Developing a Methodology for the Identification of Alternative NoSQL Data Models via Observation of Relational Database Usage

Conference Paper · August 2019

CITATIONS
0

READS
43

5 authors, including:

Paul Michael Beach
Air Force Institute of Technology
5 PUBLICATIONS **3** CITATIONS
SEE PROFILE

Brent Langhals
Air Force Institute of Technology
9 PUBLICATIONS **7** CITATIONS
SEE PROFILE

Michael R. Grimaila
Air Force Institute of Technology
141 PUBLICATIONS **1,061** CITATIONS
SEE PROFILE

Douglas D. Hodson
Air Force Institute of Technology
38 PUBLICATIONS **157** CITATIONS
SEE PROFILE

Developing a Methodology for the Identification of Alternative NoSQL Data Models via Observation of Relational Database Usage

Paul M. Beach[1], Brent T. Langhals[1], Michael R. Grimaila[1], Douglas D. Hodson[2], Ryan D. L. Engle[1]
[1]Dept. of Systems Engineering, Air Force Institute of Technology, Wright-Patterson AFB, OH
[2]Dept. of Electrical & Computer Engineering, Air Force Institute of Technology, Wright-Patterson AFB, OH

Abstract – *NoSQL databases are largely synonymous with Big Data applications, however there is existing research to support their usage for smaller scale applications (even on a single desktop). This work seeks to reinforce this notion by proposing a methodology to identify when a NoSQL database may be better suited than an existing Relational Database Management System (RDBMS). First, approaches to characterizing database workloads are discussed, along with a summary of relevant benchmarking metrics. Then, we present a methodology for evaluating the suitability of an existing RDBMS against possible alternative NoSQL databases.*

Keywords: Benchmarking, data models, database, database systems, NoSQL.

1 Introduction

The present era of Big Data has given rise to a number of novel approaches and supporting technologies for coping with the ever-increasing amounts of data that many organizations attempt to collect and use. The volumes, velocities and varieties of data collected, inherent characteristics of "Big Data", have increasingly proven traditional relational data databases as inadequate. This challenge has led to the emergence of a new class of database systems (often referred to as NoSQL systems), which are not based upon the traditional relational model [1].

Although these systems were initially borne out of a need to deal with massive datasets hosted across large clusters of computers utilizing commodity hardware, previous research has highlighted the value in considering the use of these NoSQL systems in a single desktop configuration (citing reasons such as efficient write performance and fast, low-latency access, among others) [2]. Our research extends this effort and considers the possibility that there are relational database systems operating today which may benefit from better performance using a key-value, document, column, or graph data model, i.e. one supported by a NoSQL data model.

There are several reasons why an existing system may be using a relational data model when a different data model would be more appropriate. One possibility is that the application developer was unfamiliar with the recent development of the NoSQL data models when they designed the system. Alternatively, the usage of a database system may have evolved over time from a point when a relational data model made the most sense, to something different which more closely aligns with a NoSQL data model.

Regardless of the underlying reason for the mismatch, our research serves as an exploratory effort towards proposing a methodology for identifying a more suitable NoSQL database type based solely on observations of interactions with an existing relational database. In particular, we are characterizing a user's workload against the existing database and matching the workload to relevant benchmark metrics designed to identify a more suitable solution. The remainder of the paper is organized as follows. In Section 2, a comparison of relational and NoSQL data models is presented. Section 3 discusses existing approaches towards defining database workloads and benchmark metrics are discussed. Section 4 presents the proposed methodology. Section 5 walks the reader through a hypothetical use case for the proposed methodology and the paper is concluded in Section 6.

2 Data Models

Due to its ubiquitous usage, the uses of the relational data model are likely well understood by many audiences. However, a brief review of the NoSQL data models and their relative advantages and disadvantages over the relational model provides a foundation for the topics that follow.

2.1 Key-value

Key-value (KV) stores rely on a simple data model predicated on a simple structure involving a key and its associated value. In this data model, the key is a unique identifier for the data and is comparable to maps, dictionaries, and associative arrays which all organize data using non-integer indexing. Additionally, the set of employable keys are not pre-defined. In fact, this model grants extensive freedom to the user for key definition. Likewise, the associated value may vary greatly from strings, to numbers, or even binary large objects (BLOBs) among others. This data is considered "opaque" to the model because limited information is known about the stored data itself [3]–[5].

Storage and retrieval of data is also relatively simple as operations are typically focused on keys. In fact, operations on key-value stores are typically limited to three options: storing (putting) a value by a key, retrieving (getting) a value for a key or deleting a value by its key. In effect, this model does not provide the ability to query by examining the contents of a stored value.

In contrast to a relational model, queries that are relatively easy to perform against values, e.g., SELECT * FROM Table WHERE value=X, are no longer feasible. Additionally, atomic operations can only be performed on a single key-value pair at a time. Another limitation of the key-value model is that it becomes much more difficult to efficiently store and retrieve related pieces of data, since the data model does not provide a native mechanism to track relationships between key-value pairs. If relationship management is necessary, the user must do so externally from the KV model. Furthermore, KV stores require no pre-defined schema unlike relational implementations [3].

Finally, although the key-value model theoretically can store any type of value, a practical consideration to consider is that databases employing the KV model will have an upper limit on the size of the stored value [3]. Despite these limitations, one of the primary motivations for using a database implementation based on the key-value model is that they tend to be extremely performant [5].

2.2 Document

In this data model, documents are characterized as "self-describing, hierarchical tree data structures," with a flexible schema that allows one document to differ from another [5]. This is contrasted with the relational model, in which all tuples of a given entity are required to have the same attributes [6]. Although the document data model lacks a means of joining documents the way the relational model does, the ability to flexibly store documents nested within another document is one approach to satisfying this objective. Like the other non-relational implementations, attributes may be added to documents without defining a schema [3].

Similar to the key-value model, atomic operations across documents are not possible in the document model. On the other hand, document databases are able to execute queries based on the stored values, so they differ from key-value stores but are similar to relational databases in this regard [3].

2.3 Column Family

Superficially, relational and column family databases may appear similar since they both employ rows, columns and, effectively, tables. However there are notable differences between their underlying data models. In a column family model, data are uniquely identified by both a row and column name. Additionally, columns may be grouped together to form column families. This organization improves performance when reading and writing related data because it is stored together [4], [7]. Similar to KV stores, atomic operations in column families are limited to operating only a single row. Again, this contrasts with the relational capability to execute multi-row transactions in a single operation [3].

One of the strengths of the column family over relational model is its ability to store only the data that is of interest. Thus, rows may or may not contain values for every column, i.e., be sparse. This design enables column family databases to potentially span millions of columns in an efficient manner whereas relational databases must store null values if there are non-relevant or empty fields. This approach becomes substantially less tenable for relational databases when large numbers of sparsely populated fields are commonplace. Additionally, columns may be added at any time to an existing data store which contrasts with the relational requirement to update the existing database structure definition (i.e., schema) prior to adding an attribute [4].

2.4 Graph

Graph databases represent data using nodes and relationships between nodes, which is useful for storing information about highly interrelated objects. The graph data model (and the databases that employ it) make querying across related data easier and more efficient than would be possible using a relational data model (which would require many joins, and/or recursive queries, if supported by the particular RDMBS) [3].

3 Benchmarking NoSQL Databases

The earliest usage of the term "NoSQL" in the context of non-relational databases dates back to 2009 [5], and some of the earliest examples of what we would consider today to be a NoSQL database were being designed only a few years earlier [8]. Still, in the relatively brief period since then there has been ample interest from practitioners and researchers alike to look at ways of benchmarking the performance of these systems. This section examines previous approaches of benchmarking NoSQL databases found in the literature which focus on how the usage (i.e., workload) and performance (i.e., metrics) of these systems can be characterized

3.1 Workloads

One way of defining workloads in a database system is "requests made by the users of a system" [9]. Depending on the kinds and frequency of requests that users are making to a database system, the burden upon that system will vary. Thus, a first step towards understanding how an existing database system is being employed will require a means of assessing the existing workload on that system. Three approaches were identified in the literature, including "Big Data distributed tasks", "standardized queries" and "core workloads".

A Big Data application-based approach to defining database workloads considers Big Data workloads as being comprised of one or more core computational tasks common in Big Data processing, such as sorting, word counting, grep, cryptographic hashing, matrix multiplication, random sampling, graph traversal, and so on [10], [11]. While this bottom-up, "building blocks" approach to creating a database workload is an interesting method for comparing Big Data processing frameworks (e.g., Apache Hadoop), the scope of this research (non-"Big Data", single box workloads) does not extend to such frameworks so the usefulness of this approach would be limited.

Another observed approach for testing database workloads is to employ a set of predefined "standardized queries", which are queries representative of tasks that would be performed in real-world systems (e.g., "List the top 10 states in descending order with at least 10 customers who during a given month

bought products with the price tag at least 20% higher than the average price of products in the same category") [12], [13]. Similar to the Big Data application-based approach to assembling workloads, this is an excellent top-down approach for comparing the overall performance of two or more big data processing frameworks. However, the methodology presented in section 4 relies upon the ability to generalize observable queries (i.e. not predefined), which would be a non-trivial endeavor with this approach. Therefore, a third approach is considered.

One of the most common benchmarking frameworks used to assess NoSQL databases is the Yahoo! Cloud Serving Benchmark (YCSB) [14]. This framework was designed by engineers from Yahoo! to benchmark various databases used in cloud-based settings. One of the reasons for its popularity is the fact that it was designed to be extensible so that researchers could easily incorporate additional databases or customize the workloads to suit their needs [15].

YCSB comes with a set of six predefined "core workloads" (Workloads A through F) that generically represent typical usage patterns of various types of applications. Core workloads are defined by a fractional breakdown of the read, update, insert, and scan operations that will be executed by the benchmark and a random distribution that determines which record to read or write to. For example, Workload A is considered an "update heavy" workload, and is configured to perform a 50%/50% mix of reads and writes using a Zipfian distribution in order to represent an application such as a session store (Zipfian distributions represent situations where a few records are very popular and commonly accessed, while the majority are not frequently accessed). Workload B is considered a "read mostly" workload, and is configured to perform read queries 95% of the time and write queries the remaining 5%, also using the Zipfian distribution. The example application of Workload B is photo tagging, where most operations are reading tags with an occasional write [15].

This method has the best potential for being adapted to this research, since it is conceivable to develop software that can analyze the behavior of an existing database workload by evaluating the queries based on keyword extraction (i.e., looking for "SELECT", "UPDATE", "DELETE", etc.). This approach will be discussed further in the Methodology section.

3.2 Metrics

A number of studies have been conducted to measure the performance of databases using a variety of benchmarking metrics, to include metrics related to CRUD (create, read, update, and delete) operations and system performance.

The most prevalent types of benchmarking metrics that were observed involve assessing CRUD operations on NoSQL databases. Given the simplicity of the key-value data model, many types of NoSQL databases can support this model, even those typically associated with other data models (e.g., MongoDB, Cassandra, etc.) and as such it appears to be the most widely studied [16]. The Yahoo! Cloud Serving Benchmark (YCSB), is designed to test NoSQL databases supporting this key-value model [15]. Additionally, there have been many other studies that conducted research on the performance of CRUD operations for various types of NoSQL databases [16]–[27].

Beyond simple CRUD performance measurements, other studies have looked at aspects of the host system's performance in order to gain insights about the performance of NoSQL databases and/or Big Data workloads. For instance, the utilization of system metrics such as CPU and memory utilization, I/O wait times, memory, disk and network bandwidth utilization, cache misses, branch misprediction rates and energy-efficiency are a few of the metrics that have been used to date in previous studies [10], [13], [28]–[32]. However, it has also been pointed out that such "micro benchmarks" do not necessarily indicate how a given benchmark will perform on different hardware, potentially limiting their usefulness [33].

A consolidated summarization of these benchmarking metrics is provided in Table 1.

Table 1: Benchmarking Metrics Discussed in the Literature.

Metric	Description
CRUD Operations	
Create	Time required to insert a new value
Read	Time required to read a value
Update	Time required to update a value
Delete	Time required to delete a value
System Performance	
CPU utilization rates	Percentage of CPU utilization
I/O wait times	Percentage of time CPU is waiting
Memory I/O bandwidth	Rate of memory data transfer (in MBps)
Memory utilization	Percentage of memory utilization
Disk I/O bandwidth	Rate of disk data transfer (in MBps)
Network I/O bandwidth	Rate of network data transfer (in MBps)
Cache misses per kilo instructions	Number of times a cache lookup failed
Branch mispredict rate	Percentage of branch mispredictions
Energy-efficiency	Power consumption of a system (in Watts)

4 A Proposed Methodology

In this section, we offer a proposed methodology for identifying suitable NoSQL database alternatives to an existing RDBMS.

4.1 Step 1: Monitor Current Usage

The process begins by observing the current usage of an existing RDBMS system (see Fig. 1). This step is intended to collect information about the queries being performed against the RDMBS by all users, as well as the collection of metrics identified in Table 1. Since we are attempting to characterize the usage of the system as a whole, queries for the entire system are gathered (rather than per user). Information about the queries should include the Structured Query Language (SQL) code being sent to the RDBMS as well as the response from the database. A sufficient number of queries should be collected to ensure that a representative sample has been captured. Also, information about the overall size of the database is captured.

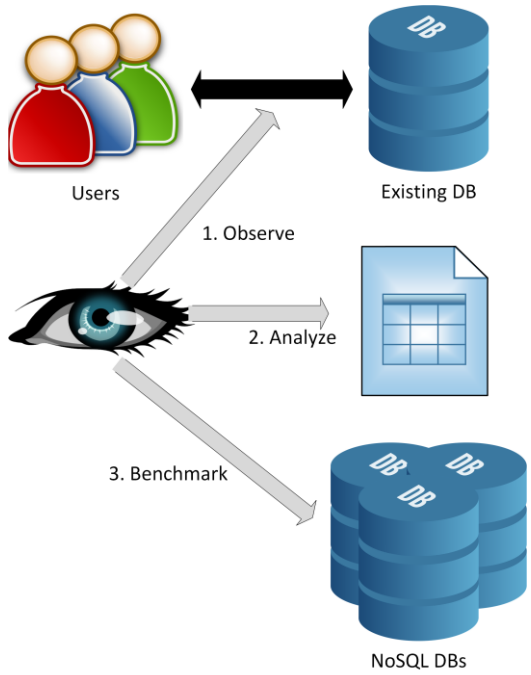

Fig. 1. Proposed methodology.

4.2 Step 2: Analyze Observed Usage

After collecting the necessary information in the first step, the next step involves analyzing the collected data in order to characterize the current system's workload. The SQL queries and associated responses are parsed in order to extract three attributes pertaining to the types of queries being presented: core workload classification(s), query frequency, and data volume.

As described in section 3.1, core workload classifications(s) refers to a fractional breakdown of the queries based on the read, update, insert, and read/modify/write mixture. To establish this ratio, the SQL commands are parsed to extract and tabulate the relevant SELECT, UPDATE, INSERT INTO, and/or DELETE commands. For the slightly more complex read/modify/write workload, a sequence of a SELECT command followed by an UPDATE/INSERT/DELETE with matching WHERE clauses would apply.

Query frequency is an average measurement of how many queries are being performed per unit of time against the database. This attribute gives a measure of the intensity of the database's usage.

Query complexity can be categorized as either simple or complex. Simple queries are those that can access data via a unique identifier or range of identifiers or values, while a complex query would be anything else (e.g., queries using aggregate function such as AVG, COUNT, MAX, MIN and SUM, or relational queries using JOIN, IN and EXISTS, etc.) [6], [34].

Finally, data volume is a measurement of the average volume of data being stored or retrieved from the database for any given query. These could be given discrete categorizations such as small for queries containing values less than 1 kilobyte in size, medium for queries with values between 1 kilobyte and 1 terabyte in size, and large for anything exceeding 1 terabyte per query [34]. Alternatively, for a simpler approach a scalar value for the observed average can be used.

4.3 Step 3: Run Benchmark Tests

Finally, the outputs from step 2 are used as inputs for a battery of benchmark tests (using YCSB or another similar benchmark tool) to assess the performance of similarly resourced NoSQL databases (i.e., in terms of CPU, memory, and storage). More concretely, the benchmark tool should be configured to use a core workload mixture analogous to the mixture observed on the existing system, queries should be ran as frequently as those observed on the existing system, and the volume of data being sent and received should similarly be based on that seen on the existing system.

Once the benchmark tests have been executed, the resulting metrics are be compared to those gathered during the observation of the existing system in step 1. NoSQL databases showing improvements in CRUD operation metrics may indicate a more suitable choice of database (assuming the improved CRUD operation(s) are of interest/value to the user). Additionally, improvements in system performance metrics may indicate that better performance could be obtained with a NoSQL system, or that similar performance can be achieved with less hardware. Finally, the relative sizes of the databases can be compared to determine if there is any advantage of one over the other in terms of storage costs.

5 Hypothetical Use Case

To provide a more concrete example of our proposed approach, we present the following hypothetical use case. In this example, we begin by observing a common RDMBS (MySQL) supporting an online forum system.

Online forums typically consist of a number of distinct entities such as subforums, discussion threads, comments, and users. Subforums organize discussion topics into related areas, which consist of discussion threads. The discussion threads are ordered by recency (the most recently commented upon threads are generally presented first, replacing threads that have not recently been commented on), and these threads are composed of individual comments. Finally, it should be noted that the comments are made by the users who visit the forum. In reality, there may be many more relevant entities in a realistic forum system (e.g., messages between users).

For the sake of this example, consider a user visiting an individual subforum, where they would be presented with a list of discussion threads that are ordered by recency of comments within each discussion thread. Usage patterns of such a system would typically involve a high-read workload (e.g., 95%) and a low-write workload (e.g., 5%), since forums are often used for community discussions where one user makes a comment that is subsequently read by many other users. Also, the most recent discussions are typically the ones being browsed at any given time, so access patterns would follow a distribution where the most recent information is accessed more often.

A typical RDBMS system such as MySQL would organize this database with a normalized schema keeping each of these entities (i.e. subforums, discussion threads, comments, and

users) separate. Thus, when the user visits a subforum a series of many SQL queries would need be issued to collect and then present the proper information to the user. For example, first the system would need to query a list of all threads in the subforum that the user was visiting, which would be a relatively simple SELECT query. However, in order to know the correct order to display them in, each of the comments for each of the discussion threads would need to be queried and sorted by most recent, requiring individual SELECT statements with an ORDER BY clause for each topic. Finally, when discussion threads are listed in subforums, they also typically include information about the most recent post, such as when it was made and by which user. Therefore, another batch of SELECT statements using JOIN operators would be required to obtain the proper usernames associated with each of the most recent comments for each discussion thread.

In the proposed methodology, we would begin by monitoring the usage of the MySQL database, collecting the SQL queries described above. As mentioned, due to the usage patterns of this system, 95% of the entries would be a SELECT statement of some type to collect the necessary discussion threads, comments and user information, and about 5% would be UPDATE or INSERT operations.

After the queries are collected, the second step of our methodology can be accomplished with a simple script to parse the collected queries for the SQL keywords of interest, tabulating each to determine the precise workload breakdown ratio. Also, the total number of queries over the observed period of time are gathered to arrive at an average frequency, and the average size of the queries and their responses are averaged to determine a representative value for the data volume to be used in the benchmark tests.

Finally, the third step of the methodology would consist of running benchmark tests using similarly sized NoSQL databases. In this scenario, the benchmark tests employ the YCSB benchmarking tool against each of the four NoSQL databases. The tool is configured with the core workload classification, target number of operations per second, and average value size. After the tests are run, the results can be compared to the performance observed in step 1. The output in Table 2 shows a representative example of the difference in performance between the two databases. The experimental setup that generated these results employed YCSB's Workload B (95% reads, 5% update) to execute 100,000 operations, modified to use a "latest" distribution (which selects records on a basis that the most recently inserted ones will be selected more often [15]), mirroring the usage pattern described above for online forums. These results were generated on virtual machines operating Ubuntu 18.04 LTS equipped with 4 GB RAM and 4 CPU cores (Intel i7-8650U @ 1.90GHz).

In contrast to the organization of the MySQL database, the nested structure of a document database (such as MongoDB), lends itself well to the structure of an online forum. Each subforum is a document, which will refence other nested documents for the individual discussion threads. These threads can contain all of the comments, along with the associated user information embedded within them since NoSQL data models are typically denormalized. As a result of this structure, reads (which comprise the majority of operations in this system) would be optimized, and can be expected to perform more quickly. Thus, it is anticipated that the results will remain consistent with real-world testing of similar scenarios, and that the benchmark test for the document database (i.e., MongoDB) would indicate that this data model may be a more suitable selection for this use case [35].

Table 2: Benchmarking Results (smaller is better).

	MySQL	**MongoDB**
Total execution time (in seconds)	22.947	15.637
Average read time (in microseconds)	184.89	147.514
Average update time (in microseconds)	933.226	250.929

6 Conclusions and Future Work

This work demonstrates a possible approach for characterizing observed workloads against an RDBMS, and further it proposes a method to suggest suitable NoSQL databases based on the observed workloads for the existing RDBMS.

Because of the abstractions involved in the underlying data models, this approach can be used to suggest when switching to a NoSQL database may be a preferred solution (in terms of increased performance and/or reduced hardware requirements), however it cannot guarantee that this is the case. Knowing for certain that switching to a NoSQL data model would be a more suitable choice would likely require empirical testing using the actual system's users and original data (properly transformed into the appropriate NoSQL data model).

Future work includes empirical testing of this methodology to further support the validity of this approach. Also, additional attributes about the observed queries such as query complexity (e.g., are the queries simple SELECT statements, or do they involve more complex operations such as JOIN), and the development of a corresponding supervised learning decision tree model to facilitate the inclusion of multiple criteria in a decision model.

Acknowledgment

Funding for this research was provided as part of the Air Force Office of Scientific Research grant 18RT0095 in support of Dynamic Data Driven Application Systems, PI: Dr. Eric Blasch.

Disclaimer

The views expressed in this paper are those of the authors and do not reflect the official policy or position of the U.S. Air Force, the Department of Defense, or the U.S. Government.

7 References

[1] A. B. M. Moniruzzaman and S. A. Hossain, "NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison," *Int. J. Database Theory Appl.*, vol. 6, no. 4, p. 14, 2013.

[2] R. D. L. Engle, B. T. Langhals, M. R. Grimaila, and D. D. Hodson, "The Case for NoSQL on a Single Desktop," presented at the International Conference on Information and Knowledge Engineering, Las Vegas, NV, 2018, p. 4.

[3] D. Sullivan, *NoSQL for mere mortals*. Hoboken, NJ: Addison-Wesley, 2015.

[4] R. Hecht and S. Jablonski, "NoSQL evaluation: A use case oriented survey," in *2011 International Conference on Cloud and Service Computing*, Hong Kong, China, 2011, pp. 336–341.
[5] P. J. Sadalage and M. Fowler, *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Upper Saddle River, NJ: Addison-Wesley, 2013.
[6] P. Pratt, *A Guide to SQL*, Seventh. Boston, MA: Thomson Course Technology.
[7] G. Copeland and S. Khoshafian, "A Decomposition Storage Model," *ACM SIGMOD Rec.*, vol. 14, no. 4, pp. 268–279, 1985.
[8] F. Chang *et al.*, "Bigtable: A Distributed Storage System for Structured Data," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, pp. 1–26, Jun. 2008.
[9] R. Jain, *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. New York: Wiley, 1991.
[10] W. Gao *et al.*, "Data Motifs: A Lens Towards Fully Understanding Big Data and AI Workloads," *ArXiv180808512 Cs*, Aug. 2018.
[11] F. Ahmad, S. Lee, M. Thottethodi, and T. N. Vijaykumar, "PUMA: Purdue MapReduce Benchmarks Suite," Purdue University, ECE Technical Reports TR-ECE-12-11, 2012.
[12] Transaction Processing Performance Council, "Transaction Processing Performance Council - TPC Express Big Bench (TPCx-BB) Standard Specification Version 1." 2016.
[13] D. Richins, T. Ahmed, R. Clapp, and V. Janapa Reddi, "Amdahl's Law in Big Data Analytics: Alive and Kicking in TPCx-BB (BigBench)," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Vienna, 2018, pp. 630–642.
[14] V. Abramova and J. Bernardino, "NoSQL databases: MongoDB vs cassandra," in *Proceedings of the International C* Conference on Computer Science and Software Engineering - C3S2E '13*, Porto, Portugal, 2013, pp. 14–22.
[15] 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 - SoCC '10*, Indianapolis, Indiana, USA, 2010, p. 143.
[16] 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)*, Victoria, BC, Canada, 2013, pp. 15–19.
[17] V. D. Jogi and A. Sinha, "Performance evaluation of MySQL, Cassandra and HBase for heavy write operation," in *2016 3rd International Conference on Recent Advances in Information Technology (RAIT)*, Dhanbad, India, 2016, pp. 586–590.
[18] A. Boicea, F. Radulescu, and L. I. Agapin, "MongoDB vs Oracle -- Database Comparison," in *2012 Third International Conference on Emerging Intelligent Data and Web Technologies*, Bucharest, Romania, 2012, pp. 330–335.
[19] S. Islam, K. Lee, A. Fekete, and A. Liu, "How a consumer can measure elasticity for cloud platforms," in *Proceedings of the third joint WOSP/SIPEW international conference on Performance Engineering - ICPE '12*, Boston, Massachusetts, USA, 2012, p. 85.
[20] J. Klein, I. Gorton, N. Ernst, P. Donohoe, K. Pham, and C. Matser, "Performance Evaluation of NoSQL Databases: A Case Study," in *Proceedings of the 1st Workshop on Performance Analysis of Big Data Systems - PABS '15*, Austin, Texas, USA, 2015, pp. 5–10.
[21] D. Carstoiu, E. Lepadatu, and M. Gaspar, "Hbase - non SQL Database, Performances Evaluation," *Int. J. Adv. Comput. Technol.*, vol. 2, no. 5, p. 11, 2010.
[22] G. Ladwig and A. Harth, "CumulusRDF: Linked Data Management on Nested Key-Value Stores," presented at the The 7th International Workshop on Scalable Semantic Web Knowledge Base Systems, 2011, p. 13.
[23] B. Atikoglu, Y. Xu, and E. Frachtenberg, "Workload Analysis of a Large-Scale Key-Value Store," in *ACM SIGMETRICS Performance Evaluation Review*, 2012, vol. 40 (1), pp. 53–64.
[24] E. Dede, M. Govindaraju, D. Gunter, R. S. Canon, and L. Ramakrishnan, "Performance evaluation of a MongoDB and hadoop platform for scientific data analysis," in *Proceedings of the 4th ACM workshop on Scientific cloud computing - Science Cloud '13*, New York, New York, USA, 2013, p. 13.
[25] A. Flores, S. Ramirez, R. Toasa, J. Vargas, R. U.- Barrionuevo, and J. M. Lavin, "Performance Evaluation of NoSQL and SQL Queries in Response Time for the E-government," in *2018 International Conference on eDemocracy & eGovernment (ICEDEG)*, Ambato, 2018, pp. 257–262.
[26] C.-O. Truică, F. Rădulescu, A. Boicea, and I. Bucur, "Performance evaluation for CRUD operations in asynchronously replicated document oriented database," *2015 20th Int. Conf. Control Syst. Comput. Sci.*, pp. 191–196, May 2015.
[27] R. Aniceto *et al.*, "Evaluating the Cassandra NoSQL Database Approach for Genomic Data Persistency," *Int. J. Genomics*, vol. 2015, pp. 1–7, 2015.
[28] H. Boral and D. DeWitt, "A methodology for database system performance evaluation," in *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, 1984.
[29] A. T. Kabakus and R. Kara, "A performance evaluation of in-memory databases," *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 29, no. 4, pp. 520–525, Oct. 2017.
[30] Z. Jia, L. Wang, J. Zhan, L. Zhang, and C. Luo, "Characterizing data analysis workloads in data centers," in *2013 IEEE International Symposium on Workload Characterization (IISWC)*, Portland, OR, USA, 2013, pp. 66–76.
[31] L. Wang *et al.*, "BigDataBench: a Big Data Benchmark Suite from Internet Services," *2014 IEEE 20th Int. Symp. High Perform. Comput. Archit. HPCA*, pp. 488–499, Feb. 2014.
[32] R. Panda and L. K. John, "Proxy Benchmarks for Emerging Big-data Workloads," in *26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2017, pp. 105–116.
[33] Y. Chen, S. Alspaugh, and R. Katz, "Interactive Analytical Processing in Big Data Systems: A Cross-Industry Study of MapReduce Workloads," *ArXiv12084174 Cs*, Aug. 2012.
[34] R. D. L. Engle, "A Methodology for Evaluating Relational and NoSQL Databases for Small-scale Storage and Retrieval," Air Force Institute of Technology, 2018.
[35] C. Gyorodi, R. Gyorodi, G. Pecherle, and A. Olah, "A comparative study: MongoDB vs. MySQL," in *2015 13th International Conference on Engineering of Modern Electric Systems (EMES)*, Oradea, Romania, 2015, pp. 1–6.