Vol. 38 No. 10 Transactions of Information Processing Society of Japan Oct. 1997

Regular Paper

A Concurrency Control Algorithm Using Serialization Graph Testing with Write Deferring

HARUMASA TADA,† MASAHIRO HIGUCHI† and MAMORU FUJII†

Several scheduling algorithms for preserving the consistency of databases have been proposed. One of such algorithm is Serialization Graph Testing (SGT). Under SGT, a scheduler maintains a graph called a serialization graph (SG). Database consistency is preserved by ensuring that the SG is acyclic. The scheduler checks the acyclicity of the SG for every operation. If the SG comes to contain a cycle, the operation is rejected. It is known that SGT achieves higher concurrency than other scheduling algorithms. However, it has some drawbacks. First, operations are forced to wait for a long time so that the acyclicity of the serialization graph can be checked. Second, a phenomenon called *cascading aborts* may occur; that is, one abortion of a transaction may cause other abortions. To deal with the first drawback of SGT, we focused on the scheduling algorithm called SGT certification. Under SGT certification, a scheduler checks the acyclicity of the SG only once for each transaction, at its termination. Therefore, all operations are executed immediately, at the cost of a delay in cycle detection. As regards the second drawback, the scheduling method called Optimistic Concurrency Control (OCC) avoids cascading aborts by using internal buffer to defer substantial write operations. However, the consistency checking of OCC differs from that of SGT, and the concurrency of OCC is not so high. Therefore, we applied this write deferment approach to SGT certification. We call our algorithm Serialization Graph Testing with Write Deferring (SGT-WD). In this paper, we present the SGT-WD algorithm and show its correctness. We also evaluate SGT-WD, SGT, and SGT certification by means of simulations on distributed database systems. The simulation results show that SGT-WD is more effective than the others.

1. Introduction

Concurrency control in database systems is an important problem that has been studied by many researchers. We have studied the scheduling algorithm called *Serialization Graph Testing* (SGT) and proposed a scheduling algorithm for distributed database systems[8),9)]. In SGT, a scheduler maintains what is called a serialization graph (SG) and schedules operations while ensuring that the SG is acyclic. It is known that SGT schedulers achieve higher concurrency of transactions than other types of schedulers[2)]. However, there are two problems associated with SGT:

(1) The time required to check the acyclicity of the serialization graphs tends to become long, and each operation must wait until the checking is complete.

(2) Executions produced by SGT may cause the phenomenon called *cascading aborts*; that is, one abortion of a transaction may cause other abortions.

To deal with the first problem, we focused on a scheduling method called *certification*[2)]. Under certification, a scheduler permits all operations to execute immediately. When it is about to schedule a commit operation, it checks whether the execution that includes the commit operation is consistent. If the execution is inconsistent, some transactions are aborted. Since operations are executed immediately, the processing time of transactions is shorter than with other types of scheduler. On the other hand, conflicts cannot be detected until the transaction is about to commit. A certification algorithm that uses SGT to check the consistency of executions is called *SGT certification.*

For the problem of cascading aborts, we consider a certification algorithm proposed by Kung and Robinson[4)], called *Optimistic Concurrency Control* (OCC). Though most certification algorithms (including SGT certification) may cause cascading aborts, OCC avoids them. The unique feature of OCC is that it first performs write operations to internal buffers, and the values are then written to the actual database at the termination of the transaction that wrote them. Since the values in the buffers cannot be accessed by other transactions, write operations are deferred practically. The consistency checking of OCC differs from that of

† Faculty of Engineering Science, Osaka University

SGT, and its concurrency is not so high. We apply this feature of OCC to SGT certification and propose a new algorithm. Since, like OCC it defers substantial write operations, we call it *Serialization Graph Testing with Write Deferring* (SGT-WD).

In this paper, we present the SGT-WD algorithm and show its correctness. We consider that its merits are apparent especially in distributed database systems that need communications for scheduling. Therefore, we evaluate the performance of SGT-WD, SGT, and SGT certification by simulations on distributed database systems.

The paper is organized as follows. In Section 2, we briefly present basic definitions. Section 3 describes serialization graph testing. In Section 4, we give an overview of certification and optimistic concurrency control. Section 5 describes the basic algorithm of SGT-WD. A correctness proof of SGT-WD appears in Section 6. We evaluate the performance of SGT-WD in Section 7, and our conclusions appear in Section 8.

2. Preliminaries

2.1 Transactions and Histories

A transaction is an execution of a program that manipulates a database. In other words, a transaction is a partial ordered set of operations including database manipulations. Without loss of generality, we can assume that transactions never read data items written by themselves. Moreover, we assume that each transaction never reads (or writes) any data item more than once. Two operations of different transactions are said to *conflict* if they both operate on a same data item and at least one of them is a *Write*.

A concurrent execution of transactions is expressed as a partial ordered set of operations called a *history*. A history indicates the order in which transaction operations were executed relatively to each other. Since some of these operations may be executed concurrently, a history is defined as a partial ordered set of executed operations. For a history H, $<_H$ denotes the ordering relation of H. A transaction T_i is said to be *active* in H if T_i is started and neither committed nor aborted in H. A history that expresses a serial execution of transactions is called a *serial* history.

2.2 Serializability

We use the property called *serializability* as a criterion for the consistency of histories. Intuitively speaking, for a history H, if there is a serial history H_s that contains the same operations as H and the relative order of all pairs of conflict operations in H and H_s are the same, then H is called serializable (SR). The strict definition is given in Bernstein, et al. [2].

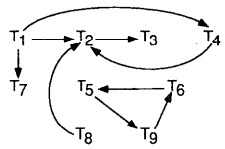

Fig. 1 An example of a serialization graph.

3. Serialization Graph Testing

3.1 Serialization Graph

The serializability of a history can be determined by analyzing a graph, called a serialization graph (SG), derived from the history.

Definition 1: For a history H, the *serialization graph* $SG(H) = \langle V, E \rangle$ is a directed graph such that
$V = \{T_i \mid T_i$ is a transaction that is already started and is not aborted in $H\}$
$E = \{(T_i, T_j) \mid$ there exist conflicting operations $o_i \in T_i$ and $o_j \in T_j$ such that $o_i <_H o_j$, where $T_i, T_j \in V\}$. □

Figure 1 shows an example of a serialization graph.

If the order of operations in each transaction is preserved in a history H, then the following theorem holds [2]:

Theorem 1: A history H is SR iff $SG(H)$ is acyclic. □

3.2 Serialization Graph Testing

SGT uses a serialization graph in order to verify the serializability of concurrent execution of transactions. In SGT, a scheduler maintains an SG throughout the execution of transactions. The scheduler behaves as follows. At first, the SG is empty (and of course acyclic). The scheduler receives an operation o of a transaction T. If a node for T does not yet exist in the SG, then the scheduler first adds the node to the SG. Then, it adds an edge from every T_j to T, where T_j includes a previously scheduled operation that conflicts with o. If the SG is still acyclic, then the scheduler can accept o and can schedule it immediately. Otherwise, the scheduler aborts T and deletes T and all edges incident with T from the SG. The aborted transaction T will restart later.

In order to detect conflicts of transactions,

for each transaction T, the SGT scheduler must maintain the sets of data items that have been respectively read and written by T. These sets are called the *readset* and *writeset* of T. For a transaction T, we define *readset*(T) and *writeset*(T) as follows.

readset(T): the set of data items that have been read by T

writeset(T): the set of data items that have been written by T

When an operation o of a transaction T reads (writes) a data item x, x is added to *readset*(T) (*writeset*(T)). For a transaction T_i such that $x \in$ *writeset*(T_i) ($x \in$ *readset*(T_i) $\cup$ *writeset*(T_i)), an edge from T_i to T is added to the SG.

The SGT scheduler must delete nodes and edges for committed transactions that are already unnecessary for scheduling, in order to avoid increasing the size of the SG. However, the details of the method for deleting unnecessary nodes from the SG are beyond the scope of this paper.

A precise description of SGT is omitted in this paper. For details, see Bernstein, et al.[2].

The major advantage of SGT is that its consistency checking is based strictly on the definition of serializability. Therefore, it achieves higher concurrency than other scheduling algorithms[2]. Most proposed scheduling algorithm are based on *Two-Phase Locking* (2PL)[3),5)~7),10)] or *Timestamp Ordering* (TO)[1]. Their concurrency is not so high as that of SGT.

On the other hand, SGT has some disadvantages. First, it takes time to check the acyclicity of the serialization graph. The time needed for the checking tends to be long, especially in distributed database systems[8]. In such systems, intersite communications are needed for the checking, because the serialization graph has a global structure. Each operation must wait until the checking is complete. Therefore, the processing time of transactions may become too long. We define the *processing time* of a transaction T as the time between the start of T and the commit of T (some abortions/restarts of T may be included). Second, there is a problem of cascading aborts. For example, consider two transactions T_1 and T_2. Suppose that T_1 has read a data item already written by T_2. If T_2 is aborted for some reason, then all its effects must also be wiped out. Of course T_1, which read a data item written by T_2, must be aborted and restarted. The phenomenon whereby aborting one transaction triggers further abortions is called *cascading aborts*[2]. Executions produced by SGT may cause cascading aborts.

4. Certification

4.1 Overview of Certification

To overcome the drawbacks of SGT, we focused on a scheduling method called *certification.*

Under many scheduling algorithms, every time a scheduler receives an operation, the scheduler decides whether to accept, reject, or delay the operation. Under certification, a scheduler immediately schedules each operation it receives. From time to time, it checks to see what it has done. If it concludes that all is well, then it continues scheduling. If it detects that it has inappropriately scheduled conflicting operations, then it aborts some transactions. In this way, operations are aggressively scheduled under certification, in the hope that no conflicts will occur. Therefore, the processing time of transactions is shorter than in other scheduling algorithms.

On the other hand, operations are scheduled even if they cause loss of consistency; this is not detected until the explicit check, which is usually done at the end of the transaction. Therefore, if conflicts occur frequently, the processing time of transactions may be much longer for certification than for other types of scheduling algorithm. Most certification algorithms are constructed as variants of conventional algorithms such as 2PL, TO, and SGT[2]. As in most scheduling algorithms, the executions produced by such certification algorithms may cause cascading aborts. However, there is another type of certification algorithm, called *Optimistic Concurrency Control* (OCC), that avoids cascading aborts.

4.2 Optimistic Concurrency Control

OCC was proposed by Kung and Robinson[4]. Like other certification schedulers, an OCC scheduler aggressively schedules operations. The unique feature of OCC is that it defers substantial write operations by using internal buffers. In OCC, an execution of a transaction T is divided into the following three phases:

Read phase: In this phase, all read operations are executed immediately, and are completely unrestricted. All write operations take place in internal buffers that can-

not be accessed by other transactions.

Validation phase: In this phase, a check is performed to determine whether the changes made by T will cause inconsistency in the database. If not, the validation is successful; otherwise, it fails.

Write phase: In this phase, the values in internal buffers are written into the actual database. At this time, the modification made by T become effective. T commits at the end of the phase.

A transaction T first enters the read phase. When T is about to execute a commit operation, it enters the validation phase. If the validation succeeds, T enters the write phase and is committed. Otherwise, T is aborted and restarted.

In OCC, in order to verify that serializability is preserved, the scheduler explicitly assigns each transaction a unique integer called *transaction number* $t(i)$ during the course of its execution. The meaning of transaction numbers in validations is as follows: there must exist a serially equivalent execution in which transaction T_i comes before transaction T_j whenever $t(i) < t(j)$. Transaction numbers are assigned at the end of the read phase. In the validation phase, the validation condition is checked. $readset(T)$ and $writeset(T)$ are defined as in Section 3.2. For each transaction T_j with transaction number $t(j)$, and for all T_i with $t(i) < t(j)$; one of the following three conditions must hold.

- T_i completes its write phase before T_j starts its read phase.
- $writeset(T_i) \cap readset(T_j)$ is empty and T_i completes its write phase before T_j starts its write phase.
- $writeset(T_i) \cap (readset(T_j) \cup writeset(T_j))$ is empty and T_i completes its read phase before T_j completes its read phase.

If none of above three conditions hold for some T_i, the validation of T_j fails. For more details, see Kung and Robinson [4].

As above, OCC checks the consistency according to overlapping of concurrent executions. The concurrency of transactions under OCC is not so high as under SGT.

OCC has two principal merits. First, operations are scheduled immediately. Second, a transaction can be aborted easily. Write operations are executed on the actual database only when the transaction commits. Therefore, when a transaction aborts, there are no data items modified by the transaction. Accordingly, no more abortions are caused by the abortion. That is, all executions produced by OCC are guaranteed not to cause cascading aborts. We call such executions *cascadeless*. It is also said that the executions *avoid cascading aborts*. The drawback of OCC is that, like other certification algorithms, it may cause many abortions when conflicts of transactions occur frequently.

5. Algorithm

5.1 The SGT-WD Algorithm

We apply the OCC approach to SGT and propose a new algorithm that overcomes the disadvantages of SGT. The important points are as follows:

- All operations are scheduled immediately and they are validated later. Therefore, the processing time of transactions is shorter than that of noncertification algorithms.
- Write operations are deferred until the validation is complete. As a result, executions produced by this algorithm are cascadeless.

We call our algorithm *Serialization Graph Testing with Write Deferring* (SGT-WD). We give the basic SGT-WD algorithm in **Table 1**. Like OCC, SGT-WD divides the execution of a transaction into three phases.

5.2 Using a Locking Scheme

Under scheduling algorithms using the SGT approach, when a transaction T reads or writes a data item x, the following processes are executed:

- Edges are added to the SG.
- x is added to $readset(T)$ ($writeset(T)$).
- x is read from (written to) the actual database.

If the above processes are interleaved with other conflicting operations, the SG may contradict with $SG(H)$. For example, suppose that a transaction T_1 tries to write a data item x. $writeset(T_1)$ has been updated, but the value of x has not been written into the actual database when another transaction T_2 reads x. The edge from T_1 to T_2 is then added to the SG according to $writeset(T_1)$. However, T_2 reads the value of the "old" x that has not been updated by T_1. The edge from T_1 to T_2 contradicts with the fact that T_2 reads x before T_1 writes it. To avoid such a case, we use a locking scheme. Before accessing a data item x, the readlock (writelock) of x is held by o to prohibit the execution of operations that conflict with o (line 2 of read phase and line 2 of validation phase).

Table 1 The basic SGT-WD algorithm.

```
when a transaction T is in the read phase
 1  if a read operation read(x) is received
 2     set the readlock of x to T
 3     for each T_i such that x ∈ writeset(T_i)
 4        add an edge T_i → T to the SG
 5     add x to readset(T)
 6     read x from the database
 7     release the readlock of x
 8  if a write operation write(x) is received
 9     store x to an internal buffer
       (do not add edges to the SG here)
10  if a commit operation commit is received
11     T enters the validation phase
when T is in the validation phase
 1  for each x_i which is stored in internal buffers by
    T.
 2     set the writelock of x_i to T
 3     for each T_j such that x_i ∈ readset(T_j) ∪
    writeset(T_j)
 4        add an edge T_j → T to the SG
 5     add x_i to writeset(T)
 6  check the SG to determine whether there is a cy-
    cle
 7  if there is a cycle then
 8     remove the node T and incident edges from
    the SG
 9     abort and restart T
10  else
11     T enters the write phase
when T is in the write phase
 1  for each x_i stored in internal buffers by T
 2     write x_i to the actual database
 3     release the writelock of x_i
 4  commit T
```

Then, the above processes are executed. The lock is released when o finishes reading (writing) x. The locking scheme used here is different from that used in 2PL. Under 2PL, the lock of a data item x is held by a transaction T when T accesses x, and is not released until T holds all the locks that are needed. Thus, the locking time depends on the internal processing of transactions. In the case of long-lived transactions, the locking time may be very long. On the other hand, under SGT-WD, the lock is held during the SG updating and the data accessing. The locking time is independent of the internal processing of transactions. Therefore, the influence of our locking scheme on the concurrency of transactions is much smaller than that of 2PL.

6. Correctness

In this section, we show the correctness of SGT-WD. In the case of the usual SGT, Theorem 1 gives the correctness of the algorithm. SGT-WD also uses the serialization graph for scheduling operations. Since an SGT-WD scheduler defers write operations of a transaction, the execution order of operations differs from the original transaction. Therefore, Theorem 1 does not hold immediately for SGT-WD.

Definition 2: For each transaction T_i in H, *a write-deferred transaction* T_i^{wd} is a transaction such that

- It has the same set of operations as T_i.
- Its read operations are executed in the same order as those of T_i.
- Its write operations are executed after the last read operation.
- It writes the same values to data items as T_i.

Note that T_i and T_i^{wd} read and write the same values. Therefore, we can say that T_i^{wd} is *equivalent* to T_i.

Theorem 2: Let H be a history produced by SGT-WD. $SG(H)$ is acyclic iff H is serializable.

[**Proof**] Although write operations are deferred in SGT-WD, the executed transaction writes the same value as the original transaction, as mentioned above. This means that an SGT-WD scheduler executes T_i^{wd} instead of T_i. Therefore, H is an execution in which T_i^{wd}s are executed concurrently. Let H_s^{wd} denote an execution in which such T_i^{wd}s are executed serially. For SGT-WD, Theorem 1 says that $SG(H)$ is acyclic iff there is an execution H_s^{wd} that is equivalent to H.

("if" part) Suppose that H is serializable. From the definition of serializability, there is a serial execution H_s that is equivalent to H. Since each T_i^{wd} is equivalent to T_i, H_s^{wd} can be obtained by replacing each T_i in H_s with T_i^{wd}, and is equivalent to H_s. Therefore, from Theorem 1, $SG(H)$ is acyclic.

("only if" part) Suppose that $SG(H)$ is acyclic. From Theorem 1, there is an execution H_s^{wd} equivalent to H. Since T_i^{wd} is equivalent to T_i, a serial execution H_s equivalent to H_s^{wd} can be obtained by replacing each T_i^{wd} in H_s^{wd} with T_i. Therefore, H is serializable. □

As mentioned in Section 5.2, SGT-WD uses readlocks and writelocks to execute read/write processes without interleaving with other conflicting operations. Therefore, the following theorem is derived immediately.

Theorem 3: The SG maintained by an SGT-WD scheduler is always equal to $SG(H)$. □

The above two theorems show that an SGT-

WD scheduler always produces serializable executions.

7. Evaluation

7.1 Comparison with SGT Certification

In this section, we compare SGT-WD with SGT certification. To explain the features of the two algorithms, we show some examples of concurrent executions in figures. In the figures, the following symbols are used:

- S means the start of the transaction.
- $R(x)$ ($W(x)$) means the read (write) operation on data item x.
- V means the validation. It includes the validation phase and the subsequent write phase.
- C means the commitment of a transaction.
- A means the abortion of a transaction.

Figure 2 shows an execution produced by SGT certification and SGT-WD. Suppose that a transaction T_1 is involved in a cycle including other transactions (not T_2) and is aborted. Consider SGT certification first (top of Fig. 2). A transaction T_1 is aborted by the failure, and then T_2, which read the data item x written by T_1, must also be aborted; that is, a cascading abort occurs. However, since T_2 has already been committed when T_1 is aborted, T_2 cannot be aborted and the database can no longer recover to a consistent state. This shows that SGT certification may produce executions that are not recoverable. To make the execution recoverable, SGT certification should defer the commit of T_2 until T_1 is committed or aborted (as shown in the middle of Fig. 2). We call this deferment *commit waiting*. Hereafter, we assume that SGT certification always carries out commit waiting if necessary. On the other hand, SGT-WD avoids such an execution by deferring the write operation of T_1. Under SGT-WD, x is not modified when T_2 reads it, while T_2 reads x, which is modified by T_1 under SGT certification. Therefore, under SGT-WD, T_2 does not need to be aborted when T_1 is aborted. Clearly, SGT-WD is more desirable than SGT certification in this case. Consider another execution, shown in **Fig. 3**. In this case, we assume that there are no transactions except T_1 and T_2. Under SGT certification, T_1 and T_2 do not form a cycle. Under SGT-WD, however, a cycle is formed, because the real execution of W(x) of T_1 is deferred until the write phase. Therefore, T_1 should be aborted and restarted. In the case of SGT certification, such a cycle is not formed and neither T_1 nor T_2 is aborted. Therefore, it may be concluded that SGT certification has an advantage over SGT-WD in this case. However, there is commit waiting for SGT certification. That is, under SGT certification, the commit operation of T_2 must wait until T_1 is committed. This means that a transaction waits for another transaction. Accordingly, the merit of certifications that aggressively schedule operations is damaged. On the other hand, no transactions wait for other transactions in SGT-WD. In this case, it is difficult to say which algorithm is more desirable.

The commit waiting mentioned above is also necessary for usual SGT. Moreover, most scheduling algorithms (including Two-Phase Locking and Timestamp Ordering) need this type of commit waiting to maintain the recoverability of executions [2]. Under SGT-WD, no commit waiting is needed, because all transactions read only data items written by committed transactions. This is one of the merits of SGT-WD.

In the case of **Fig. 4**, SGT-WD is clearly worse than SGT certification. Under SGT certification, T_1 and T_2 do not form a cycle, and

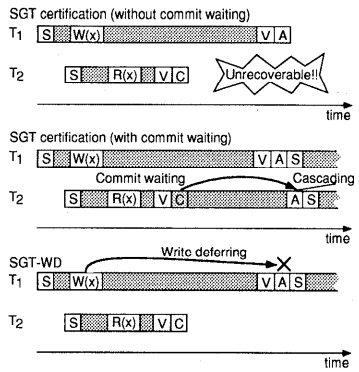

Fig. 2 Comparison of SGT certification and SGT-WD (case 1).

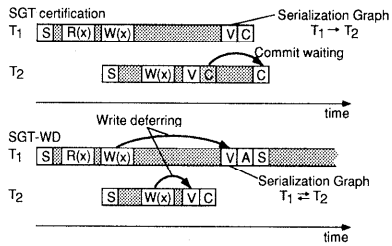

Fig. 3 Comparison of SGT certification and SGT-WD (case 2).

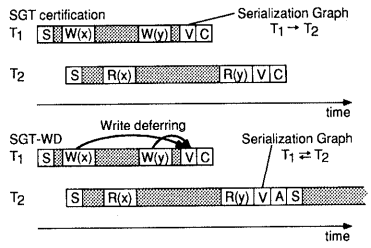

Fig. 4 Comparison of SGT certification and SGT-WD (case 3).

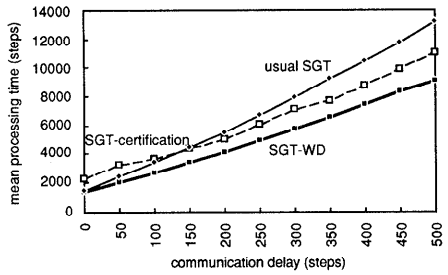

Fig. 5 Mean processing time (*arr-interval* = 200 steps, locality = 20%).

both are committed immediately (without commit waiting). Under SGT-WD, however, a cycle is formed, because the real execution of W(x) and W(y) of T_1 is deferred until the write phase. Therefore, T_2 should be aborted and restarted.

7.2 Overview of Simulations

One of our goals is to shorten the processing time of transactions when SGT is used in distributed database systems. We evaluated the processing time under SGT, SGT certification, and SGT-WD by means of simulations. We implemented the three types of scheduler on a simulator of a distributed database system. In distributed database systems, the communication cost strongly affects the processing time. To reduce the communication cost, we adopted the *fractional tag* scheme that we proposed in Tada, et al.[9]. A distributed database system is a collection of local databases, called *sites*, connected by a communication network. In the fractional tag scheme, each site has its own local SG, and messages are passed among sites to search for remote local SGs.

The following assumptions are similar to those in the abovementioned paper[9].

In our experiment, we regard a transaction as a string of operations, and we assume that each operation accesses only one data item.

We classify transactions into two types:

- Local transactions access only data items stored at the sites that execute them.
- Global transactions access data items stored at more than one site.

We define *locality* as the ratio of local transactions to all transactions.

Though there are many parameters, we fixed some parameters to simplify the simulations.

- There are 10 sites in the distributed database system.
- 100 data items are stored at one site.
- The transaction size is fixed; that is, all transactions contain 8 read/write operations.
- 25% of all operations are write operations.
- Each global transaction accesses at most 3 sites.

The time needed for executing a transaction mainly consists of the CPU processing time for scheduling, the communication delay for sending messages, and the I/O delay for data access. It seems that the CPU processing time is much smaller than the others. Simulations are executed in steps. We measure the transaction processing time by the number of steps. We assume that schedulers can process one message in one step, and that the access to a data item (including I/O delay) needs 100 steps.

To execute one operation, two types of message are needed:

- Messages for scheduling, which are needed to search for local SGs
- Messages for data access, which are needed to transfer data values

The number of messages of the former type can be suppressed by the certification approach, while messages of the latter type are still needed. Moreover, additional types of message are used to abort transactions and delete unnecessary nodes from the serialization graph.

7.3 Simulation Results

We selected the following parameters:

- The time needed to transfer an message between two sites (denoted *com-delay*).
- The mean interarrival time of transactions (denoted *arr-interval*).

Figure 5 depicts the mean processing time of transactions when *arr-interval* is fixed to 200 steps. The processing time increases with the communication delay. The rate of increase of the usual SGT is higher than that of SGT-WD and SGT certification. Of course, the influence of communication delay depends on the number of communications. Since SGT-WD adopts the certification approach, the number of communications is as small as in SGT certification.

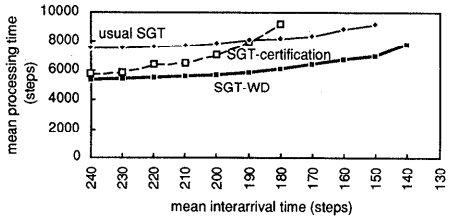

Fig. 6 Mean processing time (*com-delay* = 300 steps, locality = 20%).

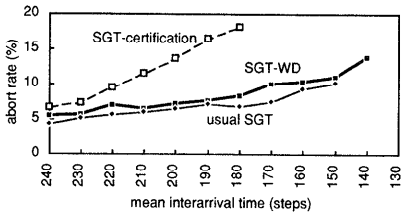

Fig. 7 Abort rate (*com-delay* = 300 steps, locality = 20%).

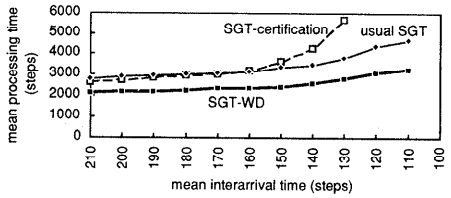

Fig. 8 Mean processing time (*com-delay* = 300 steps, locality = 80%).

Though the rates of increase of SGT-WD and SGT certification are almost equal, the absolute processing times differ. This is because SGT certification causes more abortions than SGT-WD.

In **Fig. 6**, *com-delay* is fixed at 300 steps and *arr-interval* is varied.

The shorter *arr-interval* is, the larger the load of the database. In the state of equilibrium, the numbers of arriving and leaving transactions during a period are almost equal, and the number of transactions in the database system is almost equal all the time. However, if *arr-interval* is less than a certain value, then the rate at which transactions arrive exceeds the rate at which they leave. Therefore, the number of transactions in the system keeps on increasing. That is, *saturation* of load occurs. Some points are not plotted in the graphs, because the mean processing time cannot be found as a result of the load saturation (for example the point for *arr-interval* = 130 in Fig. 6).

In Fig. 6, the mean processing time of SGT-WD is the shortest. Of course, the processing time increases as *arr-interval* decreases. However, the increase rate of SGT certification is much higher than that of SGT-WD or the usual SGT. This is because SGT certification causes many abortions as the load becomes higher. The rates of SGT-WD and the usual SGT are almost equal. This means that though SGT-WD takes the certification approach, it is as tolerant to an increase in load as the usual SGT. We consider this to be the benefit of deferment of write operations. **Figure 7** depicts the abort rate, where

$$\text{abort rate} = \frac{\#\text{ of abortions}}{\#\text{ of commitments}}$$

In Fig. 7, *com-delay* is fixed at 300 steps and the locality is 20%. The abort rate of SGT-WD is almost equal to that of the usual SGT, and is much lower than that of SGT certification. Though SGT-WD adopts the certification approach, the number of abortions is suppressed, because any cascading aborts are avoided by deferment of writes.

In **Fig. 8**, *com-delay* is also fixed at 300 steps. The difference between Fig. 6 and Fig. 8 is locality. When the locality is high, the processing time is short, because the effect of communication delays is weakened. However, the frequent occurrence of abortions seriously affects the performance, regardless of locality.

Simulation results shows that SGT-WD succeeded in introducing tolerance of the communication delay of SGT certification without spoiling the load tolerance of the usual SGT.

8. Conclusions

In this paper, we proposed a scheduling algorithm that we call Serialization Graph Testing with Write Deferring (SGT-WD). SGT-WD is a certification algorithm that defers write operations by using internal buffers. Its major advantages are as follows:

- Operations can be executed immediately.
- Only cascadeless executions are produced.

We evaluated SGT-WD, the usual SGT, and SGT certification by means of simulations on distributed database systems. Two merits of SGT-WD were recognized through these simulations. First, the influence of the communication delay on the processing time under SGT-WD is smaller than under the usual SGT, because of the suppression of communication by the certification approach. Second, SGT-WD is more tolerant of a load increase than SGT certification. This merit results from the deferment

of write operations.

The major disadvantage of SGT-WD is common to all certification algorithms. That is, transaction conflicts are detected later than in noncertification algorithms. We expected that this delay in conflict detection would cause many unnecessary abortions. However, simulation results showed that the influence on the number of abortions was less than expected.

From the above, we conclude that SGT-WD is an effective variant of SGT, especially for distributed database systems.

References

1) Bernstein, P., Rothnie, J., Goodman, N. and Papadimitriou, C.: The Concurrency Control Mechanism of SDD-1: A System for Distributed Databases (The Fully Redundant Case), *IEEE Trans.Softw.Eng.*, Vol.SE-4, No.3, pp.154–168 (1978).
2) Bernstein, P., Shipman, D. and Wong, W.: *Concurrency Control and Recovery in Database Systems*, Addison Wesley, Reading, MA (1987).
3) Jing, J., Bukhres, O. and Elmagarmid, A.: Distributed Lock Management for Mobile Transactions, *Proc. 15th IEEE Intl. Conf. on Distributed Computing Systems*, Yokohama, Japan, IEEE, pp.118–125, IEEE Computer Society Press (1995).
4) Kung, H. and Robinson, J.: On Optimistic Methods for Concurrency Control, *ACM Trans. Database Syst.*, Vol.6, No.2, pp.213–226 (1981).
5) Salem, K., Garcia-Molina, H. and Shands, J.: Altruistic Locking, *ACM Trans.Database Syst.*, Vol.19, No.1, pp.117–165 (1994).
6) Silberschatz, A. and Kedem, Z.: A Family of Locking Protocols for Database Systems that are Modeled by Directed Graphs, *IEEE Trans. Software Eng.*, Vol.SE-8, No.6, pp.558–562 (1982).
7) Son, S. and Park, S.: Scheduling Transactions for Distributed Time-Critical Applications, *Readings in Distributed Computing Systems*, Casavant, T. and Singhal, M. (Eds.), pp.592–618, IEEE Computer Society Press, Los Alamitos, CA (1994).
8) Tada, H., Higuchi, M., Fujii, M. and Okui, J.: A Scheduling Algorithm using Serialization Graph Testing for Distributed Database System, Technical Report SS94–41, IEICE (1994).
9) Tada, H., Higuchi, M., Fujii, M. and Okui, J.: A Distributed Scheduling Algorithm Using Serialization Graph Testing with Fractional Tag, *IPSJ Trans.*, Vol.38, No.1, pp.90–100 (1997).
10) Yannakakis, M.: A Theory of Safe Locking Policies in Database Systems, *J. ACM*, Vol.29, No.3, pp.718–740 (1982).

(Received January 10, 1997)
(Accepted June 3, 1997)

Harumasa Tada received his B.E. and M.E. degrees in information and computer sciences from Osaka University, Osaka, Japan, in 1993 and 1995 respectively. He is currently a Ph.D. candidate in information and computer sciences at Osaka University. His current research interests are concurrency control schemes for distributed systems. He is a member of IPSJ, ISSST and IEICE.

Masahiro Higuchi received the B.E., M.E. and Ph.D. degrees in information and computer sciences from Osaka University, Osaka, Japan, in 1983, 1985, 1995 respectively. In 1985–1990, he worked for Fujitsu Laboratories LTD. In 1991 he joined the faculty of Osaka University. Since 1995 he has been an Assistant Professor of Department of Information and Computer Sciences, Osaka University. His current research interests include concurrency control schemes for distributed systems, and methods for protocol specification, testing and verification. He is a member of IPSJ and IEICE.

Mamoru Fujii received the B.E., M.E. and Ph.D. degrees in electronic engineering from Osaka University, Osaka, Japan, in 1962, 1964 and 1970, respectively. In 1964–1967, he worked for Mitsubishi Electric Corporation. In 1967, he joined the faculty of Osaka University. In 1976–1989, he was an Associate Professor of Osaka University. In 1989–1994, he has been a Professor of College of General Education, Osaka University. Since 1994 he has been a Professor of Information and Computer Sciences Osaka University. He is a member of IPSJ and IEICE.