Hindawi Publishing Corporation
Scientific Programming
Volume 2015, Article ID 294614, 10 pages
http://dx.doi.org/10.1155/2015/294614

Research Article

Empirical Analysis of High Efficient Remote Cloud Data Center Backup Using HBase and Cassandra

Bao Rong Chang,[1] Hsiu-Fen Tsai,[2] Chia-Yen Chen,[1] and Cin-Long Guo[1]

[1]*Department of Computer Science and Information Engineering, National University of Kaohsiung, Kaohsiung 81148, Taiwan*
[2]*Department of Marketing Management, Shu-Te University, Kaohsiung 82445, Taiwan*

Correspondence should be addressed to Chia-Yen Chen; ayen@nuk.edu.tw

Received 6 September 2014; Accepted 12 December 2014

Academic Editor: Gianluigi Greco

HBase, a master-slave framework, and Cassandra, a peer-to-peer (P2P) framework, are the two most commonly used large-scale distributed NoSQL databases, especially applicable to the cloud computing with high flexibility and scalability and the ease of big data processing. Regarding storage structure, different structure adopts distinct backup strategy to reduce the risks of data loss. This paper aims to realize high efficient remote cloud data center backup using HBase and Cassandra, and in order to verify the high efficiency backup they have applied Thrift Java for cloud data center to take a stress test by performing strictly data read/write and remote database backup in the large amounts of data. Finally, in terms of the effectiveness-cost evaluation to assess the remote datacenter backup, a cost-performance ratio has been evaluated for several benchmark databases and the proposed ones. As a result, the proposed HBase approach outperforms the other databases.

1. Introduction

In recent years, cloud services [1, 2] are applicable in our daily lives. Many traditional services such as telemarketing, television and advertisement are evolving into digitized formats. As smart devices are gaining popularity and usage, the exchange of information is no longer limited to just desktop computers, but instead, information is transferred through portable smart devices [3, 4], so that humans can receive prompt and up-to-date information anytime. Due to the above reasons, data of all types and forms are constantly being produced, leaving the mass of uncorrelated or unrelated information, causing conventional databases to not be able to handle the workload in a big data environment. This leads to the emergence of nonrelational databases, of which many notable NoSQL databases that are currently being used by enterprises are HBase [5], Cassandra [6], and Mongo [7]. Generally, companies will assess the types of applications before deciding which database to use. To these companies, the data analysis of these databases can mean a matter of success or failure. For example, the mailing system, trading records, or number of hits on an advertisement, performing such retrieval, clearing, analysis, and transforming them into useful information for the user. As the types of information ever increases, the data processing abilities of nonrelational databases becomes ever challenging. The more well-known HBase and Cassandra databases are often used for a company as internal database system, and it uses its own distributed architecture to deal with data backup between different sites.

Distributed systems are often built under a single-cluster environment and contain a preventive measure against the single-point failure problem, that is, to prevent system crash or data loss. However, it could happen in such accidents as power shut-down, natural disaster, or manual error that leads to whole system collapse and then initiates a remote backup to the remote data center. Even though NoSQL database uses distributed architecture to prevent the risk of data loss, it has neglected the importance of remote data center backup. In addition to considering nodal independence and providing uninterrupted services, a good database system should also be able to support instant cross-cluster or cross-hierarchy remote backup. With this backup mechanism, data can be restored and prevent further data corruption problems. This paper will implement data center remote backup using two

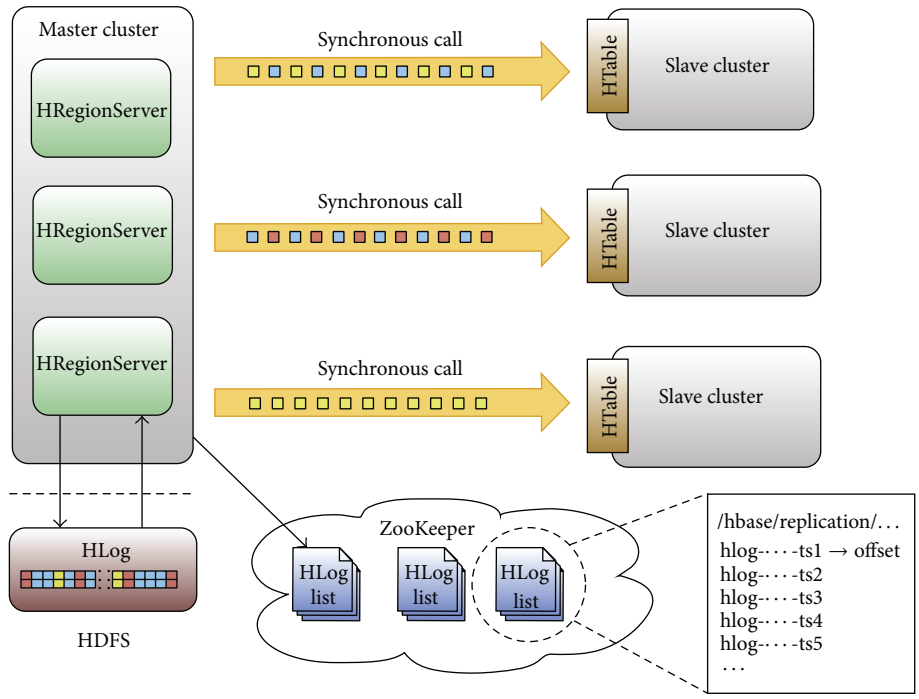

Figure 1: Remote HBase data center backup.

remarkable NoSQL databases and perform stress tests with a large scale of data, for instances, read, write, and remote data center backup. The experimental results of remote data center backup using HBase and Cassandra will show the assessment of their effectiveness and efficiency based on cost-performance ratio [8].

The following paragraphs of this paper are arranged as follows. In Section 2, large-scale database in data center will be described. The way to remote data center backup is given in Section 3. Section 4 proposes the method to implement the NoSQL database remote backup. The experimental results and discussion will be obtained in Section 5. Finally we drew a brief conclusion in Section 6.

2. Large-Scale Database in Data Center

The database storage structure can be divided into several types; currently, the more common databases are hierarchical (IBM IMS), network (Computer Associates Company IDMS), relational (MySQL, Microsoft SQL Server, Informix, PostgreSQL, and Access), and object-oriented (PostgreSQL). With the rapid growth of IT in recent years, the new data storage architecture to store large amounts of unstructured data, collectively called nonrelational database, that is, NoSQL Database (Google BigTable, Mongo DB, Apache HBase, and Apache Cassandra), was developed. NoSQL first appeared in 1998; it was developed by Carlo Strozzi as a lite, open sourced relational database, which does not provide the SQL function.

This paper will realize remote data center backup for the two distributed databases HBase and Cassandra. Both designs achieved two of the three characteristics that are consistency (C), availability (A), and partition tolerance (P) in C.A.P. theory [9].

HBase, a distributed database, works under the master-slave framework [10], where the master node assigns information to the slave node to realize the distributed data storage, meanwhile emphasizing consistency and partition tolerance characteristics. Regarding remote data center backup, a certain data center with HBase has the following advantages: (1) retain data consistency, (2) activate instant reading or writing of massive information, (3) access large-scale unstructured data, (4) expand new slave nodes, (5) provide computing resources, and (6) prevent a single-node failure problems in the cluster.

Cassandra, a distributed database, works under the peer-to-peer (P2P) [11] framework, where each node contains totally identical backup information to realize the distributed data storage with uninterrupted services, at the same time emphasizing availability and partition tolerance characteristics. As for remote data center backup, a certain data center with Cassandra has the following advantages: (1) each node shares equal information, (2) cluster setup is quick and simple, (3) cluster can dynamically expand new nodes, (4) each node has the equal priority of its precedence, and (5) cluster does not have a single-node failure problem.

3. Remote Data Center Backup

3.1. Remote HBase and Cassandra Data Centers Backup. Remote HBase data center backup architecture [12] is as shown in Figure 1. The master cluster and slave cluster must possess their own independent Zookeeper in a cluster [13]. The master cluster will establish a copy code for the data center and designate the location of the replication, so to achieve offsite or remote data center backup between different sites. Remote Cassandra data center backup architecture [14]

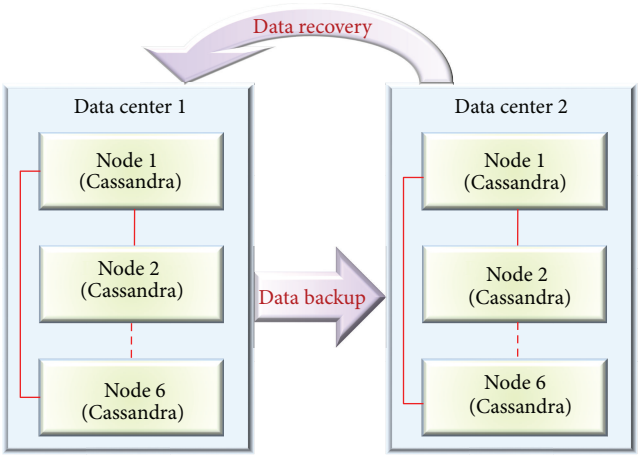

Figure 2: Remote Cassandra data center backup.

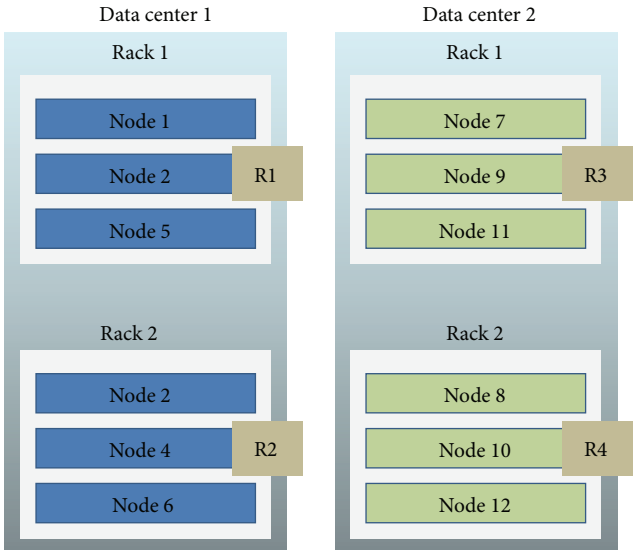

Figure 3: Illustration of rack.

is as shown in Figure 2. Cassandra is of peer-to-peer (P2P) framework connects all nodes together. When information is written into data center A, a copy of the data is immediately backed up into a designated data center B, and each node can designate a permanent storage location in a rack [15] as show in Figure 3. This paper expands the application of a single-cluster replication mechanism to the replication of data center level. Through adjusting the replication mechanism between data center and nodes, the corresponding nodes from two independent data centers are connected and linked through SSH protocol, and then information is distributed and written into these nodes by master node or seed node to achieve remote data center backup.

3.2. Cross-Platform Data Transfer Using Apache Thrift. Apache Thrift [16] was developed by the Facebook team [17], and it was donated to the Apache Foundation in 2007 to become one of the open source projects. Thrift was designed

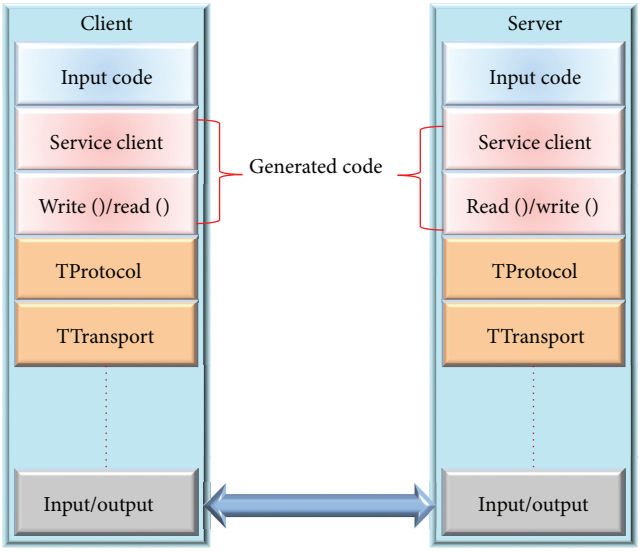

Figure 4: Apache Thrift architecture.

to solve Facebook's problem of large number of data transfers between various platforms and distinct programming languages and thus cross-platform RPC protocols. Thrift supports a number of programming languages [18], such as C++, C#, Cocoa, Erlang, Haskell, Java, Ocami, Perl, PHP, Python, Ruby, and Smalltalk. With binary high performance communication properties, Thrift supports multiple forms of RPC protocol acted as a cross-platform API. Thrift is also a transfer tool suitable for large amounts of data exchange and storage [19]; when comparing with JSON and XML, its performance and capability of large-scale data transfer is clearly superior to both of them. The basic architecture of Thrift is as shown in Figure 4. In Figure 4 the Input Code is the programming language performed by the Client. The Service Client is the Client side and Server side code framework defined by Thrift documents, and read ()/write () are codes outlined in Thrift documents to realize actual data read and write operations. The rest are Thrift's transfer framework, protocols, and underlying I/O protocols. Using Thrift, we can conveniently define a multilanguage service system, and select different transfer protocol. The Server side includes the transfer protocol and the basic transfer framework, providing both single and multithread operation modes on the Server, where the Server and browser are capable of interoperability concurrently.

4. Research Method

The following procedures will first explain how to setup HBase and Cassandra data centers using CentOS 6.4 system to achieve remote backup. Next, this system will test the performance of data centers against reading, writing, and remote backup of large amounts of information.

4.1. Implementation of HBase and Cassandra Data Centers. Data centers A and B are installed on the CentOS 6.4 operating system, and HBase and Cassandra data centers are setup

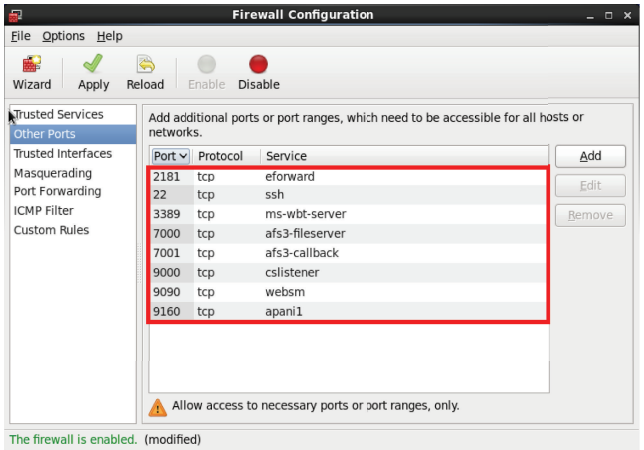

Figure 5: Settings to allow the passage through firewall.

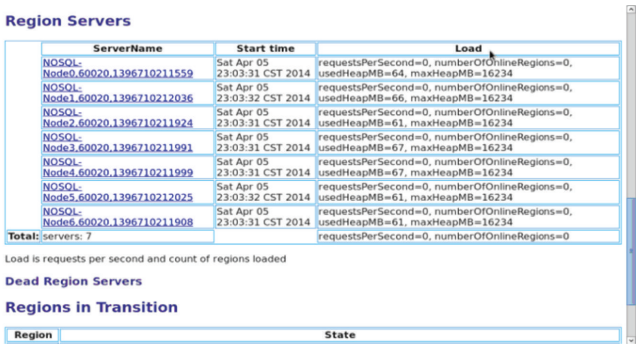

Figure 6: Examine HBase's node status.

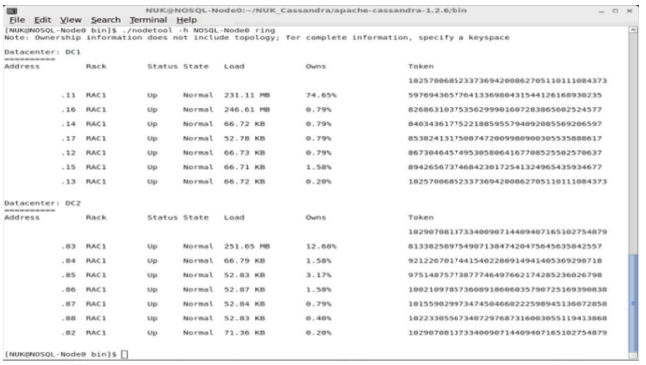

Figure 7: Examine Cassandra's node status.

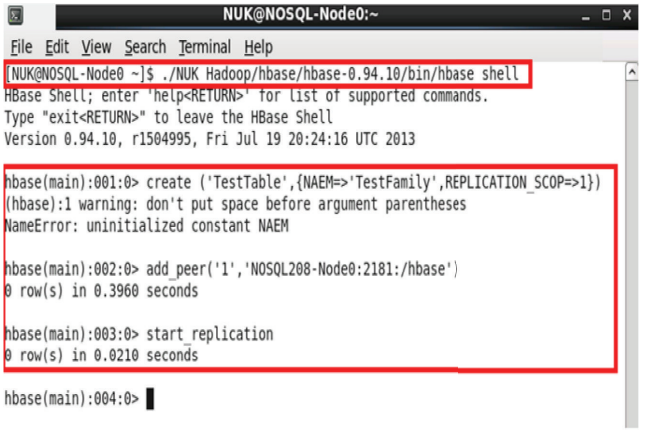

Figure 8: Create forms in HBase.

using CentOS 6.4 system. The following procedures explain how to build Cassandra and HBase data centers and backup mechanisms. Finally, we will develop test tools; the test performances include reading, writing, and remote backup in the data center:

(1) CentOS's firewall is strictly controlled; to use the transfer ports, one must preset the settings as shown in Figure 5.

(2) IT manager sets up HBase and Cassandra data centers and examines the status of all nodes as shown in Figures 6 and 7.

(3) Forms with identical names must be created in both data centers in HBase system. The primary data center will execute command (add_peer) [12], and back up the information onto the secondary data center, as shown in Figures 8 and 9.

(4) IT manager edits Cassandra's file content (cassandra-topology.properties), as shown in Figure 10 and then sets the names of the data center and the storage location of the nodes (rack number).

(5) IT manager edits Cassandra's file content (cassandra.yaml), as shown in Figure 11, and then changes the content of endpoint_snitch [14] to PropertyFileSnitch (data center management mode).

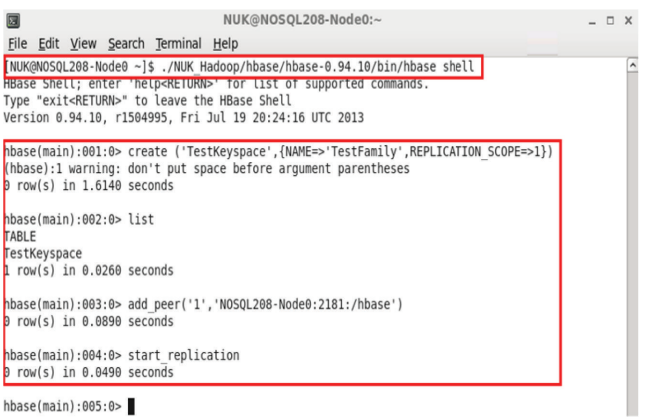

Figure 9: Initialize remote HBase data center backup.

(6) IT manager executes command (create keyspace test with strategy_options = {DC1:2, DC2:1} and placement_strategy = "NetworkTopologyStrategy") in Cassandra's primary data center and then creates a form and initializes remote backup as shown in Figure 12.

(7) IT manager eventually has to test the performance of writing, reading, and offsite data backup against large amounts of information using Thrift Java as shown in Figures 13 and 14.

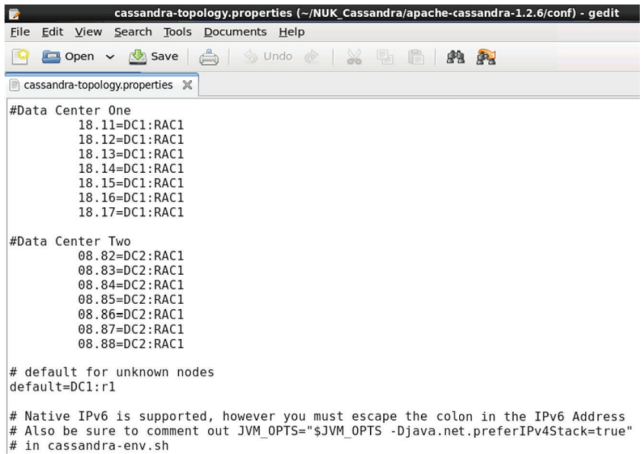

Figure 10: Setup of nodes for the data center.

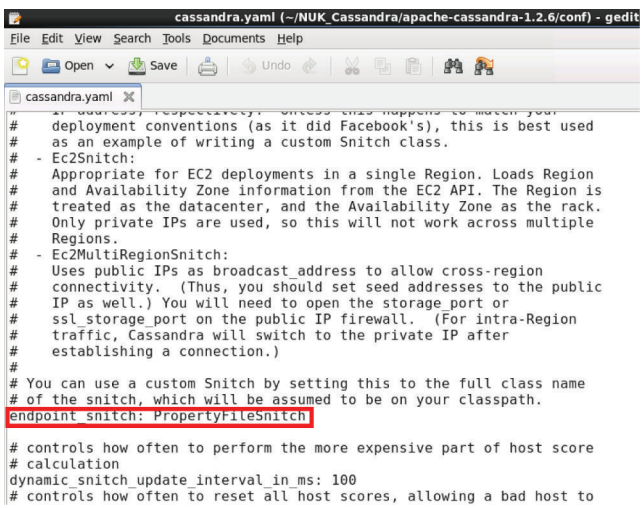

Figure 11: Setup of type for the data center.

As shown in Figure 15, the user will be connected to the database through the Server Login function, select a file folder using Server Information, and then select a data table. Having completed above instructions, the user can operate the database according to the functions described and shown in Figure 15.

4.2. Performance Index. Equation (1) calculates the average access time (AAT) for different data size. In (1), AAT_{ijk} represents the average access time with a specific data size, and N_{ik} represents the current data size:

$$\mathrm{AAT}_{s_{ijk}} = \frac{\mathrm{AAT}_{ijk}}{N_{ik}}, \quad \text{where } i = 1, 2, \ldots, l,\ j = 1, 2, \ldots, m,\ k = 1, 2, \ldots, n. \tag{1}$$

The following three formulae will evaluate the performance index (PI) [1, 2]. Equation (2) calculates the data center's average access times overall $\overline{\mathrm{AAT}_{s_{jk}}}$ (i.e., write, read, and remote

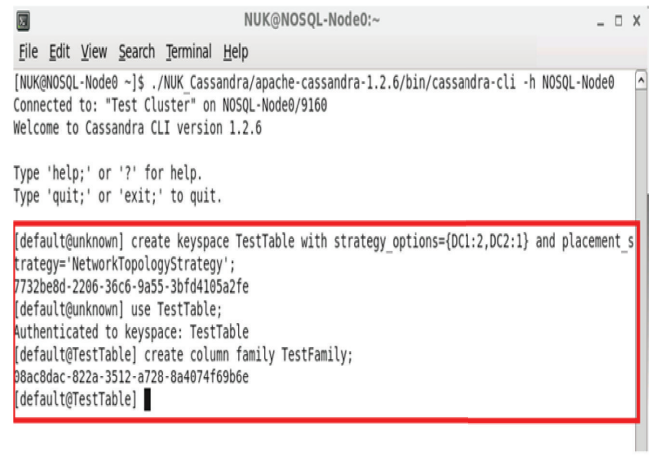

Figure 12: Created forms in Cassandra.

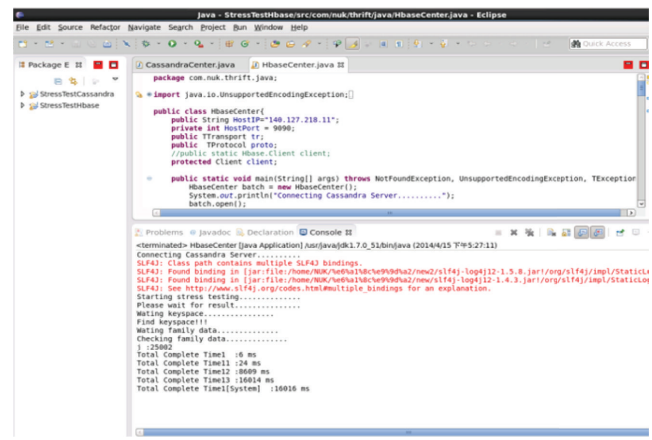

Figure 13: Test of remote HBase data backup.

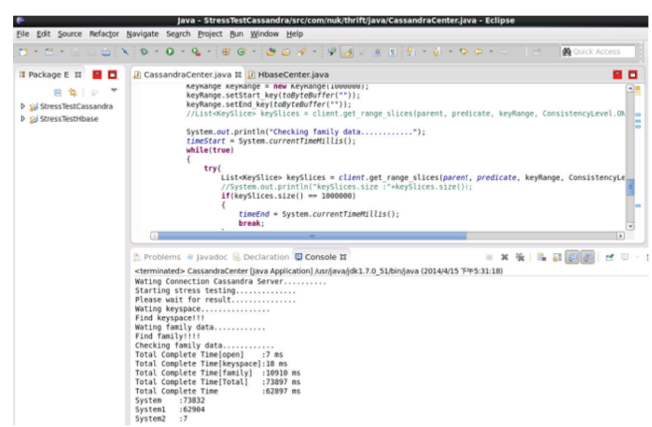

Figure 14: Test of remote Cassandra data backup.

backup), in which $\mathrm{AAT}_{s_{ijk}}$ represents the average access time of each data size; please refer back to (1). Equation (3) calculates the data center's normalized performance index. Equation (4) calculates the data center's performance index overall,

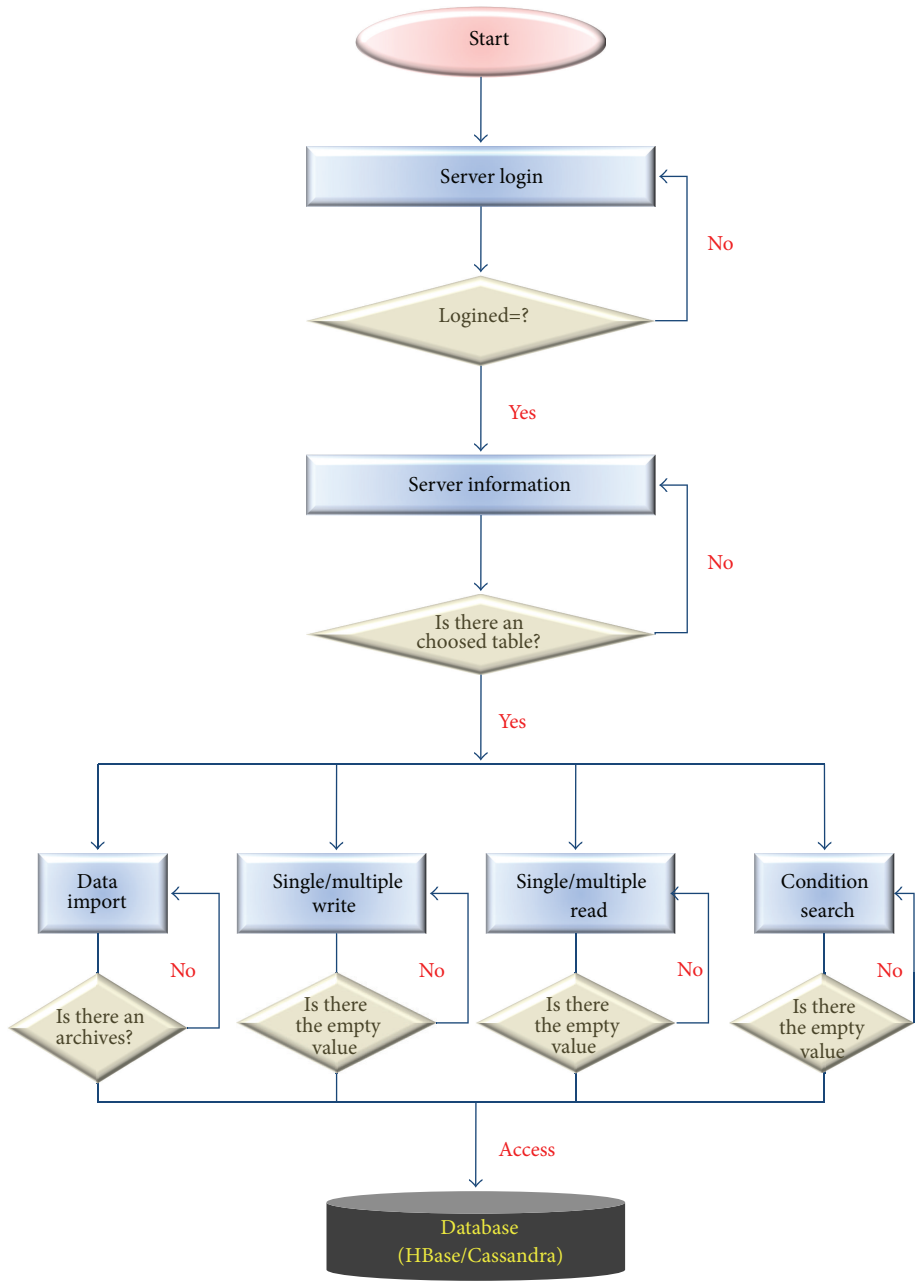

Figure 15: Flowchart of HBase and Cassandra database operation.

SF_1 is constant value and the aim is to quantify the value for observation:

$$\overline{AAT_{s_{jk}}} = \sum_{i=1}^{l} \omega_i \cdot AAT_{s_{ijk}}, \quad \text{where } j = 1, 2, \ldots, m,\ k = 1, 2, \ldots, n,\ \sum_{i=1}^{l} \omega_i = 1, \tag{2}$$

$$\overline{PI_{jk}} = \frac{1/\overline{AAT_{s_{jk}}}}{\text{Max}_{h=1,2,\ldots,m}\left(1/AAT_{s_{hk}}\right)}, \quad \text{where } j = 1, 2, \ldots, m,\ k = 1, 2, \ldots, n, \tag{3}$$

$$PI_j = \left(\sum_{k=1}^{n} W_k \cdot \overline{PI_{jk}}\right) \cdot SF_1, \quad \text{where } j = 1, 2, \ldots, m,\ k = 1, 2, \ldots, n,\ SF_1 = 10^2,\ \sum_{k=1}^{n} W_k = 1. \tag{4}$$

4.3. Total Cost of Ownership. The total cost of ownership (TCO) [1, 2] is divided into four parts: hardware costs, software costs, downtime costs, and operating expenses. The costs of a five-year period Cost_{jg} are calculated using (5) where the subscript j represents various data center and g stands for a certain period of time. Among it, we assume there

Table 1: Hardware specification.

Item	Data center A	Data center B
Server	IBM X3650 ∗ 7	IBM BladeCenter HS22 ∗ 2 IBM BladeCenter HS23 ∗ 5
CPU	Intel Xeon E5-2620 CPU ∗ 2	Intel Xeon 4C E5-2609 CPU ∗ 2
Memory	32 GB	32 GB
Disk	300 GB ∗ 4 (SAS HDD)	300 GB ∗ 2 (SAS HDD)

is an annual unexpected downtime, $\text{Cost}_{\text{downtime for server}_a}$, the monthly expenses $\text{Cost}_{\text{monthly}_b} \cdot \text{period}$, including machine room fees, installation and setup fee, provisional changing fees, and bandwidth costs:

$$\text{Cost}_{jg} = \sum_a \text{Cost}_{\text{downtime for server}_a} + \sum_b \text{Cost}_{\text{monthly}_b} \cdot \text{period} + \sum_c \text{Cost}_{\text{hardware}_c} + \sum_d \text{Cost}_{\text{software}_d}, \quad \text{where } j = 1, 2, \ldots, m,\ g = 1, 2, \ldots, o. \tag{5}$$

4.4. Cost-Performance Ratio. This section defines the cost-performance ratio (C-P ratio) [8], CP_{jg}, of each data center based on total cost of ownership, Cost_{jg}, and performance index, PI_j, as shown in (6). Equation (6) is the formula for C-P ratio where SF_2 is the constant value of scale factor, and the aim is to quantify the C-P ratio within the interval of (0, 100] to observe the differences of each data center:

$$\text{CP}_{jg} = \frac{\text{PI}_j}{\text{Cost}_{jg}} \cdot \text{SF}_2, \quad \text{where } j = 1, 2, \ldots, m,\ g = 1, 2, \ldots, o,\ \text{SF}_2 = 10^4. \tag{6}$$

5. Experimental Results and Discussion

This section will go for the remote data center backup, the stress test, as well as the evaluation of total cost of ownership and performance index among various data centers. Finally, the assessment about the effectiveness and efficiency among various data centers have done well based on cost-performance ratio.

5.1. Hardware and Software Specifications in Data Center. All of tests have performed on IBM X3650 Server and IBM BladeCenter as shown in Table 1. The copyrights of several databases applied in this paper are shown in Table 2, of which Apache HBase and Apache Cassandra are of NoSQL database proposed this paper, but otherwise Cloudera HBase, DataStax Cassandra, and Oracle MySQL are alternative databases.

Table 2: Software copyright.

Database	Copyright
Apache HBase	Free
Apache Cassandra	Free
Cloudera HBase	Free/License
DataStax Cassandra	Free/License
Oracle MySQL	License

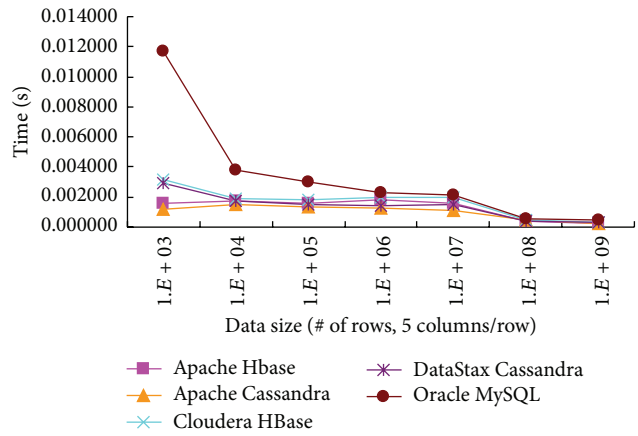

Figure 16: Average time of a single datum write in database.

5.2. Stress Test of Data Read/Write in Data Center. Writing and reading tests of large amounts of information are originating from various database data centers. A total of four varying data sizes were tested, and the average time of a single datum access was calculated for each:

(1) Data centers A and B perform large amounts of information writing test through Thrift Java. Five consecutive writing times among various data centers were recorded for each data size as listed in Table 3. We substitute the results from Table 3 into (1) to calculate the average time of a single datum write for each type of data center as shown in Figure 16.

(2) Data centers A and B perform large amounts of information reading test through Thrift. Five consecutive reading times among various data centers were recorded for each data size as listed in Table 4. We substitute the results from Table 4 into (1) to calculate the average time of a single datum read for each type of data center as shown in Figure 17.

5.3. Stress Test of Remote Data Center Backup. The remote backup testing tool, Thrift Java, is mainly used to find out how long will it take to backup each other's data remotely between data centers A and B as shown in Table 5.

As a matter of fact, tests show that the average time of a single datum access for the remote backup of Apache HBase and Apache Cassandra only takes a fraction of minisecond. Further investigations found that although the two data centers are located in different network domains, they still belonged to the same campus network. The information

Table 3: Data write test (unit: sec.).

Data size	Apache HBase	Apache Cassandra	Cloudera HBase	DataStax Cassandra	Oracle MySQL
10^3	1.6	1.2	3.2	2.9	11.7
10^4	17.1	14.9	18.9	17.4	37.8
10^5	158.5	137.8	178.4	148.3	297.2
10^6	1798.4	1277.8	1942.8	1438.4	2318.7
10^7	15983.1	11437.7	20114.2	14983.7	21291.7
10^8	41753.8	49238.3	44277.9	42829.4	53872.6
10^9	267832.2	241911.8	354219.2	336949.1	508727.6

Table 4: Data read test (unit: sec.).

Data size	Apache HBase	Apache Cassandra	Cloudera HBase	DataStax Cassandra	Oracle MySQL
10^3	1.5	1.9	3.4	4.1	11.9
10^4	4.6	5.2	9.8	10.7	67.2
10^5	44.6	64.1	55.8	70.5	378.5
10^6	604.9	658.8	694.7	732.8	672.8
10^7	5981.3	6317.1	6759.4	7189.6	7916.3
10^8	42398.1	43381.6	45792.9	46990.4	51481.1
10^9	319627.7	326960.4	344192.1	358910.7	509751.9

Table 5: Remote backup test (unit: sec.).

Data size	Apache HBase	Apache Cassandra	Cloudera HBase	DataStax Cassandra	Oracle MySQL
10^3	0.8	1.3	2.2	2.9	10.1
10^4	1.1	2	3.1	3.9	21.9
10^5	5.4	16.3	9.1	20.7	181.1
10^6	18.4	108.6	25.9	137.7	1079.7
10^7	191.3	1081.6	273.1	1281.9	4381.8
10^8	2307.3	2979.1	3209.6	3419.1	7319.1
10^9	24468.3	27953.1	29013.8	29567.3	39819.3

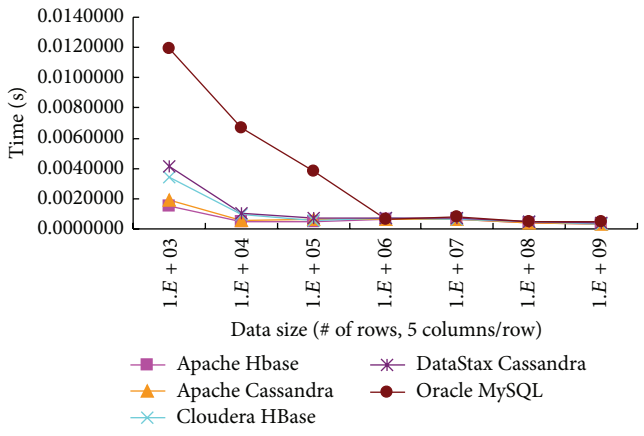

Figure 17: Average time of a single datum read in database.

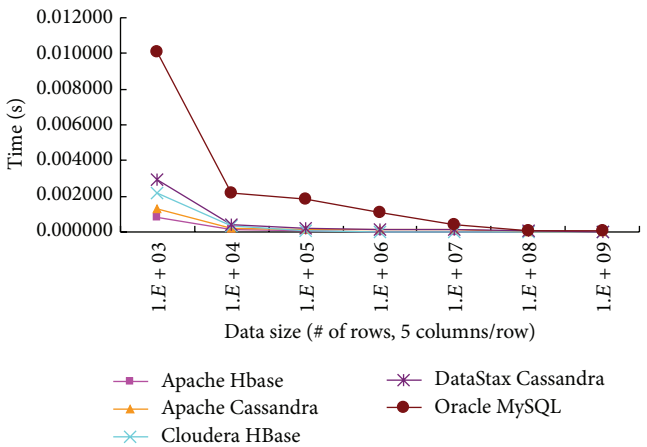

Figure 18: Average time of a single datum backup in the remote data center.

might have only passed through the campus network internally but never reaches the internet outside, leading to speedy the remote backup. Nonetheless, we do not need to set up new data centers elsewhere to conduct more detailed tests because we believe that information exchange through internet will get the almost same results just like performing the remote backup tests via intranet in campus. Five consecutive backup times among various data centers were recorded for each data size as listed in Table 5. We substitute the results from Table 5 into (1) to calculate the average time of a single datum backup for each type of data center as shown in Figure 18.

Table 6: Normalized performance index.

Function	Apache HBase	Apache Cassandra	Cloudera HBase	DataStax Cassandra	Oracle MySQL
Write	0.805	1	0.621	0.735	0.302
Read	1	0.851	0.612	0.534	0.175
Remote backup	1	0.541	0.386	0.274	0.067

Table 7: Average normalized performance index.

Database	Total average
Apache HBase	0.935
Apache Cassandra	0.798
Cloudera HBase	0.540
DataStax Cassandra	0.514
Oracle MySQL	0.181

Table 8: Performance index.

Database	Performance index
Apache HBase	93
Apache Cassandra	80
Cloudera HBase	54
DataStax Cassandra	51
Oracle MySQL	18

Table 9: Total cost of ownership over a 5-year period (unit: USD).

Database	1st year	2nd year	3rd year	4th year	5th year
Apache HBase	12686	10020	10020	10097	10171
Apache Cassandra	12686	10020	10020	10097	10171
Cloudera HBase	14240	12373	12373	12050	12109
DataStax Cassandra	14190	12727	12727	12209	12313
Oracle MySQL	15030	13373	13373	13450	13524

5.4. Evaluation of Performance Index. The following subsection will evaluate the performance index. We first substitute in the average execution times from Figures 16, 17, and 18 into (2) to find the normalized performance index of data centers for each test as listed in Table 6.

Next we substitute the numbers from Table 6 into (3) to find average normalized performance index as listed in Table 7. Finally, we substitute the numbers from Table 7 into (4) to find the performance index of data centers as listed in Table 8.

5.5. Evaluation of Total Cost of Ownership. The total cost of ownership (TCO) includes hardware costs, software costs, downtime costs, and operating expenses. TCO over a five-year period is calculated using (5) and has listed in Table 9. We estimate an annual unexpected downtime costing around USD$1000; the monthly expenses includes around USD$200 machine room fees, installation and setup fee of around USD$200/time, provisional changing fees of around USD$10/time, and bandwidth costs.

Table 10: C-P ratio over a 5-year period.

Database	1st year	2nd year	3rd year	4th year	5th year
Apache HBase	74	93	93	93	92
Apache Cassandra	63	80	80	79	78
Cloudera HBase	38	44	44	45	45
DataStax Cassandra	36	40	40	42	42
Oracle MySQL	12	14	14	13	13

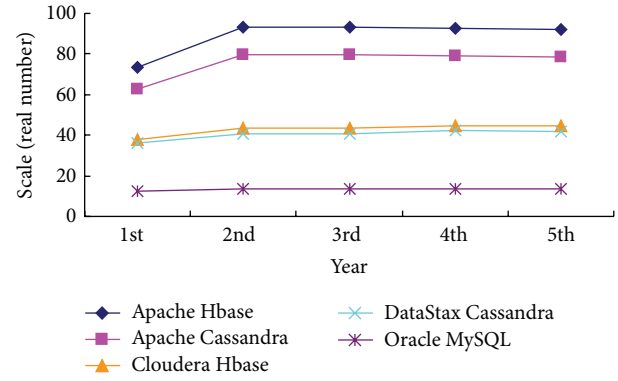

Figure 19: Cost-performance ratio for each data center.

5.6. Assessment of Cost-Performance Ratio. In (6), the formula assesses the cost-performance ratio, CP_{jg}, of each data center according to total cost of ownership, $Cost_{jg}$, and performance index, PI_j. Therefore, we substitute the numbers from Tables 8 and 9 into (6) to find the cost-performance ratio of each data center as listed in Table 10 and shown in Figure 19.

5.7. Discussion. In Figure 19, we have found that Apache HBase and Apache Cassandra obtain higher C-P ratios, whereas MySQL get the lowest one. MySQL adopts the two-dimensional array storage structure and thus each row can have multiple columns. The test data used in this paper is that considering each rowkey it has five column values, and hence MySQL will need to execute five more writing operations for each data query. In contrast, Apache HBase and Apache Cassandra adopting a single Key-Value pattern in storage, the five column values can be written into database currently, namely, no matter how many number of column values for each rowkey, only one write operation required. Figures 16 and 17 show that when comparing with other databases, MySQL consume more time; similarly, as shown in Figure 18, MySQL consumes more time in the remote backup as well. To conclude from the above results, NoSQL database has

gained better performance when facing massive information processing.

6. Conclusion

According to the experimental results of remote datacenter backup, this paper has successfully realized the remote HBase and/or Cassandra datacenter backup. In addition the effectiveness-cost evaluation using C-P ratio is employed to assess the effectiveness and efficiency of remote datacenter backup, and the assessment among various datacenters has been completed over a five-year period. As a result, both HBase and Cassandra yield the best C-P ratio when comparing with the alternatives, provided that our proposed approach indeed gives us an insight into the assessment of remote datacenter backup.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This work is supported by the Ministry of Science and Technology, Taiwan, under Grant no. MOST 103-2221-E-390-011.

References

[1] B. R. Chang, H.-F. Tsai, C.-M. Chen, and C.-F. Huang, "Analysis of virtualized cloud server together with shared storage and estimation of consolidation ratio and TCO/ROI," *Engineering Computations*, vol. 31, no. 8, pp. 1746–1760, 2014.

[2] B. R. Chang, H.-F. Tsai, and C.-M. Chen, "High-performed virtualization services for in-cloud enterprise resource planning system," *Journal of Information Hiding and Multimedia Signal Processing*, vol. 5, no. 4, pp. 614–624, 2014.

[3] B. R. Chang, C.-M. Chen, C.-F. Huang, and H.-F. Tsai, "Intelligent adaptation for UEC video/voice over IP with access control," *International Journal of Intelligent Information and Database Systems*, vol. 8, no. 1, pp. 64–80, 2014.

[4] C.-Y. Chen, B. R. Chang, and P.-S. Huang, "Multimedia augmented reality information system for museum guidance," *Personal and Ubiquitous Computing*, vol. 18, no. 2, pp. 315–322, 2014.

[5] D. Carstoiu, E. Lepadatu, and M. Gaspar, "Hbase-non SQL database, performances evaluation," *International Journal of Advanced Computer Technology*, vol. 2, no. 5, pp. 42–52, 2010.

[6] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[7] D. Ghosh, "Multiparadigm data storage for enterprise applications," *IEEE Software*, vol. 27, no. 5, pp. 57–60, 2010.

[8] B. R. Chang, H.-F. Tsai, J.-C. Cheng, and Y.-C. Tsai, "High availability and high scalability to in-cloud enterprise resource planning system," in *Intelligent Data Analysis and Its Applications, Volume II*, vol. 298 of *Advances in Intelligent Systems and Computing*, pp. 3–13, Springer, Cham, Switzerland, 2014.

[9] J. Pokorny, "NoSQL databases: a step to database scalability in web environment," *International Journal of Web Information Systems*, vol. 9, no. 1, pp. 69–82, 2013.

[10] A. Giersch, Y. Robert, and F. Vivien, "Scheduling tasks sharing files on heterogeneous master-slave platforms," *Journal of Systems Architecture*, vol. 52, no. 2, pp. 88–104, 2006.

[11] A. J. Chakravarti, G. Baumgartner, and M. Lauria, "The organic grid: Self-organizing computation on a peer-to-peer network," *IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans*, vol. 35, no. 3, pp. 373–384, 2005.

[12] L. George, *HBase: The Definitive Guide*, O'Reilly Media, Sebastopol, Calif, USA, 2011.

[13] E. Okorafor and M. K. Patrick, "Availability of Jobtracker machine in Hadoop/MapReduce zookeeper coordinated clusters," *Advanced Computing*, vol. 3, no. 3, pp. 19–30, 2012.

[14] V. Parthasarathy, *Learning Cassandra for Administrators*, Packt Publishing, Birmingham, UK, 2013.

[15] Y. Gu and R. L. Grossman, "Sector: a high performance wide area community data storage and sharing system," *Future Generation Computer Systems*, vol. 26, no. 5, pp. 720–728, 2010.

[16] M. Slee, A. Agarwal, and M. Kwiatkowski, "Thrift: scalable cross-language services implementation," Facebook White Paper 5, 2007.

[17] J. J. Maver and P. Cappy, *Essential Facebook Development: Build Successful Applications for the Facebook Platform*, Addison-Wesley, Boston, Mass, USA, 2009.

[18] R. Murthy and R. Goel, "Peregrine: low-latency queries on Hive warehouse data," *XRDS: Crossroads, The ACM Magazine for Students—Big Data*, vol. 19, no. 1, pp. 40–43, 2012.

[19] A. C. Ramo, R. G. Diaz, and A. Tsaregorodtsev, "DIRAC RESTful API," *Journal of Physics: Conference Series*, vol. 396, no. 5, Article ID 052019, 2012.