International Journal of Engineering & Technology, 7 (4.31) (2018) 375-380

SPC

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper



Comparative Study for Load Management of HBase and Cassandra Distributed Databases in Big Data

Ali Y. Aldailamy[1]*, Abdullah Muhammed[2], Waidah Ismail[3], Abduljalil Radman[4]

[1,2]*Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, (UPM), Serdang, Selangor, Malaysia*
[3,4]*Faculty of Science and Technolgy, Universiti Sains Islam Malaysia (USIM), Nilai, Selangor, Malaysia*
**Corresponding author E-mail: A.Aldailamy7@gmail.com*

Abstract

The advancement in cloud computing, the increasing size of databases and the emergence of big data have made traditional data management system to be insufficient solution to store and manage such large-scale data. Therefore, there has been an emergence of new mechanisms for data storage that can handle large-scale data. NoSQL databases are used to store and manage large amount of data. They are intended to be open source, distributed and horizontally scalable in order to provide high performance. Scalability is one of the important features of such systems, it means that by increasing the number of nodes, more requests can be served per unit of time. Distribution and scalability are always companied with load management, which provides load balancing of work among multiple nodes. Load management efficiency varies from system to another according to the used load balancing technique. In this study, HBase and Cassandra load management with scalability will be evaluated as they are the most popular NoSQL databases modeled based on BigTable. In particular, this paper will compare and analyze the load management for the distributed performance of HBase and Cassandra using standard benchmark tool named Yahoo! Cloud Serving Benchmark (YCSB). The experiments will measure the performance of database operations with a different number of connections using different numbers of operations, database size, and processing nodes. The experimental results showed that HBase can provide better performance as the number of connections increase in the presence of horizontal scalability.

Keywords: *Big Data; BigTable; Cassandra; HBase; Load Management; YCSB.*

1. Introduction

Relational databases were undoubtedly dominating the area of database. Until recent time, they were the most widely used databases to manage data practically. Jogi et al. [13] stated that the most important features of relational databases is its ability to execute transactional updates and handle the underlying consistency issues. However, with the increasing datasets, terabyte-scale data and big data, it becomes extremely difficult to store and analyze the data effectively and efficiently as well as economically using relational databases. Moreover, big data introduces new types of data in terms of structures which include structured, semi-structured and unstructured data. The advancement in cloud computing and big data has founded the needs for database that can manage and process big data in efficient way. As a result, there have been emergence of many distributed database systems developed for the purpose of serving in cloud and intended to support the three structures of data. ACID (Atomicity, Consistency, Isolation, and Durability) transactions are not supported by such emerged systems, instead, they address cloud Online Transaction Processing (OLTP) [7]. Examples of the most popular systems developed for cloud data serving include HBase [1], Cassandra [25], PNUTS [5], SimpleDB [6], CouchDB [3], and Voldemort [4]. Although there are wide range of proposed applications for cloud data serving systems, there is a lack of comparisons between these systems performance. This gap has made it difficult to understand the tradeoffs between systems, and the suitable scalability and workloads for them [7]. The main difference between such systems is the data model utilized in the system. For instance, HBase and Cassandra systems are based on column-group oriented BigTable model. In fact, it is challenging to understand the implications of these systems performance for various applications with different loads.

HBase and Cassandra are the most famous NoSQL distributed databases that have been used in cloud serving systems. Both of them are intended to operate in cloud. They are based on a column-family-oriented model and sometimes are referred as a key-value store. HBase and Cassandra have a distribution system that allows for distributing data and processing tasks [8, 11]. Therefore, the write and read operations of these two systems make utilization of distributed system technique. HBase is built on top of Hadoop Distributed File System (HDFS) and can make utilization of Hadoop MapReduce. Hadoop allows to distribute and store data among the cluster's nodes for processing [10], this approach takes advantages of data locality. On the contrary, Cassandra has its own distribution system which allows achieving great performance on multi-node setups. Partitioner is the internal component of Cassandra which is responsible for the distribution of data across the nodes of the database cluster [11].

NoSQL databases are designed to distribute data and processing work among multiple machines. Therefore, load management mechanism is required in such systems in order to provide load balancing of work among all the nodes in the cluster [26]. Load balancing is the efficient distribution of incoming requests across the available nodes in the cluster in order to avoid bottleneck problems and to utilize the available resource equally. In cases of heavy workloads, load balancing is an essential task in distributed

Copyright © 2018 Authors. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

environments to achieve maximum utilization of resources [29]. Therefore, load balancing always allows to provide better performance as a result of good resource utilization. Effective load balancing is a result of efficient load management mechanisms. However the system architecture plays an important role in the efficiency of load management [18, 19]. Konstantinou et al. [26] demonstrated that heavy workload on the system that causes better performance always leads to the fact of deploying effective and efficient load balancing mechanism.

As HBase and Cassandra systems are based on BigtTable model, it is necessary to decide which system provides better performance with regarding to various load, available scalability, and data size. The ability to efficiently perform heavy write/read operations of data in order to obtain an effective and efficient performance is of much value. At such scale, the use of distributed architectures to achieve high throughput and to serve more requests is essential. Therefore, the main goal of this study is to measure the performance in presence of different loads and how the execution time of various operations is effected for HBase and Cassandra systems with regards to horizontal scaling and data size.

The rest of this paper is structured as follow, Section 2 introduces previous benchmarking experiments of HBase and Cassandra as well as the related works to this research. Section 3 presents brief background and important components and features of BigTable model, HBase, and Cassandra. Experiments setup and Benchmarking results are showed and discussed in section 4. Finally, conclusion is presented in section 5.

2. Related Work

NoSQL databases developed to fulfil the needs of highly scalable data storage and simple retrieval of data that are indexed using single key. It also provides distribution of data and computing work among multiple machines. In fact, each NoSQL database system has its own data storage model that has strengths and weaknesses. Chang et al. [9] described the design and implementation of simple, NoSQL and column-oriented data model named BigTable. The authors reported that BigTable model provides the ability for clients to have dynamic control over data layout. According to the original design of BigTable data model, scalability is one of the most important features of this model. Scalability reflects the system ability to scale out with the increasing data size and resources in both homogenous and heterogeneous environments.

HBase and Cassandra are both NoSQL, distributed databases modelled after the concept of BigTable model [27, 28]. BigTable is designed from the beginning to be highly distributed as well as share nothing architecture. Partitioning and replication of data over commodity servers are the techniques used by this model to provide reliability and availability. Data is portioned using its key and distributed to be stored in multiple machines. Replication means that the same element of data is stored many times on many nodes [11]. HBase and Cassandra are written in java and based on the same storage model. With the aim of measuring the performance for cloud serving systems, Cooper et al. [7] introduced an open source benchmarking tool named Yahoo! Cloud Serving Benchmark (YCSB). In distributed environment, the authors use YCSB to compare four NoSQL databases including a simple sharded MySQL implementation, Yahoo!s PNUTS, HBase, and Cassandra. Unfortunately, they did not provide analysis of their performance with regards to scalability and various loads. In a different publication, experimental comparison of scalability and how it effects the performance in HBase and Cassandra systems was conducted [14]. Similarly, the authors did not highlight the issue of various loads on those systems. However, in real systems there will be tens if not hundreds or thousands of connections that need to be served in parallel which not been considered by all previous works.

Another comparison of NoSQL databases performance is what done by Rabl et al. [15]. Indeed, they performed a comprehensive performance evaluation of six NoSQL data storages including HBase and Cassandra. Although they focused on the performance evaluation of HBase and Cassandra in terms of scalability, they did not take into consideration their performance with different loads that may happen in real applied systems. More comprehensive and extended evaluation of HBase and Cassandra was done by Kuhlenkamp et al. [16]. The authors reproduced some experiments from [15] and [7] and validated in two environments: internal cluster and real cloud system of Amazon. They also performed extended experiments using YCSB with increasing loads in HBase only. In fact, the more related work to this research is what has been done by Abramova et al. [8] which provided a detailed analysis of Cassandra scalability and its performance as the number of nodes involved in the experiment was increased. In this research, the number of nodes was increased using different loads ranging from 1 to 6000 connections using YCSB. More recent study is conducted by Jogi et al. [13], in which they performed a comparison between HBase, Cassandra and MYSQL in order to evaluate their performance of heavy write operations using web based REST (Representational State Transfer) applications.

This paper varies from all previous evaluations by focusing on the implication of different loads on the performance of HBase and Cassandra. Furthermore, a detailed analysis of both systems performance in distributed environment with regards to cluster scalability and database size using different loads is provided. Finally, their performance for YCSB workload A and B in different cluster size with different database size and different loads is measured.

3. Architectures

This study investigates the effects of different loads on the scalability, data size and number of operations of the two BigTable-based storage model and distributed databases: HBase and Cassandra. In this section, we will give some brief background of their architecture.

3.1. BigTable Architecture

The development of BigTable aimed to design a platform that can scale up with the dataset size and over large number of commodity processing machines using the partitioning mechanism [9]. Column-Oriented concept is also supported by the partitioning mechanism. Unlike row-oriented layout, column-oriented layout stores each column data contiguously in separate file that can be stored in separated disk. Therefore, column family is introduced by the BigTable model in order to allow rows to have fixed number of column families defined at the creation time. However, each column family can have variable number of columns, which means that the user can add columns at the run time and the number of columns can be different in each row [9, 11]. Moreover, BigTable is designed to allow any cell to have multiple timestamp versions of data. Finally, user can access the data stored in a cell using row key, column family, column qualifier, as well as timestamp [11, 20].

BigTable model is designed with sequential write to memory which means it does not need for disk seek. Thus, the model has highly optimized all types of write operations, insert, update, and delete. As shown in Fig. 1, MemTable is the internal component used by BigTable to store data in the memory before it is written to SSTable on the disk. For writing process, the write transaction is firstly stored in the log file for recovery in case of system crash. The transaction data is then written to the in-memory MemTable. Finally, once MemTable size reaches a predefined threshold, all MemTable data are then flushed to the SSTable on the disk. Whenever read request of data is made, the model first searches for the requested data by using the row key in the MemTable. If not exists in the MemTable, the data will be retrieved from the SSTable on the disk. In order to provide optimized read operation,

the system always performs periodic data compaction. In the data compaction process, the system periodically merges many SSTables in one big SSTable which achieves higher reading efficiency [9].

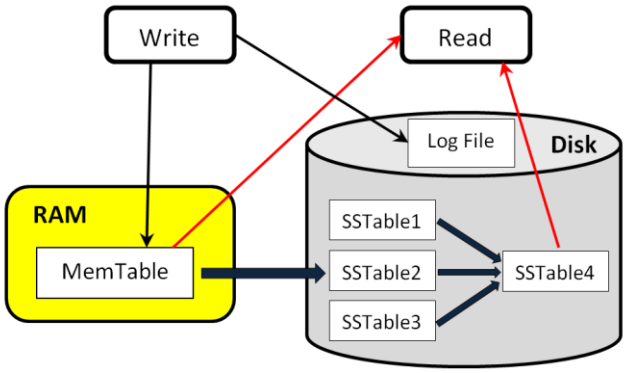


Fig. 1: Read/Write operations in the BigTable architecture.

3.2. HBase Architecture

HBase is a distributed and non-relational database based on the concept of Google's BigTable: A distributed storage system for structured data [9]. The main goal of HBase is to store huge tables with millions of columns and billions of rows. HBase is used when random, real-time CRUD (Create, Read, Update, and Delete) are needed to deal with data. It is based on famous Master-Slave architecture with one HMaster and multiple HRegionServers [22]. According to the architecture of HBase, data is distributed and stored in a group of Region Servers. Therefore, mapping of key to the corresponding server is required to locate the needed data, this mapping information is stored in tables named Mapping Tables.
HBase is an Apache open-source framework that is developed as part of Hadoop eco system. It is written in java to run on top of Hadoop Distributed Filesystem (HDFS) in order to provide BigTable functions for Hadoop [17]. Using HDFS storage system by HBase as its own data storage has advantages and disadvantages. This approach has the advantage that HDFS handles data replication, data consistency and resiliency, so HBase does not need to provide such functions and worry about storage system. Gokavarapu [11] stated that this approach has downside, HBase is constrained and limited by the HDFS characteristics, which do not provide optimized access for random read. Moreover, this architecture causes extra network latency for the additional access from the server of database to the File server (i.e. the Hadoop cluster DataNode).

3.3. Cassandra Architecture

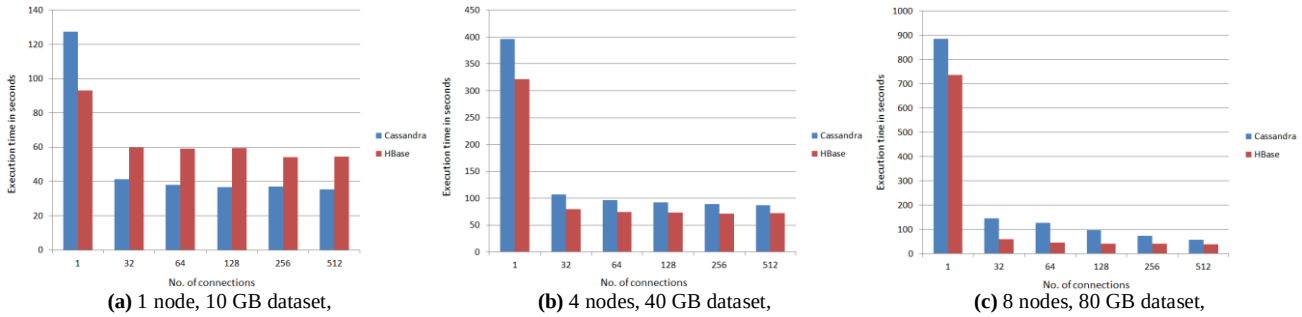
Cassandra is a NoSQL column-oriented database that was originally designed by Facebook but now is developed by Apache Software. Apache Cassandra is a distributed, NoSQL and open-source database management system developed to store and manage huge volume of data using commodity machines. It was designed to provide high availability of data without a single point of failure. As a column-oriented system, the data in Cassandra is stored and managed in groups of column families and in tables that is vertically oriented [2]. Abramova et al. [8] stated that Cassandra is best suited for applications that need heavy write. Cassandra is based on BigTable storage model for storage engine and replication of Dynamo architecture. The architecture of Apache Cassandra was designed to provide the ability for the system to scale [13]. Unlike the legacy Master-Slave architecture of HBase architecture, Apache Cassandra distributed system has a ring cluster that has peer-to-peer nodes with one elected node that has extra work as coordinator [23]. This approach of design has made Cassandra cluster easy to setup and maintain. In addition, this architecture allows the data to be distributed and stored in all the cluster's nodes, as well as making replications of the data as user requirements.
Cassandra handles the distribution, storing, and maintenance of the data automatically, so developers do not need to worry about portioning of data [21]. Thus, it allows them to direct their energies and focus on creating a value-features applications. As mentioned before, Cassandra has a built in component named partitioner which is responsible for the distribution of data among the database cluster. The Partitioner computes numerical token for primary key of each rows using hashing mechanism. It then assigns this token to one of the cluster nodes in consistent and predictable way. Although, partitioner is a built-in component of Cassandra, it can be configured according to the user requirements. The default configuration of the partitioner is intended to make random and even distribution of data among cluster nodes. Finally, Cassandra has a load balancing that is responsible of maintaining the balance of workload across the cluster nodes. It takes an automatic action when changes such as adding or removing nodes is made to the cluster in order to keep the workload balanced over all the nodes [24].

4. Experiments and Results

The experiments aim to evaluate HBase and Cassandra performance with various workloads using increasing number of nodes, increasing dataset size, and increasing number of connections. In the experiments, the execution time of each workload is measured using increasing number of threads in order to evaluate the affects in execution time of various operations for both systems.

4.1. Experiments Setup

Our experimental cluster consist of 8 identical machines equipped with Intel core i5 with quad core, 8 GB of RAM, hard disk drive of 500GB and gigabyte network adapters that are connected together using 1000 Ethernet switch. Oracle Virtual Box 5.0.26 is used to make a cluster of 8 nodes, each has 4 cores and 6GB of RAM. Linux Ubuntu 16.04 is operated in all the nodes of the cluster. For HBase experiments, the cluster is running using Hadoop 2.7.0 distributed file system (HDFS), while Apache Hbase-1.1.8 is running in all the cluster nodes in the top of HDFS. HBase works on top of HDFS that has Master-Slave architecture. Therefore one of the nodes is assigned to be NameNode and DataNode at the same time. In the experiment of Cassandra, we use Apache Cassandra- 2.2.4 in all the nodes in order to make a cluster of 8 nodes. Cassandra distribution system is decentralized, which means all nodes organized in Peer-to-Peer fashion, so one of the nodes is elected to be the coordinator of the cluster.
In order to evaluate the performance of HBase and Cassandra with various loads of work, we use Yahoo Cloud Serving Benchmark (YCSB) workloads A and B. Workload A is a mixed workload of 50% read and 50% update operations. Whereas, workload B is heavy read with 5% update operations and 95% read operations. The thread count of YCSB client allows to define the number of connections from client to server that can be served simultaneously. YCSB has data generator that builds table of 10 fields and the row key field, each field is inserted with 100 byte of random data. Therefore, the size of each row in the table is about 1KB.
YCSB has two benchmark tiers which include performance tier and scalability tier. The Performance tier is provided to measure the latency of requests issued to the database while it is under load. However, as the amount of load increases, the latency of issued requests increases, this is due to potential tradeoff between throughput and latency. The scalability tier measure the effect on the database performance as the number of machines in the system is increased. In real databases, the size of the data is increasing, so adding more machines to the system is the best solution to mainta-

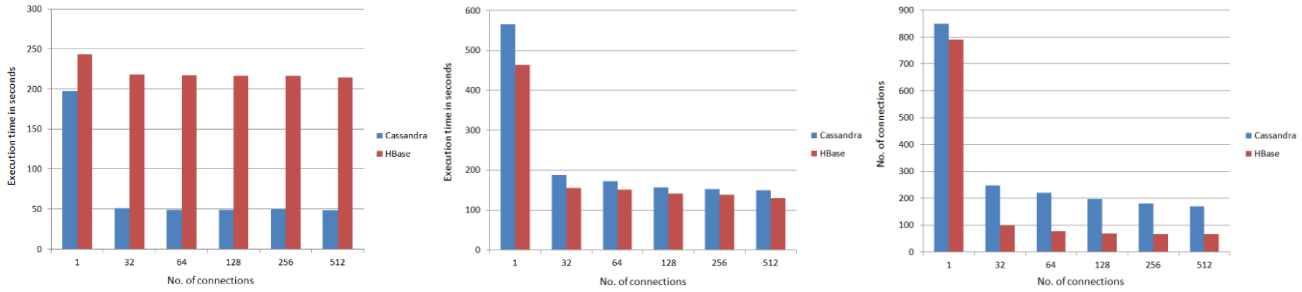


(a) 1 node, 10 GB dataset, and 10000 operation.

(b) 4 nodes, 40 GB dataset, and 40000 operation.

(c) 8 nodes, 80 GB dataset, and 80000 operation.

Fig. 2: Execution average time for Workload A.



(a) 1 node, 10 GB dataset, and 10000 operation.

(b) 4 nodes, 40 GB dataset, and 40000 operation.

(c) 8 nodes, 80 GB dataset, and 80000 operation.

Fig. 3: Execution average time for Workload B.

in the performance constant. Therefore, the database that provides good scalability should deliver the same performance as the dataset size, number of machines, and required throughput increase proportionally.

In the experiments, the database size was varied proportionally with the cluster size, we use datasets similar to the one used in [8], 10 billion records and 10 thousands operation with one node, 40 million records and 40 thousands operation with the four nodes, and finally 80 million records and 80 thousands operation with 8 nodes. However, various loads used in [14] are reused in our experiments. The number of requests is also increased proportionally with the size of the data and cluster. Table 1 summarizes the benchmarking experiments setup. The three experiments are executed for workload A and B. While execution time is the performance metric that we observed during the experiments.

Table 1: Experiments Setup.

Nodes	Dataset (GB)	Number of Operations
1	10	10,000
4	40	40,000
8	80	80,000

During the experiments, we made sure that the cluster is free of other running processes. The YCSB load process is executed first to load the table with data records. In order to make sure that flush-to-disk events of MemTable has taken place, the load of each workload and the experiments have run for long time. After each experiment, each system is reset to the state of new installed by clearing all in-memory data and deleting all data files from disk. Although, no other jobs were running in the cluster during the implementation of the experiments, each experiment is repeated 3 times and average time is taken in order to get as accurate results as possible.

4.2. Results and Analysis

The main goal of this paper is to measure the performance of HBase and Cassandra as the number of connections to the system increases. This section shows the results of the benchmarking experiments. For each workload, we present the execution time taken to execute specific number of operation as the number of connection increases. The same experiments are repeated in 1, 4, and 8 nodes with a proportional increase in the size of the dataset and the number of operations. The increase in the cluster size is to measure the scalability of HBase and Cassandra as the cluster horizontally scaled. We first report our results and point out significant values. Then, we discuss the results.

4.2.1. Workload A

In the first experiment, we run Workload A which is write-heavy workload with 50% update operations and 50% read operations. This type of workload is intended to compare the performance between read and write operations of the system. In the experiment of 1 node, HBase outperforms Cassandra with 1 connection, while HBase performance is degraded with all other connections in comparison with Cassandra as shown in Fig. 2 (a). However, the performance of HBase and Cassandra do not illustrate significant increase in the speedup even with the increase of the number of connections. This could happen when load balance mechanism runs in an environment with a very limited resources. In 4 and 8 nodes experiments as shown in Fig. 2 (b and c), HBase and Cassandra show stable increase in the speedup with the increase of number of connections, but HBase provides better performance in overall. This is an evidence that HBase has better load balance mechanism that makes good utilization of distributed resources. This good utilization is because of the use of HDFS file system of Hadoop which provides effective and efficient distributed storage system. This is similar to what reported by Cooper et al. [7], where HBase demonstrates the best write performance among other NoSQL databases. In addition, this good performance is also due to the use of Memstore which is a built-in component in HBase that is intended to allow HBase to provide an optimized write operations.

4.2.2. Workload B

The second workload is Workload B which is read-heavy workload on systems. It consists of 95% read operations and only 5% of write operations. This kind of workload is designed for its common use in many information systems of social web where

write-once and read-many paradigm is dominant. In other words, it is used to evaluate system where the read operation has the lion part of all operations. From Fig. 3 (a), Cassandra shows better performance in the experiment of 1 node with all connections. It also illustrates stable increase of the speedup in the experiments of 4 and 8 nodes. On the contrary, the performance of HBase is drastically degraded in 1 node experiments despite of increasing the number of connections. However, our results in Fig. 3 (b and c) show that HBase outperforms Cassandra in 4 and 8 nodes experiments in all connections. This is because that HBase uses Memstore to hold the latest written data for temporary time. This also enhances the reading process as the data is searched in the Memstore first.

To conclude, Ghandour et al [30] stated that HBase supports various load balancing techniques to equally divide the work among the available region servers. This techniques allow to provide an equal utilization of resources, so that the performance can be improved. On the other hand, Cassandra uses consistent hashing to partition data across the nodes of the cluster. However, the consistent hashing algorithm has some challenges such as non-equal data and load distribution among available nodes. This is a result of the random position assignment of each node on the cluster ring. Although, Lakshman et al. [25] has demonstrated that Cassandra has tackled such problem by analysing the load information of the ring and change the position of the lightly loaded nodes to reduce the nodes that are heavily loaded, but the provided solution has not produced an equal distribution of data and load among nodes of the cluster. Moreover, such solution has produced an extra computing overhead.

5. Conclusion

In this paper, load management of two famous NoSQL databases, HBase and Cassandra, were investigated within the context of their scalability. In particular, HBase and Cassandra load balancing mechanisms were evaluated as the number of connections that need to be served at the same time was increased. The experiments were conducted in order to test and analyze the performance of both databases to show how each system cope with various loads while the cluster and dataset size vary. In conclusion, Cassandra provided better performance in a single node. On the contrary, HBase performance becomes better with the increase in number of connections, the size of cluster, and dataset. This is an evidence that HBase has more efficient load balancing and good utilization of resources. In the experiments, both of the systems showed increase in the speedup as the number the number of connection increases. However, this increase of speedup is better in HBase than that of Cassandra. HBase is recommended when coping with heavy load and scalability is required. In a single node, Cassandra produced better performance in the presence of heavy loads. In summary, we deduce that HBase performs higher throughput than Cassandra taking into account the increase of connections and horizontal scalability of computing resources.

Acknowledgement

This work was supported by the International Grant USIM/INT-NEWTON/FST/IHRAM/053000/41616 under Newton-Ungku Omar Fund. This publication only reflects the authors' views.

References

[1] George L, HBase: The Definitive Guide: Random Access to Your Planet-Size Data, 2nd ed., O'Reilly Media, Inc., (August, 2017), pp: 1-522.

[2] Carpenter J and Hewitt E, Cassandra: The Definitive Guide: Distributed Data at Web Scale, 2nd ed., O'Reilly Media, Inc., (2016), pp: 1-337.

[3] Anderson JC, Lehnardt J, and Slater N, CouchDB: The Definitive Guide: Time to relax, 1st ed., O'Reilly Media, Inc., (November, 2010), pp: 1-184.

[4] Feinberg A, "Project Voldemort: Reliable distributed storage," Proceedings of the 10th IEEE International Conference on Data Engineering, Hannover, Germany, (2011).

[5] Cooper B F, Ramakrishnan R, Srivastava U, Silberstein A, Bohannon P, Jacobsen H-A, Puz N, Weaver D, and Yerneni R, "Pnuts: Yahoo!'s hosted data serving platform," Proceedings of the VLDB Endowment, vol. 1, no. 2, (January, 2008), pp. 1277–1288.

[6] Habeeb M, A developer's guide to Amazon SimpleDB, Upper Saddle River: Addison-Wesley Professional, (2010).

[7] Cooper BF, Silberstein A, Tam E, Ramakrishnan R, and Sears R, "Benchmarking cloud serving systems with YCSB," Proceedings of the 1st ACM symposium on Cloud computing - SoCC 10, (2010), pp: 143-154

[8] Abramova V, Bernardino J, and Furtado, P,"Evaluating Cassandra scalability with YCSB," Proceedings of International Conference on Database and Expert Systems Applications, Springer International Publishing, (2014), pp: 199–207.

[9] Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, and Gruber RE, "Bigtable: A distributed storage system for structured data," ACM Transactions on Computer Systems (TOCS), ACM, vol. 26, no. 2, (January, 2008), pp: 1–26.

[10] Carstoiu D, Cernian A, and Olteanu A, "Hadoop hbase-0.20.2 performance evaluation," New Trends in Information Science and Service Science (NISS), 2010 4th International Conference on. IEEE, (May, 2010), pp: 84-87.

[11] Gokavarapu H, "Exploring Cassandra and HBase with BigTable Model," Indiana University Bloomington.

[12] Abubakar Y, Adeyi TS, and Auta IG, "Performance Evaluation of NoSQL Systems using YCSB in a Resource Austere Environment," International Journal of Applied Information Systems, vol. 7, no. 8, (April, 2014), pp: 23–27.

[13] Jogi VD and Sinha A, "Performance evaluation of MySQL, Cassandra and HBase for heavy write operation," In Recent Advances in Information Technology (RAIT), 2016 3rd International Conference on IEEE, (March, 2016), pp: 586-590.

[14] Shi Y, Meng X, Zhao J, Hu X, Liu B, and Wang H, "Benchmarking cloud-based data management systems," Proceedings of the second international workshop on Cloud data management - CloudDB 10, ACM, (October, 2010), pp. 47-54.

[15] Rabl T, Gómez-Villamor S, Sadoghi M, Muntés-Mulero V, Jacobsen HA, Mankovskii S, "Solving big data challenges for enterprise application performance management," Proceedings of the VLDB Endowment. vol. 5, no. 12, (August, 2012), pp: 1724-1735.

[16] Kuhlenkamp J, Klems M, and Röss O, "Benchmarking scalability and elasticity of distributed database systems," Proceedings of the VLDB Endowment, vol. 7, no. 12, (August, 2014), pp: 1219–1230

[17] Tudorica BG and Bucur C, "A comparison between several NoSQL databases with comments and notes," 2011 RoEduNet International Conference 10th Edition: Networking in Education and Research, IEEE, (June, 2011), pp: 1-5

[18] Xiong A-P and Zou J, "Research of Dynamic Load Balancing Strategy on Hbase," Proceedings of the 5th International Conference on Information Engineering for Mechanics and Materials, (2015).

[19] Wang P and Qi Y, "Research of Load Balancing Based on NOSQL Database," In Applied Mechanics and Materials, Trans Tech Publications, vol. 602, (2014).

[20] Jmtauro C, Aravindh S, and Shreeharsha AB, "Comparative Study of the New Generation, Agile, Scalable, High Performance NOSQL Databases," International Journal of Computer Applications, vol. 48, no. 20, (June, 2012), pp: 1–4.

[21] DataStax Software, "Introduction to Apache Cassandra," DataStax Software company: https://www.datastax.com/resources/ whitepapers/intro-to-cassandra, last visit:20.01.2018

[22] Vora MN, "Hadoop-HBase for large-scale data," Computer science and network technology (ICCSNT), Proceedings of 2011 International Conference on Computer Science and Network Technology, IEEE, Vol. 1, (December, 2011), pp: 601-605 .

[23] Pirzadeh P, Tatemura J, Po O, and Hacıgümüş H, "Performance Evaluation of Range Queries in Key Value Stores," Journal of Grid Computing, vol. 10, no. 1, (March, 2012), pp: 109–132.

[24] Feng C, Zou Y, and Xu Z, "CCIndex for Cassandra: A Novel Scheme for Multi-dimensional Range Queries in Cassandra," In Semantics Knowledge and Grid (SKG), 2011 Seventh International Conference, on IEEE, (October, 2011), pp. 130-136.
[25] Lakshman A and Malik P, "Cassandra: a decentralized structured storage system," ACM SIGOPS Operating Systems Review, vol. 44, no. 2, (April, 2010), pp: 35-40.
[26] Konstantinou I, Tsoumakos D, Mytilinis I, and Koziris N, "DBalancer: distributed load balancing for NoSQL data-stores," Proceedings of the 2013 international conference on Management of data - SIGMOD 13, ACM, (June, 2013), pp: 1037-1040.
[27] Featherston D, "Cassandra: Principles and application," Department of Computer Science University of Illinois at Urbana-Champaign, (August, 2010).
[28] Bhupathiraju V and Ravuri RP, "The dawn of Big Data - HBase," 2014 Conference on IT in Business, Industry and Government (CSIBIG), IEEE, (March, 2014), pp: 1-4.
[29] Katyal M and Mishra A, "A comparative study of load balancing algorithms in cloud computing environment." arXiv preprint arXiv:1403.6918, (March, 2014).
[30] Ghandour A, Moukalled M, Jaber M, Falcone Y, "User-based Load Balancer in HBase." In Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017), (2017), pp: 364-368.