IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 23, 2019, accepted February 12, 2019, date of publication February 22, 2019, date of current version March 12, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2901115

An Implementation of Scalable High Throughput Data Platform for Logging Semiconductor Testing Results

CHEN-KUN TSUNG[1], HSIANG-YI HSIEH[2], AND CHAO-TUNG YANG[2], (Member, IEEE)
[1]Department of Computer Science and Information Engineering, National Chin-Yi University of Technology, Taichung 41170, Taiwan
[2]Department of Computer Science, Tunghai University, Taichung 40704, Taiwan

Corresponding author: Chao-Tung Yang (ctyang@thu.edu.tw)

This work was supported by the Ministry of Science and Technology (MOST), Taiwan, under Grant 107-2218-E-029-003 and Grant 107-2221-E-029-008.

ABSTRACT Unlike graded data of common semiconductor test results storing in relational databases, log data in the standard test data format (STDF) contain millions of test data entries. In a semiconductor packaging and testing factory, a semiconductor wafer or integrated circuit tests generate thousands of STDF files each day; therefore, how to store these massive databases is a crucial topic. Different products correspond to different test items and STDF content; if a relational database is used to store all forms of data, the practical operation becomes challenging. This paper used a NoSQL document-oriented database collocated with a Docker container to build a system, named the scalable STDF data (SSD) framework, for storing semiconductor test data. According to semiconductor test operations, the SSD framework first converts STDF files into an open standard format for data transmission and subsequently transfers them to the database. The use of NoSQL databases allows for flexibility of specifications of STDF content, and a Docker container exhibits features such as rapid deployment and high scalability. The SSD framework meets the requirements of semiconductor testing for throughput, latency, and parallel experimental projects; possesses excellent execution efficiency; and provides flexible data storage services in a semiconductor testing environment where processing a large quantity of data is required. From our simulation results, the major performance of the proposed system depends on the hardware properties. The higher hardware distribution degree provides better performance. Docker container provides more connections and the scalability of storage, but higher software distribution contributes limited performance enhancement.

INDEX TERMS Flexible data storage, scalable STDF data, semiconductor testing, standard test data format.

I. INTRODUCTION

The manufacturing process of an integrated circuit starts from a circuit layout. Through hundreds of repeated complex procedures (e.g., exposure, development, ion implantation, and etching), dozens of layers of circuits are accurately formed on a thin and round disk-like wafer before it undergoes final testing and packaging. The semiconductor packaging test is the final stage of semiconductor manufacturing. Test items are enumerated for wafer or chip characteristics and tested individually. During the test, the results for each item are recorded in a file of a specific format. The score directly affects the value of a wafer.

The associate editor coordinating the review of this manuscript and approving it for publication was Chin-Feng Lai.

Recently, advancement in semiconductor manufacturing has substantially enhanced the functions of integrated circuits; therefore, the unit price of each wafer is escalating, and the number of test items to verify the functions of chips is increasing. Hundreds of items are tested to verify wafer operation; after each test, a test data STDF file is generated. STDF files exhibit a particular structure. Appropriately preserving STDF files is helpful for engineers to conduct subsequent analysis, such as tracing problems encountered in the manufacturing process.

STDF files are simple and flexible files that can store data logs for millions of test items. The storage environment must be capable of storing vast quantities of test data at all times [13]; in such storage settings, users should be able to quickly find the required data in a massive database.

Conventional relational databases can improve data throughput using a clustered architecture when processing large data; however, they are ineffective for STDF files because of the diversity in wafer test data.

Based on the Docker program and MongoDB, this study developed the SSD framework to elastically store test data for wafer fabrication. Used mainframes and equipment, which have been replaced during renovation but still have computing and storage capabilities, can be used as the primary hardware to reduce system construction costs and turn passive assets into active assets. MongoDB is an open-source NoSQL database; it features high performance, high reliability, and automatic expansion that satisfies the short response time, long operating time, and scale-out in data acquisition required by the semiconductor industry.

The test cases used in the experiment were divided into five categories to highlight the context of actions such as reading, updating, and adding. Data throughput, latency, and response time are discussed separately. The performance of the SSD framework introduced in this paper exceeded the data storage requirements of a test package. Notably, in the large-scale writing of data, the SSD framework was able to respond quicker than the time estimated according to the linear growth rate of data.

In this paper, we aim at saving semiconductor testing data log, and the contributions are listed as follows:

- Implement a high throughput data log platform for saving semiconductor testing data.
- Build up the scale-out data-save service to increase the storage volume with small effort.
- Provide a solution for the reactivation of useful but not main force equipment.

II. RELATED WORKS

A. NoSQL DATABASE

STDF files generated by wafer tests exhibit a distinct structure. Although the organization of the files has a particular specification, the diversity in the content of STDF files induced by wafer characteristics hinders data normalization [17]. To support such file-oriented data, developers have begun to provide solutions for most conventional relational databases [18], [19]. NoSQL developed using the consistency, availability, and partition tolerance (CAP) theorem can satisfy the characteristics of STDF files and provide optimized data access [1], [6]. Barbierato *et al.* (year) found that using NoSQL to store or query data was superior to using relational databases [5]. Current NoSQL databases are divided into the following four categories:

1) Key-Value databases: These break the schema of relational databases; in the database, each datum is independent, and the system has high horizontal scalability. Typical kits include BigTable, Hadoop HBase, and Apache Cassandra.
2) In-memory databases: To increase the reading speed, these database storage operations rely only on memory. These are suitable for use on cached pages, reducing the number of readings of a hard disk. Prominent in-memory databases are Redis, Tuple Space, and Velocity.
3) Document databases: These are used to store loosely structured or unstructured data; in common unstructured data, such as HTML pages, each tag paragraph can contain text, images, or music. Common examples are CouchDB, MongoDB, and Riak.
4) Graph databases: These use graph structures to store data in the forms of nodes, edges, and properties. Common graph databases are Neo4j and InfoGrid.

In a series of test reports, The Software and Information Industry Association noted that when accessing documents such as STDF files, MongoDB has superior scalability and execution performance to other NoSQL databases. Practical implementations, such as accessing electronic medical records or processing turbine operation data at wind power plants, require working with data exhibiting high heterogeneity, uncertainty, and scalability; in this regard, MongoDB provides reliable services.

Except for in particular circumstances, semiconductor production lines do not stop. Therefore, hardware devices responsible for STDF data storage must have a low failure rate. Current solutions for hardware failure are as follows:

- Data cutting: The effect of failure can be minimized, and reading and writing of data can be distributed to different nodes to increase speed. In events of data loss, partial data can still be retained.
- Multiple copies: Most NoSQL implementations are based on a hot backup of data to ensure high data reliability.
- Dynamic scaling: In response to a high data growth rate, NoSQL can expand capacity without shutting down the system.
- Query support: Data stored in document databases are in the form of binary large objects (BLOBs). Using correlation between keywords and a list of attributes, the operation of the schema key is no longer restricted.

To handle huge amount of data and a board range of data types, NoSQL database is the popular solution for data scientists. NoSQL database considers schemaless to meet the requirements of elastic applications. Rational databases apply well defined structure to specific the data access rule, so programmers access the rational database via a common interface, e.g. structure query language (SQL). The property of well-defined structure increases the data access performance of rational database, but it restricts the model of acceptable data. Therefore, the schemaless is the major reason for NoSQL considered in a wide range of applications [30]. However, NoSQL database is not appropriate for all kinds of applications. We study some implementation issues that are arranged as follows to discuss the NoSQL database.

- Schemaless: NoSQL does not consider data structure, so it can be applied to wide a range of applications. Since the NoSQL is schemaless, programmers have to develop the interfaces for database access services.

Thus, the efficiency of database access depends on the service interface [29].

- Application latency: writing data into NoSQL database is more efficient than that in rational database [31]. Given a specific query order, NoSQL may not provide better performance than rational database. The major reason is that the runtime overhead results in the latency especially web applications [29].
- ACID properties (Atomicity, Consistency, Isolation and Durability): rational databases provide ACID but NoSQL databases consider Basically Available, Soft State, Eventual consistency (BASE) properties [33]. BASE is not as reliable as ACID. So some special applications require ACID rather than BASE e.g. banking industries.

LourenÃğo *et al.* [32] study current popular NoSQL databases, and figure out that the MongoDB provides reliable data access services. Even if we have alternative choices for better data read and write performance, the performance of MongoDB can be improved via data shard.

B. DOCKER

For system operations, a means to rapidly reconstruct the execution environment in the event of a hardware failure is crucial to prevent catastrophic consequences. In the past, high reliability was achieved through hardware virtualization technologies and virtual machines. However, the slow response of the setting caused the deployment of virtual machines to be time consuming, and the copying of physical files was the primary time cost. To solve this problem, Docker provides Linux containers, which use Linux to package the core libraries and codes to create the execution environment required by users. The difference from conventional virtualizations is that Docker implements virtualization at the operating system (OS) level and uses the native OS directly; the conventional approach is implemented at the hardware level. Figure 2.3 shows the operation of conventional virtualization. Conventionally, a guest OS must be built, which is time- and hardware-intensive; however, Docker does not need this. Therefore, the performance of Docker is as simple and convenient as a fast and lightweight virtual machine. The execution efficiency of Docker is high; the startup of the container completes within seconds. Docker does not consume extra system resources, except when executing built-in applications; therefore, the usage rate of the system is high, and it is helpful for companies that provide semiconductor testing.

Docker technologies share Linux kernel to provide multiple containers. However, the security issue is a critical consideration. Since containers use the same kernel, it implies the container user may access the information of other users. To keep the security, we have three major solutions: isolation, host hardening, network security, and jail [20]–[22]. In these solutions, isolation is a common solution to enhance the container security [21], [23], [24]. Managers can apply the identifiers to realize the isolation,

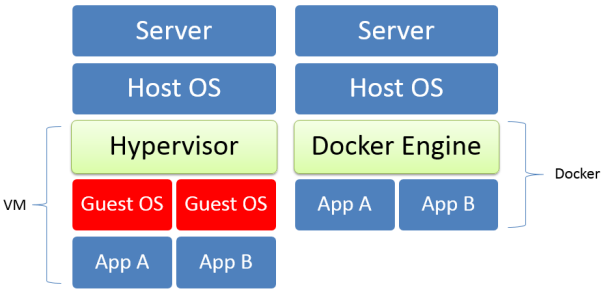

FIGURE 1. The operation of conventional virtualization.

e.g. containers' namespaces, groups and mandatory access control [24]. Moreover, Bacis *et al.* [24] propose a docker file configuration to Security-Enhanced Linux policy, so that the container security can be improved. According to the study of Dua *et al.*, isolation provides higher security [22]. Although isolation suffers a short board effect, Lin et al. propose a defense mechanism for the identified privilege escalation attacks [22]. For network security, to enhance the container security provides the access control in terms of the communication level. Manu *et al.* [25] design the balanced multilateral security prototype. The proposed architecture applied OSI/TCP/IP model to the cloud service structure. In the proposed architecture, physical layer, data link layer and network layer are rearranged in a service, i.e. infrastructure as a service. So that each communication can be controlled, and the security can be managed.

III. PROBLEM DEFINITION

Considering a semiconductor manufacturing, the test process will generate a lot of manufacturing information, including manufacturing parameters, equipment status, workpiece feedback, etc. The gateway will collect the manufacturing information and output the data in the STDF format. The STDF data will be sent to a data center for further process. In this paper, we focus on developing and implementing the data center to gather the testing record. Production managers will follow the testing record to determine the production quality is passed or not. The data center should provide following abilities.

1) High throughput: higher throughput indicates that more data can be received. Since the test processes will generate a lot of data, the semiconductor test data center must have the ability to receive grate amount data in a specific time period.
2) Low update latency: update is an important process for semiconductor test. The process should change the status according to the test result. The data center must provide low update latency so that the further process can be launched as early as possible.
3) Low insertion time: the major work of processing the test data is the data insertion. The data center must have the ability to receive the test data and write them into the storage. Therefore, the performance of the data insertion is a critical issue of the proposed data center.

IV. THE PROPOSED SOLUTION - SCALABLE SEMICONDUCTOR DATA LOGGER

Items are tested individually, and the tests generate STDF files in the direct-mounted NAS space. The SSD framework provides a resident program to continuously monitor the space. When an STDF file is generated, the AP uses STDF4j and Google GSON to convert the STDF file to JSON format and then writes the JSON data to MongoDB through a Java MongoDB driver. The test engineers query the test results through the SSD framework's graphical user interface (GUI). The usage scenario is shown in Figure 2.

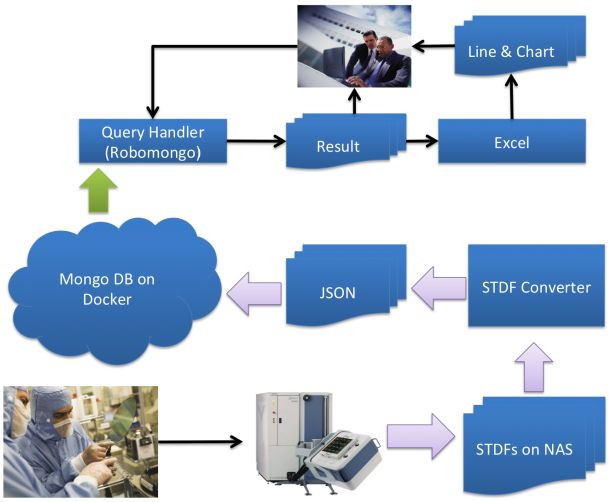

FIGURE 2. The scenario applied in the semiconductor test process.

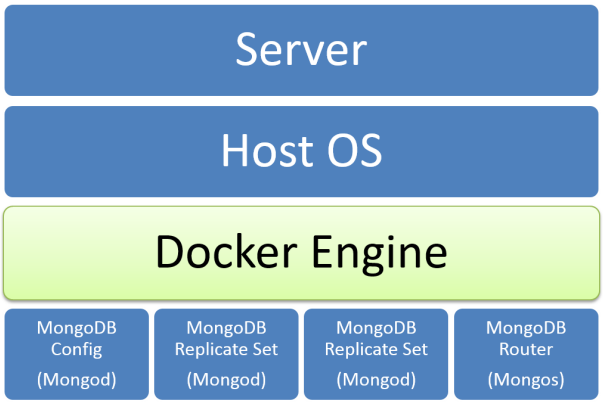

FIGURE 3. The structure of the proposed solution.

Figure 3 shows the architecture of the solutions proposed in this paper. An STDF file is converted to JSON data, and the data are transferred to Mongos of MongoDB through the MongoDB client. Subsequently, the internal algorithm of MongoDB writes the data to MongoDB for storage, and a MongoDB configuration server records the location of the data. The software architecture is shown in Figure 4.

Figure 5 illustrates the operation of a semiconductor test. Semiconductor tests can usually be divided into two types: chip probing and final testing.

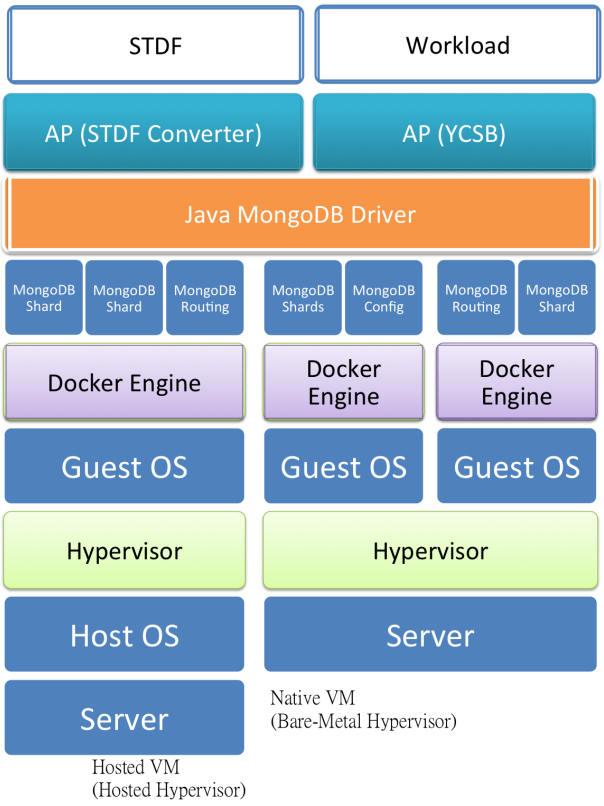

FIGURE 4. The software structure of the proposed solution.

- Chip probing: The crystal grains on the wafer are examined for defects by a test probe according to the electrical specifications of the design. The test is conducted on the wafer level; therefore, chip probing is also called wafer sorting. If the yield rate is too low, it means that problems occurred during the wafer manufacturing process. This information must be quickly reflected in adjustments to the production process.
- Final testing: This test, also known as a product test, is performed using an electrical needle on packaged products. This test ensures that products meet the design specifications after packaging.

Understanding the STDF file organization structure is necessary when converting files to or from STDF format. An STDF file is a binary file composed of different data. Each data record contains a header and the data itself. The header includes the data length, type, and subtype. In the header, the data type and subtype are integers appearing in pairs. The meaning of this data record can be found using the combination of REC_TYPE + REC_SUB.

The primary data record in an STDF file is the master information record, which is used to identify each STDF file. This record stores information about the tests, including lot information, start and end times, machine used, program used, production equipment code, and operators.

The majority of STDF records is stored as a Parametric Test Record 3.6 (PTR). Tens of thousands of wafer data entries

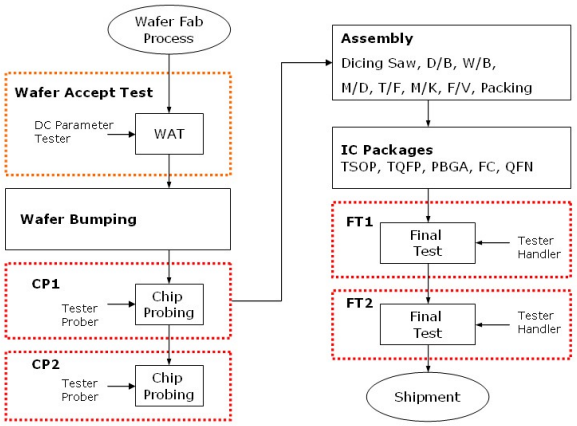

FIGURE 5. The operations of a semiconductor test.

are stored in STDF files. Each wafer has hundreds of PTRs; therefore, a file contains millions of PTRs. Preset values for the test (e.g., limits, units, and value ranges) are stored when the PTR first appears in an STDF file.

The transferred JSON data are shown in Figure 6. This experiment first transferred the PTRs, which were the majority in the STDF files, to MongoDB. Therefore, the JSON schema of the test data only stored the MIR/PTR records. The number of PTRs was n, and n depends on the quantity of test data in an STDF file.

```
{ "DEVICE_NAME":"DEVICE01" , "PROGRAM_NAME":"PGM01" ,
{ "ITEM_NAME":"TEST1" , "TESTER_ID":"TXX01" ,
"PROBER_HANDER":"P1234" , "P/C_ID":"AT001" ,
"START_TIME":["\/Date(1330444800000)\/"] ,
"END_TIME":["\/Date(1330444800000)\/"] ,
"PTR": [
{"x":"1" , "Y":"3" , "SITE":"1" , "BIN_PF":"P" , "ITEM_PF":"F" , "VALUES":"0.22"} ,
{"x":"2" , "Y":"3" , "SITE":"1" , "BIN_PF":"P" , "ITEM_PF":"F" , "VALUES":"0.23"} ,
...]}
```

FIGURE 6. A sample of transferred JSON data.

Security is a key issue for sharing Linux kernel. There are some techniques that we can use to increase the container security, e.g. isolation or communication control [22], [24], [25]. In this paper, we focus on providing the scalable log data saving service for semiconductor testing. Our major mission is to scale out the storage volume and the connection ability of saving testing data rather than providing multiple services in different containers. So that the infrastructure requirement of saving testing data is simpler than that required by the multi-purpose service.

V. SIMULATIONS

The experimental environment is shown in Figure 7. During the experiment, the user executed the command Shell Script, and specified through parameters the workload and interface to be used. The system subsequently wrote the data into the test database and began testing. After each test, the system automatically cleared the remaining data and re-entered the test data to ensure the fairness of each test.

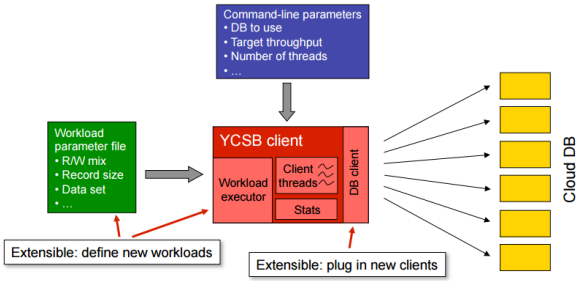

FIGURE 7. The experimental environment for semiconductor test.

TABLE 1. The test benchmark including read, update insertion operations.

	Read	Update	Insertion	Note
workload A	0%	0%	100%	Full Insertion
workload B	10%	90%	0%	Most Update
workload C	65%	10%	25%	Major Read
workload D	90%	10%	0%	Most Read
workload E	100%	0%	0%	Full Read

We consider five test scenarios based on the combinations of data read, update, and insertion. The test scenarios are listed in Table 1. According to the operations in each scenario, we have five workloads including full insertion, most update, major read, most read, and full read.

This study used the Yahoo! Cloud Serving Benchmark (YCSB) as a test data source. The YCSB is a benchmark framework designed by Yahoo! for a variety of databases to compare performance between different databases. Although the data characteristics presented in practical applications are different, the design of the benchmark is based on the following three:

- A large amount of data traffic: Data traffic is increasing with the speed of Internet service and the transformation of service applications.
- Flexibility in increasing nodes: This allows administrators to effortlessly increase the number of serving nodes to provide more storage.
- Fault tolerance: Software and hardware troubleshooting is undertaken for all possible hosts or networks.

The YCSB uses the following four levels to determine the benchmark:

- Performance: Use gradually increased loading to detect if an exception has occurred or if the performance is as expected.
- Scaling: Increase the number of service nodes, but also increase loadings in the same proportions to compare the extent of changes.
- Availability: Test the burden to performance when hardware or network problems occur.
- Replication: Individually adjust the number of service nodes provided by the database.

In this paper, our major goal is to provide a scalable data storage service for saving semiconductor testing data. Since the server replacement rate in the semiconductor industry is higher than other industries, some servers still have the ability

to provide storage service even if the computation power is not high enough for some processes. So that we consider three hardware levels:

- **Low level environment** means that the storage unit has low computing power but the storage service is still enough. We use a notebook to simulate low level environment.
- **Single server** means that the server provides high enough computing power and we would like to insert data to single server, and the server may provide some computation in the further process.
- **Two servers** means that the system can accept heavy data insertion process, so that we may use distributed structure to increase the storage performance.

To scale out the storage, we have two implementations: 1) hardware distribution strategy which means increasing the number of servers, and 2) software distribution strategy which means increasing the number of MongoDB shards. Therefore, we will discuss the performance of the proposed service in terms of the hardware and software distribution strategies.

A. THROUGHPUT

This study first tested the execution performance of versions 2.6.9 and 3.0.2 of MongoDB to determine the settings for subsequent experiments. We consider three environments in this experiment: low level environment, single server, and two servers. The simulation results are shown in Figure 8, 9, and 10 respectively.

The result of low level computing unit execution is shown in Figure 8. The horizontal axis is the test case; the vertical axis is throughput. In each data set, version 3.0.3 outperformed version 2.6.9; in the execution environment of Native VM, consistent results were also obtained (Figure 9). On two servers of Native VM in the same hardware and software environment, this study conducted comparison tests on the throughput of eight shard servers. The result of the execution is shown in Figure 10. The performance of version 3.0.2 was superior to that of 2.6.9. Therefore, the throughput performance of version 3.0.2 was superior to 2.6.9 for both single-machine and distributed execution.

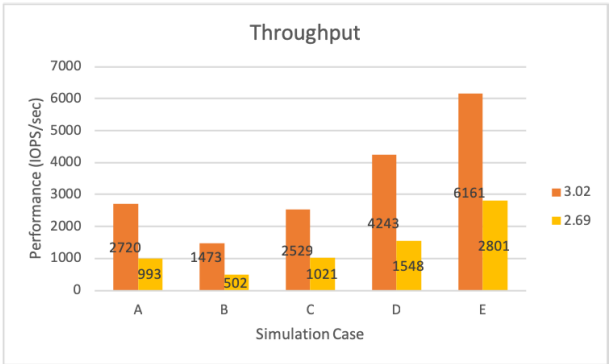

FIGURE 8. The throughput evaluation of a single server.

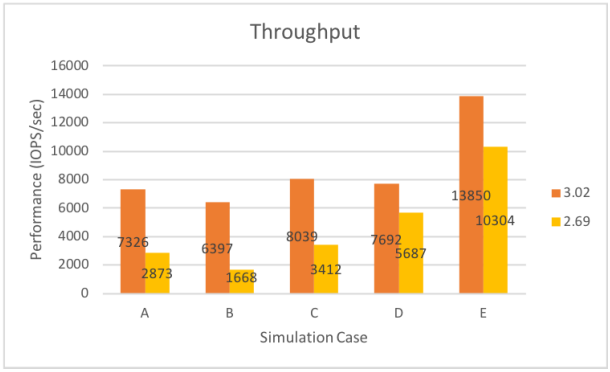

FIGURE 9. The throughput evaluation of Native VM.

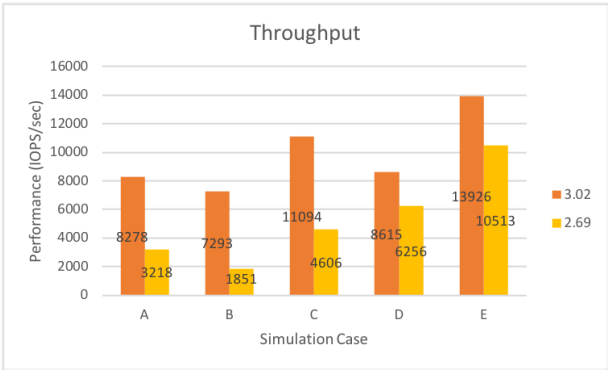

FIGURE 10. The throughput evaluation of Native VM with two servers.

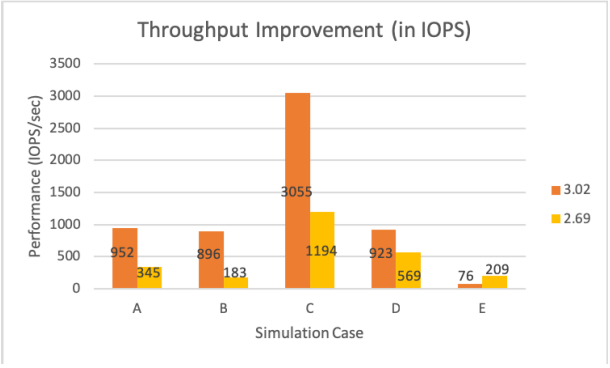

FIGURE 11. The throughput improvement comparing between single server and two servers.

The marginal performance between single server and two servers is illustrated in Figure 11 while the improvement in percentage is shown in Figure 12. The performance of insertion is similar to that of update but more powerful than that of pure read process. However, in the full read case (WorkloadE), version 2.69 has better performance the version 3.02. Therefore, considering hybrid database engine may be a performance optimization issue. For the compression in percentage, we get similar results than that of Figure 11. The result shows that increasing the hardware distribution degree provides major improvements in WorkloadC (major read case), but the improvement is not only about 13% in other cases.

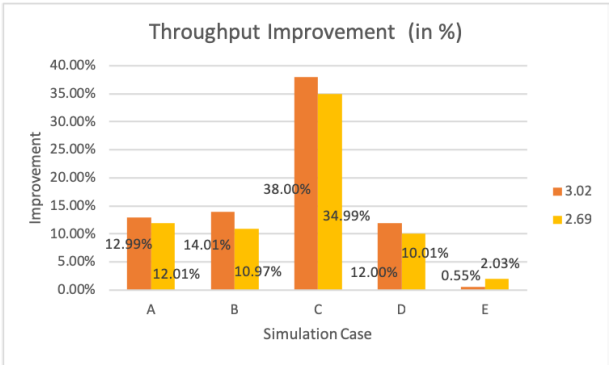

FIGURE 12. The throughput improvement percentage comparing between single server and two servers.

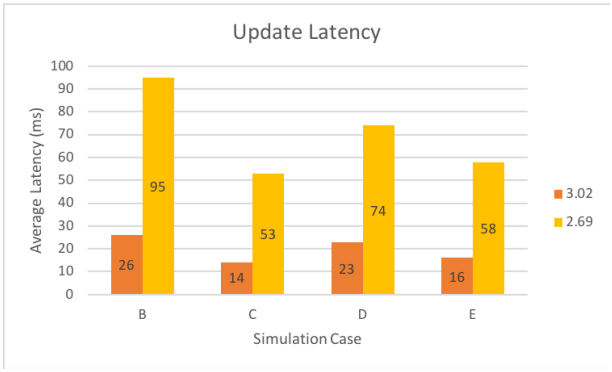

FIGURE 13. The update latency of a single server.

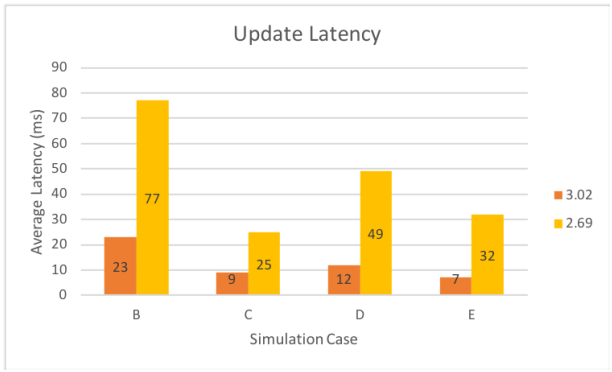

FIGURE 14. The update latency of two servers.

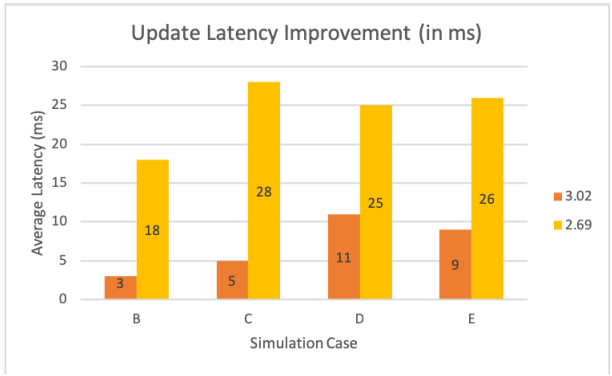

FIGURE 15. The update latency improvement between single server and two servers.

From above results, the system throughput depends on the computing power, but not on the hardware distribution degree. From Figure 12, the marginal throughput obtained by increasing the hardware distribution degree comes from WorkloadC, which includes hybrid processes. In other cases, which indicate a single process, the hardware distribution degree provides limited improvement of throughput. Especially in WorkloadE with pure read process, MongoDB with version 2.69 provides better performance than that of version 3.02.

B. UPDATE LATENCY

The execution efficiency of updating the test data is an essential reference indicator. The architecture with high execution efficiency provides low latency. This experiment compared the latency of versions 2.6.9 and 3.0.2 to understand the response speed of the solutions described in this paper. The update performance depends on the computational power of the server. We consider the simulation platforms with a single server and two servers. Since the low level environment provides less computation power than single server and two servers, we do not consider the low level environment in this simulation. Moreover, case WorkloadA includes pure insertion processes, so we only consider the cases from WorkloadB to WorkloadE.

The experimental results of a single server are shown in Figure 13. The horizontal axis is the test case, and the vertical axis is latency. The unit is ms. Version 3.0.2 had lower latency. Overall, the latency of version 3.0.2 was approximately 80% of the update latency of version 2.6.9. Particularly, in the experimental scenario of WorkloadC, MongoDB 3.0.2 substantially improved throughput and provided users favorable feedback.

In the execution environment of two nodes, the overall latency was reduced by clustered operations that distributed data traffic (Figure 14). Although lower latency was obtained in all test cases, the ratio was consistent with the results obtained with a single server.

To obtain the overall result, we compute the latency improvement in time and in the percentage, and the results are shown in figure 15 and 16 respectively. From figure 16, it is clear that the hardware distribution degree helps to improve the update latency in the cases with the read process including from WorkloadC to WorkloadE. Moreover, we obtain the improvement sequence is WorkloadE > WorkloadD > WorkloadC. It means the hardware distribution degree contributes the latency improvement for the read process.

C. THROUGHPUT IN VARIOUS SHARD

In this experiment, we focus on discussing the performance of the degree of software distribution. The degree of software distribution means the number of MongoDB shards. The degree of software distribution is directly related to throughput. Theoretically, the higher degree of software distribution is, the higher the throughput is. This experiment tested the

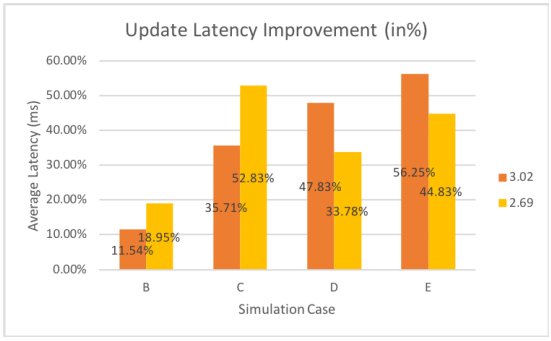

FIGURE 16. The update latency improvement between single server and two servers in percentage.

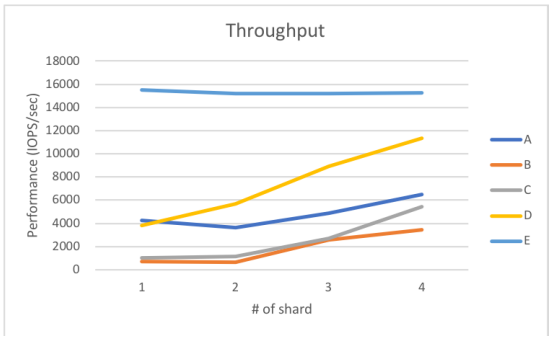

FIGURE 18. The throughput evaluations with various number of shards capture from the single server environment.

throughput performance of different numbers of shard in different scenarios.

Figure 17 shows the experimental results of four shard servers in different test scenarios captured in the low level environment. Figure 18 is captured from the single server environment. The horizontal axis is the degree of software distribution; the vertical axis is throughput. This experiment added a set of test environments with two servers and four shard servers to compare the effects of the number of shard servers on throughput.

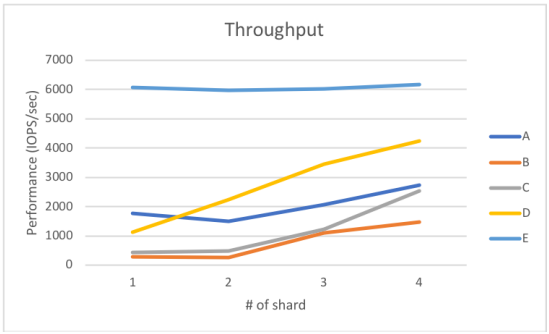

FIGURE 17. The throughput evaluations with various number of shards in low level environment.

When reading data was the primary work content, the number of shard servers was positively correlated to throughput as presented by WorkloadD; however, if only considering reading behavior, the performances of different numbers of shard servers were not notably different, as WorkloadE presented. For other updating or inserting actions, the throughput performance slightly improved with the number of shard servers.

To evaluate the benefit of the software distribution degree, we calculate the improvement of single server environment compared to the low level environment. Figure 19 is the enhancement of throughput while File 20 is the benefit in percentage. Increasing software distribution degree provides stable benefits, but the benefit is not proportional to the degree of software distribution. It indicates that increasing the hardware distribution generates higher benefit than increasing the software distribution.

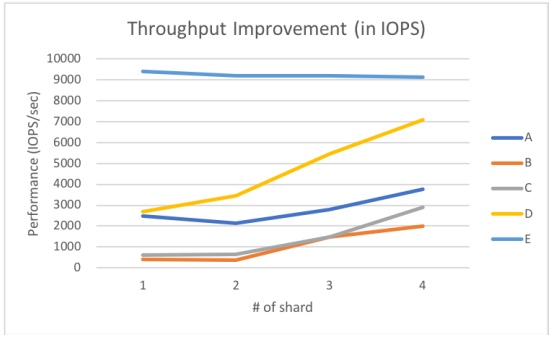

FIGURE 19. The evaluation of the throughput gap between that in Fig. 17 and Fig. 18.

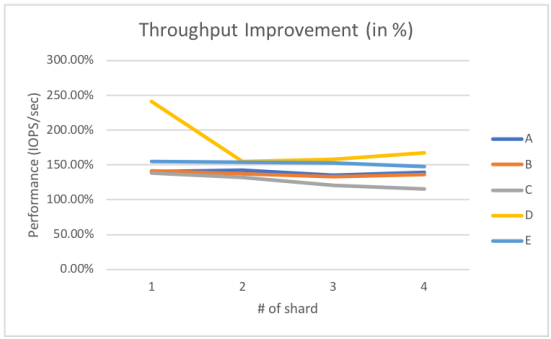

FIGURE 20. The evaluation of the throughput gap between that in Fig. 17 and Fig. 18 in percentage.

To verify our observation, we consider a comparison in 4 MongoDB shards and the simulation results are captured in single server and two servers. The result is shown in Figure 21 and Table 2. The updating and insertion performance of two servers was superior to that of a single server. However, for the test scenario emphasizing the reading of data, the throughput of two servers was not substantially different from that of a single server. Therefore, parallel operations benefit the updating and insertion of STDF files. However, intensive data reading prevented the subsequent data analysis from achieving high operational performance.

On the other hand, comparing the results in Figure 21 and Figure 21 confirms our observation in the above. Increasing the hardware distribution degree provides higher throughput performance than increasing the software distribution degree.

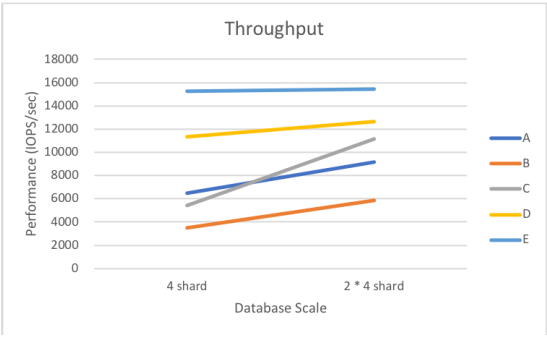

FIGURE 21. The throughput comparison in single server and two servers where each server considers four shards.

TABLE 2. The throughput improvement captured in Fig. 21.

	A	B	C	D	E
1	140.99%	141.00%	138.00%	241.02%	155.00%
2	142.00%	137.00%	132.00%	154.55%	154.00%
3	135.00%	133.00%	121.00%	157.81%	153.00%
4	139.00%	136.00%	115.00%	167.00%	148.00%

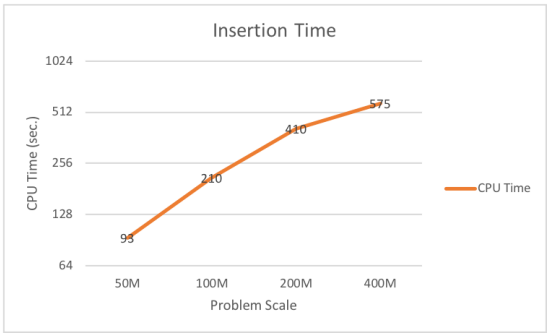

FIGURE 22. The performance of STDF data insertion.

For the scale out storage consideration, both strategies achieve the purposes.

D. INSERTING STDF DATA TO MongoDB

To capture the real world performance, we measure the total CPU time that the proposed system required to process various testing data. This experiment evaluated the efficiency of insert actions. The experimental results are shown in Figure 22. The number of insertions on the horizontal axis shows exponential growth with base 2; the vertical axis shows the time spent in execution, in seconds.

When inserting less than 2 million STDF data entries, the processing time grew in multiples of 2. However, when the data reached 4 million entries, the execution time was only slightly longer than the original 40.24%, but the data growth was 10 times greater than the original rate. Therefore, for semiconductor testing, the solution proposed in this paper effectively assists the production line to save test result data.

VI. CONCLUSION AND DISCUSSION

With the continuous advancement of processing technology, the quantity of data generated by semiconductor manufacturing and testing is a challenge for enterprises. These data can be used to not only grasp the quality of production but also trace possible flaws in the process. This study provided an STDF data storage environment for semiconductor test data that can consist of a small storage system or an idle device in the factory. This study used a Docker container and MongoDB to build a storage environment that substantially improved the response time to hardware failure. A shard server environment to build the system is recommended to the stability and security considerations of the system to provide stable access to data for an enterprise.

Semiconductor test data can be used for more than STDF files. In the future, other production-related data can also be stored in the cloud storage system. Appropriately selecting a sharding key for data sharding can conveniently divide the large quantity of data; data query could directly correspond to the required shard to enhance query speed. The direction of experimentation in the future will be as follows:

- The application integration: the saved testing data can be applied to various processes and applications. For example, production manager tracks the equipment health via the prognostics and health management (PHM) system. The MongoDB not only needs to save testing data but also outputs the specific data for further applications. So that the resource management is necessary. To manage and dispatch jobs in docker platform, Kubernetes provided by Google is a candidate solution [26]–[28]. Kubernetes provides some convenient properties in terms of managing docker resources, e.g. rapidly deploying resources and elastic applications. To integrate applications, Kubernetes is a candidate for resource management.
- The data visualization: we will try to visualize the selected query results to indicate the information provided by the semiconductor test data. To realize this issue, we have to handle the JOIN process in NoSQL database. Since NoSQL database, e.g. MongoDB, does not consider schema, the JOIN operator is not supported. There are alternatives to merge the data from two collections, e.g. lookup and aggregate functions. However, the major problem is the performance. Even if the JOIN processes can be realized between several collections, the performance should be optimized in terms of throughput and latency.
- The track of equipment health: the equipment health determines the product quality. This implies that the quality check staff can use the test data to track the equipment health. Therefore, the production manager can apply the testing data to design prognostics and health management system. On the other hand, the testing data should also be revised to keep track the equipment health. For example, we have to consider data of the life cycle time of the device (e.g., the data of the probe card) to obtain the effects of the gradual deterioration of the production equipment on the test results. Thus, the model of equipment health can be derived. For the performance optimization, the hybrid database

structure is a possible solution. For example, we may consider rational database and in-memory database to handle the cross-collection processes and large scale read-write analysis respectively.

REFERENCES

[1] B. G. Tudorica and C. Bucur, "A comparison between several NoSQL databases with comments and notes," in *Proc. RoEduNet Int. Conf. 10th Ed., Netw. Educ. Res.*, 2011, pp. 1–5.
[2] United Software Associates. (Mar. 2015). "*Comparative Benchmarks: MongoDB vs. Couchbase vs. Cassandra*. [Online]. Available: https://www.mongodb.com/collateral/comparative- benchmarks-mongodb-vs-couchbase-vs-cassandra
[3] W. Xu, Z. Zhou, H. Zhou, W. Zhang, and J. Xie, "MongoDB improves big data analysis performance on electric health record system," in *Proc. Life Syst. Modeling Simulation*, 2014, pp. 350–357.
[4] Q. Wang, Z. Shen, L. Ma, and S. Yin, "The storage of wind turbine mass data based on MongoDB," in *Computer Engineering and Networking* (Lecture Notes in Electrical Engineering), vol. 277. Springer, 2014, pp. 403–409.
[5] E. Barbierato, M. Gribaudo, and M. Iacono, Performance evaluation of NoSQL big-data applications using multi-formalism models," *Future Gener. Comput. Syst.*, vol. 37, pp. 345–353, Jul. 2014.
[6] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant Web services," *SIGACT News*, vol. 33, no. 2, pp. 51–59, Jun. 2002.
[7] E. A. Brewer, "Towards robust distributed systems (abstract)," in *Proc. 19th Annu. ACM Symp. Princ. Distrib. Comput. (PODC)*, New York, NY, USA, 2000, p. 7.
[8] T. Haerder and A. Reuter, "Principles of transaction-oriented database recovery," *ACM Comput. Surv.*, vol. 15, no. 4, pp. 287–317, Dec. 1983.
[9] *Standard Test Data Format Specification Version 4.0*, Teradyne, Boston, Ma, USA, 2007.
[10] A. Khoche, P. Burlison, J. Rowe, and G. Plowman, "A tutorial on STDF fail datalog standard," in *Proc. IEEE Int. Test Conf.*, Oct. 2008, pp. 1–10.
[11] A. Khoche *et al.*, "STDF memory fail datalog standard," in *Proc. 27th IEEE VLSI Test Symp.*, May 2009, pp. 209–214.
[12] M. Seuring, M. Braun, A. Ma, G. Eide, K. Yang, and H. Tang. "Employing the STDF V4-2007 standard for scan test data logging," *IEEE Des. Test. Comput.*, vol. 29, no. 6, pp. 91–99, Dec. 2012.
[13] M.-J. Wu, J.-S. R. Jang, AND J.-L. Chen, "Wafer map failure pattern recognition and similarity ranking for large-scale data sets," *IEEE Trans. Semicond. Manuf.*, vol. 28, no. 1, pp. 1–12, Feb. 2015.
[14] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proc. 1st ACM Symp. Cloud Comput. (SoCC)*, New York, NY, USA, 2010, pp. 143–154.
[15] S. Patil *et al.*, "YCSB++: Benchmarking and performance debugging advanced features in scalable table stores," in *Proc. 2nd ACM Symp. Cloud Comput., (SOCC)*, New York, NY, USA, 2011, pp. 9:1–9:14.
[16] V. Abramova, J. Bernardino, and P. Furtado, "Testing cloud benchmark scalability with cassandra," in *Proc. IEEE World Congr. Services*, Jun./Jul. 2014, pp. 434–441.
[17] T. T. Aye *et al.*, "Data driven framework for degraded pogo pin detection in semiconductor manufacturing," in *Proc. IEEE 10th Conf. Ind. Electron. Appl. (ICIEA)*, Jun. 2015, pp. 345–350.
[18] Z. Da, W. Yang, P. Ran, and Y. Huo, "Program design of JSON to structured data conversion," in *Proc. MATEC Web Conf.*, Les Ulis, France: EDP Sciences, vol. 139, 2017, Art. no. 00166.
[19] Z. Wang, D. Zhou, and S. Chen, "STEED: An analytical database system for tree-structured data," *Proc. VLDB Endowment*, vol. 10, no. 12, pp. 1897–1900, 2017.
[20] T. Combe, A. Martin, and R. Di Pietro, "To docker or not to docker: A security perspective," *IEEE Cloud Comput.*, vol. 3, no. 5, pp. 54–62, Sep./Oct. 2016.
[21] M. Peveler, E. Maicus, B. Holzbauer, and B. Cutler, "Analysis of container based vs. Jailed sandbox autograding systems," in *Proc. 49th ACM Tech. Symp. Comput. Sci. Educ.*, Feb. 2018, p. 1087.
[22] R. Dua, A. R. Raja, and D. Kakadia, "Virtualization vs containerization to support PaaS," in *Proc. IEEE Int. Conf. Cloud Eng. (IC2E)*, Mar. 2014, pp. 610–614.
[23] X. Lin, L. Lei, Y. Wang, J. Jing, K. Sun, and Q. Zhou, "A measurement study on linux container security: Attacks and countermeasures," in *Proc. 34th Annu. Comput. Secur. Appl. Conf.*, Dec. 2018, pp. 418–429.
[24] E. Bacis, S. Mutti, S. Capelli, and S. Paraboschi, "DockerPolicyModules: Mandatory access control for docker containers," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, Sep. 2015, pp. 749–750.
[25] A. R. Manu, J. K. Patel, S. Akhtar, V. K. Agrawal, and K. N. B. S. Murthy, "Docker container security via heuristics-based multilateral security-conceptual and pragmatic study," in *Proc. Int. Conf. Circuit, Power Comput. Technol. (ICCPCT)*, Mar. 2016, pp. 1–14.
[26] V. Medel, O. Rana, J. Á. Bañares, and U. Arronategui, "Modelling performance & resource management in kubernetes," in *Proc. IEEE/ACM 9th Int. Conf. Utility Cloud Comput. (UCC)*, Dec. 2016, pp. 257–262.
[27] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, omega, and kubernetes," *Queue*, vol. 14, no. 1, p. 10, 2016.
[28] V. Medel, R. Tolosana-Calasanz, J. Á. Bañares, U. Arronategui, and O. F. Rana, "Characterising resource management performance in Kubernetes," *Comput. Elect. Eng.*, vol. 68, pp. 286–297, May 2018.
[29] U. Störl, T. Hauf, M. Klettke, and S. Scherzinger, "Schemaless NoSQL data stores-Object-NoSQL Mappers to the rescue?" *Datenbanksysteme für Business, Technologie und Web (BTW)*, pp. 579–599, Mar. 2015.
[30] D. Vohra, "Why NoSQL?" in *Pro Couchbase Development*, Berkeley, CA, USA: Apress, pp. 1–18, 2015.
[31] V. Jain and A. Upadhyay, "MongoDB and NoSQL databases," *Int. J. Comput. Appl.*, vol. 167, no. 10, pp. 16–20, 2017.
[32] J. R. Lourenço, V. Abramova, M. Vieira, B. Cabral, and J. Bernardino, "NoSQL databases: A software engineering perspective," in *New Contributions in Information Systems and Technologies*, Cham, Switzerland: Springer, 2015, pp. 741–750.
[33] S. Binani, A. Gutti, and S. Upadhyay, "SQL vs. NoSQL vs. NewSQL-A comparative study," *Database*, vol. 6, no. 1, pp. 1–4, 2016.

CHEN-KUN TSUNG received the B.Sc. and M.Sc. degrees in computer science and information engineering from Dayeh University, Changhua, Taiwan, in 2004 and 2006, respectively, and the Ph.D. degree in computer science and information engineering from National Chung Cheng University, in 2014. He is currently an Assistant Professor of computer science and information engineering with the National Chin-Yi University of Technology, Taichung, Taiwan. His current research interests include cloud computing, big data, web-based applications, and combinatorial optimization.

HSIANG-YI HSIEH received the M.Sc. degree in computer science from Tunghai University, Taichung. His current research interests include cloud computing and data storage.

CHAO-TUNG YANG received the B.Sc. degree in computer science from Tunghai University, Taichung, Taiwan, in 1990, and the M.Sc. and Ph.D. degrees in computer science from National Chiao Tung University, Hsinchu, Taiwan, in 1992 and 1996, respectively. He is currently a Distinguished Professor of computer science with Tunghai University. His current research interests include cloud computing and big data, parallel and multicore computing, and web-based applications. He serves in a number of international journal editorial boards.