Journal of Information Hiding and Multimedia Signal Processing ©2016 ISSN 2073-4212
Ubiquitous International Volume **7**, Number **5**, September **2016**

High Performance Remote Cloud Datacenter Backup in Big Data Environment

Bao-Rong Chang
Department of Computer Science and Information Engineering
National University of Kaohsiung
700 University Road, Kaohsiung, 811, Taiwan
brchang@nuk.edu.tw

Hsiu-Fen Tsai
Department of Marketing Management
Shu-Te University
59 Hun Shang Road, Kaohsiung, 824, Taiwan
soenfen@mail.stu.edu.tw

Cin-Long Guo
Department of Computer Science and Information Engineering
National University of Kaohsiung
700 University Road, Kaohsiung, 811, Taiwan
brandon770913@gmail.com

Abstract. *HBase and Cassandra are two most commonly used large-scale distributed NoSQL database management systems; especially applicable to 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 effective remote cloud datacenter backup using Thrift Java on NoSQL database HBase and Cassandra. The binary communications protocol technology from Apache Thrift is employed to establish graphical user interface instead of using command line interface so as to ease data manipulation. In order to verify high performance, a stress test has taken on strictly data reading/writing and remote backup of a large amount of data. Finally, performance index has been evaluated for several benchmark databases including two above-mentioned databases. As a result, the proposed HBase approach outperforms the other databases.*
Keywords: NoSQL Database, Hbase, Cassandra, Remote Datacenter Backup, Performance Index, Graphical User Interface.

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 above reasons, data of all types and forms are constantly being produced, leaving the mass of un-correlated or un-related information, causing conventional databases is not able to handle the workload in a big data environment. This leads to the emergence of non-relational databases, of which many notable NoSQL databases that are currently being used by enterprises are HBase [5], Cassandra [6], and Mongo [7]. 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 be happened in such accidents as power shut-down, natural disaster, or manual error that leads to whole system collapse and then initiates an remote backup to the remote data center. Even though NoSQL database uses distributed architecture to prevent the risk of data loss, but it has neglected the importance of data center remote 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 remarkable NoSQL databases, and perform stress tests with a large scale of data, for instances, read, write, and data center remote backup through graphical user interface we proposed instead of command line interface. The experimental results of data center remote backup using database HBase and Cassandra will show normalized performance index, average normalized performance index, and performance index to indicate the performance evaluation [8].

2. **Large-scale database in data center.**

This paper will realize data center remote 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 [10] framework, where the master node assigns data to the slave node to realize the distributed data storage, meanwhile emphasizing on consistency and partition tolerance characteristics. Regarding data center remote backup, a certain data center with database HBase has the following advantages: (1) retain data consistency, (2) activate instant reading or writing of massive information, (3) access to 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 data to realize the distributed data storage with uninterrupted services, at the same time emphasizing on availability and partition tolerance characteristics. As for data center remote backup, a certain data center with database Cassandra has the following advantages: (1) each node shares equal information, (2) cluster setup is quick and simple, (3) 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 Fig. 1. The master cluster and slave cluster must possess its 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 data center remote backup between different sites. Remote Cassandra data center backup architecture [14] is as shown in Fig. 2. Cassandra is of peer-to-peer (P2P) framework connects all nodes together. When data have written into data center A, a copy of the data is immediately backed up into a designated data center B, as well, each node can designate a permanent storage location in a rack [15]. 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 data have distributed and written into these nodes by master node or seed node to achieve data center remote backup.

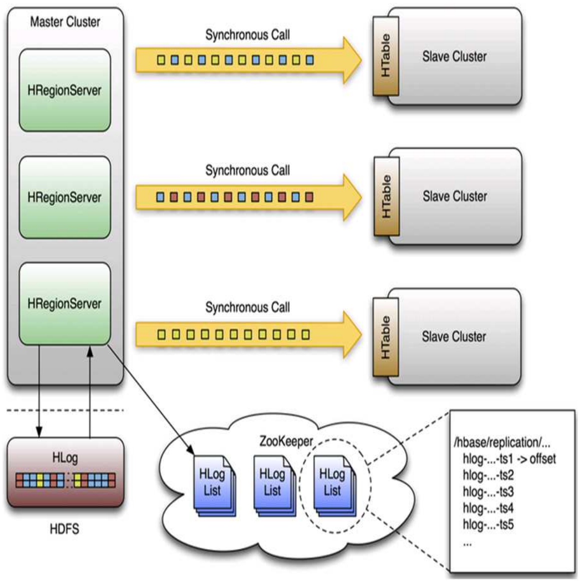


FIGURE 1. Remote HBase data center backup

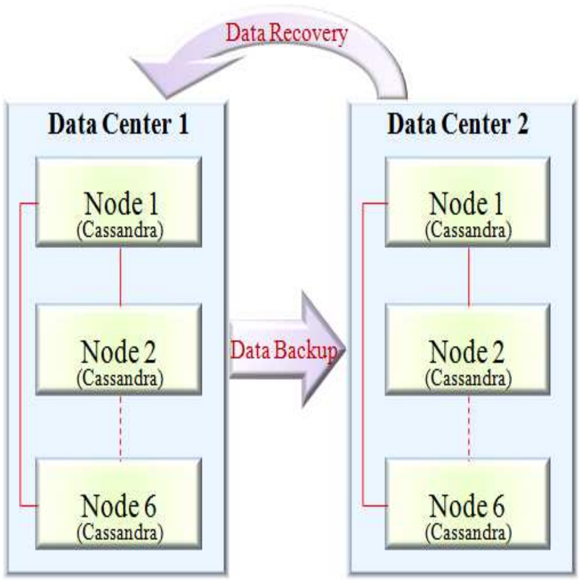


FIGURE 2. Remote Cassandra data center backup

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 to solve Facebooks 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 Fig. 3. In Fig. 3 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 Thrifts transfer framework, protocols, and underlying I/O protocols.

Using Thrift, we can conveniently define a multi-language service system, and select different transfer protocol. The Server side includes the transfer protocol and the basic transfer framework, providing both single and multi-thread operation modes on the Server, where the Server and browser are capable of interoperability concurrently. The use of Thrift to achieve cross-platform data transfer for either HBase or Cassandra in data center remote backup works in our proposed approach and a system diagram has shown below in Fig. 4 as well.

4. System implementation.

This section will realize data center HBase and Cassandra, and implement the remote backup of data center as well. After that, the efficiency of the remote backup has done according to performance index.

4.1. **Data transfer, data integrity checking, and graphical user interface.** The following algorithm will implement data centers using database HBase and Cassandra underlying the operating system CentOS 6.4, and achieve the goal of remote backup.

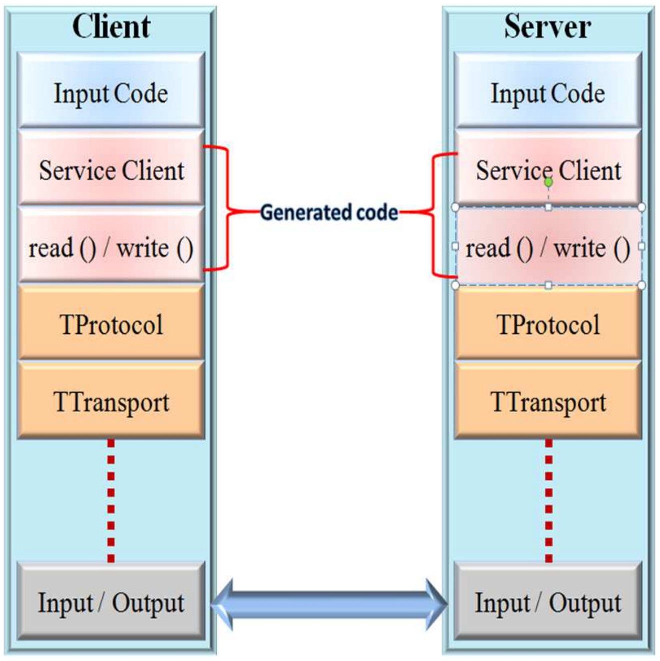


Figure 3. Apache Thrift architecture.

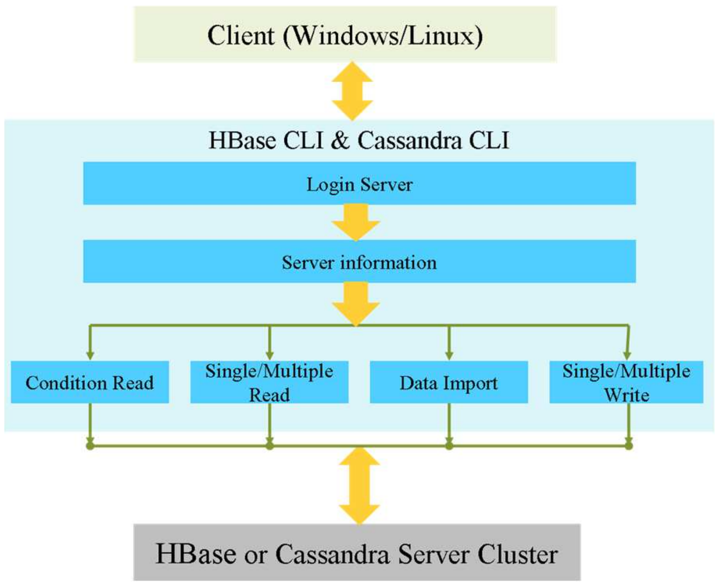
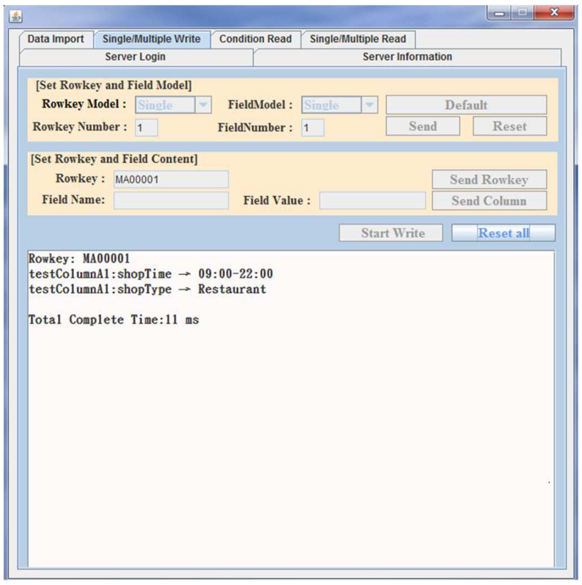


Figure 4. Structure of HBase or Cassandra in data center remote backup

Next, this paper will test the efficiency of data centers against reading, writing and remote backup of large amounts of data.
Algorithm: Datacenter setup and remote backup

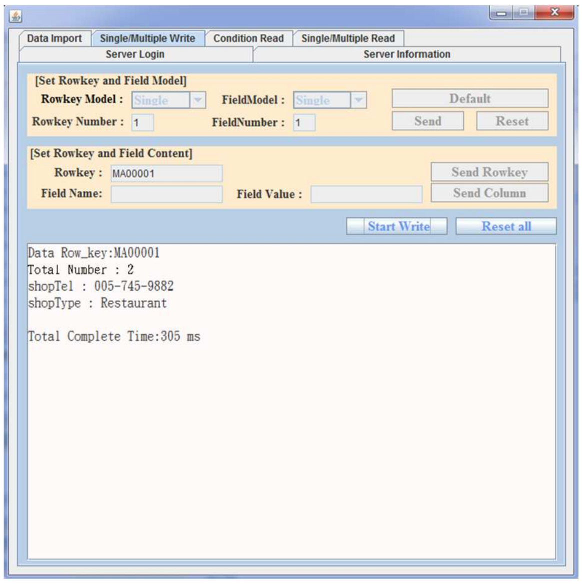
Input: CentOS, Hadoop, Cassandra, Testing data with the size of 1.5 Terabyte
Output: Remote backup of HBase and Cassandra in two datacenters
Step 1. control CentOS's firewall to use the transfer ports and pre-set the settings.
Step 2. set up HBase and Cassandra data centers and examine the status of all nodes.
Step 3. create forms with identical names in HBase system for both data centers. The primary data center will execute command (add_peer) [12], and backup the information onto the secondary data center.



```
Rowkey Model : Single    FieldModel : Single
Rowkey Number : 1        FieldNumber : 1
Rowkey : MA00001
Field Name:              Field Value :
```

```
Rowkey: MA00001
testColumnA1:shopTime → 09:00-22:00
testColumnA1:shopType → Restaurant

Total Complete Time:11 ms
```

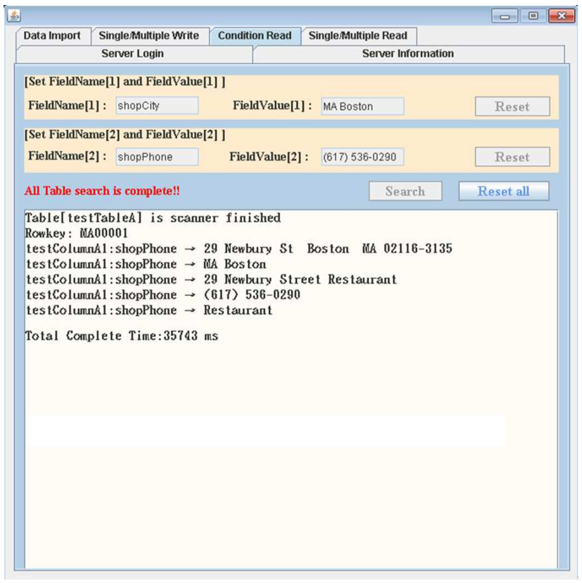
Figure 5. Writing of Data shopTime: 09:00-22:00 into Rowkey: MA00001 (HBaseX)



```
Rowkey Model : Single    FieldModel : Single
Rowkey Number : 1        FieldNumber : 1
Rowkey : MA00001
Field Name:              Field Value :
```

```
Data Row_key:MA00001
Total Number : 2
shopTel : 005-745-9882
shopType : Restaurant

Total Complete Time:305 ms
```

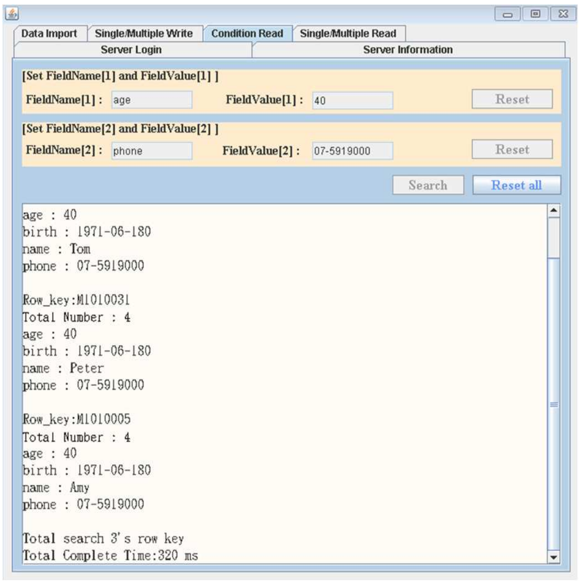
Figure 6. Writing of Data shopTel: 005-754-9982 into Rowkey: MA00001 (CanssandraX)



```
FieldName[1] : shopCity      FieldValue[1] : MA Boston
FieldName[2] : shopPhone     FieldValue[2] : (617) 536-0290
All Table search is complete!!
```

```
Table[testTableA] is scanner finished
Rowkey: MA00001
testColumnA1:shopPhone → 29 Newbury St  Boston  MA 02116-3135
testColumnA1:shopPhone → MA Boston
testColumnA1:shopPhone → 29 Newbury Street Restaurant
testColumnA1:shopPhone → (617) 536-0290
testColumnA1:shopPhone → Restaurant

Total Complete Time:35743 ms
```

Figure 7. Search shopCity: MA Boston and shopPhone: (617) 536-0290 (HBaseX)



```
FieldName[1] : age       FieldValue[1] : 40
FieldName[2] : phone     FieldValue[2] : 07-5919000
```

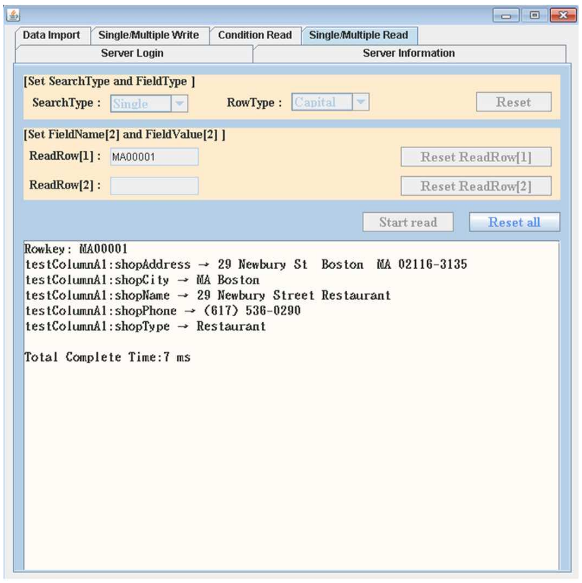
```
age : 40
birth : 1971-06-180
name : Tom
phone : 07-5919000

Row_key:M1010031
Total Number : 4
age : 40
birth : 1971-06-180
name : Peter
phone : 07-5919000

Row_key:M1010005
Total Number : 4
age : 40
birth : 1971-06-180
name : Amy
phone : 07-5919000

Total search 3's row key
Total Complete Time:320 ms
```

Figure 8. Search age: 40 and phone: 07-5919000 (CassandraX)

Step 4. edits Cassandra's file content (cassandra-topology.properties), then sets the names of data center and the storage location of the nodes (i.e., rack number).

Step 5. edit Cassandra's file content (cassandra.yaml), and then change the content of endpoint_snitch [14] to PropertyFileSnitch (data center management mode).

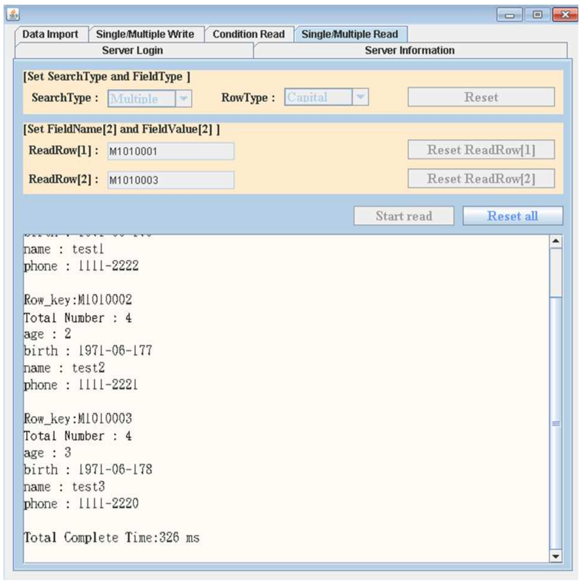
Step 6. execute command (create keyspace test with strategy_options = {DC1:2,DC2:1} and placement_strategy='NetworkTopologyStrategy') in Cassandra's primary data center, and then creates a form and initialize the remote backup.



[Set SearchType and FieldType]
SearchType : Single
RowType : Capital
[Set FieldName[2] and FieldValue[2]]
ReadRow[1] : MA00001
ReadRow[2] :

```
Rowkey: MA00001
testColumnA1:shopAddress → 29 Newbury St  Boston  MA 02116-3135
testColumnA1:shopCity → MA Boston
testColumnA1:shopName → 29 Newbury Street Restaurant
testColumnA1:shopPhone → (617) 536-0290
testColumnA1:shopType → Restaurant

Total Complete Time:7 ms
```

Figure 9. Read Single Data Rowkey: MA00001 (HBaseX)



[Set SearchType and FieldType]
SearchType : Multiple
RowType : Capital
[Set FieldName[2] and FieldValue[2]]
ReadRow[1] : M1010001
ReadRow[2] : M1010003

```
name : test1
phone : 1111-2222

Row_key:M1010002
Total Number : 4
age : 2
birth : 1971-06-177
name : test2
phone : 1111-2221

Row_key:M1010003
Total Number : 4
age : 3
birth : 1971-06-178
name : test3
phone : 1111-2220

Total Complete Time:326 ms
```

Figure 10. Read Multiple Data Rowkey M10100001 to M1010003 (CassandraX)

Step 7. test writing, reading, and offsite data backup against large amounts of data using Thrift Java, and check their performance.

The binary communications protocol technology from Apache Thrift is employed to establish graphical user interface instead of using command line interface. As shown in Figs. 5, 6, 7, 8, 9, and 10, the graphical user interface provides functions of importing of external files, and writing, reading, and secondary indexing of mass data which is similar to that of Solandra (Solr+Cassandra). But it requires constant updates to Cassandra database and Solr [20] search engine; in addition, identical query result to that of Cassandra's cannot be guaranteed for each and every reading or secondary index. The graphical user interface in this paper ensures that the data accessed each time we are connected to the database is consistent, and that we are able to connect to different data nodes without access failure.

4.2. **Performance index.** In order to develop the assessment of the proposed approach, the necessitated equations are derived from first measuring a single datum access time for a certain database on Eq. 1, next calculating average access time based on a variety of data size on Eq. 2, then inducing a normalized performance index among the databases on Eq. 3, and finally resulting in a performance index according to different tests on Eq. 4. In these equations we denote the subscript i the index of data size, j the index of database, and k the index of test category as well as the subscript s indicates a single datum. Eq. 1 calculates the average access time (AAT) for each data size. In Eq. 1 AAT_{ijk} represents average access time with the same data size, and N_{ik} represents the current data size. Eq. 2 calculates the data center's average access times overall $\overline{AAT_{S_{jk}}}$ for each test (i.e. write, read, remote backup), in which $AAT_{S_{ijk}}$ represents the average access time of each data size, please refer back to Eq. 1. The following formula will evaluate the performance index (PI) [8]. Eq. 3 calculates the data center's normalized performance index. Eq. 4 calculates the data center's performance index overall, SF_l is constant value and the aim is to quantify the value for observation.

$$AAT_{S_{ijk}} = \frac{AAT_{ijk}}{N_{ik}}, where\ i = 1, 2, ..., l, j = 1, 2, ..., m, k = 1, 2, ..., n \tag{1}$$

$$\overline{AAT_{S_{jk}}} = \sum_{i=1}^{l} w_i \cdot AAT_{S_{ijk}}, where\ j = 1, 2, ..., m, k = 1, 2, ..., n, \sum_{i=1}^{l} w_i = 1 \tag{2}$$

$$\overline{PI_{jk}} = \frac{\frac{1}{\overline{AAT_{S_{jk}}}}}{\max\limits_{h=1,2,...,m}\left(\frac{1}{\overline{AAT_{S_{hk}}}}\right)}, where\ j = 1, 2, ..., m, k = 1, 2, ..., n \tag{3}$$

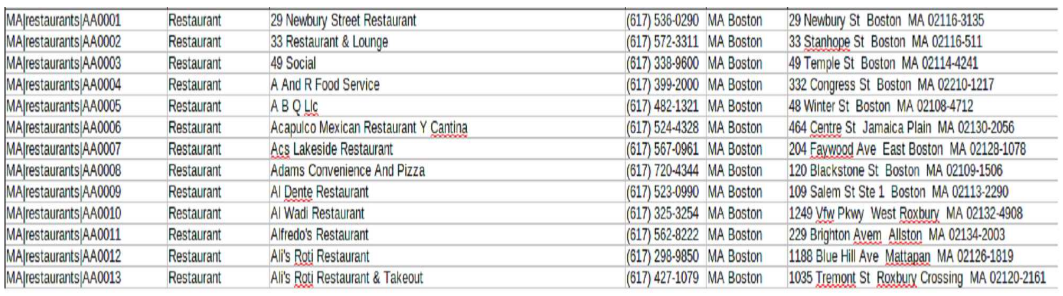
$$PI_j = \left(\sum_{k=1}^{n} W_k \cdot \overline{PI_{jk}}\right) \cdot SF_l, where\ j = 1, 2, ..., m, k = 1, 2, ..., n, SF_l = 10^2, \sum_{k=1}^{n} W_k = 1 \tag{4}$$

5. Experimental Results and discussion.

This section will go for the data center remote backup, the stress test, as well as the performance index among various data centers. Finally, the performance evaluation for a variety of data bases has carried out according to performance index.

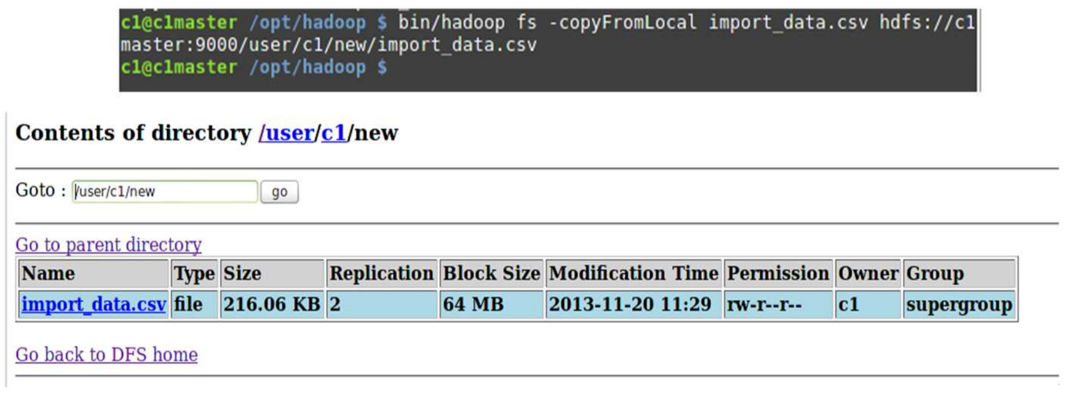
5.1. Stress test of data read/write in data center.

All of tests have performed on IBM X3650 Server (denoted data center A) and IBM BladeCenter (denoted data center B). Data centers A and B have been installed a variety of databases, that is, the latest version of Apache HBase (hbase-0.94.27), Apache Cassandra (the latest stable release 2.1.6), Cloudera HBase, DataStax Cassandra, and Oracle MySQL. We are going to launch a series of the tests to measure their system performance and efficiency. The first four databases are of remarkable NoSQL type database employed to the tests, and the last one is a well-known SQL type databases. Writing and reading tests of large amounts of data are launched to/from these databases. A number of various specific data sizes were applied, and the average time of a single datum access was logged for each data size. Hadoop provides a file system where in Fig. 11 we import prepared data, and in Fig. 12 we upload this raw data to HDFS. We need to create a definite row-key in raw data because of no providing auto-generating row-key in HBase. HBase didnt provide an auto-generated row-key format for any input. Fig. 11 presents our prepared data format, which it construct with row-key, category, shop-name, telephone, province and address. The source of these data is grabbed from the yellow page in U.S.A. We separate raw datas fields by comma qualifier.



MA|restaurants|AA0001	Restaurant	29 Newbury Street Restaurant	(617) 536-0290	MA Boston	29 Newbury St Boston MA 02116-3135
MA|restaurants|AA0002	Restaurant	33 Restaurant & Lounge	(617) 572-3311	MA Boston	33 Stanhope St Boston MA 02116-511
MA|restaurants|AA0003	Restaurant	49 Social	(617) 338-9600	MA Boston	49 Temple St Boston MA 02114-4241
MA|restaurants|AA0004	Restaurant	A And R Food Service	(617) 399-2000	MA Boston	332 Congress St Boston MA 02210-1217
MA|restaurants|AA0005	Restaurant	A B Q Llc	(617) 482-1321	MA Boston	48 Winter St Boston MA 02108-4712
MA|restaurants|AA0006	Restaurant	Acapulco Mexican Restaurant Y Cantina	(617) 524-4328	MA Boston	464 Centre St Jamaica Plain MA 02130-2056
MA|restaurants|AA0007	Restaurant	Acs Lakeside Restaurant	(617) 567-0961	MA Boston	204 Faywood Ave East Boston MA 02128-1078
MA|restaurants|AA0008	Restaurant	Adams Convenience And Pizza	(617) 720-4344	MA Boston	120 Blackstone St Boston MA 02109-1506
MA|restaurants|AA0009	Restaurant	Al Dente Restaurant	(617) 523-0990	MA Boston	109 Salem St Ste 1 Boston MA 02113-2290
MA|restaurants|AA0010	Restaurant	Al Wadi Restaurant	(617) 325-3254	MA Boston	1249 Vfw Pkwy West Roxbury MA 02132-4908
MA|restaurants|AA0011	Restaurant	Alfredo's Restaurant	(617) 562-8222	MA Boston	229 Brighton Avem Allston MA 02134-2003
MA|restaurants|AA0012	Restaurant	Ali's Roti Restaurant	(617) 298-9850	MA Boston	1188 Blue Hill Ave Mattapan MA 02126-1819
MA|restaurants|AA0013	Restaurant	Ali's Roti Restaurant & Takeout	(617) 427-1079	MA Boston	1035 Tremont St Roxbury Crossing MA 02120-2161

Figure 11. The format of inputting data.



```
c1@c1master /opt/hadoop $ bin/hadoop fs -copyFromLocal import_data.csv hdfs://c1
master:9000/user/c1/new/import_data.csv
c1@c1master /opt/hadoop $
```

Contents of directory /user/c1/new

Goto : /user/c1/new go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
import_data.csv	file	216.06 KB	2	64 MB	2013-11-20 11:29	rw-r--r--	c1	supergroup

Go back to DFS home

Figure 12. Uploading data to HDFS.

(1) Data centers A and B perform the large amounts of information writing test through Thrift Java. In Table 1, five consecutive writing times among various databases were performed where the various specific data sizes applied for writing test. We substitute the results into Eq. 1 to calculate the average time of a single datum write for several databases as shown in Fig. 13.
(2) Data centers A and B perform large amounts of information reading test through Thrift. In Table 2, five consecutive writing times among various databases were performed where the various specific data sizes applied for reading test. We substitute the results into Eq. 1 to calculate the average time of a single datum read for several databases as shown in Fig. 14.

Table 1. 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

(Note: Unit of data size is the number of rows where one row contains five columns)

5.2. **Stress test of data center remote backup.** The remote backup testing tool, Thrift Java, is mainly used to find out how long does it take to backup data each other remotely between data centers A and B as shown in Table 3. As a matter of fact, the experiment shows that the average time of a single datum access for the remote backup of Apache HBase and Apache Cassandra only takes a fraction of mini-second. Further investigations found that although the two data centers are located at different network domains, they still belonged to the same campus network. The information might have only passed through the campus network internally, but they never reach the internet outside, leading to speedy the remote backup. Nonetheless, we do not need to setup new data centers elsewhere to conduct more detailed tests because we believe that data exchange through internet will get the almost same results just like performing the remote

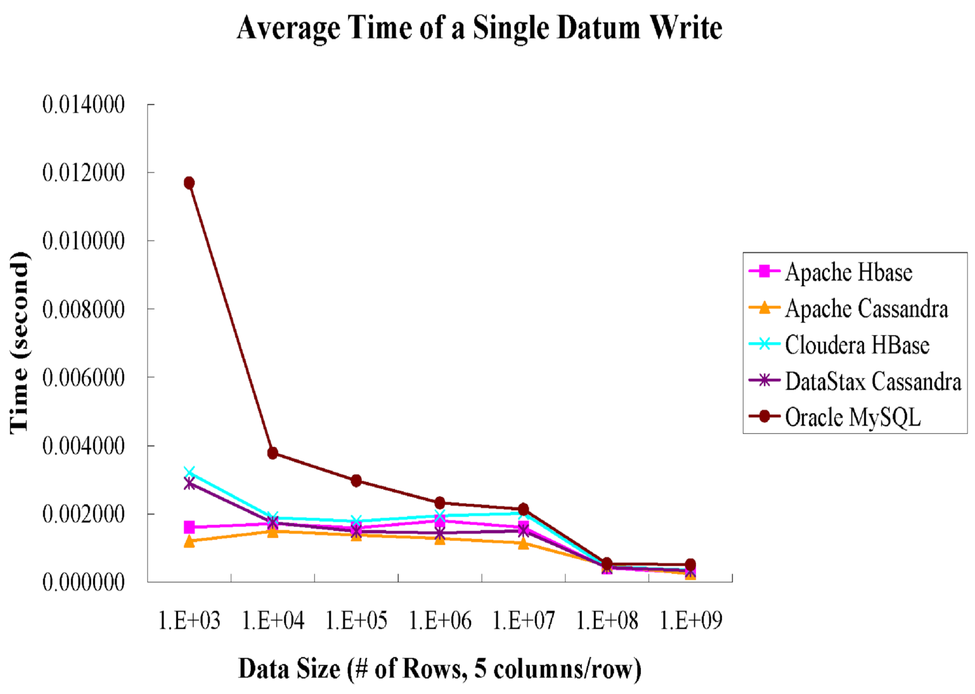


Figure 13. Average time of a single datum write in database.

Table 2. 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

(Note: Unit of data size is the number of rows where one row contains five columns)

backup tests via intranet in campus. Five consecutive backup times among various specific databases between two distinctive datacenters have been logged when applying different data sizes in the test. We substitute the results into Eq. 1 to calculate the average time of a single datum backup for several databases between two distinctive datacenter as shown in Fig. 15.

5.3. **Performance evaluation and discussion.** The following subsection will evaluate the performance index for the above-mentioned databases. We first substitute the average execution times from Tables 1, 2, and 3 into Eq. 2 to find the average access time based on a variety of data size in the test. Next we substitute those results into Eq. 3 to find the normalized performance index as listed in Table 4 and shown in Fig. 16. Finally, we substitute those results into Eq. 4 to find average normalized performance index as listed in Table 5 and the performance index of databases as listed in Table 6. In Table 6, we have found that Apache HBase and Apache Cassandra obtain higher performance index, whereas MySQL get the lowest one. MySQL adopts the two-dimensional array storage

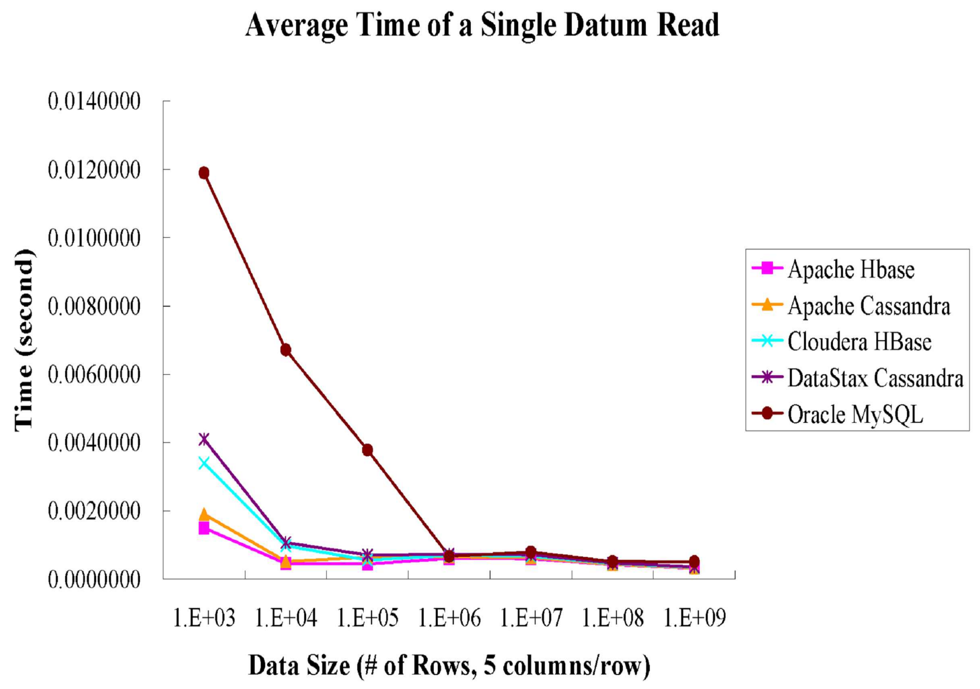


Figure 14. Average time of a single datum read in database.

Table 3. 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

(Note: Unit of data size is the number of rows where one row contains five columns)

structure and thus each row can have multiple columns. The testing 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. Figs. 13 and 14 show that when comparing with other databases, MySQL consume more time; similarly, as shown in Fig. 15, MySQL consumes more time in the remote backup as well. To draw a conclusion from the above results, NoSQL database has gained better performance when facing massive data processing. Particularly, based on C.A.P theory [9], in the remote datacenter backup, Apache HBase outperforms Apache Cassandra a lot because Apache Cassandra need to take a longer time to follow the property of partition tolerance. Conversely, in data writing test, Apache Cassandra is superior to Apache HBase because Apache HBase have to keep the property of availability to take a time-consuming work

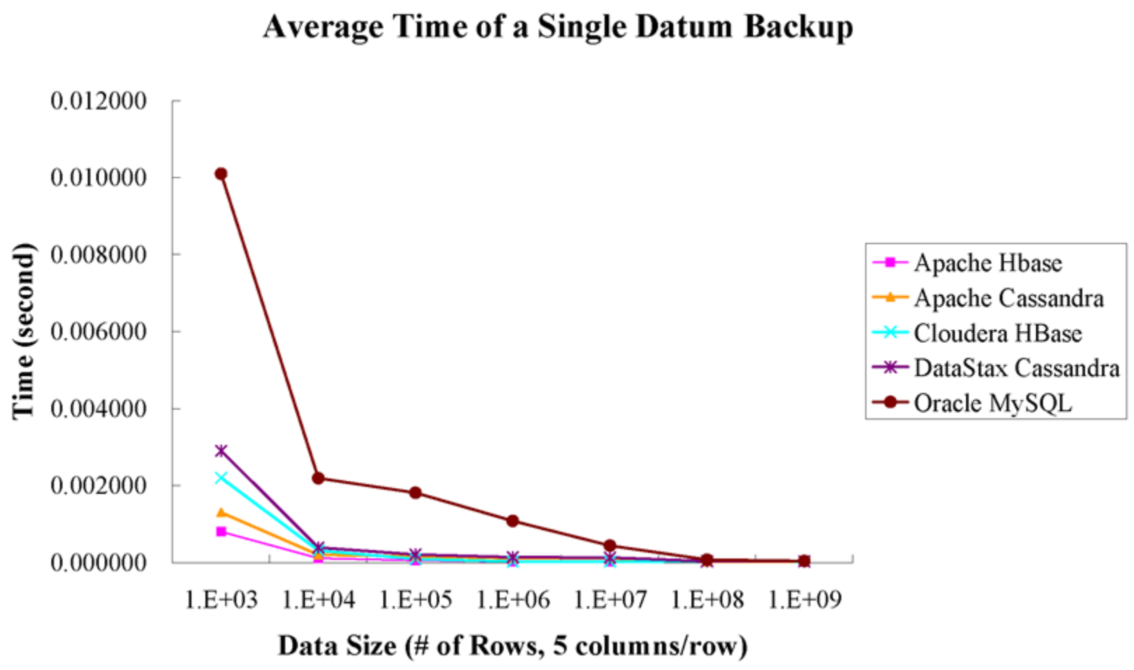


Figure 15. Average time of a single datum backup in the remote data center.

for the data replication between storage segments. The latest version of Apache HBase has performed better than Cloudera HBase so far.

Table 4. Normalized performance index

Operation	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 5. 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

6. **Conclusions.**

This paper realizes the remote data backup for HBase and Cassandra data centers, and the performances evaluation for various databases has been carried out according to the performance index. As a result, HBase and Cassandra can best perform the others, provided that this paper indeed gives us an insight into how to evaluate the performance for the remote datacenter backup.

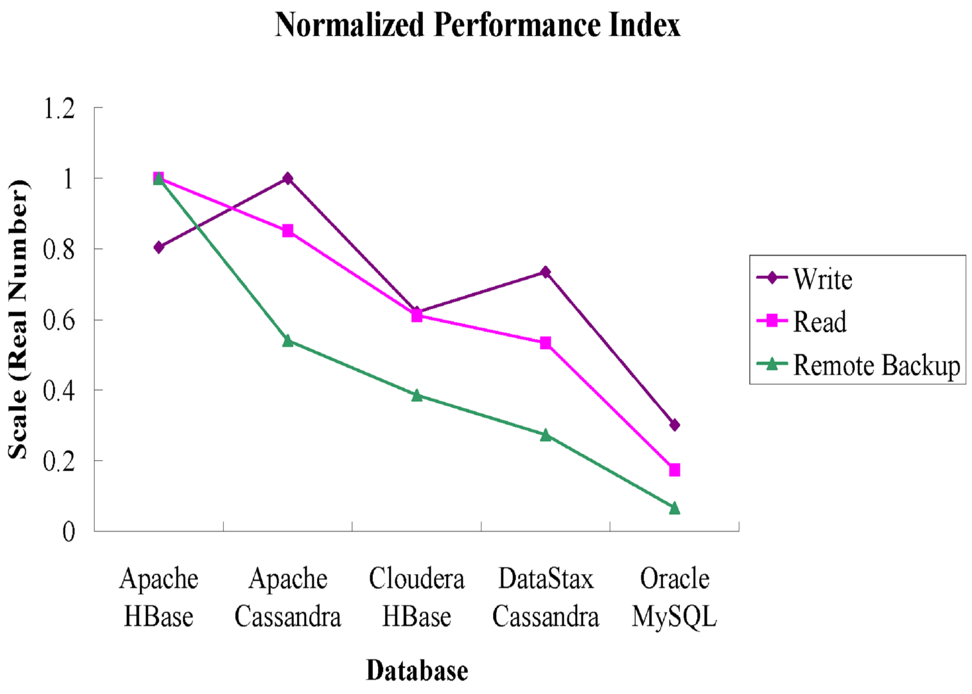


Figure 16. Normalized performance index of various databases.

Table 6. Performance index

DataBase	Performance Index
Apache HBase	93
Apache Cassandra	80
Cloudera HBase	54
DataStax Cassandra	51
Oracle MySQL	18

Acknowledgment. This work is fully supported by the Ministry of Science and Technology, Taiwan, Republic of China, under grant number **MOST 103-2221-E-390-011**.

REFERENCES

[1] B. R. Chang, H. F. Tsai, and C. M. Chen, Empirical analysis of server consolidation and desktop virtualization in cloud computing, *Mathematical Problems in Engineering*, vol.2013, article ID.947234, 11 pages, 2013.

[2] B. R. Chang, H. F. Tsai, C. Y. Chen, and Y. C. Tsai, Assessment of in-cloud enterprise resource planning system performed in a virtual cluster, *Mathematical Problems in Engineering*, vol.2014, article ID.520534, 8 pages, 2014.

[3] B. R. Chang, H. F. Tsai, C. M. Chen, and C. F. Huang, 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] N. O'Higgins, *MongoDB and Python: Patterns and Processes for the Popular Document-Oriented Database*, O'Reilly Media Inc., Sebastopol, CA, USA, 2011.

[8] B. R. Chang, H. F. Tsai, and C. M. Chen, Assessment of in-cloud enterprise resource planning system performed in a virtual cluster, *Mathematical Problems in Engineering*, vol.2014, article ID.947234, 11 pages, 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 masterslave 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 Inc., Sebastopol, CA, USA, 2011.

[13] E. Okorafor, and M. K. Patrick, Availability of jobtracker machine in hadoop/mapreduce zookeeper coordinated clusters, *Advanced Computing: An International Journal*, vol.3, no.3, pp.19–30, 2012.

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

[15] Y. Gu, 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*, vol.5, 8 pages, 2007.

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

[18] R. Murthy, R. Goel, Low-latency queries on hive warehouse data. xrds: crossroads, *The ACM Magazine for Students*, 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, ID.052019, 2012.

[20] R. Kuc, *Apache Solr 4 Cookbook*, Packt Publishing Co, Birmingham, UK, 2013.