NoSQL Database Characteristics and Benchmark System

Kota Tsuyuzaki and Makoto Onizuka

Abstract

The NTT Software Innovation Center has been researching and developing NoSQL (Not Only SQL; SQL: structured query language) databases as storage platforms for big data. In this article, we discuss the characteristics of NoSQL databases and their benchmark system. We also report the results of applying the benchmark to MongoDB, a typical example of a NoSQL database.

1. Introduction

The amount of enterprise data has been dramatically increasing, and such large-scale data is referred to as *big data*. Many services and analytical applications based on big data have emerged, and users want to obtain or mine the most recent and useful knowledge from big data. Handling big data requires advanced technology for data storage. In particular, NoSQL (Not Only SQL; SQL: structured query language) databases have attracted attention for big data storage. NoSQL is designed specifically to manage big data, a task for which commonly used relational database management systems (RDBMSs) are not well suited.

We describe the main features and characteristics of NoSQL databases and report the results of a NoSQL benchmark using MongoDB [1] as an example.

2. NoSQL

2.1 Features

The three main features of NoSQL databases are scale-out, replication, and flexible data structure (**Fig. 1**). We explain these three features below.

Scale-out refers to achieving high performance by using many general-purpose machines in a distributed manner. Distributing the data over a large number of machines enables scaling of the data set and distribution of the processing load. A common feature of many NoSQL databases is that data is automatically distributed to new machines when they are added to the cluster, so the performance is also improved. Scale-out is evaluated in terms of scalability and elasticity.

Replication is the copying of data to achieve data redundancy and load distribution. Even if data consistency has been lost among the replicas, it is eventually achieved: this is known as *eventual consistency*. Replication is evaluated in terms of consistency and availability.

A flexible data structure means that there is no need to define a structure as a *database schema*. Traditional RDBMSs require pre-defined schemas, and redefining them carries a high cost. NoSQL, on the other hand, does not require defined schemas, so users can store data with various different structures in the same database table. However, most NoSQL databases do not support high-level query languages such as SQL, which is used by RDBMSs, so products that support either simple relational operations or indexing have been released. This feature is evaluated qualitatively.

2.2 Characteristics and benchmark system

Because NoSQL databases feature scale-out and replication, a NoSQL benchmark should take scalability, elasticity, consistency, and availability into account as well as performance. We explain each characteristic and describe the benchmarking software design points concerning these characteristics.

Scalability indicates how the performance of a NoSQL database cluster scales with the number of physical machines. If performance improves as

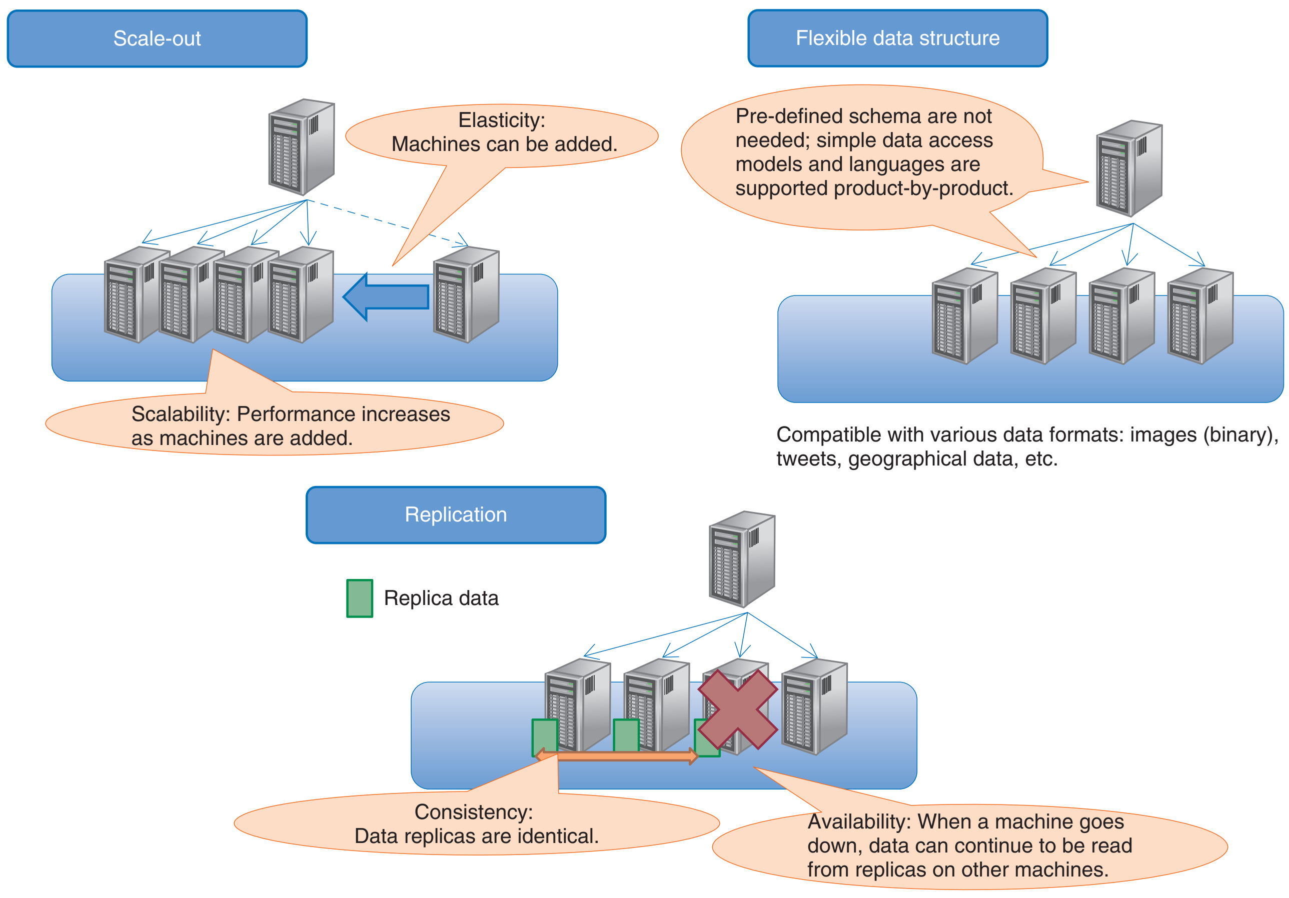
Scale-out
Elasticity: Machines can be added.
Scalability: Performance increases as machines are added.
Flexible data structure
Pre-defined schema are not needed; simple data access models and languages are supported product-by-product.
Compatible with various data formats: images (binary), tweets, geographical data, etc.
Replication
🟩 Replica data
Consistency: Data replicas are identical.
Availability: When a machine goes down, data can continue to be read from replicas on other machines.

Fig. 1. NoSQL database features and characteristics.

machines are added to a NoSQL database cluster, we can say that the NoSQL product has high scalability. In scalability benchmarking with many physical machines, the load generator, which benchmarks the NoSQL database cluster, is often a bottleneck. An effective approach for preventing this is to design benchmark software as a distributed system running on the cluster (**Fig. 2**).

Elasticity enables the addition of physical machines to a cluster while the NoSQL database is running on it. Elasticity benchmarking involves measuring the impact of adding a physical machine to the NoSQL database cluster. The benchmarking requires the addition of a machine while the performance is being measured. The impact of adding a physical machine has been reported for NoSQL benchmarking with the Yahoo! Cloud Serving Benchmark (YCSB) [2]. That paper [2] also reported a performance instability issue was arose during several hours of elasticity for Cassandra, a NoSQL database product.

Consistency is a measure of the strength of data integrity. The parameters used to evaluate it include the number of replicas and the latency within the cluster. The consistency benchmarking software must check the consistency among replicas in an update-heavy workload, so data management of that workload, in which data should be updated, is necessary.

The final characteristic, availability, refers to the ability of the overall system to continue operating during a network failure, called a *network partition*, or a physical machine failure. High availability means that the system can work without interruption and without degraded performance, even when some machines go down and the network is partitioned or both. In general, network partitioning makes it difficult to ensure both consistency and availability at the same time, so NoSQL databases are designed to prioritize one or the other in operation. Measuring the effect on performance in addition to that on operating continuity is important in the index for availability as

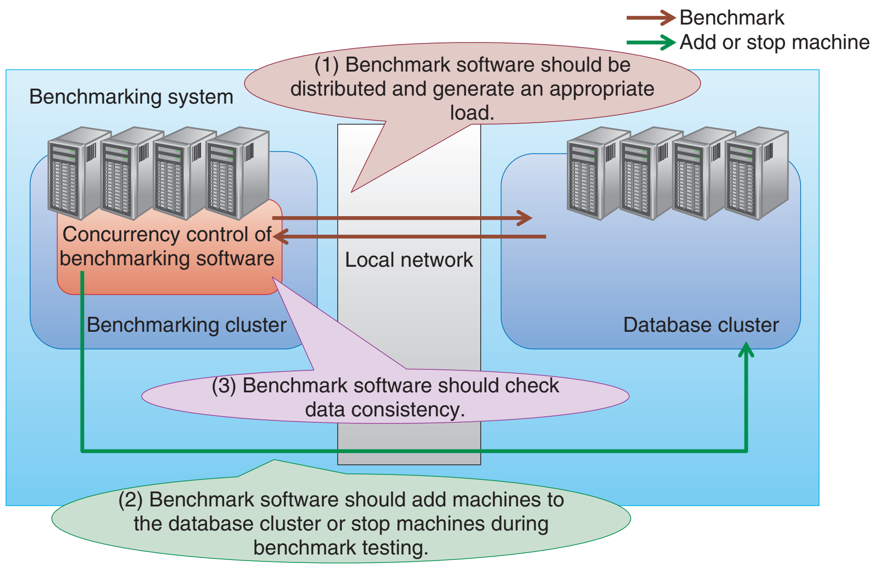
Benchmark
Add or stop machine
(1) Benchmark software should be distributed and generate an appropriate load.
Benchmarking system
Concurrency control of benchmarking software
Local network
Benchmarking cluster
Database cluster
(3) Benchmark software should check data consistency.
(2) Benchmark software should add machines to the database cluster or stop machines during benchmark testing.

Fig. 2. Important points concerning benchmarking system configuration and benchmarking.

well as that for elasticity. The important points regarding benchmark software are summarized below.

(1) It should be distributed and should generate an appropriate load.

(2) It should add machines to the database cluster or shut down machines automatically during benchmark testing.

(3) The benchmark software should check data consistency.

We present selected benchmark results obtained using custom benchmark software based on YCSB, which can automatically execute items (1) and (2).

3. Benchmarking evaluation

3.1 Benchmarking and evaluation of MongoDB

This benchmark test assumed a service in which user data is continually increasing, such as blog articles in a social networking service (SNS). The target software for the benchmarking was MongoDB, which features scale-out, replication, and a flexible data structure. Scalability is generally measured in the benchmarking of NoSQL databases [3] and MongoDB does not allow eventual consistency. Therefore, we present the results for elasticity and availability, which are particularly important when designing a system such as the SNS described above, by using a NoSQL database with data having those characteristics.

The performance of MongoDB depends on whether or not the entire data set resides in physical cache memory; therefore, we used two data sets: a small data set that could be fully stored in cache memory and a large data set that was too large to fully reside in cache memory, so data swapping was necessary.

3.2 Elasticity benchmark results

The results for elasticity are presented in **Fig. 3**, where the horizontal axis is elapsed time from the beginning of measurement and the vertical axis is performance relative to the throughput for ideal scaling from the addition of one machine. A physical machine was added 600 seconds after the beginning of measurement.

Comparing the results for the small and large data sets in Figs. 3(a) and (b), we can see that the performance of MongoDB with the large data set became unstable just after the machine was added and the performance gradually decreased. This behavior was caused by the increase in disk access during data migration of the large data set. Since there was a large reallocation load on the disks after the machine was added, the NoSQL took a lot of time to complete the data migration.

When the data set was small, as in Fig. 3(a), the impact of adding the physical machine ended after a minute or two and performance then became stable again. However, the throughput did not reach the ideal value for adding a machine in the case of the small data set. This indicates that the added machine

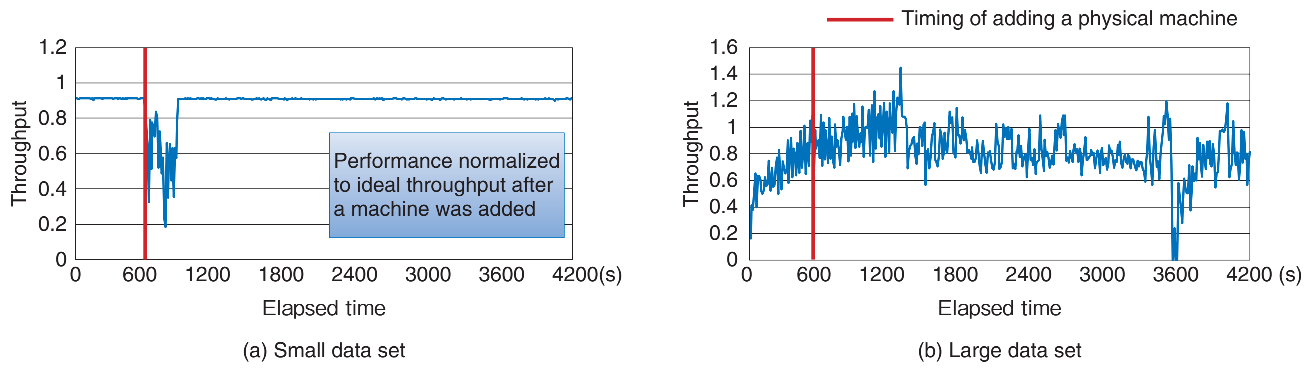

Fig. 3. Effect of elasticity benchmarking process on NoSQL database performance.

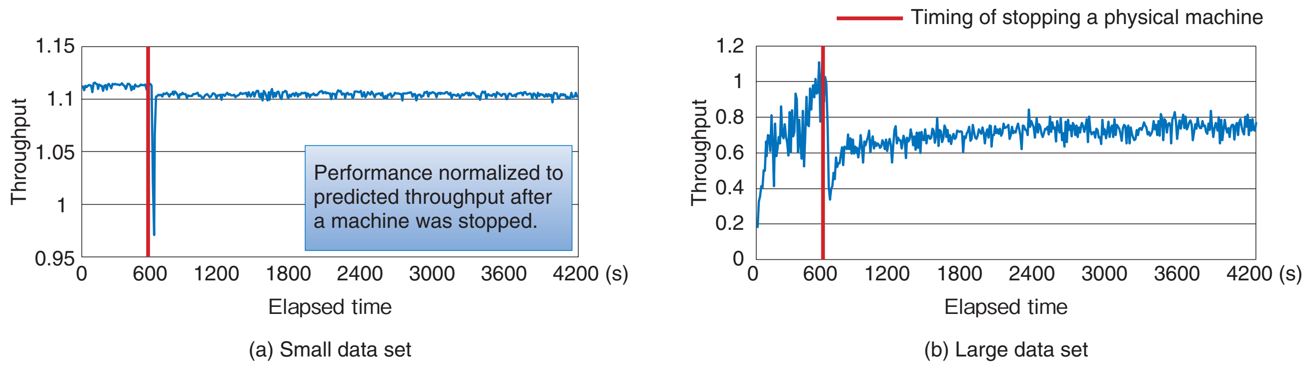

Fig. 4. Effect on performance during NoSQL database availability benchmarking.

was used inefficiently.

3.3 Availability benchmark results

The benchmark results for availability are presented in **Fig. 4**. The horizontal axis is time and the vertical axis is the measured throughput for when a machine had gone down.

Comparing the results for the small and large data sets in Figs. 4(a) and (b), we see that, after a machine was stopped, performance took longer to recover for the large data set than for the small data set. The performance for the large data set did not return to the previous level, remaining at only about 70% of the ideal performance. We believe that this was caused by increased disk access for reading data on machines that had replicas of data on the stopped machine. This increase in disk access caused a bottleneck in the entire system.

For the small data set in Fig. 4(a), the decrease in performance with the stopped machine was greater than expected. We believe that this is because there was almost no effect from the input/output load for the small data set.

4. Concluding remarks

Our benchmark testing of MongoDB for elasticity and availability revealed that data size has a significant impact on database performance when the system is extended or machines are taken off-line. However, these characteristics vary with the NoSQL product. Therefore, when designing the system configuration of actual systems that use NoSQL databases, one should benchmark elasticity and availability as well as performance and scalability. For example, the trade-off with the impact on performance and recovery time must be estimated and reflected in the system design in terms of the number of machines and the hardware configuration.

Our future work includes trying to improve the benchmarking of elasticity and availability to expand the use of NoSQL for big data. We believe that techniques for controlling the trade-off between performance and features on the basis of such benchmarking results will become more important in the future. The NTT Software Innovation Center will continue to engage in research and development of technology for handling big data.

References

[1] MongoDB. http://www.mongodb.org/
[2] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," Proc. of the ACM Symposium on Cloud Computing 2010 (ACM SOCC 2010), Indianapolis, IN, USA.
[3] T. Dory, "Comparing Scalable NOSQL Databases," http://www.slideshare.net/ThibaultDory/comparing-nosql-databases-benchmark
[4] J. Miyazaki and M. Onizuka, "Current trends and techniques on data cloud," the Information Processing Society of Japan, Vol. 52, No. 6, pp. 684–692, 2011 (in Japanese).

Kota Tsuyuzaki
Distributed Computing Technology Project, NTT Software Innovation Center.
He received the M.E. degree in information science from Waseda University, Tokyo, in 2010. Since joining NTT in 2010, he has been engaged in R&D of NoSQL databases. As a result of organizational changes in July 2012, he is now in NTT Software Innovation Center.

Makoto Onizuka
Distinguished Technical Member, Distributed Computing Technology Project, NTT Software Innovation Center and Visiting Associate Professor at the Graduate School of Information Systems, University of Electro-Communications.
He received the Ph.D. degree in computer science from Tokyo Institute of Technology in 2007. His primary research interests include systems and algorithms for data-intensive cloud computing.