


A Benchmark for Performance Evaluation of a Multi-Model Database vs. Polyglot Persistence

Feng Ye, Hohai University, China*

 <https://orcid.org/0000-0003-0005-2073>

Xinjun Sheng, Hohai University, China

Nadia Nedjah, State University of Rio de Janeiro, Brazil

Jun Sun, Hohai University, China

Peng Zhang, Jiangsu Provincial Water Conservancy Engineering Planning Office, China

ABSTRACT

As the need for handling data from various sources becomes crucial for making optimal decisions, managing multi-model data has become a key area of research. Currently, it is challenging to strike a balance between two methods: polyglot persistence and multi-model databases. Moreover, existing studies suggest that current benchmarks are not completely suitable for comparing these two methods, whether in terms of test datasets, workloads, or metrics. To address this issue, the authors introduce MDBench, an end-to-end benchmark tool. Based on the multi-model dataset and proposed workloads, the experiments reveal that ArangoDB is superior at insertion operations of graph data, while the polyglot persistence instance is better at handling the deletion operations of document data. When it comes to multi-thread and associated queries to multiple tables, the polyglot persistence outperforms ArangoDB in both execution time and resource usage. However, ArangoDB has the edge over MongoDB and Neo4j regarding reliability and availability.

KEYWORDS

Availability, Benchmark, Multi-Model Database, noSQL, Polyglot Persistence, Reliability, Workload

INTRODUCTION

There is an increasing demand for analyzing and processing multi-model data, including structured, semi-structured, and unstructured data. In particular, structured data commonly refer to relational, key-value, and graph data; semi-structured data mainly include JSON and XML documents; and unstructured data are typically text files. For multi-model data management, it is inevitable and difficult for developers to make trade-offs between multi-model databases and polyglot persistence.

DOI: 10.4018/JDM.321756

*Corresponding Author

This article published as an Open Access article distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/4.0/>) which permits unrestricted use, distribution, and production in any medium, provided the author of the original work and original publication source are properly credited.

However, existing studies suggest current benchmarks are not completely suitable for evaluating and comparing multi-model databases and polyglot persistence, whether in terms of test datasets, workloads, or metrics. First, obtaining large-scale real multi-model data is difficult and costly, and few data generators can generate multi-model test datasets. Second, the workloads of the existing benchmarks are not comprehensive and cannot cover diversified multi-model data application scenarios. Finally, most of the existing benchmarks pay more attention to the execution time of the workloads while ignoring the metrics of infrastructure resource usage and nonfunctional attributes. Specifically, in a distributed environment, database system failure is considered a normal event rather than an accident (Ghemawat et al., 2003), so collecting and measuring database resource usage and nonfunctional attributes is very important. However, as far as we know, there is no benchmark for multi-model databases and polyglot persistence that takes resource usage and nonfunctional attributes as metrics. Aiming at these problems, we propose an end-to-end benchmark named MDBench for evaluating and comparing a multi-model database and polyglot persistence. The main contributions of this paper are summarized as follows:

1. A scalable multi-model data generator is designed for generating multi-model test datasets. The key algorithm of the data generator is efficient to ensure that no matter how large the dataset is generated, it will not cause serious out-of-memory resources.
2. Four groups of representative workload experiments are designed and implemented to simulate different multi-model data application scenarios. In particular, a multi-thread workload experiment and reliability and availability experiments are conducted in the research field of evaluation and comparing multi-model databases and polyglot persistence.
3. Based on data store selection, we use MDBench to implement a comprehensive performance evaluation on the single multi-model database ArangoDB and a polyglot persistence instance that consists of MongoDB and Neo4j and systematically analyze the experimental results.

The subsequent contents are organized as follows. First, the research status of database benchmarking is summarized. In the next section, we introduce the data stores involved in the evaluation and the reasons for selecting them. Then, MDBench is introduced in detail from three aspects: multi-model data generation, workloads, and metrics mechanism. Next, the experimental results are introduced and analyzed. Finally, the paper is summarized and proposed.

OVERVIEW OF DBMS BENCHMARKS

The database benchmark can perform repeatable, comparable, quantitative tests on performance indicators. Existing database benchmarks in the industry can be divided into the following two categories: RDBMS and NoSQL benchmarks. The multi-model database benchmarks belong to NoSQL benchmarks. Because of the particularity of its data model, we will also introduce multi-model database benchmarks separately.

RDBMS Benchmarks

This kind of benchmark research work started early and had a wide range, and they are also the driving force behind the rapid development of relational database systems. For example, the Wisconsin benchmark (Bitton et al., 1983) consisted of 32 SQL statements and took the total time to execute the full workloads as the only metric. DebitCredit was designed by the Tandem team, which simulated a real-world transaction scenario, measuring the throughput and cost-effectiveness of various transaction processing systems. The TPC test benchmark, jointly established by Microsoft, Intel, HP, and others, was the standard for evaluating RDBMS. It tested the DBMS's ACID characteristics, query speed, and online transaction processing capabilities. Among the thirteen TPC benchmarks, TPC-C and

TPC-E were designed for the OLTP databases; TPC-H and TPC-DS were designed for the decision support system. Currently, these benchmarks remain the key choice for DBMSs to provide data management solutions.

NoSQL Benchmarks

The development process of data management technology is to continuously integrate semi-structured and unstructured data into DBMS to reduce cost and improve efficiency. For each NoSQL store, there are different benchmarks for evaluating and comparing related big data systems, such as XBench (Yao et al., 2004), YCSB (Matallah et al., 2017), YCSB++ (Patil et al., 2011), BG (Alabdulkarim et al., 2018), BigDataBench (Zhan et al., 2016), and CloudSuite (Ferdman et al., 2012). XBench is a stand-alone XML benchmark that covers an overall database design defined by application categories. It tested the scalability of the database and the full XQuery functionality captured in the XML query case. YCSB is an open-source tool used by Yahoo to evaluate the performance of computer programs. It compared different data stores “apple to apple” regarding performance, elasticity, and availability. The YCSB framework consists of a workload generation client and standard workloads covering all aspects of the performance space. YCSB++ extends YCSB to evaluate the advanced features of NoSQL storage. BG is a benchmark for evaluating interactive social network behavior, which simulates social network behavior by reading or updating database operations. CloudSuite is designed for scale-out cloud applications and provides popular scale-out workloads to evaluate different NoSQL stores deployed in cloud architectures. CloudSuite also provides a list of real datasets and supports the extension of these datasets.

Multi-Model Database Benchmarks

As a part of NoSQL benchmarks, the multi-model database benchmark is listed separately due to the particularity of its data model. According to Messaoudi et al. (2017, 2018), in biomedical big data, the authors selected a single multi-model database OrientDB and a polyglot persistence instance composed of MongoDB and Neo4j to carry out performance evaluation with multiple workloads, such as insertion, deletion, and search operations. The results showed that MongoDB performed better than OrientDB in processing document data, and OrientDB performed better than Neo4j in querying graph data when the depth of the graph reached three layers. Although many workloads were involved in the evaluation, evaluating execution times alone did not give a complete picture of the capabilities of different data stores. Shah et al. (2014) evaluated eight databases, including OrientDB, Neo4j, and TitanDB, from two aspects of processing time and disk space usage. They found that OrientDB, Neo4j, and TitanDB performed well in persistence. Neo4j and MongoDB performed well in terms of query performance. Despite the fewer workload types, Shah et al. evaluated a wider range of databases than other researchers. Bagga and Sharma (2020) compared six databases, including MongoDB, CouchDB, and HBase, from backup, consistency, partition, and performance. Fernandes and Bernardino (2018) evaluated the graph and multi-model databases with graph function from seven aspects: storage mode, query language, partitioning, backup, multi-model, multi-architecture, extendibility, and cloud deployment. The experimental results showed that Neo4j and ArangoDB had the best performance. We can see that both teams are more focused on functional attributes. Macak et al. (2020) compared MongoDB and Neo4j with the multi-model database OrientDB from the perspective of eight query workloads. Finally, it was found that Neo4j was more efficient than OrientDB in processing graph data with a depth of less than four, and OrientDB performed better when the depth was greater than four, while MongoDB query efficiency was much higher than OrientDB in processing document data. Although only the time consumed by the query workload was measured, the measurements from Macak et al. covered many graph and document data query workloads. This research was of great reference value for those application scenarios with many queries. Jayathilake et al. (2012) used a column, document, tuple, graph, and multi-model database to process tree data. Membase showed the lowest latency and the highest throughput during tree creation.

On the other hand, the graph database Neo4j and multi-model database have achieved excellent results in data retrieval. Tree data are an important data type, so the research results fill the gap in the NoSQL database evaluation of tree data. The experiment run by Oliveira and del val Cura (2016) compared the combination of ArangoDB and OrientDB with MongoDB and Neo4j. The experiment could be divided into two parts: insert and query. ArangoDB inserted document data efficiently, while MongoDB inserted document data efficiently when there were many fields. ArangoDB was the most efficient when inserting graph data. In the query part, when the depth was less than two, the performance of ArangoDB was better; when the depth was between two and four, the performance of OrientDB was better; and when the depth was greater than four, the performance of the combination of MongoDB and Neo4j was better. Although only insert and query workloads were evaluated, Oliveira and del val Cura's experimental findings in graph depth traversal laid the foundation for the results reported by Macak et al. (2020).

We list the characteristics of six representative benchmarks from three aspects: dataset, workload and metric in Table 1. However, it can be seen from Table 1 that in the performance evaluation of different databases, most of them focus on the execution time of workloads while ignoring the occupation of hardware resources. The type of workload is relatively singular. The most incredible thing is that all benchmarks ignore the evaluation of the multi-thread workload. Therefore, we propose an end-to-end benchmark named MDBench for multi-model databases and polyglot persistence, aiming to provide a comprehensive solution for storing and managing multi-model data.

OVERVIEW OF THE EXPERIMENTAL DATABASE

Selecting the right objects for benchmarking is the starting point. Performance, price, and energy consumption are the most common metrics for computer program evaluation (Han et al., 2017). Therefore, based on the above standards, we select outstanding databases in various data model fields for benchmarking. Here, MongoDB and Neo4j are selected as the instances of polyglot persistence, and ArangoDB is representative of a single multi-model database for this study. Next, we will introduce the characteristics and selection basis of these data stores based on the literature analysis and comparison.

MongoDB

MongoDB is a document-oriented and scalable high-performance database (Banker et al., 2016, Plugge et al., 2015), whose efficient indexing mechanism brings high-speed queries that make it stand out among NoSQL databases (Zong et al., 2017).

Truică et al. (2018) proposed T2K2 and T2K2D2 benchmarks and used them to test the performance of MongoDB, Oracle, and PostgreSQL. Experimental results showed that MongoDB performed better than Oracle and PostgreSQL in calculating top-K keywords and documents. Truică et

Table 1.
Benchmark comparison

Benchmark	Dataset Extensibility	Workloads			Metrics	
		Multi-Thread	Join Table	Transaction	Time	Resource
DebitCredit				√	√	
TPC-C			√	√	√	
TPC-H			√	√	√	
YCSB				√	√	
XBench	√		√		√	
UniBench	√		√	√	√	

al. (2021) also proposed a universal document-oriented distributed benchmark TEXTBENDS, which was used to evaluate the computational efficiency of word weighting under two different weighting schemes: TF-IDF and Okapi BM25. Comparing MongoDB, Hive, and Spark, the experimental results showed that MongoDB had the best overall performance. Mishra et al. evaluated the performance of four document databases and databases with a document model. When comparing database throughput and runtime in a single-threaded state, MongoDB outperformed other databases with the highest throughput and lowest runtime. In a comprehensive analysis of MongoDB and ArangoDB for some threads under different workloads, MongoDB outperformed ArangoDB by a high percentage.

Neo4j

Neo4j is a high-performance graph database engine whose unique Cypher language enables convenient graph data processing (Holzschuher & Peinl, 2013). It follows the characteristics of the graph data model to maintain three data structures: nodes, relationships, and attributes. In addition, it has the characteristics of reliability, transactional, high availability, and security (Miller, 2013). Although Neo4j is a relatively new open-source project, it has been used in over 100 million nodes and meets enterprise robustness and performance requirements.

Beis et al. (2015) conducted a comprehensive comparative evaluation of three popular graph databases, Titan, OrientDB, and Neo4j. Experimental results showed Neo4j was the most efficient graph database for most workloads. Only by knowing the capabilities and limitations of each system can researchers know where to focus their efforts. Therefore, Lissandrini et al. (2018) conducted a comprehensive performance evaluation and analysis of seven graph databases: ArangoDB, BlazeGraph, Neo4j, OrientDB, Sparksee, SQLG, and Titan. The results showed that Neo4j and the other three databases performed better in graph traversal. Furthermore, completing the entire set of queries in a single and batch manner was the most efficient. Dominguez et al. (2010) evaluated four of the most scalable native graph databases, Neo4j, HypergraphDB, Jena, and DEX, against the HPC extensible graph analysis benchmark and tested the performance of each database for different typical graph operations and graph sizes. The results showed that Neo4j and DEX were the most efficient graph databases.

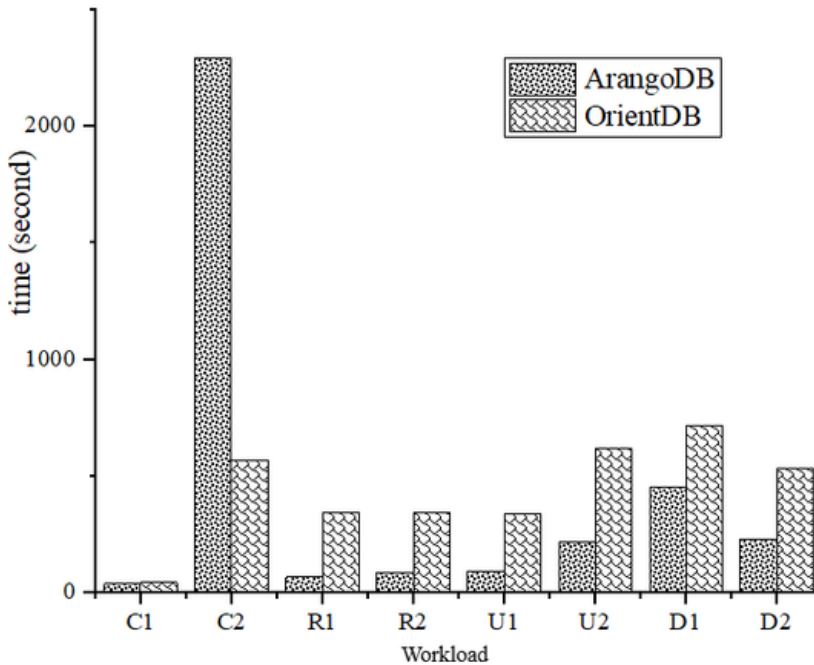
ArangoDB

In ArangoDB, documents are stored in collections. Collections use `_id` to uniquely identify each document. The `_id` can be assigned by the user at creation time or automatically generated by ArangoDB. Indexes are created for both the `_id` and `_key` attributes, where the index on the `_key` attribute is called the primary index, which exists in each collection and cannot be deleted. There are two types of sets in ArangoDB: vertex sets and edge sets. Documents in an edge collection have two additional attributes, `_from` and `_to`. Both must be bound to the corresponding vertex document's `_id` attribute. ArangoDB uses the ArangoDB Query Language (AQL) to manipulate graphs or collections. AQL syntax is different from SQL syntax, although many of the same keywords exist. Compared with SQL syntax, AQL is more powerful and read-write.

Currently, OrientDB and ArangoDB are representative and influential multi-model databases. Zhang et al. (2018) proposed the UniBench benchmark for multi-model database evaluation and evaluated OrientDB and ArangoDB, and the experimental results show that ArangoDB performed better than OrientDB in most cases.

While there are few studies on multi-model databases, there is no other relevant research except Chao's evaluation of multi-model databases above. Therefore, we should select OrientDB or ArangoDB as the multi-model database in the experiment. Based on workloads {C1, C2, R1, R2, U1, U2, D1, D2}, this paper compares the running times of OrientDB and ArangoDB. According to the experimental results shown in Figure 1, it can be seen that the overall performance of ArangoDB is indeed better than OrientDB under basic document operation and graph operation. Therefore, they chose ArangoDB as the multi-model database for the experiment.

Figure 1.
ArangoDB vs. OrientDB



In summary, by analyzing and comparing existing studies, the single multi-model database ArangoDB and a specific polyglot persistence instance composed of MongoDB and Neo4j are chosen as research objects for benchmarking and comparison.

THE BENCHMARK PROPOSED

From a macro perspective, the three elements of the benchmark are data, workload, and metrics mechanism (Xia et al., 2015). This section will present our end-to-end benchmark from the three perspectives above.

Multi-Model Data Generation

One of the challenges facing the performance evaluation of multi-model databases and polyglot persistence is the lack of a large-scale multi-model dataset. Previous data generators have focused on single-model data. Combining multiple single-model data generators can increase system instability because we tailored each data generator to a specific application scenario. Jiaheng Lu and his team proposed a multi-model data generator in UniBench that can generate JSON, XML, relational, document, and graph data. However, the data generator has different requirements on the hardware operating environment according to different workload factors. The data generator of MDBench is realized after optimization of the data generator based on UniBench. Compared with UniBench, the data generator proposed in this paper occupies very low memory and saves hardware resources to a large extent. Compared to the pre-optimized data generator, the optimized data generator frees up three-quarters of the memory space. Algorithm 1 shows the implementation process of the data generator. This data generator generates the dataset used in the relevant experiments presented in this paper. Here, we select two types of data: document data and graph data. The document data

comprises commodity and order information, while the graph dataset comprises customers and their social networks. In addition, the productId in the order points to the item's primary key, and the personId points to the customer in the graph dataset. The orderId in the suborder points to the order primary key. We can see the specific information and relationship between goods, orders, and customers in Figure 2.

Workloads

There are many types of workloads, and some database vendors focus on query performance, while others focus on transaction consistency. Different databases behave differently even though they handle the same workloads, so the workloads should be designed with broad coverage. To explore and compare the processing capability of multi-model database and polyglot persistence on different workload types, to make the evaluation scenario similar to the real big data application scenario, and reflect the use case of the real environment, a series of workloads are designed, as shown in Table 2. Each workload contains a label, brief description, data model, and quantity. Categorized from the perspective of create, delete, update and query, C = {C1, C2, C3, C4, C5, C6} are the insert workloads, R = {R1, R2, R3, R4, R5, R6, R7} are the query workloads, U = {U1, U2, U3} are the update workloads, and D = {D1, D2, D3, D4, D5, D6} are the delete workloads. From the perspective of data type, D = {C1, C3, C4, C5, R1, R3, U1, U3, D1, D3, D4, D5} are the document workloads, G = {C2, C6, R2, U2, D2, D6} are the graph workloads, and M = {R4, R5, R6, R7} are the multi-model workloads.

Measurement Mechanism

Figure 4 is an architecture diagram of the MDBench. It comprises four parts: database cluster, resource monitoring unit composed of Prometheus and Grafana, data pipeline unit composed of Zookeeper and Kafka, and workload injection unit written in Java language.

Previous metrics of database benchmarks have mainly focused on the execution time of workloads. However, at a time when data volumes are exploding, it is not reasonable to focus solely on execution

Figure 2.
Relationship between datasets



Figure 3.
Algorithm

Algorithm 1 multi-model data generator

Input: $\alpha, \beta, \gamma, \delta, \epsilon, \text{factor}$
Output: C,P,O,G

```

1: initialize: Set  $\alpha = 1000, \beta = 1000, \gamma = 1000, \delta = 5, \epsilon = 1000$ 
2:  $\alpha = \alpha * \text{factor}$ 
3:  $\beta = \beta * \text{factor}$ 
4:  $\gamma = \gamma * \text{factor}$ 
5:  $\epsilon = \epsilon * \text{factor}$ 
6: for  $t = 1, 2, \dots, \alpha$  do
7:   Customer customer = new Customer()
8:    $C \leftarrow \text{customer}$ 
9: end for
10: for  $t = 1, 2, \dots, \beta$  do
11:   Product product = new Product()
12:    $P \leftarrow \text{product}$ 
13: end for
14: for  $t = 1, 2, \dots, \gamma$  do
15:   Order order = new Order()
16:    $O \leftarrow \text{order}$ 
17:   for  $t = 1, 2, \dots, \delta$  do
18:     SubOrder suborder = new SubOrder()
19:      $\text{order} \leftarrow \text{suborder}$ 
20:   end for
21:    $\text{random}(\text{order.id}) \in \alpha$ 
22: end for
23: for  $t = 1, 2, \dots, \epsilon$  do
24:    $\text{random}(\text{edge.source}) \in \alpha$ 
25:    $\text{random}(\text{edge.target}) \in \alpha$ 
26:   if  $\text{edge} \in G$  then
27:      $t = t - 1$ 
28:     continue
29:   else
30:      $G \leftarrow \text{edge}$ 
31:   end if
32: end for

```

time. Therefore, compared with the previous database benchmark, the measurement mechanism of MDBench proposed in this paper measures the experimental results from four dimensions. The first dimension is the execution time of workloads, the second dimension is the resource occupation, the third dimension is reliability, and the fourth dimension is availability.

Execution Time

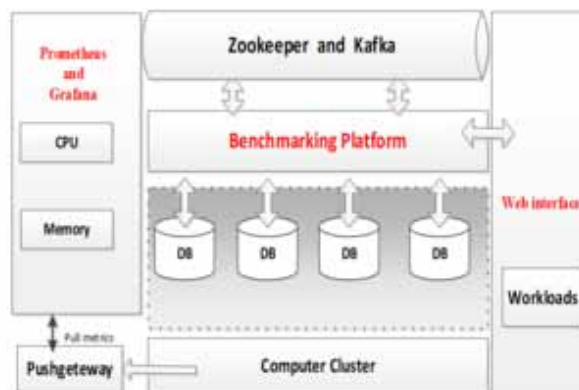
The execution time T is measured using the Timer class built into the Java language's software Development Kit (JDK). In these experiments, the statistics of execution time follow Formula (1), where w_{is} is the start time of the i_{th} workload, and w_{ie} is the end time of the i_{th} workload.

$$T = w_{ie} - w_{is} \quad (1)$$

Table 2.
Workloads

Label	Description	Data Model	Quantity
C1	Create customer information	Document	9949
C2	Create customers' social network	Graph	187810
C3	Create orders and suborders	Document	636167
C4	Create product information	Document	10116
C5	Create document data(C3+C4)	Document	846283
C6	Create graph data(C1+C2)	Graph	197759
R1	Query customer information	Document	9949
R2	Query who a customer knows	Graph	187810
R3	Query product information with the primary key	Document	10116
R4	Given the customer's name, query the total amount of orders paid by the customer	Multi-model	1000
R5	Given the customer's primary key, query which products they have purchased	Multi-model	1000
R6	Given the name of a product, query the information of customers who bought the product	Multi-model	1000
R7	Given the name of a product, query what other products the woman who bought the product has purchased	Multi-model	1000
U1	Update customer information	Document	9949
U2	Update who a customer knows	Graph	187810
U3	Update product information according to the primary key of the product	Document	10116
D1	Delete customer information	Document	9949
D2	Delete customer social networks	Graph	187810
D3	Delete orders and suborders	Document	836167
D4	Delete products	Document	10116
D5	Delete document data(D3+D4)	Document	846283
D6	Delete graph data(D1+D2)	Graph	197759

Figure 4.
The architecture of MDBench



Resource Occupation

Resource occupation is monitored by the distributed monitoring unit Prometheus, which collects server performance data. Meanwhile, time series data collected by Prometheus are presented by Grafana in the form of graphs, which is an interface tool. CPU and memory statistics follow Formula (2), where r_i is the CPU or memory consumed by the i_{th} workload.

$$w = \frac{\sum_{i=1}^n r_i}{n} \quad (2)$$

Reliability

The reliability metric is the ratio of successfully responded requests to the total number of requests, as shown in Formula (3).

$$Reliability = \frac{TotalRequest - Unsuccessful\ Responses}{TotalRequest} \quad (3)$$

Availability

The measure of availability is the ratio of effective working time to total working time, as shown in Formula (4).

$$Availability = \frac{TotalInServiceTime - Downtime}{TotalInServiceTime} \quad (4)$$

DESIGN OF EXPERIMENTS

From the perspective of the practical application of big data, this paper divides the experiment into four groups. They are the single table workload experiment, multi-thread workload experiment, multi-table joint query experiment, reliability and availability experiment. We tune each type of database for performance prior to experimentation to ensure that the database maximizes its advantages.

Experimental Configuration

The experiment runs on three servers containing one master node and two slave nodes. The master node is configured with eight-core 16 GB memory, and the two slave nodes are configured with four-core CPU and 8 GB memory. Three databases related to the experiment, including MongoDB, Neo4j, and ArangoDB, are installed and deployed on three servers in a distributed architecture. The evaluation platform is implemented in Java and runs on a single slave node, so the server has a preinstalled dependency environment. The measurement of experimental results is divided into two parts: execution time and resource occupancy. The execution time of the workload is measured using the Java Development Kit (JDK). The resource occupancy is measured by distributed resource monitoring platforms Prometheus and Grafana. We show the software and hardware parameters in Table 3.

Table 3.
Software and hardware parameters

OS	CentOS 7.6
CPU	Intel(R) Xeon(R) Platinum 8269CY CPU @ 2.50 GHz
Turbo Boost	Active
Hyper Threading	Active
JDK version	1.8.0_291
The heap size	4GB(master),2GB(slave)
ArangoDB version	3.9.0
MongoDB version	5.0.6
Neo4j version	4.3.10
Zookeeper version	3.7.0
Kafka version	2.13-2.8.0
Prometheus version	2.28.0
Grafana version	8.2.3

COMPARISON OF THE MULTI-MODEL DATABASE AGAINST POLYGLOT PERSISTENCE

This paper's experiments on the performance evaluation of a multi-model database and polyglot persistence comprise four parts: a single-table workload experiment, a multi-thread workload experiment, a multi-table joint query experiment, and a reliability and availability experiment.

The first part of the experiment is a single-table workload experiment. This is because we must migrate the data in the database for practical applications, and persistence is inevitable in iterative operation systems. For example, in data migration, the consumption of time and resources to the downstream consumer must be predictable. Otherwise, it will directly affect the normal operation of the downstream system. Therefore, it is necessary to measure this kind of single-table workload.

The second part of the experiment is the multi-thread workload experiment. With the popularization of the Internet and the intelligence of mobile devices, servers are facing increasing concurrency pressure. The processing of sudden and high concurrent requests is the ability that distributed databases should have, and it is also the necessary condition for databases to be put into production and life.

The third part is the multi-table joint query experiment. In the real application scenario, with time and business development, the data in the tables will increase, increasing the cost of database operations. Therefore, at the beginning of the system, developers will cut data separately based on factors such as function modules and data relationships, using tables to store them. While these data are needed, we can query them via an associated query to multiple tables. Except for data migration and persistence involving only one table, most operations involve associated queries to multiple tables. Therefore, the effect of multi-model databases and polyglot persistence-associated queries on multiple tables is also a concern of users. The above three experiments will use measurements from execution time and resource occupation.

The fourth part is the reliability and availability experiment. E Bauer and R Adams proposed the calculation formula for reliability and availability in 2012 (Bauer and Adams., 2012). In this experiment, we sent 1000 requests to the evaluation database. During the request process, we adopt fault injection to simulate the restart of the server after power failure. The quantified reliability result is obtained by calculating the ratio of the number of successfully responded requests to the total number

of requests. The availability is quantified by the time the last request responded before the restart and the time the first request responded to after the restart. We will perform the reliability and availability tests for each database five times, averaging the remaining three times by removing one maximum and one minimum. The reliability and availability of polyglot persistence follow the Cannikin law.

RESULTS AND DISCUSSION

Single Table Workload Experiment

Single table workloads are the simplest of all workload types and are the basis of all workload types. The measurement of single table workloads includes a total of eight workloads. C5, R3, U3, and D5 are the CRUD workloads of document data, while C6, R2, U2, and D6 are the CRUD workloads of graph data. We measure experiments from execution time and resource occupation. Figure 5 shows the comparison of processing time for single-table workloads, and Figure 6 shows the comparison of resource occupation for single-table workloads.

Figure 5 shows that the ArangoDB takes almost twice as long to delete document data as polyglot persistence. When inserting graph data, polyglot persistence took nearly three times as long as ArangoDB. In the case of handling other workloads, there is little difference between polyglot persistence and ArangoDB.

From the comparison of resource occupation of single table workloads in Figure 6, we can see that in most cases, the CPU consumption of polyglot persistence is higher than that of ArangoDB, and the memory consumption of ArangoDB is generally higher than that of polyglot persistence.

Figure 5. T
ime consumption of ArangoDB (MD) and polyglot persistence (PP) when processing a single table workload

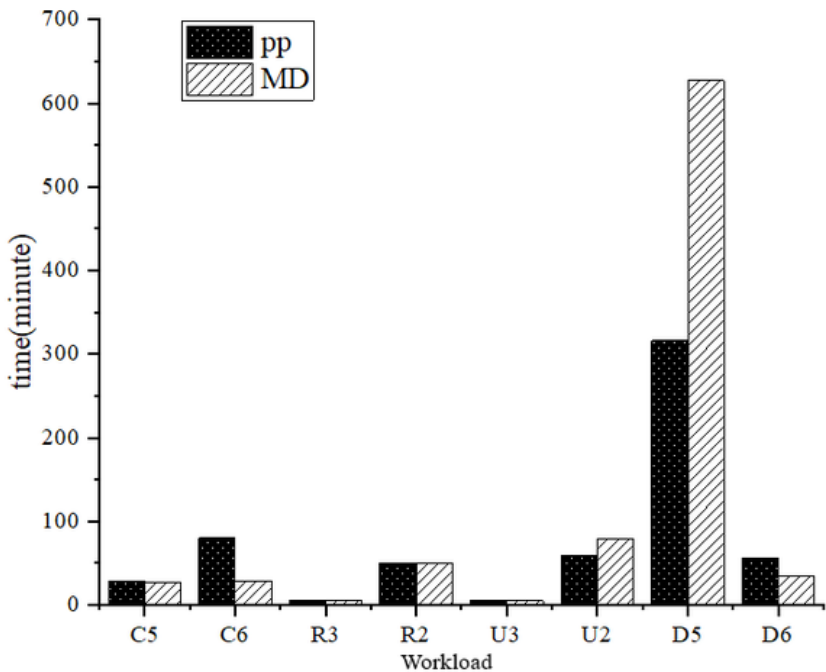
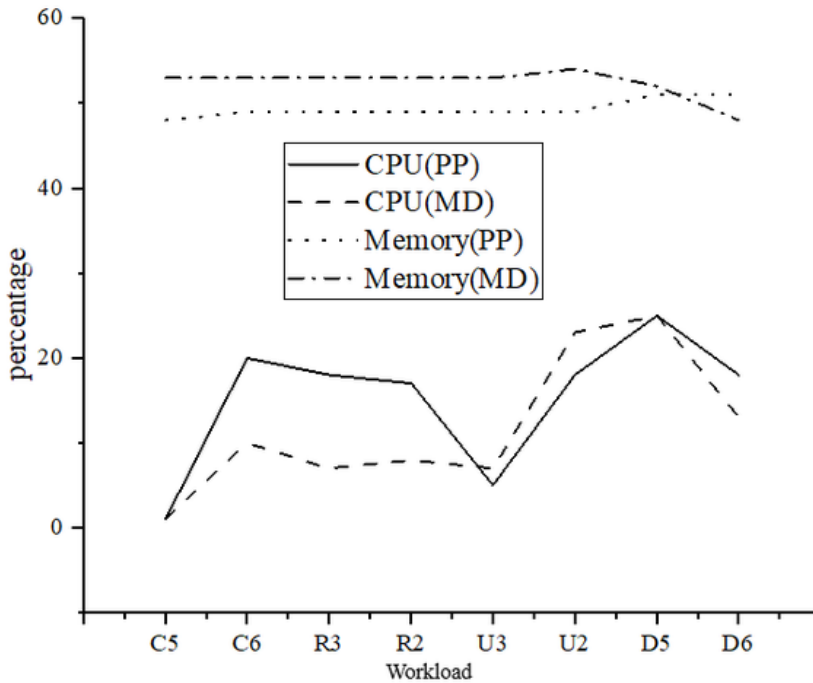


Figure 6.
CPU and memory usage of ArangoDB (MD) and polyglot persistence (PP) when processing a single table workload



Multi-Thread Workload Experiment

The multi-thread experiment simulates a high-concurrency application scenario by controlling the number of threads created by the evaluation platform. R4 is selected to carry out the multi-thread workload experiment. R4 is a mixed workload involving document and graph operations. The evaluation platform measures the experimental results from two dimensions: execution time and resource occupation.

Figure 7 shows the comparison of ArangoDB and polyglot persistence under multi-thread workloads. As we observe in Figure 7, the execution time changes significantly as the number of threads increases from 1 to 5. This is because both ArangoDB and polyglot persistence can handle high concurrency scenarios. However, when the number of threads increases from 5 to 80, the ArangoDB and polyglot persistence processing times do not change, which is normal because all systems that support high concurrency have a performance ceiling on the number of concurrent processes they can support.

Figure 8 shows the resource occupation comparison of ArangoDB and polyglot persistence under multi-thread workloads. As the figure shows, the memory occupation of ArangoDB and polyglot persistence is remarkably stable, consistently at 40%. We find that when the number of threads is small, the CPU usage of polyglot persistence is significantly higher than that of ArangoDB. As the number of threads increases, the CPU usage of both approaches 100%.

Multi-Table Joint Query Experiment

This paper selects R4, R5, R6, and R7 to conduct the multi-table joint query experiment. This part of the experiment uses a vector-based method to represent the results of the associated query to multiple tables. Specifically, based on known parameters, an intermediate result is first queried, and then the eventual result is obtained progressively based on the intermediate result. For example, R4 is given the

Figure 7.
Time consumption of ArangoDB (MD) and polyglot persistence (PP) when processing workload R4

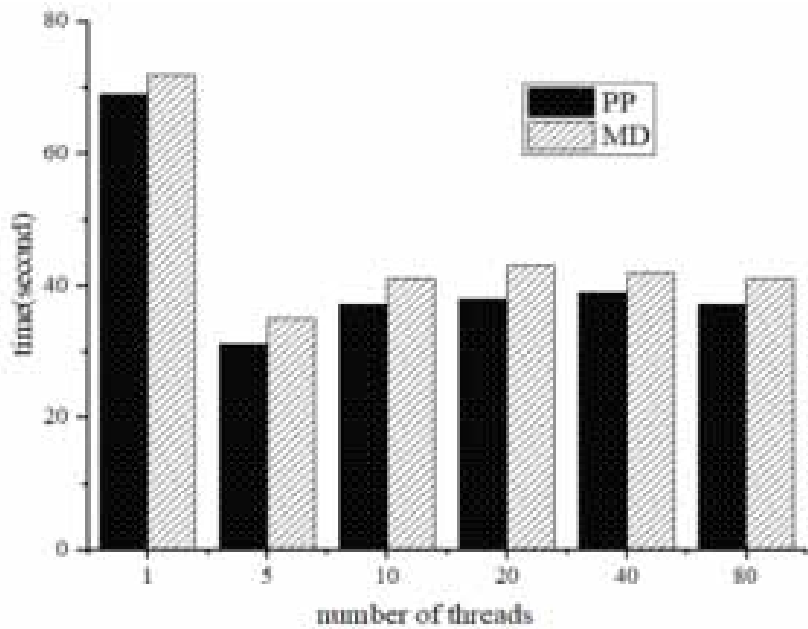
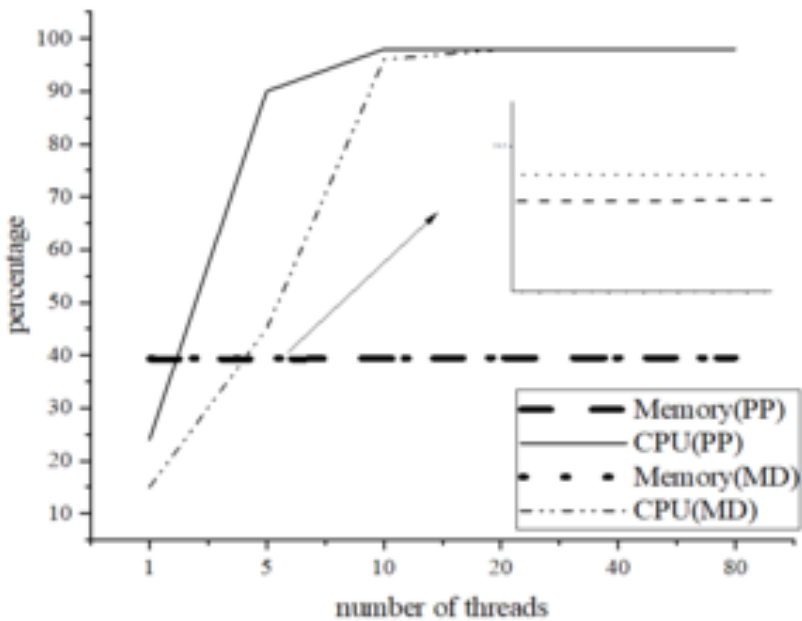


Figure 8.
CPU and memory usage of ArangoDB (MD) and polyglot persistence (PP) when processing workload R4



customer’s name and queries the total price of orders paid by the customer. According to the customer’s name, a null vector can be calculated (|C|, |O|, |CO|), including the size of the vector |C| on behalf of the customer table, |O| represents the size of the order table, and |CO| on behalf of the associated query result. In the same way, R5 can be expressed as (|O|, |S|, |OSPI|), where |S| represents the size of the order table and |PI| represents the size of the goods table. R6 can be expressed as (|PI|, |S|, |O|, |C|, |PSOC|). R7 can be expressed as (|X|, |Y|, |XY|), where $X = (|PI|, |S|, |O|, |C|, |PSOC|)$, $Y = (|O|, |S|, |OSI|)$. This method can reflect how many associated queries the workload contains and the size of the intermediate results at each step.

Figure 9 compares the time taken by ArangoDB and polyglot persistence under the workload of the associated query. It can be seen from the figure that with the increase in the number of associated tables, the time taken by ArangoDB and polyglot persistence increases gradually. At the same time, we can find that the time of polyglot persistence is always less than that of ArangoDB.

Figure 10 displays the resource occupation comparison of ArangoDB and polyglot persistence under the associated query to multiple tables. It can be seen from the figure that the number of associated tables has little influence on CPU and memory consumption, except that the consumption of CPU and memory of polyglot persistence increases slightly when the number of associated tables increases from 4 (R6) to 6 (R7).

Reliability and Availability Experiment

Table 4 shows the number of failed response requests among 1000 in the reliability experiment. Table 5 shows the time in milliseconds for each database processing fault in the availability experiment. Reliability and availability calculations follow the Cannikin law to avoid experimental contingency, removing one maximum and one minimum and averaging the remaining three values. In the end, the reliability of ArangoDB was 97.40%, and the availability was 97.19%. Polyglot persistence has

Figure 9.
Time consumption of ArangoDB (MD) and polyglot persistence (PP) when processing the associated query to multiple tables

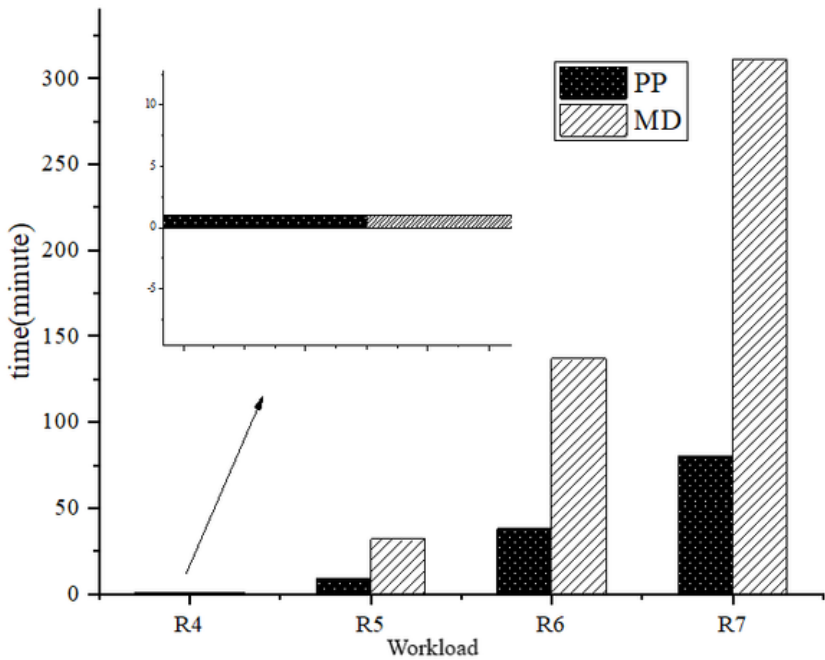


Figure 10.
CPU and memory usage of ArangoDB (MD) and polyglot persistence (PP) when processing the associated query to multiple tables

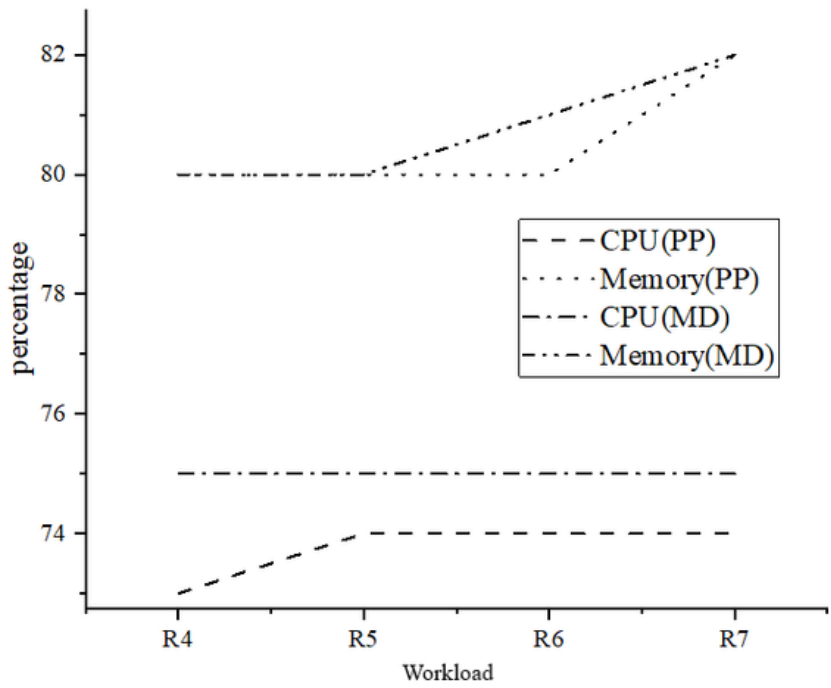


Table 4.
The number of failed response requests

MongoDB	Neo4j	ArangoDB
2	3	55
1	5	5
1	5	18
2	2	44
2	3	16

Table 5.
The time of database processing fault

MongoDB	Neo4j	ArangoDB
41	7365	3192
189	6540	2353
69	6889	2679
47	4706	3226
66	7282	2568

99.63% reliability and 93.10% availability. It can be seen that the availability of ArangoDB is higher than polyglot persistence, and the reliability is not as good as polyglot persistence.

Discussion

This section measures three sets of experiments: a single-table workload experiment, a multi-thread workload experiment, and a multi-table joint query experiment regarding execution time and resource occupation. In addition, we also performed reliability and availability experiments. According to the experimental results, we can draw the following conclusions.

We recommend ArangoDB for scenarios with heavy single-table workloads and a high proportion of graph data creation operations. We recommend polyglot persistence for a high percentage of document data deletion operations. If server memory is tight, polyglot persistence is recommended. If the server CPU is tight, we advise using ArangoDB.

We recommend polyglot persistence for high concurrency scenarios regarding execution time and resource occupancy. At the same time, it also shows that although research on multi-model databases has made remarkable progress in recent years, there are still many deficiencies. As we can see from this part of the experiment, ArangoDB is not as good at handling concurrency as polyglot persistence.

For applications where business operations are more complex, polyglot persistence is still better than ArangoDB regarding execution time and resource occupancy. However, especially in terms of execution time, as the number of tables involved increases (R4 to R7), so does the gap between polyglot persistence and ArangoDB.

We recommend ArangoDB if the requirements for availability are high. We recommend polyglot persistence if the requirements for reliability are high.

SUMMARY AND PROSPECTS

The benchmark MDBench proposed in this paper evaluates the selected multi-model database and polyglot persistence from execution time, resource occupation, reliability, and availability. The evaluation experiment comprises four parts: the first part is the single table workload experiment, the second part is the multi-thread workload experiment, the third part is the multi-table joint query experiment, and the fourth part is the reliability and availability experiment. Through the four experiments, we can comprehensively understand the execution time, resource occupation, reliability, and availability of ArangoDB and polyglot persistence to provide a reference for users to store multi-model data.

At present, MDBench has not evaluated read-write mixed workloads, but workloads are mixed in real application scenarios. Therefore, the evaluation of read-write mixed workloads will be a part of the work in the next stage.

ACKNOWLEDGMENT

The paper was supported by the National Key R&D Program of China (2019YFE0109900); the Jiangsu Province Water Conservancy Science and Technology Project (2022003); The Key technology research project of Tide Control Safety System Improvement in Nansha District of Guangzhou (823005916).

REFERENCES

- Alabdulkarim, Y., Barahmand, S., & Ghandeharizadeh, S. (2018). BG: A scalable benchmark for interactive social networking actions. *Future Generation Computer Systems*, 85, 29–38. doi:10.1016/j.future.2018.02.031
- Bagga, S., & Sharma, A. (2020). A comparative study of NoSQL databases. In P. K. Singh, Y. Singh, M. H. Kolekar, A. K. Kar, J. K. Chhabra, & A. Sen (Eds.), *Recent Innovations in Computing* (pp. 51–61). Springer. doi:10.1007/978-981-15-8297-4_5
- Banker, K., Garrett, D., Bakkum, P., Verch, S., Garret, D., & Hawkins, T. (2016). *MongoDB in action: Covers MongoDB version 3.0*. Simon and Schuster.
- Bauer, E., & Adams, R. (2012). *Reliability and availability of cloud computing*. Wiley. doi:10.1002/9781118393994
- Beis, S., Papadopoulos, S., & Kompatsiaris, Y. (2015). Benchmarking graph databases on the problem of community detection. In N. Bassiliades, M. Ivanovic, M. Kon-Popovska, Y. Manolopoulos, T. Palpanas, G. Trajcevski, & A. Vakali (Eds.), *New Trends in Database and Information Systems II* (pp. 3–14). Springer International Publishing. doi:10.1007/978-3-319-10518-5_1
- Bitton, D., DeWitt, D. J., & Turbyfill, C. (1983). *Benchmarking database systems-A systematic approach*. University of Wisconsin-Madison, Department of Computer Sciences. <https://pages.cs.wisc.edu/~dewitt/includes/benchmarking/vldb83.pdf>
- Dominguez-Sal, D., Urbón-Bayes, P., Giménez-Vanó, A., Gómez-Villamor, S., Martínez-Bazan, N., & Larriba-Pey, J. L. (2010). Survey of graph database performance on the hpc scalable graph analysis benchmark. In H. T. Shen, J. Pei, M. T. Özsu, L. Zou, J. Lu, T.-W. Ling, G. Yu, Y. Zhuang, & J. Shao (Eds.), *Web-Age Information Management* (pp. 37–48). Springer. doi:10.1007/978-3-642-16720-1_4
- Ferdman, M., Adileh, A., Kocberber, O., Volos, S., Alisafae, M., Jevdjic, D., Kaynak, C., Popescu, A. D., Ailamaki, A., & Falsafi, B. (2012). Clearing the clouds: A study of emerging scale-out workloads on modern hardware. *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, 37–48. doi:10.1145/2150976.2150982
- Fernandes, D., & Bernardino, J. (2018) Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB. *7th International Conference on Data Science, Technology and Applications*, 373-380. doi:10.5220/0006910203730380
- Ghemawat, S., Gobioff, H., & Leung, S. T. (2003). The Google file system. *Proceedings of the nineteenth ACM symposium on Operating systems principles*, 29-43. doi:10.1145/945445.945450
- Han, R., John, L. K., & Zhan, J. (2017). Benchmarking big data systems: A review. *IEEE Transactions on Services Computing*, 11(3), 580–597. doi:10.1109/TSC.2017.2730882
- Holzschuher, F., & Peinl, R. (2013). Performance of graph query languages: comparison of cypher, gremlin and native access in neo4j. *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, 195-204. doi:10.1145/2457317.2457351
- Jayathilake, D., Sooriaarachchi, C., Gunawardena, T., Kulasuriya, B., & Dayaratne, T. (2012). A study into the capabilities of NoSQL databases in handling a highly heterogeneous tree. *2012 IEEE 6th International Conference on Information and Automation for Sustainability*, 106-111. doi:10.1109/ICIAFS.2012.6419890
- Lissandrini, M., Brugnara, M., & Velegrakis, Y. (2018). Beyond macrobenchmarks: microbenchmark-based graph database evaluation. *Proceedings of the VLDB Endowment*, 12(4), 390-403. doi:10.14778/3297753.3297759
- Macak, M., Stovcik, M., Buhnova, B., & Merjavý, M. (2020) How well a multi-model database performs against its single-model variants: Benchmarking OrientDB with Neo4j and MongoDB. *2020 15th Conference on Computer Science and Information Systems (FedCSIS)*, 463-470. doi:10.15439/2020F76
- Matallah, H., Belalem, G., & Bouamrane, K. (2017). Experimental comparative study of NoSQL databases: HBASE versus MongoDB by YCSB. *Computer Systems Science and Engineering*, 32(4), 307–317.
- Messaoudi, C., Amrou, M., Fissoune, R., & Hassan, B. (2017). A performance study of NoSQL stores for biomedical data. *The Sixth International Conference on Innovation and New Trends in Information Systems*.

- Messaoudi, C., Fissoune, R., & Badir, H. (2018). A performance evaluation of NoSQL databases to manage proteomics data. *International Journal of Data Mining and Bioinformatics*, 21(1), 70–89. doi:10.1504/IJDMB.2018.095556
- Miller, J. J. (2013). Graph database applications and concepts with Neo4j. *Proceedings of the Southern Association for Information Systems Conference*, 2324(36).
- Oliveira, F. R., & del Val Cura, L. (2016). Performance evaluation of NoSQL multi-model data stores in polyglot persistence applications. *Proceedings of the 20th International Database Engineering & Applications Symposium*, 230-235. doi:10.1145/2938503.2938518
- Patil, S., Polte, M., Ren, K., Tantisiriroj, W., Xiao, L., López, J., Gibson, G., Fuchs, A., & Rinaldi, B. (2011). YCSB++: Benchmarking and performance debugging advanced features in scalable table stores. *Proceedings of the 2nd ACM Symposium on Cloud Computing*, 9. doi:10.1145/2038916.2038925
- Plugge, E., Hows, D., Membrey, P., & Hawkins, T. (2015). *The definitive guide to mongodb: A complete guide to dealing with big data using MongoDB*. Apress.
- Shah, S. M., Wei, R., Kolovos, D. S., Rose, L. M., Paige, R. F., & Barmpis, K. (2014). A framework to benchmark NoSQL data stores for large-scale model persistence. In J. Dingel, W. Schulte, I. Ramos, S. Abrahão, & E. Insfran (Eds.), *Model-Driven Engineering Languages and Systems* (pp. 586-601). Springer International Publishing. doi:10.1007/978-3-319-11653-2_36
- Truică, C. O., Apostol, E. S., Darmont, J., & Assent, I. (2021). TextBenDS: A generic textual data benchmark for distributed systems. *Information Systems Frontiers*, 23(1), 81–100. doi:10.1007/s10796-020-09999-y
- Truică, C. O., Darmont, J., Boicea, A., & Rădulescu, F. (2018). Benchmarking top-k keyword and top-k document processing with T2K2 and T2K2D2. *Future Generation Computer Systems*, 85, 60–75. doi:10.1016/j.future.2018.02.037
- Xia, F., Zhou, M., & Jin, C. (2015). Challenges and progress of big data management system benchmarks. *Big Data Research*, 1(1), 81–95. doi:10.11959/j.issn.2096-0271.2015.01.008
- Yao, B. B., Ozsu, M. T., & Khandelwal, N. (2004). XBench benchmark and performance testing of XML DBMSs. *Proceedings of the 20th International Conference on Data Engineering*, 621-632. doi:10.1109/ICDE.2004.1320032
- Zhan, J.-F., Gao, W.-L., Lei, W., Li, J.-W., Wei, K., Luo, C.-J., Han, R., Tian, X.-H., & Jiang, C.-Y. (2016). BigDataBench: An open-source big data benchmark suite. *Chinese Journal of Computers*, 39(1), 196–211.
- Zhang, C., Lu, J., Xu, P., & Chen, Y. (2018). UniBench: A benchmark for multi-model database management systems. In R. Nambiar & M. Poess (Eds.), *Technology conference on performance evaluation and benchmarking for the era of artificial intelligence* (pp. 7–23). Springer International Publishing. doi:10.1007/978-3-030-11404-6_2
- Zong, P., & Lei, Li. (2017). Performance comparison of PostgreSQL and MongoDB dealing with unstructured data. *Computer Engineering and Applications*, 53(7), 104–108. doi:10.3778/j.issn.1002-8331.1508-0203

Feng Ye is a lecturer and member of CCF. His research interests include big data analysis, digital twin, and water conservancy informatization.

Xinjun Sheng is a graduate student at Hohai University and has research interests, including data mining and big data.

Prof. Nadia Nedjah graduated in 1987 in Systems Engineering and Computation and, in 1990, obtained an M.Sc. degree in Systems Engineering and Computation. Both degrees were obtained from the University of Annaba, Algeria. Since 1997, she has held a Ph.D. from the University of Manchester – Institute of Science and Technology, UK. She joined the Department of Electronics Engineering and Telecommunications of the Engineering Faculty of the State University of Rio de Janeiro as an Associate Professor. Between 2009 and 2013, she was the head of the Intelligent Systems research area in the Electronics Engineering Post-graduate program of the State University of Rio de Janeiro, Brazil. (More details can be found on her homepage: <http://www.eng.uerj.br/~nadia/english.html>.)

Jun Sun(1998-), born in Suzhou, Jiangsu province, China; graduate student in Hohai University. Research interests include data mining and big data.

Peng Zhang, Ph.D. in Physical Geography, Nanjing University, Adjunct Professor of Hohai University, Researcher-level Senior Engineer. He is also the vice chairman of the Hydraulic Committee of Jiangsu Hydraulic Society, and the editorial board member of Jiangsu Water Conservancy Magazine. He has presided over several major plans, such as Jiangsu River and Lake Protection Planning and Jiangsu Provincial Water Resources Comprehensive Planning.