Future Generation Computer Systems 65 (2016) 111–121

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Data adapter for querying and transformation between SQL and NoSQL database

Ying-Ti Liao [a], Jiazheng Zhou [a], Chia-Hung Lu [a], Shih-Chang Chen [a], Ching-Hsien Hsu [b,c,*], Wenguang Chen [d], Mon-Fong Jiang [e], Yeh-Ching Chung [a]

[a] Department of Computer Science, National Tsing Hua University, Hsinchu, 30013, Taiwan, ROC
[b] School of Mathematics and Big Data, Foshan University, China
[c] Department of Computer Science and Information Engineering, Chung Hua University, Hsinchu, Taiwan, ROC
[d] Department of Computer Science and Technology, Tsinghua University, Beijing, China
[e] is-land Systems Inc., Hsinchu, 300, Taiwan, ROC

HIGHLIGHTS

- This paper presents data adapter to make possible the automated transformation of multi-structured data in Relational Database (RDB) and NoSQL systems.
- With the proposed data adapter, a seamless mechanism is provided for constructing hybrid database systems.
- With the proposed data adapter, hybrid database systems can be performed in an elastic manner, i.e., access can be either RDB or NoSQL, depending on the size of data.

ARTICLE INFO

Article history:
Received 23 July 2015
Received in revised form
6 February 2016
Accepted 10 February 2016
Available online 10 March 2016

Keywords:
Big data
NoSQL
Data adapter
Hybrid database
Cloud computing
Database services

ABSTRACT

As the growing of applications with big data in cloud computing become popular, many existing systems expect to expand their service to support the explosive increase of data. We propose a *data adapter* system to support hybrid database architecture including a relational database (RDB) and NoSQL database. It can support query from application and deal with database transformation at the same time. We provide three modes of query approach in *data adapter* system: blocking transformation mode (*BT* mode), blocking dump mode (*BD* mode), and direct access mode (*DA* mode). We provide a data synchronization mechanism and describe the design and implementation in detail. This paper focuses on velocity with proposed three modes and partly variety with data stored in RDB, NoSQL database and temporary files. With the proposed *data adapter* system, we can provide a seamless mechanism to use RDB and NoSQL database at the same time.

1. Introduction

BIG data and hybrid database system are becoming popular as cloud service blooms. NoSQL databases are also growing in popularity for big data applications. Most of the existing systems are based on RDB, but with the growth of data size, enterprise tends to handle big data with NoSQL database for analysis or wants to get faster access on big data. Instead of replacing RDB with NoSQL database, enterprises and research organizations integrate the both databases. User applications interact with RDB to handle small and middle scale of data; NoSQL database serves as system back-end data pool for analysis and batched read/write operations, or periodic back-up destinations from RDB.

The database integration may affect the original system design. In the original system, application interacts with relational database using SQL. Since NoSQL database cannot be accessed by SQL, application needs to modify the design to access both RDB and NoSQL database. Mechanism of data transformation from RDB to NoSQL database is needed when integrating the original system with NoSQL database. The transformation process forces application to suspend and to wait for data synchronization. The transformation may take a long time if data is in large scale. It is a critical issue for some real-time, non-stopping service like scientific analysis or online web applications.

* Corresponding author at: School of Mathematics and Big Data, Foshan University, China.
E-mail address: robertchh@gmail.com (C.-H. Hsu).

http://dx.doi.org/10.1016/j.future.2016.02.002
0167-739X/

This paper proposes a *data adapter* system that integrates RDB and NoSQL database, and can handle database transformation. The main features of the *data adapter* are listed as follows.

1. **SQL Interface to RDB and NoSQL Database.** We offer a general SQL interface to access both RDB and NoSQL database. It consists of a SQL query parser and Apache Phoenix [1] as a SQL translator to connect HBase [2] as a NoSQL database, and MySQL JDBC driver as a RDB connector. With this SQL interface, application does not need to modify the queries or handle NoSQL queries, and can remain the original system design to access both MySQL and HBase.
2. **DB Converter.** We design a database converter to handle database transformation with a table synchronization mechanism. The database converter transforms data from MySQL to HBase database with Apache Sqoop and Apache Phoenix bulk load tool. The synchronization mechanism synchronizes data after finishing transformation for each MySQL table by patching the blocked queries during transformation.
3. **Query Approach.** We propose three modes of query approach: blocking transformation mode (*BT* mode), blocking dump mode (*BD* mode), and direct access mode (*DA* mode). Each mode provides different policies of how application can access RDB.

This paper integrates above query approach and tools for querying and data transformation between RDB and NoSQL databases. The rest of paper is organized as follows. Section 2 describes existing problems and related work. Section 3 shows the design concept and introduces each component of the *data adapter* we propose. Section 4 points out the database consistency problem, and shows how to perform synchronization mechanism, along with three modes of query approach. Section 5 gives the theoretical analysis of synchronization time and synchronization overhead. Section 6 shows the experimental results and analysis of the *data adapter* system. Section 7 concludes this paper and shows the future work.

2. Related work

A cluster is a powerful architecture for computer science applications in many perspectives. For instance, a Hadoop cluster can be built with commodity hardware to access large amount of data. Furthermore, users can build their cloud platform with OpenStack [3], an open source cloud computing software. Users can decide the frameworks to be used but have to handle maintenance issues on their own. While the software stack for big data store, computing and analysis is determined, there are still some important issues needed to be considered for integration, such as security. Ali et al. [4] provide a survey which shows security issue of sharing resource on cloud platform. Chang et al. [5] present a cloud computing adoption framework to meet the requirements of business cloud. They consolidate the proposed framework with OpenStack security and multilayered security. In other words, users who build their cloud platform will have to not only solve the security issues but also encounter lots of challenges. Consequently, users will have less time for developing big data applications. Some use online cloud platforms instead of building their own clusters to focus on the design and implementation of big data applications. Hashem et al. [6] give a comparison of Google, Microsoft, Amazon and Cloudera big data cloud platforms and classify big data for users to understand the relationship between cloud platforms and big data. A lot of tools are developed for developing big data analytics system, but there is no one-size-fits-all solution. Chen and Zhang [7] discuss big data tools in different perspectives and suggest 7 principles for designing a big data system. They also show both opportunities and challenges while handling big data issues. For developers who try to leverage big data frameworks with expected performance, Barbierato et al. [8] propose a way to evaluate performance of a big data system via SIMTHESys framework. Authors use elements of the SIMTHESysBigData modelling language on this framework to represent the main elements of MapReduce paradigm. They also take Apache Hive [9], which generates MapReduce tasks, as an example to demonstrate how to model HiveQL queries with SIMTHESysBigData modelling language.

There have been numbers of works on different NoSQL databases [10], e.g. BigTable [11], HBase [2,12], MongoDB [13], and Cassandra [14], for big data [15,16]. NoSQL databases provide efficient big data storage and access requirements. In this paper, HBase is as a NoSQL database in the *data adapter* system. HBase is built on top of Hadoop distributed file system (HDFS) [17], which is a distributed framework that allows for distributed processing of large data set across clusters of computers. MapReduce framework [18] provides scalable computing services on Hadoop [19].

While NoSQL database has ability to manage big data, RDB still has superiority with middle or small scale of data. There are many studies of hybrid database system trying to integrate both databases. Cattell [20] examines a number of SQL and NoSQL data stores designed to scale simple OLTP-style application; the authors in the literature [21] point out the need of hybrid data storage in Internet of Things (IoT) area, and present a two-layer architecture based on a hybrid storage system that is able to support a federated cloud scenario in Platform as a Service (PaaS).

The design of hybrid database system architecture and the way of performing data transformation depend on the types of application services. Doshi et al. [22] classify the application the types of data growth enterprises experience, namely *Vertical Growth (VG)*, *Chronological Growth (CG)* and *Horizontal Growth (HG)*. Appropriate approaches are provided for blending SQL and NewSQL platforms for each data growth. There is an integrator used to synchronize data between RDB and NewSQL database. HBase and Hive backend are integrated to facilitate programming sophistication.

This paper focuses on the *CG*-like category which transforms data from RDB to NoSQL database. The architecture, which integrates RDB and NoSQL database, offers the capabilities to manage dramatically growing data and handle real-time queries. Thus, methods of SQL-to-NoSQL translator and schema mapping are needed when performing queries among different databases with migrated data. There are two basic strategies to migrate tables from RDB to HBase. One is to migrate all tables of a database in RDB to a table in HBase and gives different column family names for each RDB tables. The other way is to create a table in HBase for each table in RDB. JackHare [23] migrates data from MySQL to HBase with later one because it is not suggested to have too many column families in a HBase table. A schema mapping strategy is also proposed to translate data model from MySQL to HBase. JackHare performs logic operations of SQL commands via MapReduce programs. Authors describe the way JackHare supports SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY, JOIN, and AGGREGATE functions via MapReduce for most frequently used SQL commands. Rith et al. [24] identify a subset of SQL commands to access NoSQL databases. Cassandra and MongoDB are integrated because CQL, a query language for Cassandra, is similar to SQL and MongoDB is allowed to perform complex queries. Therefore authors translate SQL commands to connected NoSQL databases by implementing a middleware using C# with ANTLR as a SQL parser and SQL grammar based on Macroscope, a.Net library, to narrow the gap of using NoSQL databases. Roijackers [25] proposes an abstraction architecture with triple notation data model to store data. This work hides details of NoSQL. Users can understand this system easier. Simple queries are used to access both RDB and NoSQL database instead of ANSI-SQL commands. Transformation methods to triples are needed to be implemented for different NoSQL database. Performance

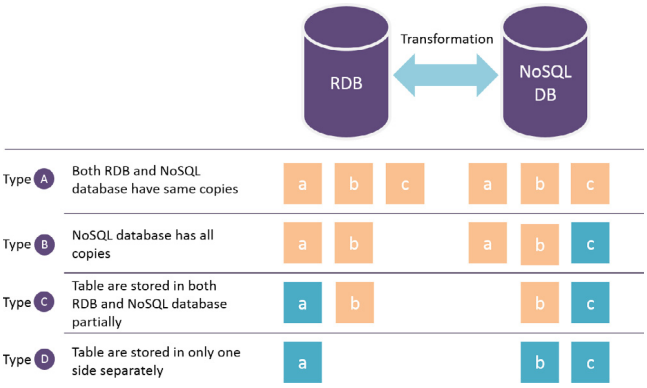


Fig. 1. Transformation types between RDB and NoSQL databases.

of INSERT/UPDATE query is bad since this paper makes nested data very complicated for enhancing read performance. Li [26] proposes a two-phase transformation process of relational tables from relational databases to HBase. The first phase is a heuristic approach transform a relational schema of a relational database to a HBase schema with data model and features required by HBase. The second phase helps with data mapping between source and target schema via an extended technique of nested mapping based on [27]. Maghfirah [28] proposes a data model to keep constraints information and enhance the process of database migration from MySQL to HBase. Constraints and relationships between tables can be stored in XML files. It helps systems to improve the process of SQL validation. With constraints information, system can check if any INSERT/UPDATE/DELETE/DROP query violates integrity constraints.

Fig. 1 shows data transformation in two aspects. One is the type of data distributions between databases while the other one is the direction of data transformation to be performed between databases. Our work focuses on *Type A* that both RDB and NoSQL database have same copies of tables, and the direction of data transformation is from RDB to NoSQL database.

The design of a flexible and modularized data converter is important. We provide a *data adapter* that contains a database converter using Sqoop [29] to perform data dump. Sqoop is a data converter designed for efficiently transforming bulk data between RDB and NoSQL database. Some researchers also use Sqoop as data converter in hybrid database system [30,31]. Transformation between two databases encounters table synchronization problem. Cho and Garcia-Molina [32] show how to refresh a local copy of an autonomous data source to maintain data consistency. They further define synchronization policies and analytical study how effective the various policies are, and show their improvement.

3. The data adapter system

The *data adapter* system is highly modularized, layered between application and databases. It is responsible for performing queries from applications and data transformation between databases at the same time. The system provides a SQL interface parsing query statements to access both a relational database and a NoSQL database.

We offer a mechanism to control the database transformation process and let applications perform queries whether target data (table) are being transformed or not. After data are transformed, we provide a patch mechanism to synchronize inconsistent tables. We present the *data adapter* system with its design and implementation in following sections.

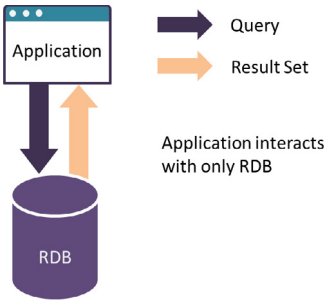


Fig. 2. Original system with RDB only.

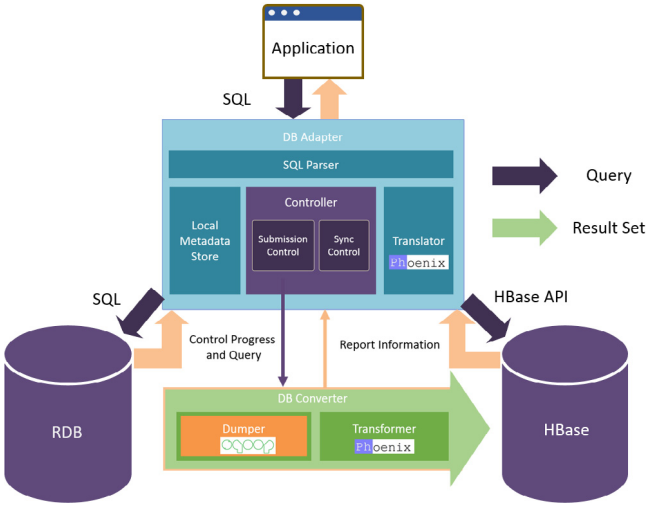


Fig. 3. System architecture with *data adapter* and its components.

3.1. System architecture

Most of the applications usually interact with relational databases as shown in Fig. 2. If the developers decide to use NoSQL database due to the growth of data along with the original relational database, the transformation between these two kinds of databases is needed. Without the proposed system, developers have to stop their service, modify application design to connect to NoSQL database for service expansion or data analysis. In order to provide a non-stopping service while the transformation is performed, we propose the *data adapter* system.

Without the *data adapter*, the original system allows application to only connect to a relational database. Fig. 3 gives the architecture of the proposed data adapter system which consists of four components: (1) a relational database, (2) a NoSQL database, (3) *DB Adapter*, and (4) *DB Converter*. The system is the coordinator between applications and two databases. It controls query flow and transformation process. The *DB Converter* is responsible for data transformation and reporting transformation progress to *DB Adapter* for further actions.

In the proposal, applications access databases through the *DB Adapter*. The *DB Adapter* parses query, submits query, and gets result set from databases. It needs some necessary information such as transformation progress from *DB Converter*, and then decides when the query can be performed to access database. The *DB Converter* transforms data from a relational database to a NoSQL database. The *data adapter* system accepts queries while the transformation is performed, the data in two databases may not be consistent. The *DB Adapter* will detect and ask *DB Converter* to perform synchronization process to maintain data consistency.

```
hbase(main):015:0> scan 'TEST'
ROW                          COLUMN+CELL
 \x80\x00\x00\x00\x00\x00\x00\x01   column=_0:DATA, timestamp=1404291520810, value=value1
 \x80\x00\x00\x00\x00\x00\x00\x01   column=_0:_0, timestamp=1404291520810, value=
```

Fig. 4. A Phoenix table in HBase.

3.2. Components design and implementation

The *data adapter* system consists of two parts as shown in Fig. 3: *DB Adapter* and *DB Converter*. The *DB Adapter* is responsible for communicating with applications, two databases, and *DB Converter*. *DB Converter* is responsible for converting data from a relational database to HBase, and synchronizing inconsistent tables. We describe the design and implementation of each component as follows.

Apache HBase is a scalable NoSQL database based on Hadoop framework. Data models of tables in HBase are quite different from ones in MySQL. To solve this issue, Phoenix is employed to create tables as clones of MySQL tables. Rowkey, column family and column qualifier of HBase are handle by Phoenix, too.

Apache Phoenix is a SQL translator for HBase. It allows database users who are familiar with SQL to access HBase with frequently used SQL commands. Instead of creating MapReduce jobs, Phoenix accesses HBase with coprocessor and makes results of queries returned faster. However, the value of rowkey and name of column family must be generated specifically when creating tables by Phoenix. Fig. 4 shows the value of rowkey is the value of column of primary key and name of column family is "*_0*". We need to convert data according to the requirements, otherwise, Phoenix cannot access any data in HBase.

DB adapter system can be designed to connect with different databases as data source. In this paper, it is designed to support MySQL and HBase. MySQL JDBC driver is used to connect with MySQL while Phoenix provides client and server jar files used to connect with HBase. We perform SQL queries from application through translator and let the translator handle SQL statement translation. When users need different NoSQL database instead of HBase, it is necessary to find a proper SQL translator for *data adapter*. Besides, we have to develop new methods for data converter to migrate data from RDB to NoSQL database.

SQL parser is an interface which accepts queries from applications, parses queries, extracts and sends necessary information to controller. Parser can tell the difference between read and write queries and pass the information to controller to put write queries, which might be affected by transformation progresses, in a queue if necessary.

Controller controls the progress of table transformation, query flow, and table synchronization according to proposed modes of query approach. Queries which perform insert, delete or update operations on a table which is being transformed to HBase are put in a queue by controller. Data in tables is not allowed to be modified in specific steps for different strategies. *Submission control* and *sync control* are two components in controllers. Submission control not only communications with converter but also recorders the transformation progress in local metadata store. In this paper, a table is a transformation unit. The order of tables to be transformed by converter is also decided by submission control. Sync control is responsible for performing synchronization process after each table is transformed. A SQLite database is used to record all necessary information such as table transformation status, and this database is kept updating by controller and DB converter.

DB converter consists of two parts: *dumper* and *transformer*. The data transformation flow in *DB converter* is shown in Fig. 5. Sqoop is employed as the *dumper* to export data from RDB to CSV format files for *transformer* phase. Sqoop first selects a range of table and divides data into splits namely files of partial data. Each split is handled by a mapper. Sqoop does not block tables

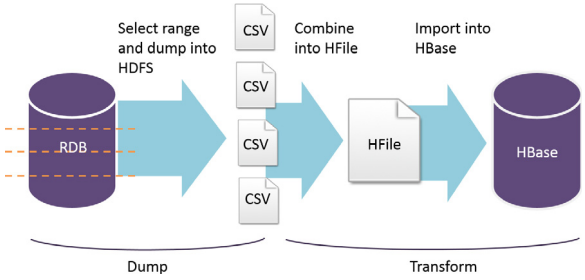


Fig. 5. Transformation flow.

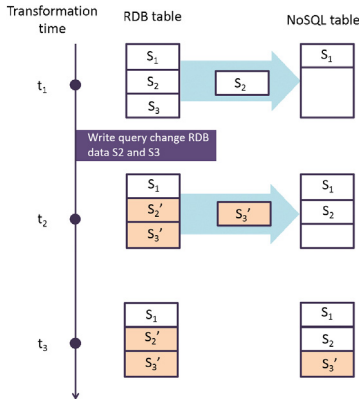


Fig. 6. The example of data inconsistency.

because of performing operations using MapReduce. Applications can submit queries to access any table which is involved in a data transformation process. In transformer phase of *DB converter*, Phoenix Bulk Load is used to load CSV files and convert data into HFiles for HBase via MapReduce operation. Another reason to use Phoenix Bulk Load is because Phoenix is a SQL translator in this system. Phoenix needs specific information while accessing tables in HBase. Phoenix Bulk Load not only coverts data into HBase tables but also generates information required by Phoenix. In our system design, transformer and translator are considered as a pair of components. The data format used in this system such as data type and schema mapping must be compatible with both transformer and translator. Otherwise, translator cannot understand the output result of transformer.

4. Query approach

The *data adapter* provides a mechanism that application can access both relational and NoSQL database whether data transformation is performing or not. An important design is took into consideration which is to decide when and how to execute query. Database accessing may change data and affect the tables in different *transformation stages*. Hence, data inconsistence between source database and destination database may occur when performing queries and data transformation on a table at the same time. Fig. 6 gives an example of how tables becoming inconsistent during the transformation process.

In Fig. 6, there is an RDB table on the left-hand side and we want to dump and transform data into NoSQL database on the right-hand side. The RDB table is divided into 3 splits S1, S2, and S3. The *DB converter* performs the transformation process in the order of S1, S2, and S3. At time t1, the *DB converter* completes the transformation for S1, and performs the transformation for S2. At time t2, the *DB converter* completes the transformation for S2, and performs the transformation of S3. Meanwhile, there is a write query arrives in our system and it affects S2 and S3. The data in RDB table and NoSQL table becomes inconsistent (S2 and S2'). At time

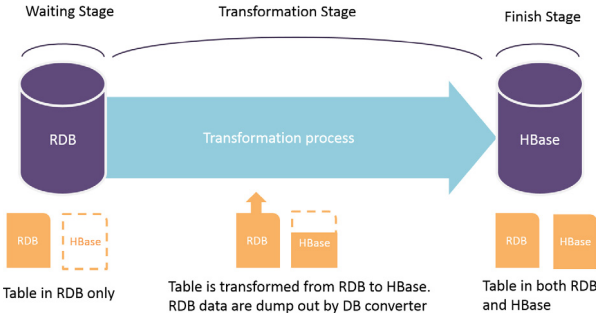


Fig. 7. Blocking transformation mode.

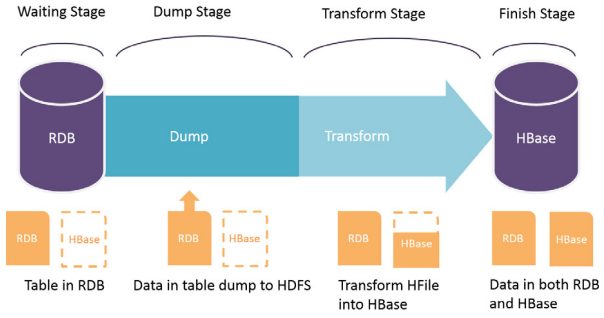


Fig. 8. Blocking dump mode.

t3, *DB converter* completes the transformation of S3' and the whole process is done. Since S3 becomes S3' before transformation, we can find S3' in RDB table which is identical to S3' in NoSQL table. As the result shows, the data between RDB table (S1, S2', S3') and NoSQL table (S1, S2, S3') is inconsistent.

The main idea of synchronizing tables is to perform the same query (patch) on inconsistent NoSQL tables. We propose three modes of query approach for applications in our system. The *data adapter* blocks queries according to each modes with different strategies. After data is transformed from RDB table to NoSQL table, there may be some inconsistent data needed to be synchronized. *Data adapter* will then patch the queries on NoSQL tables. After that, data between RDB and NoSQL database will be the same. In following section, three modes of query approach, i.e. *BT*, *BD* and *DA*, are proposed to solve this issue.

4.1. Blocking transformation mode (BT mode)

Fig. 7 shows details of *Blocking Transformation* mode (*BT* mode). Since read query will not affect RDB tables, for all coming read queries, the *data adapter* will execute them immediately. Therefore, we need to concentrate on dealing with write queries. In *BT* mode, the main strategy is that the *data adapter* will block all queries that will affect the tables being transformed.

In the transformation process, we treat a table as a transformation unit. There are three stages in *BT* mode transformation flow: *waiting stage*, *transformation stage* and *finish stage*. *Waiting stage* means the tables stay in RDB and are not transformed. Queries from application will be performed only on RDB in this stage. In the *transformation stage*, RDB table is accessed and transformed by *DB converter* into HBase table. Meanwhile, if there is a query wants to access the transformed table in *transformation stage*, the query will be blocked by controller and wait for transformation finishes. In *finish stage*, table finished transformation from RDB to HBase. The *data adapter* will then patch the blocked queries on HBase. After the synchronization is done, for the following queries, the *data adapter* can perform them on the table both in RDB and HBase to keep data synchronized. The performance of BT mode will be affected seriously by transformation time.

4.2. Blocking dump mode (BD mode)

Blocking Dump mode (*BD* mode) improves *BT* mode by further dividing *transformation stage* into *dump stage* and *transform stage*. It can reduce the influence by transformation time. The details of *BD* mode are shown in Fig. 8.

To find the opportune moment to perform write queries in RDB as early as possible during the transformation, we want to find a point in transformation process. Since transformation consists of two stages (*dump stage* and *transform stage*), we find that the best point is the moment right after *dump stage*, i.e., the point between *dump stage* and *transform stage*. The reasons are described as follows. In *dump stage*, data in tables are dumped from RDB to dump files (CSV format files) in HDFS, and queries will be blocked to prevent from data consistency between RDB tables and dump files. In *transform stage*, Phoenix Bulk Load reads CSV format files in HDFS to create HBase table. Clearly, the transformer will not access RDB in *transform stage*. So in *transform stage*, we can perform queries in RDB, but need to block queries to patch to HBase later. After a table finishes transformation, it will enter *finish stage*. We perform patch process at the beginning of *finish stage* to synchronize tables between RDB and HBase.

In *BD* mode, transformation will cause table inconsistent in *transform stage*. However, queries of application will only be blocked in *dump stage*. It can improve the performance tremendously, but application still has to wait until *dump stage* finishes.

4.3. Direct access mode (DA mode)

The strategy of *Direct Access* mode (*DA* mode) is to isolate the application execution and database transformation process. Application can perform queries at any stage on RDB. No matter table is in *dump stage* or *transform stage*, the queries will be performed on RDB immediately, and the queries will be in a local queue waiting to be patched later.

Data inconsistency problem in *DA* mode is more serious than the problem in *BD* mode since query result may be transformed partially due to query interrupt in *dump stage*. It causes new and old data of the result set will in HBase. But databases will be eventually consistent after performing synchronization process. *DA* mode uses synchronization mechanism to solve data inconsistency problem so that application can totally ignore the data transformation process. Since we allow queries enter *dump stage*, the competition between executed queries and dumper will affect RDB performance slightly.

In Table 1, we compare how the queries from application are applied on RDB during transformation. We can see *DA* mode can offer best accessibility for application. Both BT mode and *BD* mode may let application wait until transformation process is done to enter *finish stage*.

In addition, applying queries on HBase has to wait until table finishes transformation. Queries from application in *waiting stage* will be executed right away because at this time table does not start transformation. Queries in *dump and transform stage* will be put into a queue, waiting for synchronization process in *finish stage*. If queries from application arrive in *finish stage*, the queries can be performed directly on RDB and HBase since they are already synchronized.

5. Theoretical analysis

In this section, we analyse the performance of the *data adapter* regarding to synchronization time and overhead. Different modes of query approach all need to apply patches since they perform

Table 1
Query access on RDB with three modes.

	Waiting stage	Dump stage	Transform stage	Finish stage
BT mode	○	×	×	○
BD mode	○	×	○	○
DA mode	○	○	○	○

Table 2
The algorithm of synchronization overhead minimization.

```
For (i = 1; i ≤ n; i++)
  For (j = i; j ≤ n; j++)
    find all combinations of T1, T2, ..., Tn starts with Ti
    store found combinations in list L
  EndFor
EndFor
For (i = 1; i ≤ n; i++)
  find highest query frequency fi of Ti during the period of time thf
  remove the combination related to the found Ti with the highest query frequency fi
from L
EndFor
While (L still has table combination)
  calculate the number of queries needs to be patched
EndWhile
choose the table combination with smallest number of queries needs to be patched
```

queries and transformation at the same time. To synchronize the data in RDB and HBase, different number of patches have to be performed after each table is transformed. In the view of HBase, we analyse the synchronization time and the number of patches to be applied to have consistent data. We define two terms as follows for better understanding:

1. **Synchronization Time.** The synchronization time means the very first time both RDB and HBase have the same data.
2. **Synchronization Overhead.** During database transformation, we use patches and apply them on HBase to keep data consistency between RDB and HBase. The number of patches is the synchronization overhead in the view of the *data adapter*.

5.1. Synchronization time

To synchronize tables between RDB and HBase requires operations of converting tables and patching queries. Fig. 9 illustrates different period of time of operations in the synchronization process. Tables are converted one by one. The order given in Fig. 9 is $T_1, T_2, \ldots, T_{n-1}, T_n$ for n tables. Table T_1 is converted first, related *write* queries, P_1, is patched later in HBase. After finishing the operation of converting table T_1, the operation of P_1 is then performed and data converter starts converting table T_2 at the same time. Assume the patching time is much less than converting time for each table and most of the patching time is overlapped with converting time of next table. The synchronization time, t_{total}, is the sum of converting time for all tables and patching time for the last table. Therefore, the t_{total} is

$$t_{total} = \sum_{k=1}^{n} t_k + \alpha * P_n \tag{1}$$

where t_k is the time for converting table k, α is the average single-query patching time, and P_n is the number of queries to be patched which is given as follows.

$$p_n = q_n * f_n \tag{2}$$

where q_n denotes the number of all queries to be performed while converting table t_n, and f_n denotes the frequency of queries related to table t_n.

Eq. (1) shows to find the best synchronization time is to find the table with smallest number of queries to be patched. However, it will be hard to the table if the frequency of queries to be performed varies and tables are asked to be converted in a specified period of time, e.g. 1:00–2:00 AM.

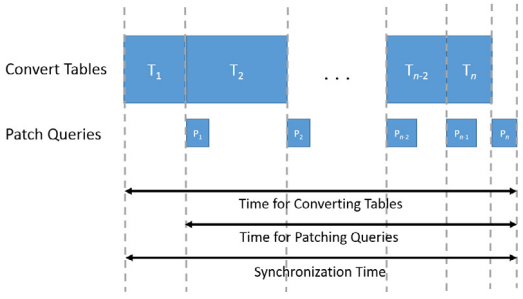


Fig. 9. Time of converting tables and patch queries.

5.2. Synchronization overhead

The number of patches represents the synchronization overhead. Some applications may frequently access one specific table in a period of time. For example, an online shopping service may frequently insert new transactions into a table, and other actions like updating customers' information or goods information do not occur so frequently. If we transform the table that is being modified with large number of transactions, it will cause large number of patches, and increase the synchronization overhead. At the same time, when *data adapter* apply the patches in HBase, it will also affect the performance of transformation. To minimize synchronization overhead of table T_n is to find the period of time t_{hf} that queries accessing T_n with highest frequency f_n and avoid converting this table during t_{hf}. So we need to find an optimal table order to minimize the total number of patches. We propose an algorithm showing in Table 2. We give a double for loop to find every table combinations at first. A for loop is then used to filter out the table combinations with highest cost in terms of patching queries for each table. A while loop is used after the for loop to calculate the number of queries needs to be patched for the left table combinations. Finally, we choose the table combination with smallest cost in terms of patching queries.

Table 3
The environment information in detail.

	Component information	
Hardware and OS	CPU	AMD Opteron(tm) Processor with 32 cores, 64-bit, 2600 MHz
	Memory	128 GB
	OS	Ubuntu 13.04 server version
	RDB	MySQL 5.5 37
Data adapter	NoSQL database	Apache HBase 0.94
	Hadoop	Apache Hadoop 1.2.1
	Phoenix	Phoenix 2.2.2
	Sqoop	Sqoop 1.4.4
Implementation	JAVA language	

Table 4
RDB table information.

Set	Table	Row count	Size
A	Books	7,984,586	500 MB
	Customers	6,676,899	500 MB
	Transactions	11,233,058	530 MB
B	Books	15,969,171	1 GB
	Customers	13,353,798	1 GB
	Transactions	22,466,116	1.1 GB
C	Books	15,969,171	1 GB
	Customers	6,676,899	500 MB
	Transactions	33,699,174	1.6 GB

The table order of minimum synchronization time and the table order of minimum synchronization overhead may not be the same. In the view of application, it concerns optimal synchronization time, so it can get consistent state of RDB and HBase faster. In the view of *data adapter*, it concerns how to minimize number of patches to reduce overhead during table transformation, and it also can reduce the competition of patching and transformation in HBase.

6. Experimental results

This section reports our empirical comparison of the proposed three modes of query approach. We will evaluate the performances of our *data adapter* system and analyse results.

6.1. Evaluation environment

Table 3 gives the experimental environment and configurations. In MapReduce process, we can set the max number of mappers by ourselves. In the experiments, we set this value to the number of CPU cores, i.e. 32 mappers, to get the best performance. We also divide RDB into 4 splits/file in dumper; this means we use 4 mappers in Sqoop import process in all following experiments. Transformer (Phoenix Bulk Load) decides appropriate number of mappers to transform the data according to the data size.

We use Amazon Elastic MapReduce test data [33] as our RDB data source. The database contains three tables: books, customers, and transactions. We generate three sets with different sizes for experiments. Table 4 lists the detailed information.

Based on the AWS test data schema, the query generator generates synthetic read/write queries, and the ratio of read/write queries is 50/50. A read query only contains SELECT statement which selects one row with specific primary key value, and always can be applied to RDB because RDB has full set of data. A write query may contain one of INSERT, UPDATE and DELETE, and it will only affect one row with specific primary key. We will focus on the discussion of write queries in experiments.

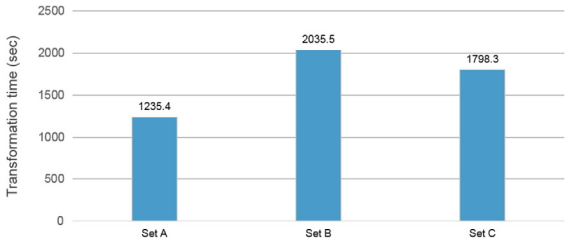


Fig. 10. DB converter transformation time.

6.2. DB converter transformation

We use 3 sets of data to perform transformation from MySQL to HBase and measure the performance of *DB converter* with each set. First, we define some terms as follows:

1. **Application Turnaround Time.** We define turnaround time as the time period from application is submitted into our system with *data adapter* to the time application finishes.
2. **Application Idle Time.** Idle time means that during the application is running, application may be blocked by the *data adapter* depending on different stages with different modes of query approach. We sum the total blocking time by the *data adapter* and define it as the idle time.
3. **Application Waiting Time.** An application submitted into the system may have to wait for previous application to be accepted by the *data adapter*, and the waiting time varies according to modes of query approach. Waiting time measures the time period from application submission to the time application starts execution.

Fig. 10 gives the transformation time of *DB converter* without any incoming query. Transformation time is affected by RDB table size. Time for Set B and set C take is longer than the time for set A. Set C takes less time than set B does although sizes of both sets are the same. The reason is the number of mappers is decided by Phoenix Bulk Load depends on the input file size. The largest table in set C is larger than the largest table in set B, so Phoenix Bulk Load asks more computing resources to transform data and results in less transformation time. The numbers of mappers in dumper and transformer are given in Table 5.

Transformation time consists of three parts: (1) the time of dumper exports data from RDB to HDFS, (2) the time of transformer imports data into HBase, (3) the time of system sets up and cleans up the job. We show the ratio of transformation time with three sets in Fig. 11. The proportions of RDB dump in three sets are very close because dumper dumps data into HDFS with MapReduce in parallel. Transformer takes the most of the transformation time because it uses only one reducer to write data into HBase, and it cannot be parallelized; the proportions of transformation in three sets are similar. The proportions of setup and cleanup of three sets are similar. We can notice that the proportion in Set A is a little larger than proportions in Set B and Set C since the size of Set A is smaller.

Table 5
Size and number of mappers in DB converter.

	Table	Size	#Mappers in dumper	#Mappers in transformer
Set A	Books	500 MB	4	8
	Customers	500 MB	4	8
	Transactions	530 MB	4	8
Set B	Books	1 GB	4	16
	Customers	1 GB	4	16
	Transactions	1.1 GB	4	18
Set C	Books	1 GB	4	16
	Customers	500 MB	4	8
	Transactions	1.6 GB	4	28

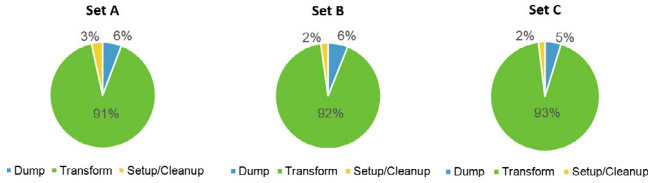


Fig. 11. Time ratio of transformation.

6.3. Single application with multiple queries

In this section, we want to simulate behaviours of a single threaded application that one query comes after another sequentially. We generate 10,000 queries for a single application. Queries are performed serially and *data adapter* accepts queries one by one. There is no dependency among queries. For instance, a record is updated by a query. This record will not be deleted by another query later. After submitting an application into the system with *data adapter*, we can observer if the turnaround time is affected by different modes of query approach, and the size of database to be transformed in the experimental results. We check the correctness of data in tables every time after each experiment and data is correct. Besides, controller in the proposal receives information of status of tables from transformer. Queries are performed according to the information. If a table is being transformed, controller will make queries accessing the proper tables in proper time.

Fig. 12 shows application turnaround time with three query modes. Application takes longest time to finish its job in *BT* mode since queries will be blocked until the involved tables finish transformation. We also observe that data transformation time is longer when data size is larger, so application turnaround time with *BT* mode in Set B and Set C will be longer than the time in Set. We can find that application turnaround time in *BD* mode and *DA* mode drop significantly comparing with *BT* mode. Although *BD* mode blocks queries with involved tables in *dump stage*, the dump time is extremely short comparing to transformation time. In *DA* mode, it will not block any query both in *dump* and *transform stages*, so the *DA* performs a little bit better than *BD* mode does.

The influence of data size is obvious in *BT* mode, but not in *BD* and *DA* modes. Because dump time does not greatly arise with larger data size, so there is only little influence on *BD* mode. *DA* mode takes shortest turnaround time among three modes. The turnaround time is nearly close to the application execution time RDB since *DA* mode does not block any query of application. The main influence factor of application turnaround time is the blocking time of different modes of query approach.

The application turnaround time is equal to the sum of application execution time and application idle time. In Fig. 13, we further show the application idle time. Since the application execution time is almost the same within three modes, we can find that the application idle time is proportional to application turnaround time.

Fig. 12. Application turnaround time.

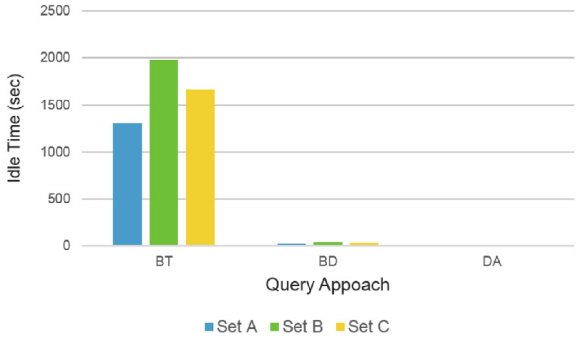


Fig. 13. Application idle time.

6.4. Multiple applications behaviours

Since the performance of single application would be affected by different factors observed in the previous section, we are interested in the behaviour of multiple applications in the *data adapter* system and find that the results are similar to those presented in single application experiments except the application waiting time. We examine the waiting time of multiple applications by submitting only one query for one application every fixed period during DB transformation is performed, and observe how different modes of query approach affect application waiting time. We only set one query for one application and observe that the result is obvious. If one application can contain more than one query, the result would be far more obvious.

The result shows in Fig. 14. We use Set B in the experiments, and set the application submission time to 1 s. For application waiting time, since it only considers the application arrival time and when it will be executed in the system, we find that results in three modes are very different. The average application waiting time of BT mode is the longest one among three modes.

In our experiments, we make average of the total waiting time of each application submitted during transformation process. We

Table 6
RDB table information.

Table	Size (GB)	Row count	#Mappers in dumper	#Mappers in transformer
Books	10	157,377,584	4	161
Customers	10	131,912,211	4	161
Transactions	10	211,603,925	4	162

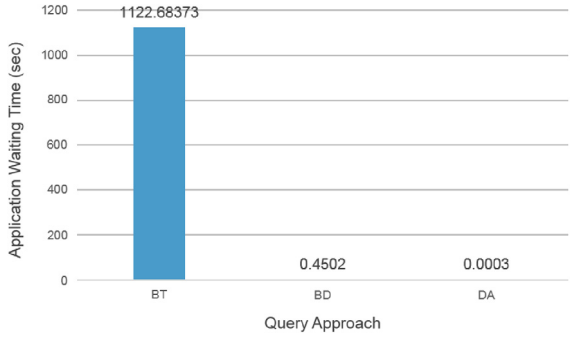


Fig. 14. Average application waiting time.

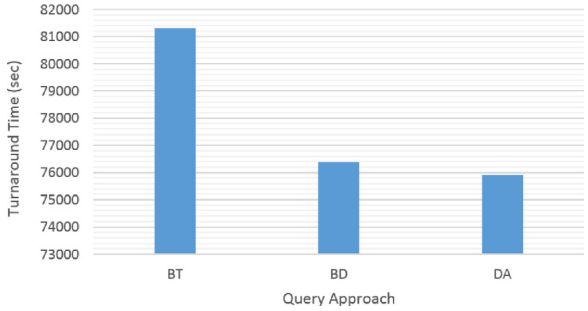


Fig. 15. Application turnaround time.

can find waiting time in *BT* mode is extremely larger than the time in *BD* mode and *DA* mode because application with write queries is blocked with involved tables in transformation. Before an application can be executed, the previous applications in the queue need to be executed first. If the previous applications are blocked, the waiting time of current application will also be affected. In *BD* mode, average waiting time is short since the proportion of dump time is short in transformation process. In *DA* mode, there is nearly no waiting time since the applications will not be blocked. The *data adapter* only spends very little time to parse each query to get necessary information, so the waiting time in *DA* mode is equal to the parsing overhead of each query.

6.5. Results on cloud platform

To verify performance of the proposal, we examine *BT*, *BD* and *DA* on an OpenStack (Grizzly) cloud platform hosted at NTHU with larger data set B. Table 6 give the details of data set. We generate 100,000 write queries as a single application and submit it to data adapter while performing data transformation.

Due to limited resources, there are two virtual machines (VMs) used for the experiment. Both VMs contains 12 virtual CPUs and 42 GB memory. The VM as master node has 4TB disk space while the VM as datanode has 2TB disk space. Fig. 15 gives the application turnaround time for *BT*, *BD* and *DA* modes of the proposal. The turnaround time increased but the performance observed is similar to the performance given in Section 6.3. Because of blocking queries in both *dump stage* and *transform stage*, *BT* requires more time than *BD* and *DA* do. Besides, large data set requires more time for *dump stage* and *transform stage*. It takes all three modes more than 75,000 s for data transformation process. *BT* needs 6.7% of its turnaround time to patch queries since it blocks all queries in data transformation process. While blocking queries only in *dump stage*, it costs less than 1% of the turnaround time for *BD* to patch queries after *transform stage*. *DA* blocks no queries during the process of data transformation and needs nearly no extra time for patching queries after the process of data transformation. Comparing results in Sections 6.3 and 6.5, we have similar observation that patching queries requires more turnaround time especially when queries are blocked during the whole process of data transformation, e.g. 88% in Section 6.3 and 6.7% in Section 6.5 for *BT*.

6.6. Summary

We discuss the advantages and disadvantages of three modes of query approach in this section:

BT mode is suitable for batch applications instead of real-time services. Some applications can tolerate maintaining time required by systems which will stop systems from executing queries. *BT* mode has less impact on these applications which submit queries during a specific time period. The advantage of *BT* mode is with less complexity of *Patch* processes since write-related queries are blocked during *dump* and *transform stages*. The disadvantage of *BT* mode is with longest data synchronization time when comparing *BT* mode with *BD* and *DA* modes.

BD mode allows queries to be executed after *dump stage* and is suitable for applications which allows little delay in terms of performing write-related queries. The advantage of *BD* mode is that *patch* process can be started earlier but the disadvantage is with higher *Patch* cost.

DA mode allows performing write-related queries without blocking queries in *dump stage* because Sqoop is employed to convert data from MySQL to CSV files. Sqoop does not lock tables and it makes perform queries in *dump stage* possible. The advantage of *DA* mode is almost delay-free during data transformation in *dump and transform stages*. The disadvantage is *DA* mode has highest cost of *Patch* process among three modes.

Experiments shows the pros and cons of the proposal. Users can choose *BT*, *BD* and *DA* modes of query approach for different scenarios, namely batch applications, real-time services or other kinds of systems with specific requirements of performing queries.

7. Conclusions and future work

A flexible and highly modularized *data adapter* for hybrid database system is proposed in this paper. The *data adapter* uses a general SQL layer accepting queries from application services, so that original application does not need to change the design. The *data adapter* also controls query flow during database transformation. We implement a prototype system and show the design concept of whole system. We also present three modes (*BT*, *BD*, and *DA* modes) of query approach with different blocking policies to perform data transformation from MySQL to HBase. In order words, this paper focuses on velocity with *BT*, *BD*, and *DA* modes of query approach and partly variety with data stored in MySQL, CSV files and HBase for different stages.

We provide theoretical analysis of synchronization time and synchronization overhead. The most important two factors are table transformation order and the characteristics of queries. We

provide a solution to minimize the synchronization time. We also calculate the number of patches and offer an algorithm to minimize the synchronization overhead.

We examine factors that influence application performance in the *data adapter*, including different database sizes, different table sizes, and application types; each of them is examined with different modes of query approach. The results show size affects the turnaround time of single application. In *BT* mode, it is influenced the most because of the long blocking time in data transformation stage. The idle time of single application in *BT* mode is the longest, and it is the shortest in *DA* mode, since application in *DA* mode can totally be executed regardless of database transformation. The application waiting time in *BT* mode takes the longest time.

In the future, we will focus on speeding up *DB converter* and try to evaluate performance with suitable models. As showed in our experiments, data size directly affects performance of applications and *data adapter*. We also want to support more complicated SQL queries by enhancing the SQL parser and translator between *data adapter* and applications. The *data adapter* now can offer real-time access with NoSQL database but cannot efficiently perform batch operations on NoSQL database. We will put emphasis on how to speed up SQL query execution. With the flexibility of the proposed *data adapter* system, we will offer more connectors to deal with different types of databases to support various services. Security is also an important issue as mention in related work. We mainly describe the functionalities, e.g. SQL query, data migration and data synchronization. To improve the *data adapter* to avoid data being hacked and compromised, it is necessary to design and integrate security components in the future.

Acknowledgement

The work was supported by the Ministry of Science and Technology of Taiwan (No. NSC 101-2221-E-007-028).

References

[1] Apache Phoenix. Available: https://phoenix.apache.org/.
[2] Apache HBase. Available: http://hbase.apache.org/.
[3] OpenStack. Available: https://www.openstack.org/.
[4] M. Ali, S.U. Khan, A.V. Vasilakos, Security in cloud computing: Opportunities and challenges, Inform. Sci. 305 (2015) 357–383.
[5] V. Chang, Y.-H. Kuo, M. Ramachandran, Cloud computing adoption framework: A security framework for business clouds, Future Gener. Comput. Syst. 57 (2016) 24–41.
[6] I.A.T. Hashem, I. Yaqoob, N.B. Anuar, S. Mokhtar, A. Gani, S.U. Khan, The rise of "big data" on cloud computing: Review and open research issues, Inf. Syst. 47 (2015) 98–115.
[7] C.L.P. Chen, C.-Y. Zhang, Data-intensive applications, challenges, techniques and technologies: A survey on Big Data, Inform. Sci. 275 (2014) 314–347.
[8] E. Barbierato, M. Gribaudo, M. Iacono, Performance evaluation of NoSQL big-data applications using multi-formalism models, Future Gener. Comput. Syst. 37 (2014) 345–353.
[9] Apache Hive. Available: https://hive.apache.org/.
[10] J. Han, E. Haihong, G. Le, J. Du, Survey on NoSQL database, in: 6th International Conference on Pervasive Computing and Applications, ICPCA, 2011, pp. 363–366.
[11] F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, et al., Bigtable: A distributed storage system for structured data, ACM Trans. Comput. Syst. (TOCS) 26 (2008) 4.
[12] M.N. Vora, Hadoop-HBase for large-scale data, in: International Conference on Computer Science and Network Technology, ICCSNT, 2011, pp. 601–605.
[13] K. Chodorow, MongoDB: The Definitive Guide, O'Reilly Media, Inc., 2013.
[14] A. Lakshman, P. Malik, Cassandra: a decentralized structured storage system, ACM SIGOPS Oper. Syst. Rev. 44 (2010) 35–40.
[15] N. Leavitt, Will NoSQL databases live up to their promise? Computer 43 (2010) 12–14.
[16] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, et al. Big data: The next frontier for innovation, competition, and productivity, 2011.
[17] D. Borthakur, HDFS architecture guide, HADOOP APACHE PROJECT, 2008. http://hadoop.apache.org/common/docs/current/hdfsdesign.
[18] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Commun. ACM 51 (2008) 107–113.
[19] Apache Hadoop. Available: http://hadoop.apache.org/.
[20] R. Cattell, Scalable SQL and NoSQL data stores, ACM SIGMOD Rec. 39 (2011) 12–27.
[21] M. Fazio, A. Celesti, M. Villari, A. Puliafito, The need of a hybrid storage approach for IoT in PaaS cloud federation, in: 28th International Conference on Advanced Information Networking and Applications Workshops, WAINA, 2014, pp. 779–784.
[22] K.A. Doshi, T. Zhong, Z. Lu, X. Tang, T. Lou, G. Deng, Blending SQL and NewSQL approaches: Reference architectures for enterprise big data challenges, in: 2013 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, CyberC, 2013, pp. 163–170.
[23] W.-C. Chung, H.-P. Lin, S.-C. Chen, M.-F. Jiang, Y.-C. Chung, JackHare: a framework for SQL to NoSQL translation using MapReduce, Autom. Softw. Eng. (2013) 1–20.
[24] J. Rith, P.S. Lehmayr, K. Meyer-Wegener, Speaking in tongues: SQL access to NoSQL systems, in: Proceedings of the 29th Annual ACM Symposium on Applied Computing, SAC'14, 2014, pp. 855–857.
[25] J. Roijackers, Bridging SQL and NoSQL (Master's thesis), Eindhoven University of Technology, 2012.
[26] C. Li, Transforming relational database into HBase: A case study, in: 2010 IEEE International Conference on Software Engineering and Service Sciences, ICSESS, 2010, pp. 683–687.
[27] A. Fuxman, M. Hernandez, C. Ho, R. Miller, P. Papotti, L. Popa, Nested mappings: schema mapping reloaded, in: Proceedings of the 32nd International Conference on Very Large Data Bases, VLDB, 2006, pp. 67–78.
[28] I. Maghfirah, Constraints preserving in schema transformation to enhance database migration from MySQL to HBase (Master's thesis), National Tsing Hua University, 2014.
[29] Apache Sqoop. Available: http://sqoop.apache.org/.
[30] O.V. Joldzic, D.R. Vukovic, The impact of cluster characteristics on HiveQL query optimization, in: 21st Telecommunications Forum, TELFOR, 2013, pp. 837–840.
[31] T. Kim, H. Chung, W. Choi, J. Choi, J. Kim, Cost-based join processing scheme in a hybrid RDB and hive system.
[32] J. Cho, H. Garcia-Molina, Synchronizing a database to improve freshness, in: ACM Sigmod Record, 2000, pp. 117–128.
[33] Amazon Elastic MapReduce Testing Data. Available: http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/query-impala-generate-data.html.

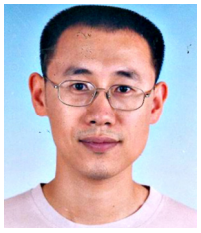
Ying-Ti Liao received her B.S. degree in Computer Science from National Taipei University in 2012, M.S. degree in Computer Science from National Tsing Hua University in 2014. She is currently a research assistant in the Department of Computer Science, National Tsing Hua University. Her research interests include Cloud Computing and Big Data.

Jiazheng Zhou received his B.S. degree in Computer Science from National Chengchi University in 2002, M.S. degree and Ph.D. degree in Computer Science from National Tsing Hua University in 2004 and 2011. He is currently a Postdoc in the Department of Computer Science, National Tsing Hua University. His research interests include Cluster Computing, Interconnection Network, High Performance Computing, Cloud Computing, and Big Data.

Chia-Hung Lu is a master student of Computer Science in National Tsing Hua University. He received his B.S degree in Computer Science from National Tsing Hua University in 2014. His research interests are related to Big Data and Cloud Infrastructure.

Shih-Chang Chen received his B.S. and M.S. degrees in Computer Science from Chung Hua University, Taiwan, in 2003 and 2005, Ph.D. degree in Ph.D. Program in Engineering Science from Chung Hua University in 2010. He is currently a postdoctoral research fellow in Computer & Communication Research Center at National Tsing Hua University. His research interests include Parallel and Distributed Systems, Cloud Computing and Big Data.

Ching-Hsien Hsu is a professor in department of computer science and information engineering at Chung Hua University, Taiwan. His research includes high performance computing, cloud computing, big data intelligence, parallel and distributed systems, ubiquitous/pervasive computing and intelligence. Dr. Hsu is an IEEE senior member.



Wenguang Chen received the B.S. and Ph.D. degrees in computer science from Tsinghua University in 1995 and 2000 respectively. He was the CTO of Opportunity International Inc. from 2000 to 2002. Since January 2003, he joined Tsinghua University. He is now a professor and associate head in Department of Computer Science and Technology, Tsinghua University. His research interest is in parallel and distributed computing, programming model and mobile cloud computing. He is leading the PACMAN Group now.

Mon-Fong Jiang received a B.S. degree in Applied Mathematics from National Chung Hsing University in 1994, and the M.S. and Ph.D. degrees in Computer and Information Science from National Chiao Tung University in 1996 and 2000, respectively. His research interests include machine learning, parallel computing, and semiconductor engineering data analysis systems. He is currently Vice President of is-land Systems Inc., a Hsinchu Science Park company in Taiwan.

Yeh-Ching Chung received a B.S. degree in Information Engineering from Chung Yuan Christian University in 1983, and the M.S. and Ph.D. degrees in Computer and Information Science from Syracuse University in 1988 and 1992, respectively. He joined the Department of Information Engineering at Feng Chia University as an associate professor in 1992 and became a full professor in 1999. From 1998 to 2001, he was the chairman of the department. In 2002, he joined the Department of Computer Science at National Tsing Hua University as a full professor. His research interests include parallel and distributed processing, cloud computing, and embedded systems. He is a senior member of the IEEE computer society.