Evaluation of Lump-sum Update Methods for Nonstop Service System

Tsukasa Kudo†, Yui Takeda‡, Masahiko Ishino*, Kenji Saotome**, and Nobuhiro Kataoka***

†Faculty of Comprehensive Informatics, Shizuoka Institute of Science and Technology, Japan
‡Mitsubishi Electric Information Systems Corporation, Japan
* Department of Management Information Science, Fukui University of Technology, Japan
** Hosei Business School of Innovation Management, Japan
*** Interprise Laboratory, Japan
kudo@cs.sist.ac.jp

Abstract - In many mission-critical systems, the lump-sum update of large amounts of data is performed. On the one hand, with the development of internet business, nonstop online services have become to be provided in many mission-critical systems. So, the lump-sum update has to be performed concurrently with the online entry. However, in the actual mission-critical systems, there are various kinds of lump-sum update operations corresponding to their business. In this paper, we define the lump-sum update models from the point of view of both the actual business process and characteristics of the target data, and show the problem of the conventional update methods. Then, we propose a novel update method for this problem, which utilizes the transaction time database, and show its evaluation results of the efficiency of both the lump-sum update and online entry comparing with the conventional update method. Based on these results, we show the proposal method is effective in the case where the update data is related to each other.

Keywords: Database, batch processing, mini-batch, transaction, mission-critical system, nonstop service.

1 INTRODUCTION

In many mission-critical systems, their databases are usually updated by two methods. The first is entries from online terminals (hereinafter "online entry") such as ATM (Automatic Teller Machine) in a banking system, which is performed at any time in the online service time zone and its result is immediately reflected in the database. Because the online entries are performed concurrently by many users, their ACID properties are maintained by the transaction processing based on the lock function of the database. So, the result becomes as if they were performed in a certain order.

The second is the lump-sum update of large amounts of data in the database. For example, large amounts of account transfer in a banking system, which is entrusted by a company, is performed as a lump-sum update. This process is not required rigorous immediacy, so it is performed at the designated time by the system administrator. Therefore, in the old days, it was performed as the night batch to avoid the online service time zone by the method locking the whole target data and updating them in a lump (hereinafter "batch update"). However, in recent years, the electronic commerce has been expanding due to the progress of the internet business, and many systems have become to provide the nonstop online service, such as above-mentioned ATM. As the result, it has become necessary that the batch update is performed concurrently with the online entry.

On the other hand, the mini-batch has been put to practical use, which divides the lump-sum update into small update units to reduce the individual lock time and performs them sequentially [2]. However, because the mini-batch updates data one after another, the state on the way of the update is queried, in which some data is not updated yet and the other is already updated. That is, the ACID properties of the transaction are not maintained as the whole mini-batch processing.

Here, in our previous study on query methods in a mission-critical system, we showed that there are various kinds of batch operations and the appropriate method should be adopted for each case [3]. This suggests that the various requirements exist for the lump-sum update process according to the business operations, too. So, in this paper, we focus on a local government system as an example of the mission-critical systems and define the lump-sum update models from the point of view of both the actual business process and characteristics of the target data, in which lump-sum update is divided by both the conflict with the online entry and relevance between the update data. Then, we show the requirement of the lump-sum update method for each update model. And, in some case where it is performed concurrently with the online entry, we show there is the problem that the consistency of the data cannot be maintained by the conventional update methods.

For this problem, we propose a novel update method to maintain the ACID properties even in the above-mentioned case. It utilizes the transaction time database, which is a kind of temporal database and supports the record management on the transaction time when some fact existed in the database [4]. Hereinafter we call this method "temporal update". Moreover, we evaluate the efficiency of both the lump-sum update and online entry about the following update methods by developing prototypes: the batch update, the mini-batch and the temporal update. Based on these results, we show the database can be updated in the practical efficiency by the temporal update method, even for the update model that was challenging for the conventional methods. In addition, we show that an appropriate method has to be adopted based on the business operations, for not only the lump-sum update but also the online entry.

In Section 2, we define the lump-sum update models from

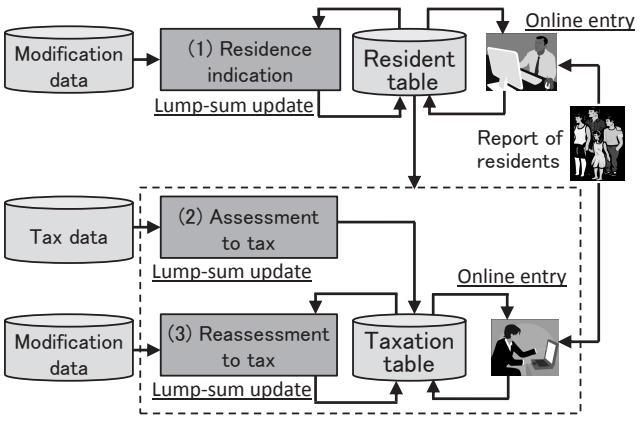

Figure 1: Business processes with lump-sum update

the point of view of the business operations, in Section 3 we propose the temporal update method and in Section 4 we show the prototype to evaluate its efficiency and characteristic. In Section 5, we evaluate the update methods from the point of view of update models, and in Section 6 we show our considerations.

2 MODELING OF LUMP-SUM UPDATE IN MISSION-CRITICAL SYSTEMS

2.1 Business Process with Lump-sum Update

In the actual mission-critical systems, there are various kinds of lump-sum update processing corresponding with each business process. Figure 1 shows examples of them about the tables in a local government system, which is updated also by online entries. We show the requirement of the lump-sum update based on these cases below.

Resident table of Fig. 1 stores the data of resident cards, which is used by the various business of the local government office for the attribute information of the resident: name, address and so on. Here, because residents belong to each household, the consistency of the resident data in the same household has to be maintained. In addition, as for a resident, since a series of records from birth to death and so on is managed, the consistency among the records also has to be maintained. In the online entry for this table, the change of the resident such as moving and birth is reflected in the table immediately. And, if the resident requests his or her resident card simultaneously, it is published immediately reflecting the change. On the other hand, for the example of the lump-sum update of this table, the residence indication is given. This business is performed to change addresses to be easy to understand, so it is performed in the whole target district at the same time. That is, since a great deal of data is updated for this business process, it is performed by the lump-sum update in the local government system.

Similarly, Taxation table in Fig. 1 stores the taxation data of the residents. There is no correlation among the data, because taxation is performed for each resident individually. Since the taxation is managed by the fiscal year, the assessment to tax is performed to add the tax data of the target year at first.

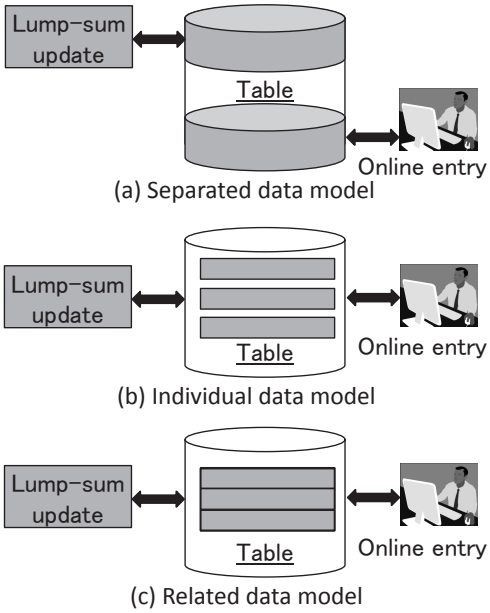

Figure 2: Lump-sum update models about business

Here, several tax declarations from residents are late for this assessment, and several changes of residents also occur after it. So, the reassessment to tax is performed at regular interval to correct the taxation. Since these business processes are performed for a large number of residents, they are executed by the lump-sum update. On the other hand, when a resident is going to move out, his or her taxation is calculated based on the change by the online entry at the report window of the local government office and reflected in Taxation table immediately. On this basis, the settlement of tax is performed at the same time.

2.2 Lump-sum Update Model about Business

The lump-sum update during the online entry, which is shown in Fig. 1 from the viewpoint of the business processes, corresponds to the following three types of lump-sum update models from the viewpoint of the update data, which is shown in Fig .2. Here, from the viewpoint of the business requirement, we assume that the online entry can update optional target data at the optional time and the update cannot be predicted beforehand. In other words, since it is the business of the report window about residents, the online entry cannot be suspended even during the lump-sum update.

(a) **Separated data model:** the case that the lump-sum updated data and online entry data are isolated as the business process. It corresponds to "(2) Assessment to tax" in Fig .1. In this case, the lump-sum update can be executed without considering the online entry. As the other example of this case, there is the business process to append the budget data of the new fiscal year in accounting systems.

(b) **Individual data model:** the case that the lump-sum update and online entry are concurrently executed on the same data, and this data independent from the other data.

It corresponds to "(3) Reassessment to tax" in Fig .1. In this case, these updates don't effect to the other data, even if there are conflicts between the lump-sum update and online entry. As the other example of this case, there is the process of a great deal of the account transfer in banking systems.

(c) **Related data model:** the case that the lump-sum update and online entry are concurrently executed on the same data, which is related to the other data. It corresponds to "(1) Residence indication" in Fig .1. As for the residents' information of the same household and the records of each resident, their consistency has to be maintained before and after the update. On the other hand, the change of a resident is processed by online entry: transference between households by moving, addition to a household by moving in and so on. Therefore, the lump-sum update has to be processed as a transaction that satisfies the ACID properties for online entries.

2.3 Problem of Conventional Lump-sum Update Method

The row lock function is provided by present database management systems, by which each single data of the table can be locked [5]. So, as for "(a) Separated data model" of Fig .2, we can execute the lump-sum update without affecting the online entry by locking only its target data, because the target data is not covered by online entry. So, it can be executed as the transaction processing by the batch update as follows: its commit is executed if the update succeeded; its rollback is executed if the update failed. In addition, in this model, the mini-batch can be also used for this lump-sum update updating data sequentially, because its target data is not covered by online entry. However, in this method, when the update failure occurred, it is necessary to perform the separate compensating transaction to cancel the whole update [2].

As for "(b) Individual data model" of Fig .2, the online entry becomes a waiting state when it competes with the lump-sum update, because the both may update the same data. So, the lump-sum update is executed by the mini-batch, because the batch update suspends the online entry for a long while. It is the method to update data one after another using the row lock function, which locks the currently updated data only, and it makes the influence on the online entry smaller because the update time of the individual data is short [2]. However, it performs the commit to each update. So, even though the failure occurred and the rollback was executed, the data already committed remains in the state of having been updated. That is, the committed data cannot be canceled in this method, because it may have been already used by the online entry. So, it is necessary to complete all the updates finally with removing the cause of the failure and continuing the update process.

On the other hand, as for "(c) Related data model" of Fig .2, it is difficult to update data by these conventional methods. First, as for the batch update, when the target data is being tried to update by the online entry concurrently, it obstructs the online entry in the same way as the individual data model. Next, as for the mini-batch, the ACID properties cannot be

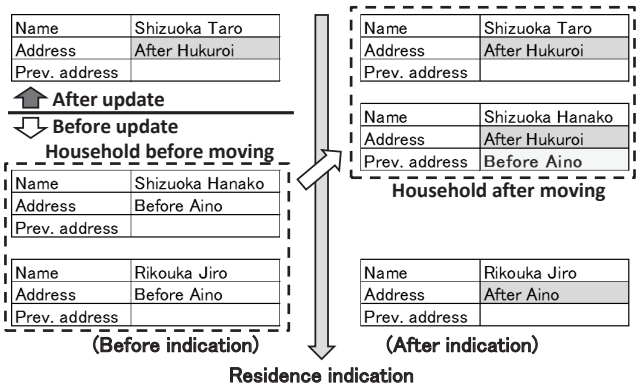

Figure 3: Online entry example of the related data model

maintained as the whole lump-sum update because each update is executed as the individual transaction, although its influence on the online entry is small. Therefore, as for the related data model, there is a problem that the integrity of data isn't able to be maintained by the conventional lump-sum update methods.

For an example of this, we show the case of a resident transference between households by moving, during the residents indication processed by the mini-batch in Fig .3. Here, the household that he or her belonged before this moving is not updated yet by the resident indication; the household after this moving was already updated. On the other hand, both of the present address and previous address are listed in the resident card. And, when this moving is processed by online entry, both of the before and after moving household data is locked by the transaction. However, because only the after moving household data has been updated, two types of addresses are listed in the resident card of this resident at the same time: the previous address is before the update; the present address is after the update. Thus, the problem that the integrity of the data isn't maintained occurs.

3 PROPOSAL OF A NOVEL LUMP-SUM UPDATE METHOD

For the problem shown in Section 2.3, we propose a novel update method, that is, temporal update method. It utilizes the transaction time database that is a kind of temporal database.

In the transaction time database, the time history that some fact was valid in the database is managed. The data once stored in the database is not deleted physically, and the time when the data became invalid is set to delete the data logically. The relation [1] of the transaction time database is expressed as $R(K, T, D)$. First, attribute K expresses the set of attributes constituting the primary key of the snapshot queried at the designated transaction time. Second, T is the time period attribute of the transaction time, which is generated by the system and isn't made public to users. T is expressed by the time set $\{T_a, T_d\}$: T_a shows the addition time that data was added to the database; T_d shows the deletion time that data was logically deleted from the database. As long as the data doesn't be deleted yet, the instance of the attribute T_d is expressed by "now", which shows the current time and

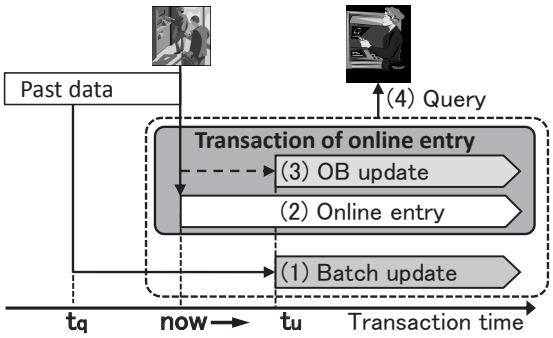

Figure 4: An example of temporal update method

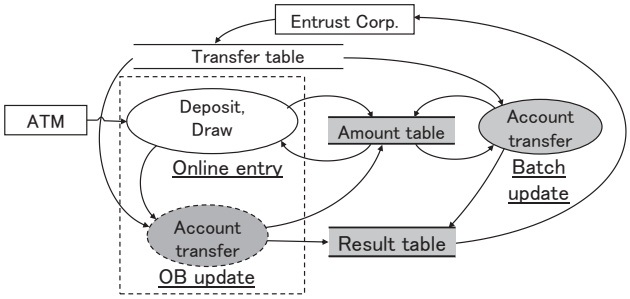

Figure 5: Dataflow of prototype

changes with the passage of time [7]. Third, D expresses the other attributes. Therefore, since the time history is managed, the snapshot at any designated past time can be queried. Moreover, the query result for a designated transaction time t becomes the snapshot of the time, and it is similar to the usual database that is called a snapshot database.

For the proposal method, We extend K by adding P, which shows the update process: the online entry, batch update and so on. So, the configuration of K is expressed by the attribute set $\{K_1, K_2, ..., K_n, P\}$. Here, n is the number of attributes except P. Figure 4 shows an example of this method, in which the account transfer is executed by the lump-sum update during the online entry from ATM in the banking system. Here, the time period of the lump-sum update process is between t_q and t_u. Using the transaction time database, the integrity of the snapshot result at the past time t_q can be maintained even during the online entry, because it updates the data at now.

As shown in (1) of Fig. 4, we perform the account transfer using this snapshot by the lump-sum update, and add the updated result to the database as the data which addition time is t_u. Here, since this data is separated from the online entry data by the above-mentioned primary key attribute P, we can add it by the batch update in the same way as (a) of Fig. 2. On the other hand, data is updated by the online entry from the ATM concurrently with this update as shown by (2). However, as shown in Fig. 3, since the batch update result is not reflected in the online entry, the process of the account transfer has to be executed individually in the same transaction of this online entry as shown by (3). Hereinafter, we call this process "OB update". Thus, since three types of data are added by different update process classified by P, the valid data is sorted out in the query process (4).

Briefly, in the temporal update, if the online entry is executed during the batch update, the process of the later is also executed individually as the OB update in the same transaction of the former. Incidentally, the OB update continues until the completion time t_u, because t_u have to be set previously.

4 EXPERIMENTS

4.1 Composition of Prototype

To confirm that we can put the temporal update method to practical use, we constructed the prototypes of both this method and conventional methods, which are the mini-batch and batch update, and evaluated their efficiency and characteristics. The prototype intends for the processing of a banking system shown in Fig. 4, and we show its data flow in Fig. 5. That is, the withdrawals and deposits to the bank accounts from the ATM are processed by the online entry to update the balance of "$Amount\ table$". On the other hand, large amounts of the account transfers, which are ordered by the trust company, are processed by the lump-sum update. Basing on the bank account and debit of "$Transfer\ table$", this process updates the balance of $Amount\ table$ and adds its result to "$Result\ table$". Each table is expressed by the following relations. Here, $Transfer\ table$ doesn't need to be the transaction time database, because it isn't updated.

$$Amount\ table(Account, Balance, T, P)$$
$$Result\ table(Account, Result, T, P)$$
$$Transfer\ table(Account, Debit)$$

Here, each attribute shows the following data: "$Account$" shows the bank account; "$Balance$" shows the bank balance of it; "$Result$" shows the result of account transfer from the bank account; "T" and "P" shows what described above. Incidentally, the instance set of P is as follows.

$$P = \{Batch\ update, Online\ entry, OB\ update\}$$

As for the bank account which account transfer is successful, $Balance$ of $Amount\ table$ is updated, and the result data is added to $Result\ table$, of which $Result$ is "0" (success). On the other hand, as for the bank account which doesn't exist or doesn't have sufficient balance, $Amount\ table$ isn't updated, and the result data is added to $Result\ table$, of which $Result$ is "1" (failure).

That is, the process of this lump-sum update was so complex that we implemented its prototype by Java, because it varies depending on the bank account presence, account balance and debit. And, we used MySQL for the DBMS (database management system); its storage engine InnoDB for transaction feature; JDBC to access the database with the row lock from Java.

We show the procedure of each lump-sum method below.

(1) **The mini-batch:** the row lock with the update mode is executed before each update of $Amount\ table$, and its commit is executed every specified update number. In this experiment, we used 1 and 80 for this number.

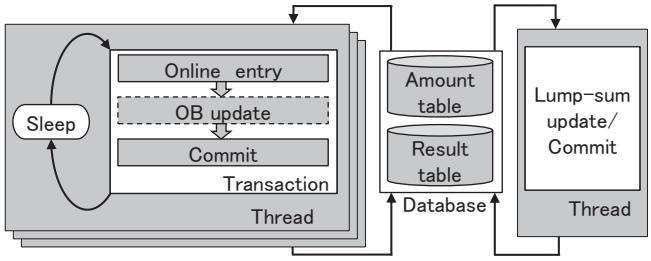

Figure 6: Program composition of prototype

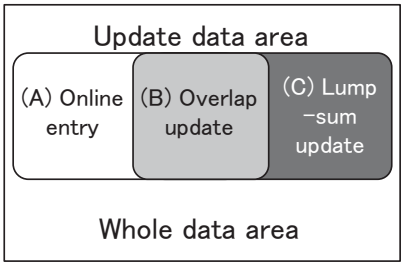

Figure 7: Evaluation data about deterioration of efficiency

(2) **The batch update:** at first the row lock on all the target data with the update mode is executed; and then, the lump-sum update by executeBatch statement of Java and the commit at the end are executed.

(3) **The temporal update:** though the target data isn't locked specifically before the batch update of this method, the added data is locked as the row lock until the commit by the InnoDB feature. Here, the commit is executed after the last addition. Incidentally, as shown in Fig. 4, the corresponding OB update is executed in the online entry transaction.

4.2 Experimental Environment

We performed this experiment by the Core i5 PC (Windows 7) in a stand-alone environment with MySQL5.1.40 and InnoDB. Here, we set InnoDB as follows: the isolation level is Repeatable read; "innodb_locks_unsafe_for_binlog" of startup option is "1" (enabled) to suppress the next-key lock [6].

We simulated the behavior of this prototype using thread programs of Java as shown in Fig. 6. That is, for the online entry, plural thread programs are executed to simulate the concurrent processing from multiple terminals. Here, the execution interval of each terminal was set to 0.5 second to simulate the load of practical environment. That is, supposing that the actual online entry interval of each terminal is 30 seconds, 16 terminals simulate the load by about 1000 terminals that are 60 times of 16 terminals. We used "sleep" method for this process. And, the commit was executed at every processing, and the OB update was also executed between the online entry and commit during the temporal update.

As the data environment, we stored 100 thousand data in $Amount\ table$, and performed 80 thousand of account transfer by the lump-sum update, such that those all succeed. On the other hand, to evaluate the deterioration of efficiency by

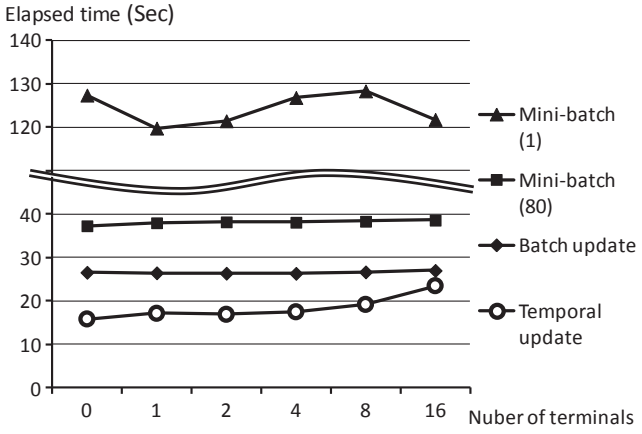

Figure 8: Elapsed times of each lump-sum update methods

Table 1: Elapsed time of lump-sum update (Sec)

Method	No-conflict	Half conflict
Mini-batch (1)	121.8	133.3
Mini-batch (80)	38.6	40.1
Batch update	27.0	29.0
Temporal update	23.5	22.4

the competition between the online entry and lump-sum update, we set the data of $Transfer\ table$ so that the data of $Amount\ table$ is classified as shown in Fig. 7 as follows: (A) updated by only the online entry, (B) updated by both of the online entry and lump-sum update and (C) updated by only the lump-sum update. We change their number based on each experimental purpose.

5 EVALUATIONS OF LUMP-SUM UPDATE METHODS

To evaluate the efficiency of each update method, we executed them without conflicts with the online entry, that is, there is no overlap update data area shown at (B) in Fig. 7. Figure 8 shows their elapsed time. Incidentally, the elapsed time is not the transaction time T but the real time measured by "currentTimeMillis" method of "System" class of Java. Its horizontal axis shows the number of online entry terminals, that is, the number of thread programs executed concurrently. Here, the case that only the lump-sum update was executed is shown at "0" of the scale. As shown in Fig. 8, the elapsed time of the mini-batch to commit at every update (hereinafter "mini-batch (1)") is more than 3 times the mini-batch to commit at every 80 update (hereinafter "mini-batch (80)"), and it is about 5 times the batch update. In addition, the temporal update is most efficient, but the elapsed time become long gradually with increasing the number of terminals. It is considered that this is an influence of the OB update shown in Fig. 6, which is performed only in the temporal update process. We discuss this in Section 6.1.

To evaluate the efficiency and characteristics of the lump-sum update and online entry in the case of their conflict, we

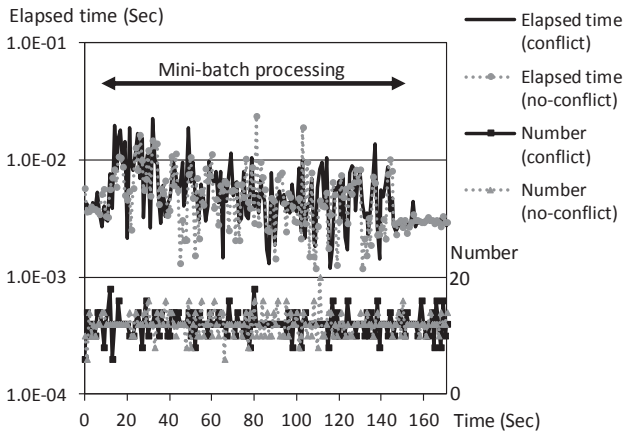

Figure 9: Efficiency of online entry during mini-batch (1)

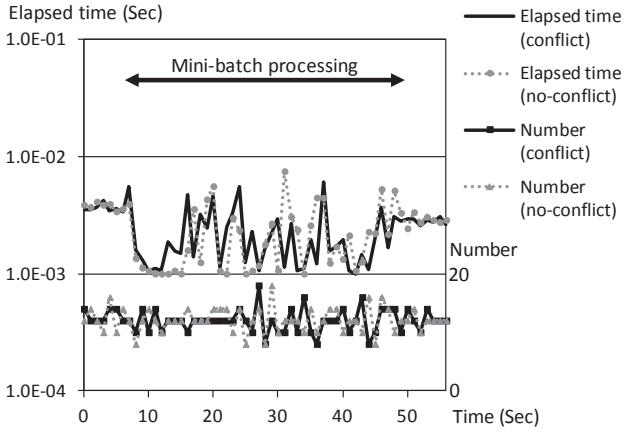

Figure 10: Efficiency of online entry during mini-batch (80)

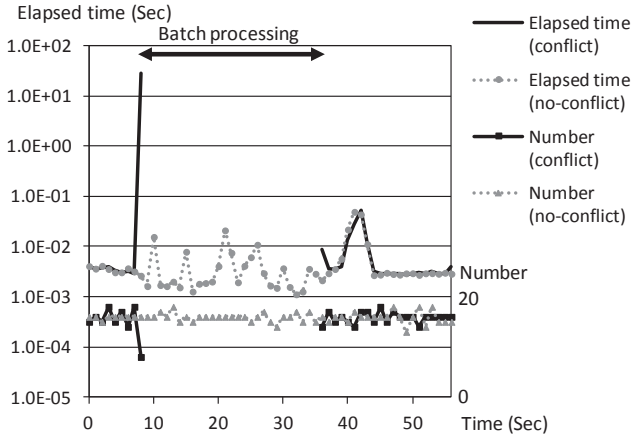

Figure 11: Efficiency of online entry during batch update

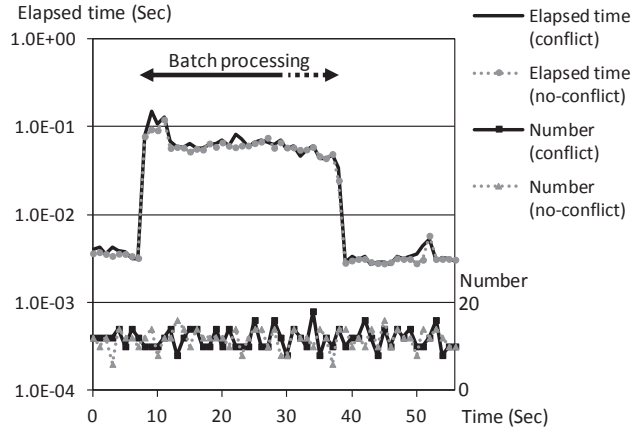

Figure 12: Efficiency of online entry during temporal update

executed them as following: the number of online entry terminals is 16; 8 of them conflict with the lump-sum update as shown at (B) in Fig. 7; the other doesn't as shown at (A). As for the conflicting data, to avoid a deadlock, it was updated in ascending order of bank account by both of the lump-sum update and online entry. Table1 shows the elapsed time of each lump-sum update method in both of the following case side by side: in the left side, there is no conflict as shown at (A) in Fig. 7, and it corresponds to the data which number of terminals is 0 in Fig. 8; in the right side, half of the terminals cause the above-mentioned conflict. As for the temporal update, both of the elapsed time is similar, whereas the other lump-sum update methods take more time in the case of the conflict. Therefore, the elapsed time of the temporal update is also least in the case of the conflict.

Next, from Fig. 9 to Fig. 12 show the efficiency of online entries conflicting with each lump-sum update method as for both the elapsed time and number of starting transactions per second. Here, the elapsed time is the average time of the update starting at the corresponding time. The left vertical axis of each figure shows the elapsed time by the logarithmic scale and the data is divided as follows: the data of terminals with conflict; the data of the other terminals without conflict (shown "no-conflict" in these figures). Similarly, the right axis shows the number of starting transactions. In addition, these figures show the time zone of the lump-sum update. Here, as for the temporal update, the completion time t_u of Fig. 4 is set beforehand and the OB update continues until t_u. So, its time zone of the batch update is shown by the solid line, and its OB update after the batch update is shown by the broken line. Incidentally, the elapsed time of the temporal update, which is shown in Table1 and so on, corresponds to the time zone of this batch update shown by the solid line.

The elapsed time of online entries is fluctuating during the execution of the mini-batch or batch update, and it is the least in the mini-batch (80). On the other hand, as for the batch update, the online entries, which conflict with it, is waited until its completion. As for the temporal update, though no online entry waited for a long while, the elapsed time of online entry transactions became more than 10 times. Because they include the OB updates. But, the elapsed time fluctuations of online entries are smaller than the other methods.

Table2 shows the evaluation about the lump-sum update models shown in Fig. 2. There are constraints of lump-update method to apply it to each model as shown in Section 2.3. As mentioned above, as for the separated data model and indi-

Table 2: Evaluations about lump-sum update model

Lump-sum update model	Online entry elapsed time	Lump-sum update elapsed time	Available method about the model
Separated data model	MB (80)	T	MB, B, T
Individual data model	MB (80)	T	MB, T
Related data model	T	T	T

(Notes) MB: Mini-batch; B: Batch update; T: Temporal update

vidual data model, the mini-batch (80) gives the least impact to the online entry. However, since it cannot be applied to the related data model, the temporal update method has to be applied to this model. Moreover, the elapsed time of the temporal update method was the least among the target lump-sum update methods of this simulation.

6 CONSIDERATIONS

In the actual mission-critical system, it is expected that there are various kinds of system operations and restrictions about both of the lump-sum update and online entry. In this section, we discuss the temporal update method based on the evaluation results in Section 5.

6.1 Efficiency of Temporal Update Method

The temporal update had the highest efficiency about the elapsed time of the lu -sum update in the above-mentioned experiment. The reason for this is because only the insertion of data is executed in the temporal update, whereas the batch update executes querying of the data to update it. And, as for the temporal update, since the commit is executed collectively after updates, the increase of the load by the commit was suppressed as well as the batch update comparing with the mini-batch. Moreover, as for the temporal update, the updated data isn't queried until the completion time t_u even if its commit is executed. So, in the case that the target data is increased, its update process can be executed one after another of dividing set with maintaining the ACID properties. That is, even in the case of extremely large number of updates, it is possible to apply the temporal update by executing them one after another.

In addition, the temporal update maintains its efficiency even in the case of conflict with the online entry as shown in Table1, whereas there are declines in the other methods. Its reason is because the other update methods have to wait for the lock completion of the online entries, whereas the temporal update method executes only the data insertion that doesn't need to lock them. By the way, since the online entry in the actual system operations updates data randomly, it often causes the deadlock with the batch update or mini-batch (80). On this point, the temporal update has an advantage, because it doesn't lock the data being updated by the online entry.

On the other hand, the elapsed time of the temporal update become longer with the increase of the online entry terminals as shown in Fig. 8. As mentioned above, there isn't the conflict between the lump-sum update and online entry. So, this is considered to be caused by the load of the OB update, to which the processing such as the update of $Result\ table$ is added. And, its adjustment is the future challenge.

Here, as for the temporal update method, its completion time needs to set beforehand, so some margin is necessary for it. But, we consider it is valid from the view point of efficiency not only for the related data model but also for the other models. For example, it is valid for the update that has to be processed in a short time and processing time can be estimated beforehand. That is, the appropriate update method should be selected based on the business requirements.

6.2 Online Entry Method

Since the OB update is executed in the temporal update, the online entry takes longer time than in the mini-batch as shown in Fig. 9, 10 and 12. Though the adjustment of this part is the future challenge as mentioned above, this time is within about 0.1 seconds, which is different from the wait time for the lock in the batch update as shown in Fig. 11. Therefore, it is considered that we can apply it to actual mission-critical systems within a certain range of load even under the present condition.

As for the mini-batch (80), since it updates the plural data collectively, the online entries have to complete in a short time. That is, when its target data is locked by an online entry, its update process has to wait with locking the other data updating collectively. Moreover, it makes the other online entries that try to update these data await state. In other words, there is the problem that the conflicts may spread to the unrelated online entries. Therefore, the online entry transaction cannot include the processing that needs a long while, such as waiting for user input. This problem is similar as for the temporal update, because the data updated by the batch update become to be valid at the completion time t_u as shown in Fig. 4. That is, the following updates have to be serialized: the online update before t_u, the validation of the batch update and the online update after t_u.

Therefore, based on the requirement of the target business, the appropriate method has to be selected for not only the lump-sum method but also the online entry. For example, as for above-mentioned case, there are some choices. As for the mini-batch (1), it is appropriate for the case where the various kinds of online entry methods are used though the restriction on the lump-sum update time is loose. On the other hand, if the online entry time is short, the appropriate lump-sum method can be selected based on the requirement of the

business: efficiency, the lump-sum update models and so on. Incidentally, the batch update is not appropriate for the case of conflict with the online entry as shown in Fig. 11.

7 CONCLUSIONS

With the spread of nonstop online services caused by the development of the internet business, the lump-sum update has to be executed concurrently with the online entry in the mission-critical systems. In this paper, first, we showed the lump-sum update model from the view point of the businesses of mission-critical systems, and showed that the conventional update methods have the problem in the case to update data relating to each other. Second, we proposed the temporal update method for this problem, and showed it has the practical efficiency through the evaluations by the prototype. Third, we showed that it is necessary to select the appropriate method for both of the lump-sum update and online entry, based on the evaluations including both the proposal method and conventional methods.

Future study will focus on the implementation method of the temporal update for the actual mission-critical system, especially the improvement of the online entry response.

ACKNOWLEDGMENT

This work was supported by KAKENHI(24500132).

REFERENCES

[1] E. F. Codd, “Extending the database relational model to capture more meaning,” ACM Transactions on Database Systems, Vol. 4, No. 4, pp. 397–434 (1979).
[2] J. Gray, and A. Reuter, “Transaction Processing: Concept and Techniques,” Morgan Kaufmann, San Francisco (1992).
[3] T. Kudo, Y. Takeda, M. Ishino, K. Saotome, K. Mutou, and N. Kataoka, “A Correction Reflected Query Method of Database during Online Entry,” International Journal of Infomatics Society, Vol. 3, No. 1, pp. 3–11 (2011).
[4] R. Snodgrass, and I. Ahn, “Temporal Databases,” IEEE COMPUTER, Vol. 19, No. 9, pp. 35–42 (1986).
[5] ORACLE, MySQL Documentation: MySQL Reference Manuals, http://dev.mysql.com/doc/refman/5.5/en/index.html.
[6] ORACLE, MySQL Documentation: MySQL Reference Manuals (14.3.9.4. InnoDB Record, Gap, and Next-Key Locks), http://dev.mysql.com/doc/refman/5.5/en/innodb-record-level-locks.html.
[7] B. Stantic, J. Thornton, and A. Sattar, “A Novel Approach to Model NOW in Temporal Databases,” Proceedings 10th International Symposium on Temporal Representation and Reasoning and Fourth International Conference on Temporal Logic, pp. 174–180 (2003).

(Received October 13, 2012)

Tsukasa Kudo received the M. Eng. from Hokkaido University in 1980 and the Dr. Eng. in industrial science and engineering from Shizuoka University, Japan in 2008. In 1980, he joined Mitsubishi Electric Corp. He was a researcher of parallel computer architecture, an engineer of application packaged software and business information systems. Since 2010, he is a professor of Shizuoka Institute of Science and Technology. Now, his research interests include database application and software engineering. He is a member of IEIEC, Information Processing Society of Japan and The Society of Project Management.

Yui Takeda received the B.E. from Keio University, Japan in 1987. In 1987, she joined Mitsubishi Electric Corp. She was an engineer of artificial intelligence and application software. Since 2001, she joined Mitsubishi Electric Information Systems Corp. Now, she manages intellectual property rights.

Masahiko Ishino received the master’s degree in science and technology from Keio University in 1979 and received the Ph.D. degree in industrial science and engineering from graduate school of Science and technology of Shizuoka University, Japan in 2007. In 1979, he joined Mitsubishi Electric Corp. Since 2009, he is a professor of Fukui University of Technology. Now, His research interests include Management Information Systems, Ubiquitous Systems, Application Systems of Datamining, and Information Security Systems. He is a member of Information Processing Society of Japan, Japan Industrial Management Association and Japan Society for Management Information.

Kenji Saotome received the B.E. from Osaka University, Japan in 1979, and the Dr. Eng in Information Engineering from Shizuoka University, Japan in 2008. From 1979 to 2007, he was with Mitsubishi Electric Corp., Japan. Since 2004, he has been a professor of Hosei business school of innovation management. His current research areas include LDAP directory applications and single sign-on system. He is a member of the Information Processing Society of Japan.

Nobuhiro Kataoka received the master’s degree in electronics from Osaka University, Japan in 1968 and the Ph.D. in information science from Tohoku University, Japan in 2000. From 1968 to 2000, he was with Mitsubishi Electric Corp. From 2000 to 2008, he was a professor of Tokai University in Japan. He is currently the president of Interprise Laboratory. His research interests include business model and modeling of information systems. He is a fellow of IEIEC.