A Synchronization Algorithm of Mobile Database for Cloud Computing

Ranjeet Singh¹, Chiranjit Dutta²

Faculty of information Technology,
SRM University, NCR campus, India

Abstract
The existing synchronization solutions of mobile database are not independent from the database server because they use database dependent information such as metadata or use specific functions of database server such as trigger and time stamp. These constraints are critical weak point in cloud environment because various applications are running in various devices and servers in ubiquitous environment. Therefore, there is must need synchronization algorithms of mobile database which is independent of the vendor of synchronization solution and mobile database in cloud environment. This paper suggests SAMD (Synchronization Algorithms based on Message Digest) in order to resolve the problems mentioned above. SAMD resolves synchronization problems using only standard SQL queries as certified by the ISO (International Organization for Standardization). This is followed by a possible synchronization of any data combination regardless of the kind of database of server side or mobile database. In conclusion, the SAMD is effective solution for mobile database synchronization in cloud environment.

Keywords: SAMD, MDV, DSDT, MCDT.

1. Introduction
Recent advances in mobile technology and equipment have led to the emergence of a new computing environment and a variety of small sized mobile devices such as PDAs (personal digital assistants), smart mobile phones, HPCs (handheld PCs) and Pocket PCs have been popularized. As various network technologies are increasingly being associated with such mobile devices, the processing of business information can be available using mobile devices. As a result, business models that rely on mobile technologies are appeared [1]. Mobile devices do not have much computing power and rely on batteries. Additionally, constant access to network is difficult due to narrow bandwidth [2] [3]. Therefore, it is not easy to process a large size of stored data and maintain a continuous connection with the server-side database. For these reasons, mobile devices have mobile databases in order to achieve stable data processing. Mobile devices download replications of limited data from a connected server-side database using a synchronization device that has a stable wire communication function. Mobile devices process various tasks using the data downloaded in an off-line state. The work on the network disconnected condition is a crucial point for mobility support [4]. In a disconnected environment, there are inevitable inconsistencies between the server-side database and the mobile database. Synchronization techniques can solve the data inconsistencies and guarantee the integrity of the data. Consequently, synchronization is an essential subject in mobile device computing environments [5]. Commercial DBMS vendors offer various solutions to data synchronization in a mobile environment [6], [7], [8]. However, these solutions are not independent of the server-side database because they use database dependent information such as metadata or use specific functions of server-side database such as trigger and time stamp. In other words, the mobile database vendor should be equivalent to the server-side database vendor.

The solution of operating a separate synchronization server in the middle tier is independent of the server-side database but dedicated to the mobile database. That is, the synchronization solution and the mobile database should be the identical vendor product. Additionally, when a client programmer develops mobile applications that are embedded in mobile devices, the developer should uses a particular library that is provided by the vendor of mobile database or modify existing mobile applications for synchronization process. Because of these restrictions, the extensibility, adaptability and flexibility of mobile business systems are markedly decrease. This problem must be solved in order to build efficient mobile business systems because upcoming mobile environments will have heterogeneous characteristics in which diverse mobile devices, mobile databases and RDBMS exist.
2. LITERATURE SURVEY

2.1 Synchronization Framework
Fig. 1 represents a synchronization framework using a synchronization server in a mobile business environment. The whole framework consists of a server-side database, synchronization server (AnySyn) and multiple mobile devices with internal mobile databases. The server-side database maintains all of the data required for business, and the mobile database downloads copies of data the user needs from the server-side database. The synchronization server is located between the two databases to synchronize the data and manage.

![SAMD Synchronization Framework](image1)

**Figure 1: SAMD Synchronization Framework**

Additional information required for synchronization. The AnySyn synchronization server performs synchronization based on the SAMD algorithm. The synchronization policy is established in AnySyn and the load caused by accessing the server-side database is minimized by operating a connection pool. Every mobile device uses a separate toolkit to access the AnySyn server over a wired network to perform synchronization.

2.2 Existing System
The existing system uses a Synchronization Algorithm based on comparison of message digest values of the selected rows of both server-side database and mobile database needed for synchronization.

2.3 Message Digest
Message digest consists of a unidirectional hash function that maps a message of a random length to a fixed-length hash value. Message digest $h$ is created by the hash function $H$, which can be expressed as follows:

$$h = H(M)$$

$M$ is a message of a random length and $H(M)$ is a fixed-length message digest. Even a single bit changed in the message causes a change of message digest value.

3. SAMD SYNCHRONIZATION ALGORITHM

3.1 Objective of SAMD
In order to guarantee independence of database vendor and synchronization solution vendor in a mobile business environment that has diverse mobile devices, mobile databases, and RDBMS, the SAMD synchronization algorithm satisfies the following objectives.

![Message Digest in table](image2)

**Fig 2: Message Digest in table**
1. Independence of vendors.
   Does not use metadata or internal functions dedicated to a particular database.
2. Synchronization using only standard SQL statements.
   Perform synchronization using only standard SQL queries and data manipulation language specified in ISO standard. Therefore, any data processing using trigger is not allowed.
3. Disallows schematic modification of data table of the server-side database.
   The data table schema cannot be modified to add data necessary for synchronization. In other words, synchronization must be performed independent from the existing data table schema. Therefore, additional information such as time stamps cannot be added to the data table.
4. Disallows adding restrictions in implementing applications. There can be no restrictions such as performing additional works to an application code or having to use a specific library in order to perform synchronization.

3.2 SAMD Synchronization Algorithms

The SAMD synchronization algorithm is applied for the table schema of the server-side database and the mobile database. Both databases have a data table (DSDT: Database Server Data Table, MCDT: Mobile Client Data Table) and a message digest table (DSMDT: Database Server Message Digest Table, MCMDT: Mobile Client Message Digest Table). The data table contains the business data, and the message digest table stores the message digest value from the data table. The message digest table consists of a PK (primary Key) column of data table, message digest value (MDV) column, flag (F) column and mobile device ID (Mid) column. The flag column signals an inconsistency that has occurred in the corresponding column; therefore, the flag column is used to identify a row that requires synchronization. The mobile device ID is a unique number of the mobile device, so this column is used to identify a mobile device that requires synchronization.

![Fig 3: Table Structure of SAMD](image)

In Fig. 3, if a row’s PK value is A1, this value is identical to the two message digest values and there is no need for synchronization. However, if a row has a PK value of C1, the value of MDV in MCDMT is different from the value of MDV in DSMDT and the MCMDT flag value is 1. Consequently, synchronization is necessary. The synchronization process is performed for each row to resolve all of the inconsistencies mentioned in Section II/B. For example, if there is an inconsistency in row C1, synchronization takes place from the mobile database to the server-side database and DSDT’s PK C1 row is replaced with the MCDT’s C1 row.

The synchronization algorithm consists of Synchronizations 1, 2 and 3, as shown in Fig. 3. Synchronizations 1 and 2 synchronize the data table and message digest table. Therefore, the two are identical synchronization algorithms applied to different tables. Here, the message digest values that are created with each row value of the data table, and the message digest values of the message digest table, are compared. If the values are identical, there has been no change in the data and synchronization is not necessary. If the values are different, it means that the data table value has been changed, in which case the message digest table has to be updated with new message digest values and the flag has to be set to 1. The flag value is used to identify a row that needs synchronization. The server-side database has one DSMDT for every DSDT. Although the size of the MCMDT is smaller than that of the DSMDT, there is an MCMDT for every mobile device that has a unique ID. It is very inefficient to perform Synchronization 2 for every row of the DSDT every time there is a synchronization request from a mobile device. Therefore when the mobile device requests synchronization, the mobile device ID value is sent to the server-side database and then the SAMD algorithms select the row from DSMDT whose value of Mid column is the same as the mobile device ID value and Synchronization 2 is only applied to the selected rows. For example, a mobile device whose mobile device ID value is ‘md1’ requests synchronization, the rows whose value of Mid column is ‘md1’ are selected and then only used in Synchronization 2.

After SAMD algorithms analyze the type of inconsistency using the flag values of both messages digest tables,
primary key, which is used to identify the row. Therefore, Synchronization 3 is performed between two data tables for each inconsistent type. Upon completion of synchronization, the flag of the synchronized row is set to 0 in the message digest table.

Most mobile devices have limited resources, and the load on the device should be minimized during the synchronization process. Accordingly, all message digest tables are located in the server-side database to economize storage space of the mobile device, as shown in Fig. 3, while there is the load caused by accessing the network in Synchronization 1 but the data size of MCDT is smaller than the server. Furthermore, the MCDT data necessary for Synchronization 1 is sent to the server-side database in a single transmission over a wired network using an SQL query capable of batch processing.

After this point, there is no load on the mobile device, which reduces the load caused by network access in the Synchronization 1 stage.

The SAMD synchronization algorithm must keep the following restrictions.
1) Every database table must have a primary key.
2) The primary keys of the data table and the message digest table have an identical value for a given row.
3) A new row is inserted into the mobile database and another one into the server-side database; the primary key values of the two rows cannot be identical.

The relational database model involves every table having restrictions 1) and 2) are basic conditions. Restriction 3) implies that there is no integrity collision for the primary key during the synchronization process between the mobile database and the server-side database. Even though identical primary key values can be inserted at both ends, this problem is not taken into consideration since it can easily be resolved by application-level processing or the synchronization policy.

Fig. 4 exhibits the flow chart for the SAMD synchronization algorithm.

Steps S1–S3 represent the Synchronization 2 stage of Fig. 3. When the DSDT and DSMDT are FullOuterJoined, DSDT rows, for which Steps S1, S2 and S3 should be applied, can be identified by dangling rows.

**Fig. 4. Table structure of SAMD Algorithms**

Steps S2–S3 are the synchronization process for cases in which row is inserted into or deleted from the DSDT. If a row is inserted, its primary key and message digest values of the new row are added in the DSMDT. For a deleted row, its MDV column value is set to NULL in the DSMDT. If the row is deleted without setting the column value to NULL, it is distinguishable from inserting a row into the DSDT. Step S7–S12 display the Synchronization 3 stage of Fig. 3 When the DSMDT and MCMDT are Full Outer Joined, the rows that are subject to synchronization and the inconsistent types
are identified using the dangling rows and the DSMDT and MDCMDT flags and then the synchronization between DSDT and MCDT is achieved.

4. IMPLEMENTATION AND EVALUATION

The SAMD algorithm was implemented using JAVA and were linked with databases using JDBC (Java Database Connectivity). The commercial synchronization solution, which is vendor-independent of server-side database and has middle-tier architecture, was used in order to compare SAMD. Because the properties of the commercial synchronization solution and SAMD are similar, this comparison is reasonable. As for the message digest, JCE (Java Cryptography Extension) [10] was used.

For performance evaluation, the commercial RDBMS and mobile database were installed on one machine to eliminate the network effect factors. First, 1000 randomly generated rows were inserted into RDBMS, sent through synchronization and the same 100 rows were inserted into database server. Then, we inserted 50 rows into RDBMS, and then inserted another 50 rows into mobile database. Afterwards, 50 rows were deleted from mobile database and another 50 rows modified. The modified and deleted data were programmed to be equally spaced among the 100 rows.

Now this times changes the modified the mobile client database, by retaining the most popular words and their associated postings in main memory, we may be able to reduce the number of accesses to the disk resident portion of the index. With sentence chunking, we expect the access pattern to be more random due to very large size of the sentence vocabulary.
The commercial synchronization solution requires that additional data table, procedures and triggers be created for synchronizing with a server side database that is not a same vendor of the commercial synchronization solution [7].

For the commercial synchronization solution, a synchronization SQL or a JAVA script must be written [9]. This is a very cumbersome and inconvenient process that requires a substantial amount of time to become familiar with. Furthermore, some cases in TABLE I are not supported. Therefore, our evaluation could not be performed for all cases listed in TABLE I but only for the four cases mentioned above. The commercial synchronization solution requires that additional data table, procedures and triggers be created for synchronizing with a server-side database that is not a same vendor of the commercial synchronization solution [9].

There are no such requirements for the SAMD algorithm. Whereas the vendor of the commercial synchronization solution and the mobile database must be the same, but SAMD mandates no restrictions regarding the types of the server-side databases or the mobile databases. Therefore, there is an advantage in terms of having a wide use property in addition to performance.

However, SAMD maintains a separate message digest table, which is a disadvantage from the perspective of storage efficiency. However, since large scale storage is relatively.

5. INTERFACE OF SAMD MOBILE DEVICE
CONCLUSIONS AND FUTURE WORK

This paper has suggested an SMAD synchronization algorithm based on message digest for synchronizing between server-side databases and mobile databases. The SAMD algorithm is performed with only SQL functions of relational databases, so that it is not dedicated to particular vendors and is available for use in combination with any server-side databases and mobile databases. The extensibility, adaptability and flexibility are guaranteed when a mobile business system is authorized. This feature is important in order to build an efficient mobile business system because the upcoming mobile business environment has heterogeneous characteristics in which diverse mobile devices, mobile databases and RDBMS exist.

REFERENCES