UniTest: A Universal Testing Framework for Database Management Systems

Gengyuan Shi, Chaokun Wang(✉), Bingyang Huang, Hao Feng, and Binbin Wang

School of Software, Tsinghua University, Beijing 100084, China
{shigy19,hby17,fh20,wbb18}@mails.thu.edu.cn, chaokun@tsinghua.edu.cn

Abstract. With the continuous development of data collection, network transmission, and data storage, Big Data are now rapidly expanding in all science and engineering domains. Considering the characteristics of Big Data including quick generation, large size, and diverse data models, higher requirements are placed on the functionality and performance of database management systems. Therefore, it is essential for users to choose a stable and reliable database management system. However, finding the best way to evaluate the reliability and stability of database management systems is still a huge challenge, and it is difficult for users to design their own test cases for evaluating these systems.

In order to address this problem, we carefully design a universal testing framework, called UniTest, which can perform effective functional testing and performance testing for different types of database management systems. Extensive testing experiments on multiple types of database management systems show the universality and efficiency of our framework.

Keywords: Database management system · Functional testing · Performance testing · UniTest

1 Introduction

In recent years, the fast development of Big Data technologies including cloud computing, the Internet of Things, and social network analysis [3], has greatly changed the way people live. However, massive data not only bring exceptional opportunities to the development of the technologies but also bring huge challenges to data management. Considering the variety of Big Data, efficient multi-model data management has become a fundamental requirement in real-world scenarios with the perspective that "no one size fits all" [16]. How to manage such heterogeneous data effectively and efficiently is still a big challenge for many industries. In order to utilize the existing DBMSs for providing more efficient multi-model data management, there is an urgent need for techniques to evaluate different types of Database Management Systems (DBMSs) [7,8]. Based on

© Springer Nature Switzerland AG 2021
C. S. Jensen et al. (Eds.): DASFAA 2021, LNCS 12681, pp. 96–104, 2021.
https://doi.org/10.1007/978-3-030-73194-6_7

these database evaluation techniques, users can accurately and effectively evaluate the overall abilities of the DBMSs to ensure that, in real-world scenarios, these systems can provide efficient and stable services.

There are many studies on the DBMSs comparison [1,10,13]. However, there is little work available for universally evaluating multiple types of DBMSs. For most non-relational databases, as well as for different types of data, there is currently no widely used testing platform [7]. Another problem with most of the existing database testing platforms and benchmarks is that the test cases are designed in advance. As a result, the testers cannot create test cases easily according to their concerns, and hence face great difficulties when testing DBMSs that support different data models.

The challenge behind the problems above is the difficulty in conducting the test for different types of DBMSs universally, both the data models and data management technologies of these systems differ. In order to address this problem, this paper designs UniTest, a universal framework for evaluating a variety of DBMSs in terms of both functionality and performance. UniTest allows users to easily design and execute new test cases for different types of DBMSs.

The main contributions of this paper are summarized as follows:

1. We carefully design UniTest, a universal framework for testing multiple types of DBMSs. Besides the universal pre-defined cases for both functional and performance testing, UniTest allows users to design new statements of different types of query languages as test cases. To the best of our knowledge, UniTest is the first testing framework that can be used to test more than five types of DBMSs universally.
2. We implement the UniTest system with effectiveness and high extensibility. We provide an easy-to-use interface for testers to configure the system environment, select test cases, and check the test results in detail. UniTest can also be easily extended by integrating new DBMSs and designing new test cases. The experimental results show that UniTest can effectively test DBMSs that support different data models.

The rest of this paper is organized as follows. We discuss the related work in Sect. 2. In Sect. 3, we present the architecture of UniTest. We report the experimental results in Sect. 4 and finally give our conclusions in Sect. 5.

2 Related Work

The overall performance of one DBMS is related to many factors such as the system architecture, the scale of data, the hardware environment, and so on. To compare different DBMSs, it is necessary to conduct test cases under the same conditions. Typical performance testing methods include benchmark testing, load testing, and so on. Different test methods perform evaluations of the DBMSs from different perspectives as required [2,4,5,9,11,12,19].

2.1 Benchmark Testing

Benchmark testing refers to the test method used to quantify some particular performance metrics of the systems under test. The benchmark testing process mainly consists of three key steps: test data generation, load type selection, and test indicator selection [15]. Various aspects of the systems under test need to be evaluated, such as reliability, time efficiency, resource consumption, cost performance, and so on.

Some benchmark testing tools are designed for specific typical applications. Facebook's LinkBench is mainly used to test the DBMSs in the social network scenarios [2]. Yahoo's YCSB is proposed for testing NoSQL cloud databases [4]. BigDataBench runs tests using different business models [14,18].

However, there are still some drawbacks to these tools. Firstly, the test methods mainly focus on one or a small number of data models and are difficult to test database management systems of multiple types just using one testing tool. Secondly, the lack of a graphical interface is a big problem for many of them, since testers cannot easily set test configurations and get the test results. For example, LinkBench mainly provides test cases including adding, deleting, and changing for graph data but does not support modification of data of other models. YCSB and BigDataBench do not provide direct supports for testing many types (e.g., graph and message queue) of DBMSs.

2.2 Load Testing

The load testing technologies evaluate the processing limit of the DBMSs by simulating the business scenarios. Specifically, these technologies continuously increase the pressure on the system under test until a certain performance measure (such as the response time) of the system exceeds the expected value or a certain resource is exhausted. By means of the load testing, we can understand the performance capacity of the system, discover possible problems in the system, or provide a reference for system performance tuning.

The data loading model performs load testing on the DBMS under test by generating a large amount of data. It is necessary to consider the representativeness, extensiveness, and data distribution of the test dataset when generating the test data, since these characteristics have a great influence on the performance of the application.

Using the load testing tools, the tester can simulate a series of virtual operations in a real business scenario, thereby testing and evaluating various aspects of the system under test. Currently, many load testing tools help testers conduct performance tests through automated testing. JMeter [6] is one of the typical load testing tools.

JMeter is an open-source software application designed to test the server or client architecture and simulate a massive load to test the stability and performance of the DBMSs. Testers can get test results by creating, configuring, and executing test plans.

3 The UniTest

This section proposes UniTest, a universal framework for testing multiple types of DBMSs. The framework architecture can be divided into three parts: the *User Interface*, the *Test Management* module, and the *Test Execution* module. Firstly, we briefly introduce the architecture of UniTest. Then, the three major components are presented in detail in Sect. 3.1, 3.2 and 3.3, respectively.

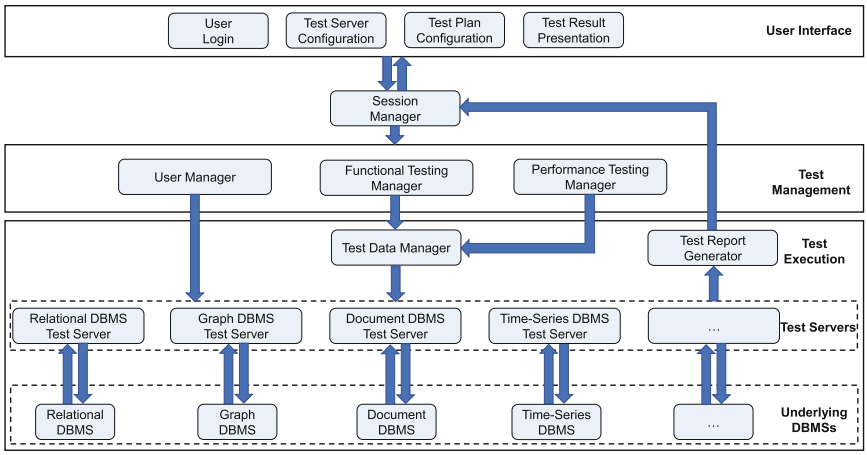

Fig. 1. The architecture of UniTest.

The architecture of UniTest is shown in Fig. 1. The user interface interacts with the tester for test case customization, system configuration, results presentation, and so forth. The test execution module processes the test request sent by the tester, and then forwards it to the corresponding test server and finally obtains the test result from the test server for displaying in the user interface. There is one test server for each of the data models, which is responsible for receiving and executing the specific test cases on the DBMS under test. The test management module receives the test results from the test execution module and returns them to the user interface to display.

With the support of this framework, we also have an evaluation method for testing different types of DBMSs. The tester can log in through the user interface, generate a test plan by selecting the target DBMS and the test cases, and pass the test plan to the test management module. The test server in the test execution module connects to the server where the DBMS under test, namely the target DBMS, is located according to the test plan, and performs corresponding tests. During testing, the test execution module prepares test data by loading the corresponding test dataset or generating test data. The test results are automatically collected and returned to the user interface along with the generated test report.

3.1 User Interface

The user interface includes a user login pane, a test server configuration pane, a test plan configuration pane, and a test result presentation pane. Through the user interface, the tester can complete operations such as logging in, configuring and conducting tests, and viewing test results.

In the test server configuration pane, the tester can configure the address, the user name, and the password for each DBMS test server.

In the test plan configuration pane, the tester can select test cases for each DBMS from the list of test cases given by UniTest, combine them into a test plan, and execute the test. UniTest automatically generates the test plan and passes the plan to the test management module.

In the test result display pane, the log information and the test results returned from the test server are displayed. The tester can download the automatically generated test report.

3.2 Test Management Module

The test management module, consisting of a user manager, a functional testing manager, and a performance testing manager, is responsible for performing all test plans in the framework.

In the user manager, the user permissions of the testers of UniTest can be managed to ensure the security of the DBMSs under test. Roles of the users are divided into two categories: administrator users and ordinary users. The administrator user can add users or delete existing users, and can modify the passwords of other users; ordinary users only have the right to log in and modify their passwords.

The test management module refines the received test plan. There may be precondition or inclusion relationships between a series of given test cases, i.e., test case A must be executed at first, or test case A should contain test case B. UniTest automatically checks these relationships and forms an integrated test plan.

In the functional testing manager, different test cases for different data types are preset for the tester to select. One or more test cases can be selected to be freely combined to form a test plan.

The performance testing manager includes performance test cases for different DBMSs, including data migration efficiency (import/export) and query execution efficiency.

3.3 Test Execution Module

The test execution module receives the test plan from the test management module, dispatches the test cases to the corresponding test servers, and returns the test results and the test report. The test execution module includes a test data manager, a test report generator, and a test server for each type of the DBMSs.

The test data manager prepares test datasets for the target DBMS when the test is performed. For example, this component generates a testing dataset automatically before executing the data migration test which needs a large amount of data to be imported.

The test report generator monitors the test process and collects the test results. It automatically generates a test report document of the current test according to information on the process and results of the test.

For each type of DBMSs, the test server controls the corresponding DBMS server and executes the user-specified test cases. After the test plan is forwarded by the test management module to the specific test server, the test server conducts test cases according to the test plan.

For performance testing, the test execution module conducts the user-specified test plan. The test server controls the target DBMSs to test the performance of data migration and query execution, and returns the results including the response time and data migration speed.

4 Experiments

In this section, we test some typical DBMSs under Ubuntu 16.04 with a 10-core Intel Xeon E5-2630 (2.20 GHz) and 320 GB main memory. This process consists of two parts: functional testing and performance testing, which are the core of our framework.

We choose representative DBMSs of different types in the performance testing experiment to check the performance of the data migration and query execution for different models of data. Specifically, we test the performance of MySQL for relational data management, Neo4j for graph data management, InfluxDB for time series data management, CouchDB for document data management, Redis for key-value data management, MySQL Blob for binary big object data management, and Kafka for message queue data management.

4.1 Functional Testing

We have implemented the testing system according to the UniTest architecture. The pre-defined test cases in UniTest cover seven categories of functionalities, such as the separation of service instances, the authentication of database users, and the ability to trace data sources.

Test cases for functionality testing are conducted for representative DBMSs of different types. For the space limitation, the results of functional testing of all the DBMSs are not presented.

4.2 Performance Testing

We also conduct performance testing for many types of DBMSs. Here, two typical categories of test cases are considered. In the data migration efficiency test, we

test the speed of data importing and data exporting. In the query execution efficiency test, we test the speed of query execution under different conditions.

The results of performance testing on Neo4j is shown in Table 1. The graph datasets are generated by FastSGG [17]. The data migration efficiency is tested varying the sizes of nodes and edges, while the query execution efficiency test cases are executed varying the sizes of the graph. For the limit of space, test results of other types of representative DBMSs are not displayed in this paper.

Table 1. Results of performance testing for Neo4j.

Description		Average time of 9 cases
Data migration	Importing node data (56 KB with 1893 nodes)	9333 KB/s, 6 ms
	Importing edge data (162 KB with 4641 edges)	123 KB/s, 1.32 s
	Backuping 1893 nodes with attributes	519 nodes/s
	Restoring 1893 nodes with attributes	2146 nodes/s
Query execution	$\|V\| = 100$, $\|E\| = 1000$	35.2 ms
	$\|V\| = 1000$, $\|E\| = 10000$	472.5 ms
	$\|V\| = 10000$, $\|E\| = 100000$	12802.6 ms

In summary, UniTest can test different types of database management systems universally, and effectively supports testing all of these systems in a specific evaluation method, including functional and performance testing.

5 Conclusion

This paper proposes a universal testing framework called UniTest with a specific evaluation method for conducting evaluation on various types of database management systems. UniTest provides a rich set of test cases with an easy-to-use interface for different types of DBMSs. We carry out extensive experiments of functional and performance testing on some typical DBMSs. The experimental results show that UniTest provides a universal evaluation environment for different types of DBMSs and plays an important role in the application of database products in real-world scenarios.

Acknowledgments. This work is supported in part by the Intelligent Manufacturing Comprehensive Standardization and New Pattern Application Project of MIIT (Experimental validation of key technical standards for trusted services in industrial Internet), and the National Natural Science Foundation of China (No. 61872207).

References

1. Abramova, V., Bernardino, J.: NoSQL databases: MongoDB vs cassandra. In: Proceedings of the International C* Conference on Computer Science and Software Engineering, pp. 14–22. ACM (2013)
2. Armstrong, T.G., Ponnekanti, V., Borthakur, D., Callaghan, M.: LinkBench: a database benchmark based on the Facebook social graph. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pp. 1185–1196. ACM (2013)
3. Cai, L., Zhu, Y.: The challenges of data quality and data quality assessment in the big data era. Data Sci. J. **14**, 2 (2015)
4. Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R.: Benchmarking cloud serving systems with YCSB. In: Proceedings of the 1st ACM Symposium on Cloud Computing, pp. 143–154. ACM (2010)
5. Difallah, D.E., Pavlo, A., Curino, C., Cudre-Mauroux, P.: OLTP-bench: an extensible testbed for benchmarking relational databases. Proc. VLDB Endow. **7**(4), 277–288 (2013)
6. Halili, E.H.: Apache JMeter: A Practical Beginner's Guide to Automated Testing and Performance Measurement for Your Websites. Packt Publishing Ltd., Olton (2008)
7. Han, R., John, L.K., Zhan, J.: Benchmarking big data systems: a review. IEEE Trans. Serv. Comput. **11**(3), 580–597 (2017)
8. Han, R., Lu, X., Xu, J.: On big data benchmarking. In: Zhan, J., Han, R., Weng, C. (eds.) BPOE 2014. LNCS, vol. 8807, pp. 3–18. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13021-7_1
9. Iosup, A., et al.: LDBC Graphalytics: a benchmark for large-scale graph analysis on parallel and distributed platforms. Proc. VLDB Endow. **9**(13), 1317–1328 (2016)
10. Jouili, S., Vansteenberghe, V.: An empirical comparison of graph databases. In: 2013 International Conference on Social Computing, pp. 708–715. IEEE (2013)
11. Kasture, H., Sanchez, D.: Tailbench: a benchmark suite and evaluation methodology for latency-critical applications. In: 2016 IEEE International Symposium on Workload Characterization (IISWC), pp. 1–10. IEEE (2016)
12. Li, M., Tan, J., Wang, Y., Zhang, L., Salapura, V.: SparkBench: a comprehensive benchmarking suite for in memory data analytic platform spark. In: Proceedings of the 12th ACM International Conference on Computing Frontiers, p. 53. ACM (2015)
13. Li, Y., Manoharan, S.: A performance comparison of SQL and NoSQL databases. In: 2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM), pp. 15–19. IEEE (2013)
14. Liang, F., Feng, C., Lu, X., Xu, Z.: Performance benefits of DataMPI: a case study with BigDataBench. In: Zhan, J., Han, R., Weng, C. (eds.) BPOE 2014. LNCS, vol. 8807, pp. 111–123. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13021-7_9
15. Ming, Z., et al.: BDGS: a scalable big data generator suite in big data benchmarking. In: Rabl, T., Jacobsen, H.-A., Raghunath, N., Poess, M., Bhandarkar, M., Baru, C. (eds.) WBDB 2013. LNCS, vol. 8585, pp. 138–154. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10596-3_11
16. Stonebraker, M., Çetintemel, U.: "one size fits all" an idea whose time has come and gone. In: Making Databases Work: The Pragmatic Wisdom of Michael Stonebraker, pp. 441–462 (2018)

17. Wang, C., Wang, B., Huang, B., Song, S., Li, Z.: FastSGG: efficient social graph generation using a degree distribution generation model. In: Proceedings of the IEEE 37th International Conference on Data Engineering (ICDE), Chania, Greece (2021)
18. Wang, L., et al.: BigDataBench: a big data benchmark suite from internet services. In: 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), pp. 488–499. IEEE (2014)
19. Wang, M., Wang, C., Yu, J.X., Zhang, J.: Community detection in social networks: an in-depth benchmarking study with a procedure-oriented framework. Proc. VLDB Endow. **8**(10), 998–1009 (2015)