2000年 3月 電子工學會論文誌 第 37 卷 CI編 第 2 號

論文2000-37CI-3-1

데이터베이스 시스템에서 거래관리를 위한 두단계기부잠금규약 (Two-Way Donation Locking for Transaction Management in Database Systems)

李 惠 卿 *, 金 應 模 **
(Hae-Kyung Rhee and Ung-Mo Kim)

요 약

기존의 syntax 위주의 직력성 이론만 가지고서는 거래의 실행시간상 차별화 특성을 수용하면서 다수의 거래에 대한 단위시간당 처리 생산성을 높이기는 힘든 형편이다. 이러한 상황을 해결하기 위하여 이타적 잠금기법(altruistic locking : *AL*)은 거래가 객체를 사용한 다음 더이상 그 객체를 요구하지 않을 때 다른 거래들이 그 객체를 로크할 수 있도록 미리 객체에 대한 로크를 해제함으로써 거래들의 대기시간을 줄이기 위한 취지에서 제안된 것이다. 확장형 이타적 잠금(extended altruistic locking : *XAL*)기법은 *AL*을 자취의 확장 측면에서 개선한 잠금기법으로서 *AL*이 근본적으로 안고 있는 반드시 기부된 객체만을 처리해야 한다는 부담을 보다 완화한 기법이다. 본 논문에서는 우선 장기거래로 인한 단기거래의 장기적 대기현상 완화 측면에서의 *AL*과 *XAL*의 공통적 한계점을 분석하였다. Client-server 환경하에서 장기거래로 인한 단기거래의 장기적 대기현상을 최소화하도록 줄임으로써 작업처리율을 높이는 반면, 거래간의 평균 대기시간을 줄일 수 있는 새로운 확장형 이타적 잠금기법인 전후진방식의 신형 확장 기법인 2DL(two-way donation locking)을 제안하였다. 모의실험에 의한 성능평가 결과 장기거래의 길이가 5이상, 9이하인 상황에서 2DL은 2PL 보다 작업 처리율과 거래의 평균 대기시간 면에서 우수한 결과를 나타내었다.

Abstract

Traditional syntax-oriented serializability notions are considered to be not enough to handle in particular various types of transaction in terms of duration of execution. To deal with this situation, altruistic locking has attempted to reduce delay effect associated with lock release moment by use of the idea of donation. An improved form of altruism has also been deployed in extended altruistic locking in a way that scope of data to be early released is enlarged to include even data initially not intended to be donated. In this paper, we first of all investigated limitations inherent in both altruistic schemes from the perspective of alleviating starvation occasions for transactions in particular of short-lived nature. The idea of two-way donation locking*(2DL)* has then been experimented to see the effect of more than single donation in client-server database systems. Simulation experiments shows that *2DL* outperforms the conventional two-phase locking in terms of the degree of concurrency and average transaction waiting time under the circumstances that the size of long-transaction is in between 5 and 9.

* 正會員, 敬仁女大 멀티미디어情報電算學部
(KyungIn Women's College Division of Multimedia Information Computer science)
** 正會員, 成均館大學校 電氣電子컴퓨터工學部
(SungKyunKwan University Electrical & Computer Engineering)
接受日字:1999年 10月 21日, 수정완료일 : 2000年 4月12日

I. Introduction

Although liveness duration might not be a serious issue in the arena of standard on-line transaction processing, in which transactions are normally expected to finish shortly, it certainly matters in circumstances where a number of

long-lived ones are supposed to access a substantial number of data. In case database correctness is guaranteed by standard transaction scheduling schemes like *two-phase locking* (2PL)[1] for the context of concurrent execution environment in which short-lived ones are normally mixed with long-lived ones, degree of concurrency might be hampered by selfishness associated with lock retention.

This sort of reluctance for early release of locks is essentially due to their discipline. Lazy release in turn could aggravate fate of misfortune for long-lived ones in that they are more vulnerable to get involved in deadlock situations. This could the other way around aggravate the fate of short-lived ones as well in a way that they suffer from starvation or livelock affected by long-lived ones.

To reduce the degree of livelock, the idea of altruism has been suggested in the literature. *Altruistic locking*[2], *AL* for short, is basically an extension to 2PL in the sense that several transactions may hold locks on an object simultaneously under certain conditions. Such conditions are signaled by an operation donate. Like yet another primitive unlock, donate is used to inform the scheduler that further access to a certain data item is no longer required by a transaction entity of that donation. The basic philosophy behind *AL* is to allow long lived transactions to release their locks early, once it has determined a set of data to which the locks protect will no longer be accessed. In this respect, effect of donate is actually to increase the degree. In order to allow more freedom, an entity of donation is let continue to acquire new locks. This implies that donate and lock operations need not be strictly two-phase.

In *AL*, the basic concept is to allow long lived transactions to release their locks early, once it is determined that the data which the locks protect will no longer be accessed. Actually, the effect of donate operation is applied to *AL*, there are still some transactions that suffer from intolerable delay.

The idea of donating could further be exploited to pursue an enhanced degree of concurrency. *Extended altruistic locking*[2], *XAL* for short, attempted to expand the scope of donation in a way that data to be early disengaged is augmented by extra data originally not conceived to be rendered. Example 1 shows this.

Example 1(Not-Allowable Schedule under *AL* but Allowable in *XAL*): Suppose that T1 attempts to access data items *A, B, C, D* and *E* in an orderly manner. Note that data items *F, G* and *H* shall never be accessed by T_1 at all. Presume that T_1 has already locked and successfully donated *A* and *B*. T_1 now is supposed in the stage of accessing *C*. Suppose also that there are three more transactions concurrently in execution along with $T_1 : T_2$ wishing for *A* and *B*, T_3 wishing for *B* and *F*, and T_4 wishing for *F* and *H* (Figure 1).

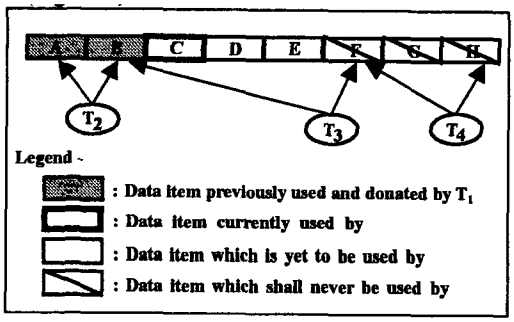

그림 1. 기부된 data사용을 위해 경쟁적인 트랜잭션들의 예

Fig. 1. Concurrent transactions competing for same data donated.

If we apply *AL* for these transactions, lock request for *A* and *B* by T_2 would be allowed for the purpose of obeying the notion of serializability. T_2 could be allowed to access since they are already included in the donating list. However T_3 would be rejected because of the restriction of *AL*. While T_3 experiences delay, T_4 would be

permitted to access F and H because any conflict arises for this access.

In case *XAL* is adopted rather than *AL*, T_3 could fortunately be allowed to access B and F without any delay since B is already included in the donating list and F also can be included in the donating list by protocol. T_4, in this case, each would proceed in the same manner as in *AL*.

Under *XAL*, T_3 can access B which has been donated by T_1, it can access the data item F which shall not be accessed by T_1. Extended donation gives transactions more chance to access data items than *AL*. Furthermore it may increase the degree of concurrency than *AL*.
End of Example 1.

Delay can be reduced using donate operation which allows other transactions access data items that has already been locked by long lived transactions. *AL* has some restrictions that must be used only the donated ones. However, *XAL* allows the schedule which requests the data items that will not be locked by long lived transaction.

AL and *XAL* maintain a certain information concerning what data items a transaction will access or when a transaction does not access which data item any more. If such information is utilized carefully, the previous overhead may be overcome and thus we expect the protocol which may increase the degree of concurrency and have better waiting time.

This paper focuses on devising a novel version of altruistic locking which could be as well applicable for distributed computing environment.

Ⅱ. Related Work

In this section, we will describe the *AL*. We will assume temporarily that transactions perform only one type of operation on database objects, and we will not distinguish between reads and writes operations. Furthermore, we will assume that transactions always commit. These initial assumptions are for the sake of our discussion only.

Concurrency control operations of transactions must be submitted to a scheduler for the sake of preventing lock conflicts between transactions. It is well known that schedules of well-formed two-phase transactions that observe this rule are correct[1,3]. 2PL protocols ensure that these conditions are met, they produce serializable schedules.

AL is like *2PL* except for the concept of wakes. If transactions do not make use of the Donate operation, altruistic scheduling reduces to *2PL* since no transactions will create wakes. However, when donations are made and transactions create wakes, an altruistic scheduler can allow a transaction to run within a wake, provided it remains completely within the wake.

While the donation of wake is rigid in *AL* in terms of fixedness of it size, a dynamic way of forming a wake could be devised given that serializability is never violated. This was realized in XAL by simply letting data originally not intended to bestowed to be dynamically included in a wake predefined. The rule is that wake expansion comes true only after a short transaction has already accessed data in its predefined wake list. So, the presumption made for *XAL* is that a short transaction still restlessly wishes to access data of its wake-dependent long transaction even after it has done with data in its wake list. The assumption could be called data-in-wake-list-first/other-data-later access fashion. *XAL* therefore performs inevitably badly if others-first wake-later access paradigm is in fact to be observed.

To resolve this sort of chained delay, others-first wake-later approach could be made viable in a way of including others, not honored before, to a wake list. This enhancement is one of substances, made in our proposed scheme, which could be considered as backward donation, compared to *XAL*, which is based on forward donation. *XAL* can be viewed as uni-donation

scheme in that it deals with donation principle involving only one single long transaction. One other major substance of our proposed scheme is to let more than one long transaction donate while serializability is preserved. The notion of *multiple serializability* is thus developed in our scheme. Our solution, *multiple-donation* scheme, allows donation from more than one long transaction but for the sake of presentation simplicity, degree of donation is limited to two in this paper.

Ⅲ. Transaction Processing Model

In this paper, the multiplicity is rendered to the case of two to measure the effect of donation variety. Two-way donation altruistic locking protocol, *2DL* for short, can be pseudo-coded as follows(Algorithm Wake Expansion).

1. Transaction Processing Model

As client-server architectures have become a dominant paradigm in distributed computing, we employ the client-server computing model to provide both responsiveness to users and support for database-related processing in distributed environments.

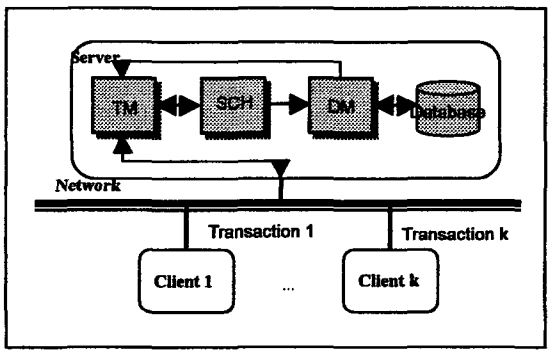

그림 2. 트랜잭션 처리 모형
Fig. 2. Transaction processing model.

In a client-server database model, the database resides on a machine other than those that run the application processing components. Thus, the database software needs to be split between the client system that actually runs the application program and the server system that stores the database. Figure 2 shows the transaction processing model.

2. Operation Instance of *2DL*

In case donated data items are used under *XAL*, it is allowed to request data items which are donated by only one transaction. Under *2DL*, in contrast, short-lived transactions are treated to be given more freedom in accessing donated data items by eliminating the single-donation constraint. Short-lived transactions can access data items donated by two different long lived transactions.

2DL also permits short-lived transactions request data items which have been donated by two different long-lived transactions. A way to conduct a two-way donation is shown, in Example 2, with two separate long transactions and a single short transaction.

Example 2 (Allowing Proceeding of Short Transaction with Two Concurrent Long Ones): Suppose that T_1, a long transaction, attempts to access data items, *A, B, C, D* and E, in an orderly manner. Presume that T_1 has already locked and successfully donated A and B. T_1 now is supposed in the stage of accessing *C*. Suppose also that there are two more concurrent transactions in execution along with T_1: T_2, long, wishing for data items, *F, G, H, I* and *J*, in an orderly manner and T_3, short, wishing for *B, G* and *K* similarly. Presume that T_2 has already locked and successfully donated *F* and *G*. T_2 now is supposed in the stage of accessing *H*(Figure 3).

If we apply *XAL* for these transactions, a lock request for *B* by T_3 would be allowed to be granted but a lock request *G* would not because *G* has already been donated by another long-lived transaction. Only after T_2 commits, can be tossed to T_3.

In case *2DL* is adopted rather than *XAL*, T_3 could fortunately be allowed to access without any delay. This is made possible by simply

including the wake of T_2 into the wake of T_1.
End of Example 2.

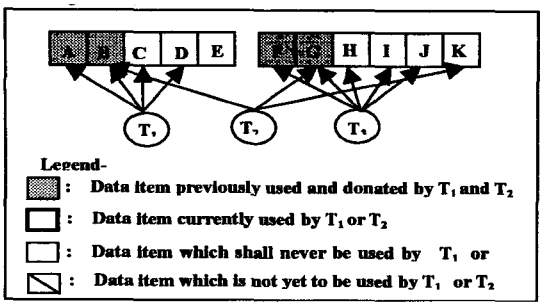

그림 3. 두 개의 장기거래와 함께 동시적으로 실행되는 T_3
Fig. 3. Execution of T_3 with two concurrent long-lived transactions.

Ⅳ. Two-Way Donation Locking

1. Algorithm

Short-lived transactions are treated to be given more freedom in accessing donated data items by eliminating the single-donation constraint under *2DL*. *2DL* permits short-lived transactions request data items which have been donated by two different long-lived transactions.

Algorithm(*Wake Expansion Rule of 2DL*)
Input: LT1; LT2; ST
/* ST:short trans;
LT1, LT2:long trans */
BEGIN
FOREACH LockRequest
IF(LockRequest.ST.data = *Lock*)
THEN
/* Locks being requested by ST already granted to long trans other than LT1 and LT2 */
Reply:=ScheduleWait(LockRequest);
ELSE IF(LockRequest.ST.data = Donated) **THEN**
/* Locks being requested by ST donated by long trans other than LT1 and LT2 */
FOREACH (ST.wake ∈ LT1 OR LT2)
IF(ST.wake = LT1) THEN
/* Donation conducted by LT1? */
IF(ST.data ∈ LT1.marking-set) **THEN**
/* Data being requested by ST to be later accessed by LT1 ? */
Reply:=ScheduleWait(LockRequest)
ELSE
Reply:=ScheduleDonated(LockRequest)
ENDIF
ELSE
IF(ST.data ∈ LT2.marking-set) **THEN**
/* Data being requested by ST to be later accessed by LT2 ? */
Reply := ScheduleWait(LockRequest)
ELSE
Reply:=ScheduleDonated(LockRequest)
ENDIF
ENDIF
ENDFOR
ELSE
Reply := ScheduleLock(LockRequest)
ENDIF
IF(Reply = Abort) THEN
/* Lock request of ST aborted */
Abort Transaction(Transactionid);
Send(Abort);
Return();
ENDIF
ENDFOR
END

2. Correctness

We will prove that the schedule which the scheduler of *2DL* may produce is serializable in this section. To do so, we will make use of the serializability theorem[1], the definition of Crest Before[2] and a lemma used in proving the correctness of *AL*[2]. The serializability theorem states that a history H is serializable iff its serialization graph is acyclic, and the definition of Crest Before states that for two transactions, say T_i and $T_j \rightarrow {}_uT_j$ if T_I unlocks some data items before T_j locks some data items.

The notations used in this correctness proof are as follows. We use $O_i[x]$, $P_i[x]$ or $q_i[x]$ to denote the execution of either read or write operation issued by a transaction, T_i, on a data item, x.

Locking operation for either read or write is also represented by $ol_i[x]$, $pl_i[x]$ or $ql_i[x]$. Unlock and donate operations are denoted by $u_i[x]$ and $d_i[x]$ respectively. H represents a history which may be produced by *2DL* and O(H) is a history obtained by deleting all operations of aborted transactions from H. The characteristics of histories which may be produced by *2DL* are as follows.

Property 1(Two-Phase Property): If $ol_i[x]$ and $u_i[y]$ are in O(H), $ol_i[x] < u_i[y]$. This property represents two-phase rule.

Property 2(Lock Property): If $o_i[x]$ is in O(H), $ol_i[x] < o_i[x] < u_i[x]$. That is, a data item is locked before and unlocked after it is accessed.

Property 3(Donate Property): If $ol_i[x]$ and $d_i[x]$ is in O(H), $o_i[x] < d_i[x]$. That is, T_i cannot be in its own wake.

Property 4(Unlock Property): If $di[x]$ and $u_i[x]$ is in O(H), $d_i[x] < u_i[x]$. That is, T_i can be unlocked after it is donated.

Property 5(Indebtedness Property) : If T_j is indebted to T_i, then for every $o_j[x]$ in O(H), either $o_j[x]$ is in the wake of T_i, or there exists $u_i[y]$ in O(H) such that $u_i[y] < o_j[x]$.

If a transaction is indebted to another, it must remain completely in the other's wake until it begins to unlock objects.

Lemma 1(Altruism) : If $p_i[x]$ and $q_j[x]$ (ij) are conflicting operations in O(H) and $qi[x] < qj[x]$, then $u_i[x] < ql_j[x]$ or $d_i[x] < ql_j[x]$. That is, two transactions are prohibited to hold locks on the same data item unless one of them has unlocked it or donated it.

Proof : A data item must be locked before and unlocked after it is accessed by Property 1. In Wake Expansion Rule of *2DL*, a conflict lock on the data item, say a, is allowed only when no transaction locks a or the transactions which hold locks on a has donated it. Thus, the history, O(H), satisfies Lemma 1.

End of Lemma 1.

Lemma 2(Complexity-In-Wake): If $T_1 \rightarrow T_2$ is in serialization graph, then either $T_1 \rightarrow {}_uT_2$ or $T_1 \; {}_dT_2$.

Proof: $T_1 \rightarrow T_2$ in serialization graph means that there exist conflicting operations, say $p_1[x]$ and $q_2[x]$, in H such that $p_1[x] < q_2[x]$. There are only two cases that may occur for this by Lemma 1. One is that there is $p_1[x] < d_1[x] < ql_2[x] < q_2[x]$ in O(H), i.e., T_2 accesses the data items donated by T_1.

A transaction T_2 has to access only wake of another transaction T_1 , once T_2 makes conflict locks on the data items donated by T_1. T_2 must be completely in the wake of T_1 if T_2 has accessed any of the wake of T_1. This is ensured by the first else if condition in algorithm. Even if T_2 has already accessed any data items which do not belong to the wake of T_1 , such data items would be included into the wake of T_1 as long as T_1 does not access any of such data items at all for its execution. If the data items locked by T_2 will be accessed by T_1, the access of T_2 to the data items donated by T_1 is not allowed by the second foreach condition. Thus, $T_1 \rightarrow T_2$ corresponds to $T_1 \rightarrow {}_dT_2$ in the case that $p_1[x] < d_1[x] < ql_2[x] < q_2[x]$ in H, or in the case that $p_1[x] < u_1[x] < ql_2[x] < q_2[x]$ in O(H) by Lemma 1. Thus, $T_1 \rightarrow T_2$ corresponds to $T_1 \rightarrow {}_uT_2$ in the case.

End of Lemma 2.

Lemma 3(Correctness of *AL*): Consider a path $T_1 \rightarrow \ldots T_{n-1} \rightarrow T_n$ in O(H). Either $T_1 \rightarrow {}_uT_2$, or there exists some Ti on the path such that $T_1 \rightarrow {}_uT_i$.

Proof: We will use induction on the path length n. By Lemma 2, the lemma is true for n = 2. Assume the lemma is true for paths of length n-1, and consider a path of length n. By the inductive hypothesis, there are two cases:

1) There is a T_I between T_1 and T_{n-1} such that $T_1 \rightarrow {}_uT_k$. The lemma is also true for paths of length n.

2) $T_1 \rightarrow {}_dT_{n-1}$. T_n and T_{n-1} conflicts on at least one object, x. Since T_{n-1} is completely in the wake of T_1, we must have $d_1[x] < ql_{n-1}[x]$ in O(H). By Property 1, Tn must lock x. By Property 4, T_1 must unlock x. Either $u_1[x] < ol_n[x]$ or $ol_n[x] < u_1[x]$. In the first case, we have

that $T_1 \rightarrow {}_uT_n$, i.e., T_n is the T_k of the lemma. In the second case, T_n is indebted to T_1. By Property 5, Tn is completely in the wake of T_1($T_1 \rightarrow {}_dT_n$) or $T_1 \rightarrow {}_uT_n$.

Theorem 1(Serializability of *2DL*): If O(H) is acyclic, O(H) is serializable.

Proof: Assume that there exists a cyclic $T_1 \rightarrow \ldots T_{n-1} \rightarrow T_n$ in serialization graph. By Lemma 3, $T_1 \rightarrow {}_dT_1$, or $T_1 \rightarrow {}_uT_i$. By Property 3, only $T_1 \rightarrow {}_uT_i$ is possible. Since T_i is prohibited to lock any more data items once T_1 unlocks any one, T_i cannot be T_1. Again, by applying Lemma 3 to the same cycle $T_1 \rightarrow T_{i+1} \rightarrow \ldots \rightarrow T_i$, we get $T_i \rightarrow {}_uT_k$. for the same reason and thus we get $T_1 \rightarrow {}_uT_i \rightarrow {}_uT_k$ in all. Since the relation u is transitive, $T_1 \rightarrow {}_uT_k$ is satisfied. Thus, T_k cannot be any of T_1 and T_i. If we are allowed to continue to apply Lemma 3 to the given cycle n-3 times more in this manner, we will get a path $T_1 \rightarrow {}_uT_i \rightarrow {}_uT_k \rightarrow {}_u\ldots \rightarrow {}_uT_m$ containing all transactions, i.e., T_1 through Tn. If we apply Lemma 3 to the given cycle starting from T_m one more time, we are enforced to get a cycle $T_1 \rightarrow {}_uT_i \rightarrow {}_uT_k \rightarrow {}_u\ldots \rightarrow {}_uT_m \rightarrow {}_uT_1$ and we get a contradiction of violating Property 1 or Lemma 3. Thus serialization graph is acyclic and by the serializability theorem O(H) is serializable.

End of Theorem 1.

V. Performance Evaluation

In this section, we experimented the performance behavior of *2DL*. Performance comparison is made against *2PL* under various workloads. Major metrics chosen are transaction throughput and average transaction waiting time. A simulation model has first of all been established.

1. Simulation Model

(1) Assumptions

To cultivate the model in detail, a number of assumptions have been brought in.

1(*Reliable System Resources*) : Client machines as well as the server are perfect in the sense that they are always operable. System network is also perfectly reliable in that transmitted messages are never lost.

2(*Read-Once Policy*): A transaction does not read a data item again after a transaction has already read or written the same data item.

3(*Fake Restart*) : Whenever a transaction experiences a restart, it is replaced by a new, independent transaction.

4(*Number of Long Transactions*) : At most one long lived transaction may be active at any time.

5(*Commit Policy*) : Long - lived transactions always commit. Due to a nature of long execution duration, they are vulnerable to be aborted before reaching their end. To reflect this view, a short transaction in conflict against any outstanding long ones is treated to be restarted.

6(*Resource Service Policy*) : There are two resource type in our model. One is CPU and the other is input/output devices(I/O).

Service for read or write is simulated by the use of CPU and I/O resources. When a read or write needs CPU service, a CPU available at the instant is allocated. Services for I/O are processed analogously. Although requests for CPU or I/O is first-come-first-served, requests from SCH for a CPU service have a priority over other requests from other components.

(2) Simulation Parameters

Number of data items in database denoted by(db_size for short).
Number of CPUs(*num_cpus*)
Number of disks(*num_disks*)
Mean size of short transactions(*short_tran_size*)
Mean size of long transactions(*long_tran_size*)
Mean time for creating a transaction
(*tran_creation_time*)
Simulation length(*sim_leng*)
Mean time for deadlock time out(*timeout*)

The simulation input parameters used, as

follows, are classified into two categories: those of which values are fixed throughout simulation and those of which values vary(Table 1).

표 1. 모의실험에 사용된 인수들
Table 1. Parameters Setting for Simulation.

Parameters	Values
db_size	100
num_cpus	2
num_disks	4
short_tran_size	2,3,4,5
long_tran_size	4,5,6,7,8,9,10
tran_creation_time	5 units
timeout	30,50
sim_leng	100 - 1500

In case a transaction is exposed to a substantial delay, even exceeding a certain timeout, once it has been blocked, it is judged to be involved in deadlock situations. Deadlock resolution shall then be followed.

This simulation model has been implemented using Scheme[4] discrete-event simulation language. Virtual-time features in Scheme are considered to be ideal enough for dealing with concurrent execution environment.

2. Simulation Results and Their Interpretations

We now discuss the results of simulation experiments performed for the three different replication control schemes: *2PL, XAL, and 2DL*. Our simulation experiments were focused on the effects of sensitive parameters in the performance indices to measure their performance behaviors.

(1) Effect of Multiprogramming Level

This experiment is used to investigate the effect of multiprogramming level on the performance of the three schemes. The overall system throughput is depicted in Figure 4. Its corresponding average transaction waiting behavior is dipicted in Figure 5. For this workload, *2DL* in general appears to outperform *2PL* in terms of average waiting time. The best throughput performance is exhibited by 2PL and the worst average waiting time is portrayed by *XAL*.

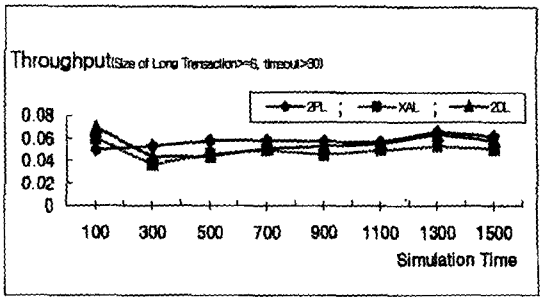

그림 4. 작업 처리율
Fig. 4. Throughput.

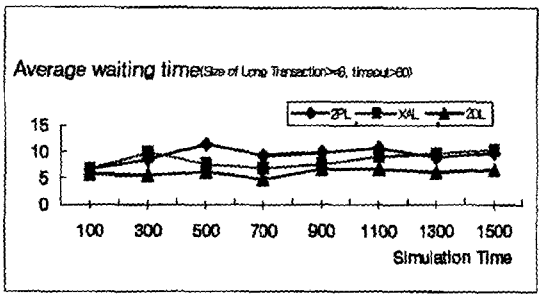

그림 5. 평균 작업 대기시간
Fig. 5. Average waiting time.

The major force behind prevailness of *2DL* mainly comes from capitalizing advantage from maintaining two different transaction wakes. Performance gain of *2DL* against *2PL* is about 155 per cent in terms of average waiting time. *2PL* however outperforms *XAL* and *2DL* with 105 per cent of performance at transaction throughput. This is because both *XAL* and *2DL* have a certain overheads to reserve data objects to be accessed.

2DL and *2PL* eventually perform similarly in terms of throughput, at simulation time 1500. It seems that the performance between two schemes appear to be about the same, however the average waiting time of *2DL* exhibits slightly better behavior than *2PL*. *2DL* outperforms the other schemes due mainly to enhanced degree of freedom given to *2DL* in accessing donated data by extending to multiple donation.

(2) Effect of Long-Lived Transaction Size

We have found that accessing donated data by extending to multiple-donation provides better performance than the other schemes. Accordingly,

we ran the simulation by varying the size of long-lived transactions to evaluate the performance under various level of donation. As the size of long-lived transaction is getting shorter, short-lived transactions can get more chance to use donated object. This experiment is used to investigate the effect of the size of long-lived transaction on the performance of concurrency control schemes, as the degree of donations varies. Figures 6 and 7 show transaction performance as a function of the throughput and average transaction waiting time when the size of long-lived transaction is greater than 3. For this experiment, all the other simulation parameters are the same as those that we already used before at Figures 4 and 5.

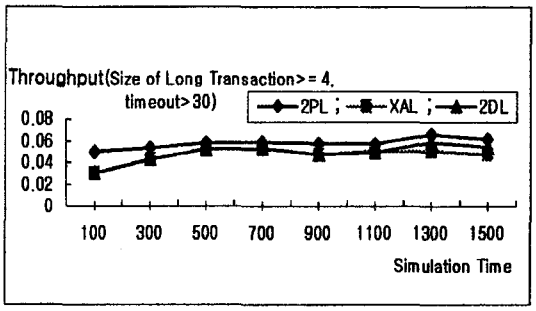

그림 6. 작업 처리율
Fig. 6. Throughput with larger long-lived transaction.

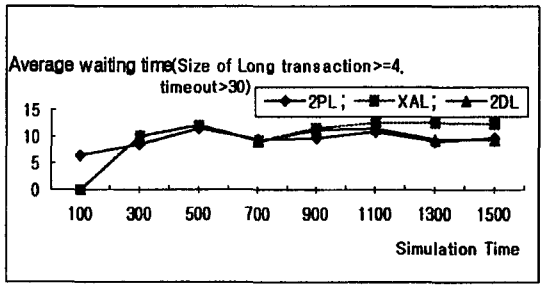

그림 7. 평균 작업 대기시간
Fig. 7. Average waiting time with larger long-lived transaction.

2DL and *XAL* eventually perform similarly in terms of throughput and their average waiting time, from simulation time 100 through 900. *2DL* and *2PL* however outperforms *XAL* from simulation time 1100.

We can observe that the throughput curves of three schemes tend to be flat although simulation length is getting longer. *2PL* performs best in terms of throughput however, *2DL* performs best in terms of average waiting time owing to two-way donation which contributes to give transactions more chance to use the objects than one-way donation. It has been observed that average waiting time of *XAL* continually increased from simulation point 700. Overall phenomenon shows us that the performance of *XAL* is the worst than the others since as the size of long-lived transaction is getting shorter, there are more donations from the long-lived transactions. Once wake-dependency has been established, the short is all of sudden forced to wait to be executed even though it attempts access to data of never antagonistic conflict to each other. This wake-dependency may cause a lot of burdens for performing the submitted transactions. This is because *XAL* has a certain overheads to reserve data objects to be accessed.

We presumed a certain value of timeout in order to remove the transaction, which is exposed to a substantial delay, from transaction waiting queue even exceeding a certain timeout. We do not investigate to find which one is judged to be involved in deadlock situation correctly in this paper. As the timeout is getting longer, the unblocked transactions can get more chance to accessing their jobs since we could not make a decision with only long transaction waiting.

Throughputs of *2PL*, *XAL*, and *2DL* tend in particular to degrade from a certain point of simulation time, for instance the point of 100. Waiting overheads of *XAL* behaves worst among three schemes. *XAL* performs worst in terms of throughput and average waiting time. Overall behaviors have been revealed that as the number of transactions increases, *2PL* generally outperforms in terms of throughput and waiting time. This shows a possibility that performance gain of *2DL* against *2PL* could be deteriorated sharply if

the simulation time is far extended beyond a certain point, say 700. Further experimentation is being conducted at the moment to investigate the degree of sharpness. The degree of extensiveness for upcoming measurements could though depend on a configuration of computing platform employed.

VI. Conclusions

Multiple-donation locking is definitely recommended in particular for environments where benefit of concurrency degree improvement exceeds overheads associated with aborts of long-lived transactions. The two-way donation scheme presented in this paper can be easily extended to three-way donation. Four-way donation could as well be devised on basis of 2DL, but serializability concern could arise due to complexity of wake-dependency relationships. As the complexity rises, multiple-donation altruism could be rendered to a simple-minded locking in which even database integrity is violated. In this respect, optimal number of long-lived transactions to volunteer for donation must be cautiously sought. 2DL is considered to be a practical solution to take in real world environment where long-lived transactions naturally coexist with short-lived ones.

This wake-dependency may cause a lot of burdens for performing the submitted transactions. This is because *XAL* and *2DL* have a certain overheads to reserve data objects to be accessed.

참 고 문 헌

[1] P. A. Bernstein, V. Hadzilacos and N. Goodman, "Concurrency Control and Recovery in Database Systems," Addison-Wesley, Massachusetts, U.S.A., 1987.

[2] K. Salem, H. Garcia-Molina and J. Shands, "Altruistic Locking," ACM Transactions on Database Systems, Vol. 19, No. 1, pp. 117-169, March 1994.

[3] K. P., Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger, "The notion of consistency and predicate locks in a database system," ACM Commun., Vol. 19, pp. 624-633, November 1976.

[4] H. Bartley, C. Jensen and W. Oxley, "Scheme User's Guide and Language Reference Manual," Texas Instruments, Texas, U.S.A., 1988.

저 자 소 개

李 惠 卿(正會員)

1979년 2월 : 숭실대학교 전자계산학과 졸업. 1985년 4월 : University of Illinois (Urbana-Champaign) 전산학과 석사. 1988년 3월~1989년 2월 : 국립천안공업전문대학 전자계산과 전임강사. 1992년 3월~현재 경인여자대학 멀티미디어정보 전산학부 조교수. 2000년 2월 : 성균관대학교 전기전자컴퓨터 공학부 박사. 주관심분야는 온라인 거래처리, 분산데이터베이스, 이동데이터베이스

金 應 模(正會員)

1981년 2월 : 성균관대학교 수학과 학사. 1986년 5월 : Old Dominion University 전산학과 석사. 1990년 2월 : Northwestern University 전산학과 박사. 1997년 8월~1998년 7월 : University of California, Irvine 전산학과 방문교수. 1991년 1월~현재 한국정보처리학회 논문지 편집부 위원장. 1990년 3월~현재 성균관대학교 전기전자 및 컴퓨터공학부 교수. 주관심분야는 데이터마이닝, Web 데이터베이스, 객체지향 DB, 트랜잭션관리