to heavier scenarios, at a fraction of the time required. The impact of the hardware configuration is also different. We have selected two machines and a single thread with an "Optimistic Lock" policy, and we perform cold evaluations. We call this tuning setup $TS1_e$.

In Figure 15 we report the outcome of a set of tuning tasks performed, like for Cassandra, including and not including the default configuration, and with budgets of 500, 1000 and 2000 experiments. To test the scalability, we consider also a test setup $TS2_e$ with 100K rows and operations. The results show that using $TS1_e$ it is possible to obtain consistent performance improvements.

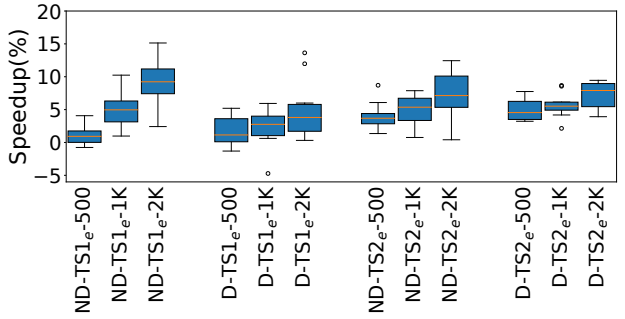


**Figure 15: Speedup with respect to the default configuration obtained in 10K experiments tuning 5 parameters tested on $TS1_e$ and $TS2_e$.**

## 4.3 Discussion

The determination of a proper experimental setup is a complex and computationally expensive task, with several factors involved. A full factorial analysis that takes all the components of the setup into account is clearly impractical, and we have therefore broken down the task into five basic steps. Some preliminary experiments, not reported in this work, are anyway necessary to determine an appropriate range of values to consider.

For both the databases considered we have devised an experimental setup to allow for both consistent and scalable evaluations, in particular for what concerns the amount of data and operations to be used. Other important factors that affect the evaluations are also considered, in particular the amount of machines and threads, for which we analyzed the impact on the performance. For both databases we have also noted how cold evaluations are the most appropriate way of performing sequential evaluations. This is important in particular when optimizing a database. This set of experiments is anyway not a one-off procedure, but it serves as a blueprint for the determination and validation of the experimental setup before a practical database benchmarking and optimization.

In a practical application, in fact, the practitioner may need to take into account other constraints such as, for example, a certain data structure, a minimum level of service to maintain, or a given hardware infrastructure. In particular, for database optimization it is recommended to design an experimental setup that resembles as closely as possible the application for which the database will be used, such as for example the hardware configuration, the data distribution, or the workload characteristics.

# 5 CONCLUSIONS

Benchkarking is a necessary operation in the development and maintenance of data-centric and simulation-based applications, and a fundamental step in their optimization. In this work we have outlined a sequence of experiments to determine an experimental setup that allows to balance the consistency of the results and the time necessary to obtain them. We have followed our procedure to devise an experimental setup for the Cassandra and Elasticsearch databases, using the YCSB benchmarking tool. We have demonstrated the usefulness of the setups obtained by optimizing the parameter configuration of Cassandra with the irace configurator, observing how the experimental setup we obtained makes it possible to both obtain reliable measurements and to scale the performance obtained in the optimization process also to heavier scenarios. Conversely, a more lightweight evaluation setup does not allow neither to obtain reliable measurements, nor to scale to more challenging scenarios.

Beyond the basic elements considered in this work, other factors that impact the performance of a database can be taken into account when devising an experimental setup beyond the basic ones we considered, such as for example the distribution of operations to be performed. Moreover, known constraints and information on the application for which the database is deployed should be exploited whenever available, to devise a setup as precise as possible. Our procedure could also be automated, in order to perform it as an

autonomous preliminary step in a database optimization application. The procedure can also be used as a guideline for devising an experimental setup for other IT systems.

## ACKNOWLEDGMENTS

This work has been partially supported by the CHIST-ERA project CHIST-ERA-17-BDSI-001 ABIDI "Context-aware and Veracious Big Data Analytics for Industrial IoT".

## REFERENCES

[1] Veronika Abramova, Jorge Bernardino, and Pedro Furtado. 2014. Evaluating cassandra scalability with YCSB. In *International Conference on Database and Expert Systems Applications*. Springer, 199–207.

[2] Yusuf Abubakar, Thankgod Sani Adeyi, and Ibrahim Gambo Auta. 2014. Performance evaluation of NoSQL systems using YCSB in a resource austere environment. *Performance Evaluation* 7, 8 (2014), 23–27.

[3] Cassandra Apache. 2021. Cassandra Stress. *Website. Available online at https://cassandra.apache.org/doc/latest/tools/cassandra_stress.html* 1 (2021).

[4] Fuad Bajaber, Sherif Sakr, Omar Batarfi, Abdulrahman Altalhi, and Ahmed Barnawi. 2020. Benchmarking big data systems: A survey. *Computer Communications* 149 (2020), 241–251.

[5] Chaitanya Baru, Milind Bhandarkar, Raghunath Nambiar, Meikel Poess, and Tilmann Rabl. 2012. Setting the direction for big data benchmark standards. In *Technology Conference on Performance Evaluation and Benchmarking*. Springer, 197–208.

[6] Andrzej Białecki, Robert Muir, Grant Ingersoll, and Lucid Imagination. 2012. Apache lucene 4. In *SIGIR workshop on open source information retrieval*. 17.

[7] Lin Cai, Yong Qi, Wei Wei, Jinsong Wu, and Jingwei Li. 2019. mrMoulder: A recommendation-based adaptive parameter tuning approach for big data processing platform. *Future Generation Computer Systems* 93 (2019), 570–582.

[8] Zhen Cao, Vasily Tarasov, Sachin Tiwari, and Erez Zadok. 2018. Towards better understanding of black-box auto-tuning: A comparative analysis for storage systems. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. 893–907.

[9] Apache Cassandra. 2014. Apache cassandra. *Website. Available online at https://www.datastax.com/cassandra* 13 (2014).

[10] Yu-Chun Chen, Hsiao-Yun Yeh, Jau-Ching Wu, Ingo Haschler, Tzeng-Ji Chen, and Thomas Wetter. 2011. Taiwan's National Health Insurance Research Database: administrative health care database as study object in bibliometrics. *Scientometrics* 86, 2 (2011), 365–380.

[11] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM symposium on Cloud computing*. 143–154.

[12] Alejandro Corbellini, Cristian Mateos, Alejandro Zunino, Daniela Godoy, and Silvia Schiaffino. 2017. Persisting big-data: The NoSQL landscape. *Information Systems* 63 (2017), 1–23.

[13] Miyuru Dayarathna and Toyotaro Suzumura. 2012. Xgdbench: A benchmarking platform for graph stores in exascale clouds. In *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*. IEEE, 363–370.

[14] Hui Dou, Pengfei Chen, and Zibin Zheng. 2020. Hdconfigor: Automatically Tuning High Dimensional Configuration Parameters for Log Search Engines. *IEEE Access* 8 (apr 2020), 80638–80653.

[15] Elasticsearch. 2021. Rally 2.1.0. *Website. Available online at https://esrally.readthedocs.io/en/stable/index.html* 1 (2021).

[16] Adrian Filip, Vadim Doga, Tatiana Poleacov, and Nina Cavcaliuc. 2019. SEPT-search engine and processing tool (Query DSL). (Sep. 2019).

[17] Alberto Franzin, Raphäel Gyory, Jean-Charles Nadé, Guillaume Aubert, Georges Klenkle, and Hugues Bersini. 2020. Philéas: Anomaly Detection for IoT Monitoring.. In *Proceedings of the 32nd Benelux Conference on Artificial Intelligence*. 56–70.

[18] Christine Gertosio and Alan Dussauchoy. 2004. Knowledge discovery from industrial databases. *Journal of Intelligent Manufacturing* 15, 1 (2004), 29–37.

[19] Clinton Gormley and Zachary Tong. 2015. *Elasticsearch: the definitive guide: a distributed real-time search and analytics engine*. O'Reilly Media, Inc.

[20] Gerard Haughian, Rasha Osman, and W. Knottenbelt. 2016. Benchmarking Replication in Cassandra and MongoDB NoSQL Datastores. In *DEXA*.

[21] Clive Humby. 2006. Data is the new oil. *Proc. ANA Sr. Marketer's Summit. Evanston, IL, USA* (2006).

[22] Marko Javornik, Nives Nadoh, and Dustin Lange. 2019. Data is the new oil. In *Towards User-Centric Transport in Europe*. Springer, 295–308.

[23] Samiya Khan, Xiufeng Liu, Syed Arshad Ali, and Mansaf Alam. 2019. Storage solutions for big data systems: A qualitative study and comparison. *arXiv preprint arXiv:1904.11498* (2019).

[24] Sanjay Krishnan, Zongheng Yang, Ken Goldberg, Joseph Hellerstein, and Ion Stoica. 2018. Learning to optimize join queries with deep reinforcement learning. *arXiv preprint arXiv:1808.03196* (2018), 1–19.

[25] Jörn Kuhlenkamp, Markus Klems, and Oliver Röss. 2014. Benchmarking scalability and elasticity of distributed database systems. *Proceedings of the VLDB Endowment* 7, 12 (2014), 1219–1230.

[26] Latrice G Landry, Nadya Ali, David R Williams, Heidi L Rehm, and Vence L Bonham. 2018. Lack of diversity in genomic databases is a barrier to translating precision medicine research into practice. *Health Affairs* 37, 5 (2018), 780–785.

[27] Mingyu Li, Zhiqiang Liu, Xuanhua Shi, and Hai Jin. 2020. ATCS: Auto-Tuning Configurations of Big Data Frameworks Based on Generative Adversarial Nets. *IEEE Access* 8 (2020), 50485–50496.

[28] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Thomas Stützle, and Mauro Birattari. 2016. The `irace` package: Iterated Racing for Automatic Algorithm Configuration. *Operations Research Perspectives* 3 (2016), 43–58. https://doi.org/10.1016/j.orp.2016.09.002

[29] Jiaheng Lu, Yuxing Chen, Herodotos Herodotou, and Shivnath Babu. 2019. Speedup your analytics: Automatic parameter tuning for databases and big data systems. *Proceedings of the VLDB Endowment* 12, 12 (2019), 1970–1973.

[30] Divya Mahajan, Cody Blakeney, and Ziliang Zong. 2019. Improving the energy efficiency of relational and NoSQL databases via query optimizations. *Sustainable Computing: Informatics and Systems* 22 (2019), 120–133.

[31] Ashraf Mahgoub, Sachandhan Ganesh, Folker Meyer, Ananth Grama, and Somali Chaterji. 2017. Suitability of nosql systems—cassandra and scylladb—for iot workloads. In *2017 9th International Conference on Communication Systems and Networks (COMSNETS)*. IEEE, 476–479.

[32] Ashraf Mahgoub, Paul Wood, Sachandhan Ganesh, Subrata Mitra, Wolfgang Gerlach, Travis Harrison, Folker Meyer, Ananth Grama, Saurabh Bagchi, and Somali Chaterji. 2017. Rafiki: a middleware for parameter tuning of nosql datastores for dynamic metagenomics workloads. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*. 28–40.

[33] Swapnil Patil, Milo Polte, Kai Ren, Wittawat Tantisiriroj, Lin Xiao, Julio López, Garth Gibson, Adam Fuchs, and Billie Rinaldi. 2011. Ycsb++ benchmarking and performance debugging advanced features in scalable table stores. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*. 1–14.

[34] Mark Raasveldt, Pedro Holanda, Tim Gubner, and Hannes Mühleisen. 2018. Fair benchmarking considered difficult: Common pitfalls in database performance testing. In *Proceedings of the Workshop on Testing Database Systems*. 1–6.

[35] Vincent Reniers, Dimitri Van Landuyt, Ansar Rafique, and Wouter Joosen. 2017. On the state of nosql benchmarks. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*. 107–112.

[36] Moisés Silva-Muñoz, Alberto Franzin, and Hugues Bersini. 2021. Automatic configuration of the Cassandra database using irace. (2021). Under review.

[37] Moisés Silva-Muñoz, Alberto Franzin, and Hugues Bersini. 2020. Supplementaty Material for: Automatic configuration of the Cassandra database using irace. http://iridia.ulb.ac.be/supp/IridiaSupp2020-014.

[38] Mohamad Sobhie. 2019. *Tuning of Elasticsearch Configuration-Parameter Optimization Through Simultaneous Perturbation Stochastic Approximation Algorithm*. Master's thesis.

[39] Surya Narayanan Swaminathan and Ramez Elmasri. 2016. Quantitative analysis of scalable NoSQL databases. In *2016 IEEE International Congress on Big Data (BigData Congress)*. IEEE, 323–326.

[40] Transaction Processing Performance Council TPC. 2010. TPC BENCHMARK™E. (2010).

[41] Dana Van Aken, Andrew Pavlo, Geoffrey J Gordon, and Bohan Zhang. 2017. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 1009–1024.

[42] Guoxi Wang and Jianfeng Tang. 2012. The nosql principles and basic application of cassandra model. In *2012 international conference on computer science and service system*. IEEE, 1332–1335.

[43] Huajin Wang, Jianhui Li, Haiming Zhang, and Yuanchun Zhou. 2014. Benchmarking replication and consistency strategies in cloud serving databases: Hbase and cassandra. In *Workshop on Big Data Benchmarks, Performance Optimization, and Emerging Hardware*. Springer, 71–82.

[44] Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, et al. 2019. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In *Proceedings of the 2019 International Conference on Management of Data*. 415–432.

[45] Conghuan Zheng, Zuohua Ding, and Jueliang Hu. 2014. Self-tuning performance of database systems with neural network. In *International Conference on Intelligent Computing*. Springer, 1–12.

[46] Xuanhe Zhou, Chengliang Chai, Guoliang Li, and Ji Sun. 2020. Database Meets Artificial Intelligence: A Survey. *IEEE Transactions on Knowledge and Data Engineering* (2020).

[47] Yuqing Zhu, Jianxun Liu, Mengying Guo, Yungang Bao, Wenlong Ma, Zhuoyue Liu, Kunpeng Song, and Yingchun Yang. 2017. Bestconfig: tapping the performance potential of systems via automatic configuration tuning. In *Proceedings of the 2017 Symposium on Cloud Computing*. 338–350.