A SYNCHRONIZATION ALGORITHM FOR MOBILE DATABASES USING SAMD

[1] P.Kalyanakumar [2] A.Sangeetha

[1] *Asst Professor, Department of Computer Science Latha mathavan Engineering College*
[2] *Asst Professor, Department of Information Technology PSNA college of Engineering and Technology*

----------***----------

Abstract - *Synchronization Algorithms based on Message Digest) algorithm based on message digest in order to facilitate data synchronization between a server- side database and a mobile database. The SAMD algorithm makes the images at the server -side database and the mobile database uses message digest tables to compare two images in order to select the rows needed for synchronization. If the two images are different, the synchronization progresses according to synchronization policy. The SAMD algorithm does not use techniques that are dependent on specific database venders; neither does it use triggers, stored procedures or timestamps. The SAMD uses only the standard SQL functions for the synchronization. Therefore the SAMD algorithm can be used in any combinations of server-side database and mobile database because of its independence of database vender. This feature is important in order to build efficient mobile business systems because the upcoming mobile business environment has heterogeneous characteristics in which diverse mobile devices, mobile databases, and RDBMS exist.*

Key Words: Mobile Device, Synchronization, Mobile Database

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 toThe corresponding author is Chang-Joo Moon.

Mi-Young Choi, Eun-Ae Cho and Doo-Kwon Baik are with the Software System Lab. in the College of Information and Communication, Korea University, Seoul, Korea (e-mail: miche11e@korea.ac.kr, eacho@korea.ac.kr, baikdk@korea.ac.kr) Dae-Ha Park is with the Divi. of IT and Media , Korea Digital University, Seoul, Korea (summer69@kdu.edu) Chang-Joo Moon are with the Department of Aerospace Information Engineering, Konkuk University, Seoul, Korea (cjmoon@konkuk.ac.kr)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 venders 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 vender should be equivalent to the server-side database vender. 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 vender 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 vender 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.

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 server-side database or mobile database. The SAMD therefore would provide extensibility, adaptability and flexibility. The SAMD makes the images at the table of the server-side database and the mobile database using a message digest algorithm; then the images, and the message digest values, are saved in the message digest tables on both sides. The SAMD algorithm compares two images in order to select the rows needed for synchronization. If the value of message digest regarding identical rows is different for both sides, it means the duplicated rows have been changed and synchronization is necessary using SAMD. Message digest is used to detect falsification of data transferred mainly via security protocols. In this procedure, because a large volume of data is compressed into a small volume, we can simplify the detection of data inconsistencies and minimize wasted storage space. Message digest functions work fast even with limited resources, so that they reduce the burden placed on mobile devices that have small computing power.This paper is organized into several sections. Section 2 introduces the background knowledge that is needed to understand this paper. A proposed SAMD synchronization algorithm based on message digest is explained in Section 3. In Section 4, performance evaluation and quality evaluation is achieved in implementing SAMD. Finally, Section 5 concludes the paper.

2. BACKGROUND KNOWLEDGE

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.

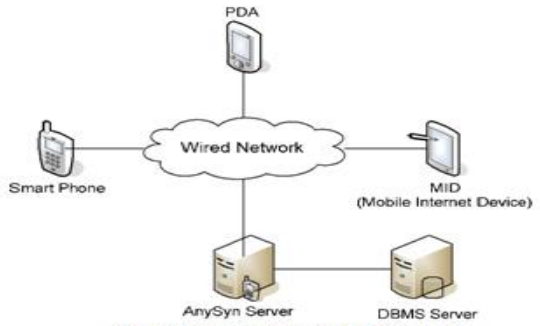

Fig. 1. SAMD Synchronization framework

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 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. Rows Inconsistency

An inconsistency refers to a state in which the published data in the server-side database and the subscribed data in the mobile database carry different values due to a change at either side. The two databases add, delete and modify data independently, which makes inconsistency inevitable. TABLE I displays every case for an inconsistency for a single row.

Among the 16 cases indicated in TABLE I, Cases 6, 7, 8, 10 and 14 include the ADD operation, which cannot occur for a single row. For example, in Case 7 the row added at the server side is different from the row modified at the client; therefore, it cannot be considered an inconsistency. Case 7 is equivalent to Case 3 and Case 5 occurring independently. Similar reasoning can be made for Cases 6, 8, 10 and 14, so SAMD does not consider the five cases.

TABLE I
INCONSISTENCY ANALYSIS

C	Mobile DB	DB Server	C	Mobile DB	DB Server	C	Mobile DB	DB Server	C	Mobile DB	DB Server
1	UC	UC	5	UC	ADD	9	UC	MOD	13	UC	DEL
2	ADD	UC	6	ADD	ADD	10	ADD	MOD	14	ADD	DEL
3	MOD	UC	7	MOD	ADD	11	MOD	MOD	15	MOD	DEL
4	DEL	UC	8	DEL	ADD	12	DEL	MOD	16	DEL	DEL

(C:Case, UC:Unchanged, ADD:Added, MOD:Modified, DEL:Deleted)

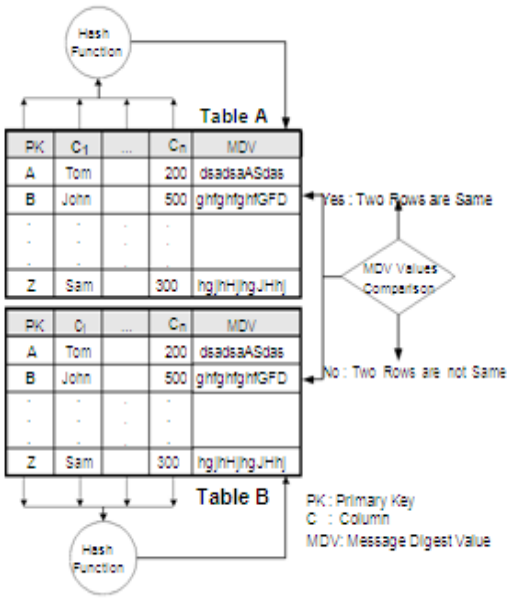

Fig. 2. Message Digest in Data Table

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 [9]. Fig. 2 demonstrates how this message digest mechanism can be applied to a relational database to examine data identity between rows of two tables.

Data in two rows are identical if two rows in Tables A and B have identical message digest values. If the two values are different, it means that the two rows have one or more different column values. Accordingly, this method can be useful in detecting inconsistency between two rows. Once a row with an inconsistency is detected, the row is copied using the primary key in the direction of synchronization according to the synchronization policy. This synchronization algorithm identifies a modified row without relying on the database's internal functions, logs or metadata to enable synchronization that is independent of the database vender.

3. SAMD SYNCHRONIZATION ALGORITHM

3.1. Objective of SAMD

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

O1) Independence of venders.

- Does not use metadata or internal functions dedicated to a particular database.

O2) 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.

O3) 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.

O4) 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 Algorithm

Fig. 3 displays the table schema of the server-side database and the mobile database where the SAMD synchronization algorithm is applied. 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 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.

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 MCMDT 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,

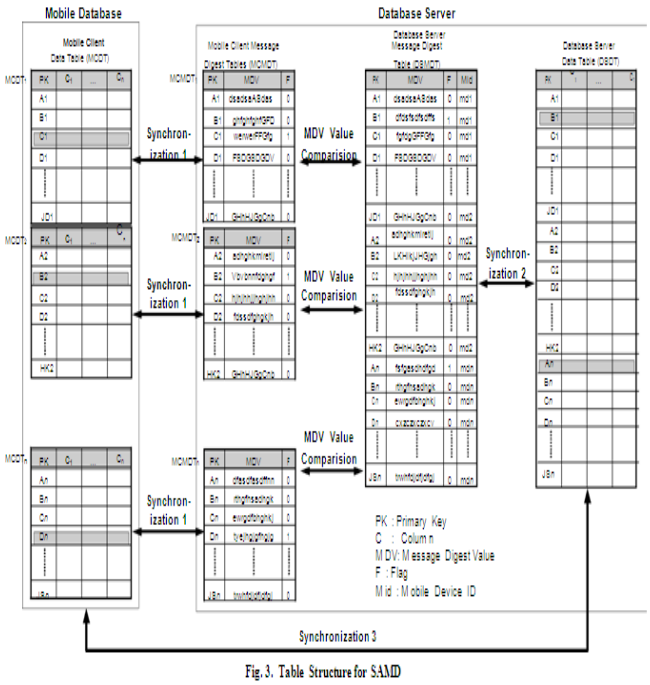

Fig. 3. Table Structure for SAMD

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 a 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.

Step S1 is the synchronization process with the DSMDT when a row of the DSDT has been modified. The message digest value is calculated for the DSDT row, which is compared to the MDV column value of the DSMDT. If the two values are identical, it means that there has not been a change at the DSDT row, and vice versa. If a change is detected, DSMDT's MDV column value is replaced with the message digest value of DSDT's row, and the flag column is set to 1. Steps S2~S3 are the synchronization process for cases in which a 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 indistinguishable from inserting a row into the DSDT.

Steps S4~S6 indicate the Synchronization 1 stage of Fig. 3. This stage involves synchronizing the MCDT and MCMDT; the basic algorithm is identical to that used in Steps S1~S3. However, the MCDTT and MDMDT are internal data tables of different vender databases that are physically separated, so FullOuterJoin as in Steps S1~S3 is not feasible. In this case, a temporary table has to be created for the MCDT table to be copied to the server side, and Synchronization 1 process is performed, after which the copied data is deleted. This single transaction by batch processing guarantees independence of the SAMD algorithm for the database vender.

Steps S7~S12 display the Synchronization 3 stage of Fig. 3. When the DSMDT and MCMDT are FullOuterJoined, 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 the DSDT and MCDT is achieved.

Step S7 involves synchronizing a modified row or one deleted from the MCDT with the DSDT. Under the D1 condition, Step S7 searches for a row with an MCMDT flag value of 1 and a DSMDT flag value of 0. The flag values indicate that the row was modified or deleted from the MCDT. A null MDV column of the MDMDT signifies a deletion from the MCMT. Otherwise, there has been a modification. In the case of deletion, rows that correspond to the rows deleted from the MCDT should be deleted from DSDT, DSMDT and MDMDT. In the case of modification, the DSDT row value is replaced with the MCDT row value, and DSMDT row value is replaced with the MDMDT row value. Upon completion of synchronization, the flag values of the synchronized rows of the DSMDT and MCMDT are set to 0. This process resolves the inconsistency cases C3 and C4.

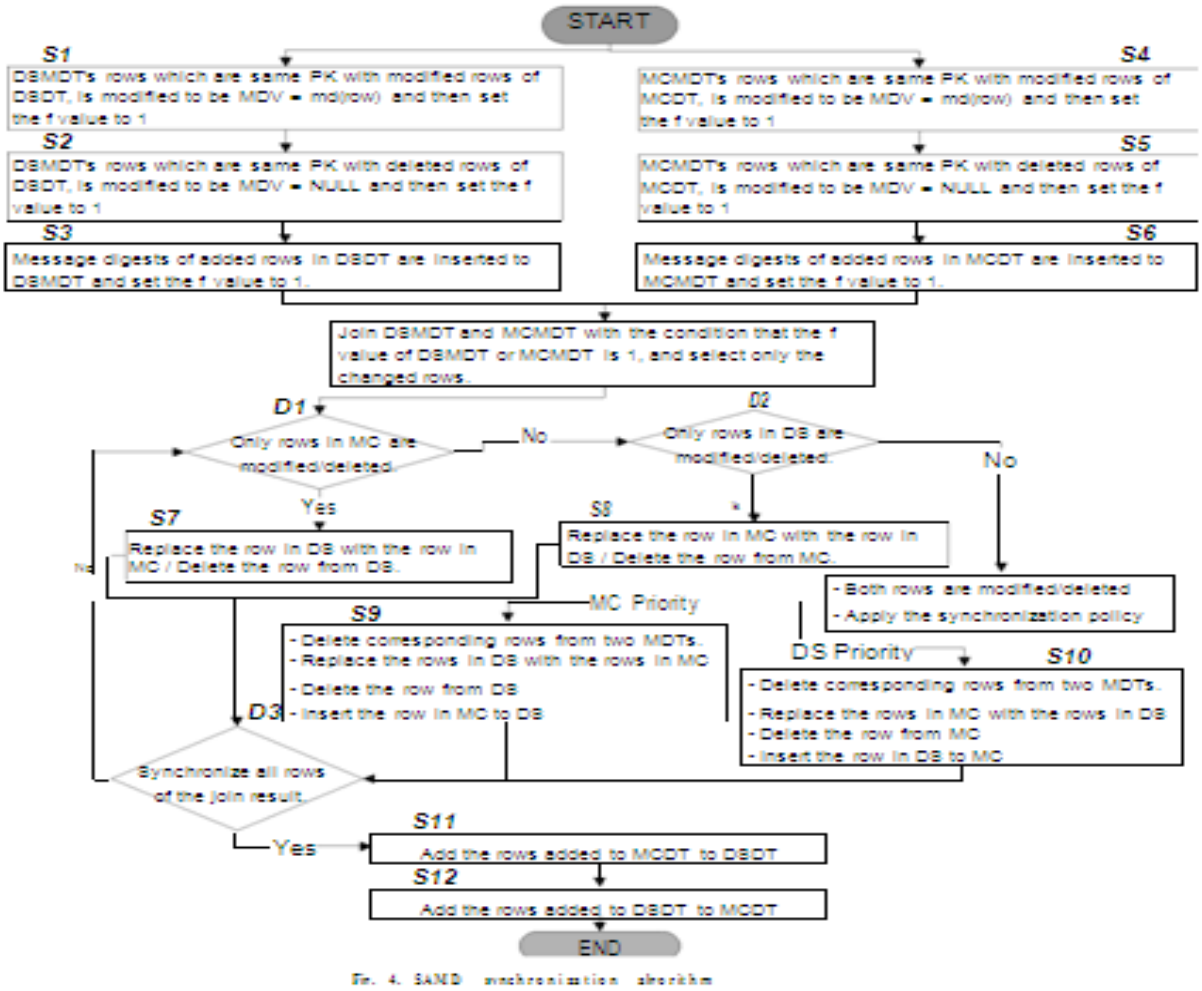

Fig. 4. SAMD synchronization algorithm

Steps S4~S6 indicate the Synchronization 1 stage of Fig. 3. This stage involves synchronizing the MCDT and MCMDT; the basic algorithm is identical to that used in Steps S1~S3. However, the MCDTT and MDMDT are internal data tables of different vender databases that are physically separated, so FullOuterJoin as in Steps S1~S3 is not feasible. In this case, a temporary table has to be created for the MCDT table to be copied to the server side, and Synchronization 1 process is performed, after which the copied data is deleted. This single transaction by batch processing guarantees independence of the SAMD algorithm for the database vender.

Steps S7~S12 display the Synchronization 3 stage of Fig. 3. When the DSMDT and MCMDT are FullOuterJoined, 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 the DSDT and MCDT is achieved.

Step S7 involves synchronizing a modified row or one deleted from the MCDT with the DSDT. Under the D1 condition, Step S7 searches for a row with an MCMDT flag value of 1 and a DSMDT flag value of 0. The flag values indicate that the row was modified or deleted from the MCDT. A null MDV column of the MDMDT signifies a deletion from the MCMT. Otherwise, there has been a modification. In the case of deletion, rows that correspond to the rows deleted from the MCDT should be deleted from DSDT, DSMDT and MDMDT. In the case of modification, the DSDT row value is replaced with the MCDT row value, and DSMDT row value is replaced with the MDMDT row value. Upon completion of synchronization, the flag values of the synchronized rows of the DSMDT and MCMDT are set to 0. This process resolves the inconsistency cases C3 and C4.

Step S8 involves synchronizing the modified or deleted rows from the DSDT with the MCDT. This step is identical

to the algorithm of Step S7, but synchronization takes place from the DSDT towards the MCDT. Upon completion, this process resolves the inconsistency cases C9 and C13.

When modification or deletion occurs in both DSDT and MCDT, Steps S9 and S10 perform synchronization in the direction from the DSDT towards the MCDT or in the reverse direction, according to the synchronization policy. The rows subject to synchronization are those with flag values of 1 for both the DSMDT and MCMDT. Four cases should be considered in Steps S9 and S10, as shown in TABLE II.

TABLE II
CASE ANALYSIS

Case	Mobile database	Server-side database
1	Deletion	Deletion
2	Modification	Deletion
3	Deletion	Modification
4	Modification	Modification

In Step S9, synchronization is performed form the MCDT towards the DSDT. In Case 1, identical rows of the DSDT and MCDT are deleted, so the corresponding rows of the DSMDT and MCMDT should be deleted as well. For Case 3, the rows that correspond to those deleted from the MCDT after completion of Case 1 should be deleted from the DSDT. Once the two cases are complete, the inconsistency cases C12 and C16 are resolved. Cases 2 and 4 modify the row in DSMDT with the modified row of the MCMDT. For Case 2, since the row in the DSDT is deleted, the MCDT row value is inserted into the DSDT. For Case 4, since the row in the DSDT is modified, the corresponding row in the DSDT is replaced with the MCDT row value. Once the two cases are complete, the inconsistency cases C11 and C15 are resolved.

For Step S10, synchronization takes place from the DSDT towards the MCDT. The algorithm is identical to that used in Step S9 with a different synchronization direction. Upon completion of Step S10, the inconsistency cases C15, C16, C12 and C11 are resolved.

Step S11 involves reflecting the row inserted into the MCDT to the DSDT. Step S11 is applied to the row for which the flag value of the MCMDT is 1 and which is a dangling row. The rows inserted into the MCDT and MCMDT are also inserted into the DSDT and DSMDT. Completing the process resolves the inconsistency case C2.

Step S12 reflects the row inserted into the DSDT on the MCDT. The algorithm is identical to that used in Step S11 but with a different synchronization direction. Completion of this step resolves the inconsistency case C5.

Executing the SAMD algorithm resolves all of the inconsistencies listed in TABLE I through the synchronization process. Therefore, it can be concluded that the SAMD algorithm synchronizes every possible form of inconsistency.

4. IMPLEMENTATION AND EVALUATION

4.1. Performance Evaluation

The SAMD algorithm was implemented using JAVA and were linked with databases using JDBC (Java Database Connectivity). The commercial synchronization solution, which is vender-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, 5000 randomly generated rows were inserted into RDBMS, sent through synchronization and the same 5000 rows were inserted into mobile database. Then, we inserted 100 rows into RDBMS, and then inserted another 100 rows into mobile database. Afterwards, 100 rows were deleted from mobile database and another 100 rows modified. The modified and deleted data were programmed to be equally spaced among the 5000 rows. In this condition, performance was evaluated by comparing the synchronization time of SAMD and the commercial synchronization solution. We evaluated synchronization time ten times using different data in the same conditions and calculated the average. As shown in TABLE III, the performance evaluation result indicates that SAMD is faster than the commercial synchronization solution by an average of 0.64 seconds.

TABLE III
PERFORMANCE RESULTS

	SAMD	Commercial	Difference
Avg.	1.55	2.18	0.64

(Unit: Seconds)

For the commercial synchronization solution, a synchronization SQL or a JAVA script must be written [11]. 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 vender of the commercial synchronization solution [11]. There are no such requirements for the SAMD algorithm. Whereas the vender 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 inexpensive and scalable, this is not a major issue. SAMD only offers implementation of the essential elements for synchronization and not supports various other aspects required for a commercial product. In addition, it lacks sufficient generality to be applicable in an actual working environment. Therefore, it may not be completely appropriate to compare SAMD with a commercial product. Under the circumstances, however, SAMD does not fall far behind the commercial synchronization solution and there are the advantages explained above including user friendliness, simple synchronization and complete vender independence.

4.2. Quality Evaluation

A SAMD algorithm was designed to maintain the objective of algorithm as mentioned in part *A* of Section III. The mentioned objectives are problems that existing commercial synchronization products have, so if these objectives are not achieved, the wide use property for application of synchronization algorithm would not be fulfilled. The widely used technologies for mobile database synchronization are timestamp & snapshot technique, using staging table in middle-tier, integrated RDBMS which use message and before-image technique. The commercial synchronization solutions which use these technologies were compared with SAMD on the basis of mentioned algorithm objectives. TABLE represents algorithm designing objectives and maintaining or not as to objectives of each technologies/SAMD.

TABLE IV
QUALITY EVALUATION

Objective	Timestamp & Snapshot	Staging Table[5]	Integrated DBMS	SAMD
01	○	X	X	○
02	X	X	X	○
03	○	○	○	○
04	○	○	○	○

5. CONCLUSION

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 venders and is available for use in combination with any server-side databases and mobile databases. Therefore, extensibility, adaptability and flexibility are guaranteed when a mobile business system is authorized. This feature is important in order to build efficient mobile business systems because the upcoming mobile business environment has heterogeneous characteristics in which diverse mobile devices, mobile databases and RDBMS exist.

REFERENCES

[1] Sang-ouk Kim, Se-Bong Oh, Sung-Young Son, Jin-Ho Lee, "The framework for synchronization in embedded database environment.", Journal of Computing Science and Engineering, Vol.20, Num.7, pp. 14-21, 2002.Tomasz Imielinski and B. R. Badrinath, "Mobile wireless computing: challenges in data management", Communications of the ACM, Volume Issue 10, pp. 18-28, 1994.

[2] Barbara, D., "Mobile Computing and Databases - A Survey", IEEE Transactions on Knowledge and Data Engineering, Vol. 11 No. 1, pp 108-117, 1999.

[3] EPFL, U. Grenoble, INRIA-Nancy, INT-Evry, U. Montpellier, "Mobile Database: a Selection of Open Issues and Research Direction", SIGMOD Record, Vol.33, No.2, pp.78-83, June, 2004.

[4] My-Sun Choi, Young-Guk Kim, "Introduction of mobile database and research status", Journal of Database Research, Vol.17, Num.3, pp. 3-16, 2001.

[5] Joshua Savill, "MobiLink Synchronization Profiles", A Whitepaper from Sybase iAnywhere., October 17th, 2008

[6] Thomas Fanghänel, Jonas S Karlsson, Cliff Leung, " DB2

Everyplace Database Release 8.1: Architecture and Key Features", Datenbank-Spektrum, pp. 1~15, 5/2003

[7] Gye-Jeong Kim, Seung-Cheon Baek, Hyun-Sook Lee, Han-Deok Lee, Moon Jeung Joe, " LGeDBMS: a small DBMS for embedded system with flash memory", 32nd international conference on very large data bases, pp. 1255~1258, 2006

[8] john E. Canavan, "Fundamentals of Network Security", ARTECH HOUSE, INC., 2001, 61~62

[9] Jonathan Knudsen, "WIRELESS JAVA : Developing with Java 2, Micro Edition", A press, 2001, pp. 155.

"MobiLink Synchronization User's Guide", Sybase, Inc., 2004, pp. 37 ~ 68, pp. 392~402

BIOGRAPHIES

Assistant professor , department of computer science and engineering, Latha mathavan Engineering college

Assistant professor , department of Information Technology ,PSNA college of Engineering and Technology

