분산망 거래관리를 위한 기아현상 극소화 잠금규약

정회원 이 혜 경*, 김 응 모**

LIVELOCK-THIN LOCKING PROTOCOL FOR TRANSACTION SCHEDULING IN DISTRIBUTED DATA NETWORK MANAGEMENT

Hae-kyung Rhee*, Ung-mo Kim** *Regular Members*

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 distributed 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.

Ⅰ. 서 론

Although liveness duration might not be a serious 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.

If long transactions are out-favored, degradation of concurrency degree with regard to short ones is inevitable. In case the degree of concurrency needs to substantially rise, for instance in multiprogramming environment, this sort of delay effect could cause domino phenomenon. As long as long transactions and short transactions live

* 경인여자대학 멀티미디어 정보 전산학부
논문번호:99347-0830 접수일자:1999년8월30일

together, we could have to live up with this kind of dilemma. 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.

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 *T1* at all. Presume that *T1* has already locked and successfully donated *A* and *B*. *T1* now is supposed in the stage of accessing *C*. Suppose also that there are three more transactions concurrently in execution along with *T1*: *T2* wishing for *A* and *B*, *T3* wishing for *B* and *F*, and *T4* wishing for *F* and *H*(Figure 1).

If we apply *AL* for these transactions, lock request for *A* and *B* by *T2* would be allowed for the purpose of obeying the notion of serializability. *T2* could be allowed to access since they are already included in the donating list. However *T3* would be rejected because of the restriction of *AL*. While *T3* experiences delay, *T4* would be permitted to access *E* and *F* because any conflict arises for this access.

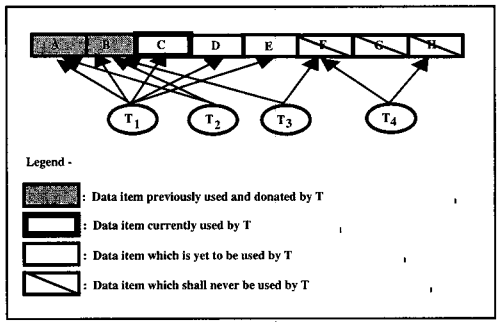

Fig. 1 Concurrent Transactions Competing for Same Data Donated

In case *XAL* is adopted rather than *AL*, *T3* 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. *T4*, in this case, each would proceed in the same manner as in *AL*.

Under *XAL*, *T3* can access *B* which has been donated by *T1*, it can access the data item *F* which shall not be accessed by *T1*. *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.

Ⅱ. Related Work

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. Example 2 shows this.

Example 2(Delay Effect Caused by Donation Extension): Suppose that *T1* attempts to access data items, *A*, *B*, *C* and *D*, in an orderly manner. Note that data items, *E*, *F*, *G*, and *H* shall never be accessed by *T1* at all. Presume that *T1* has already locked and successfully donated *A*, *B* and *C*. *T1* now is supposed in the stage of accessing D. Suppose also that there are three more transactions concurrently in execution along with *T1*: *T2* wishing for *B* and *E*, *T3* wishing for *E* and *F*, and *T4* wishing for *F* and *J*(Figure 2).

If we apply *XAL* for this situation, *T2* could in some circumstances fortunately be allowed to access both *B* and *E* without experiencing any delay.

In case *T2* initially requests *B* first rather than *E*, *T2* is able to access not only *B* but *E* as well, since *T2* is fully in the wake of *T1*. *T2* therefore succeeds to commit. *T3* then could acquire *E* released by *T2*. *T4* could thereafter acquire *F* released by *T3*.

In case, however, *T2* initially requests *E* first rather than *B*, *T2* can certainly acquire *E* but it fails for *B* because wake relationship cannot honor *E* as a member of the wake list. Once this sort of wake dependency is detected, *T2* can be allowed to access *B* only after it is finally released by *T1*. *T2* in this case is therefore blocked. *T3* must then be blocked for *E* to be released by *T2*. *T4* as well must be blocked for *F* to be released by *T3*, forging a chain of blockage.

End of Example 2.

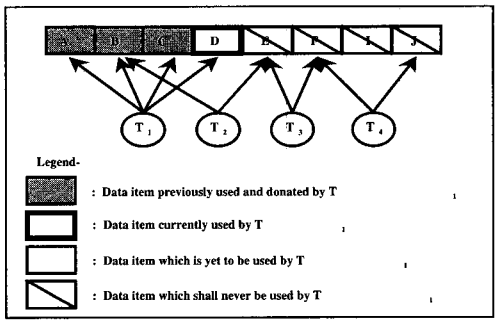

Fig. 2 Four Transactions, T1 through T4, Competing for Same Data Donated

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

3.1 Assumptions

To describe wake expansion rule in detail, simplifications were made mainly with regard to transaction management principle.

1(*Donation Privilege*): Only long-lived transactions are privileged to use donate operation.

2(*Commit Policy*): A long-lived transaction eventually commits.

3(*Deadlock Handling*): If a transaction happens to fall into deadlock situation, that transaction will be eliminated by using a certain deadlock timeout

scheme.

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*).

3.2 Transaction Processing Model

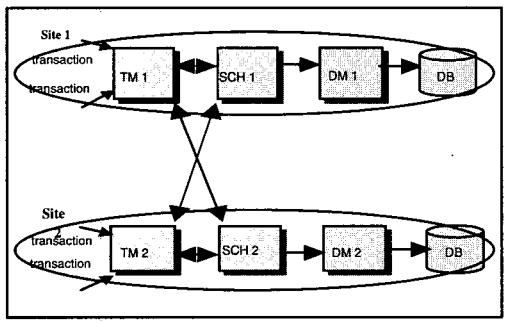

Fig. 3 2DL Transaction Processing Model

3.3 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 3, with two separate long transactions and a single short transaction.

Example 3(Allowing Proceeding of Short Transaction with Two Concurrent Long Ones): Suppose that *T1*, a long transaction, attempts to access data items, *A*, *B*, *C*, *D* and *E*, in an orderly manner. Presume that *T1* has already locked and successfully donated *A* and *B*. *T1* now is supposed in the stage of accessing *C*. Suppose also that there are two more concurrent transactions in execution along with *T1*: *T2*, long, wishing for data items, *F*, *G*, *H*, *I* and *J*, in an orderly manner and *T3*, short, wishing for *B*, *G* and *K* similarly. Presume that *T2* has already locked and successfully donated *F* and *G*. *T2* now is supposed in the stage of accessing *H*(Figure 4).

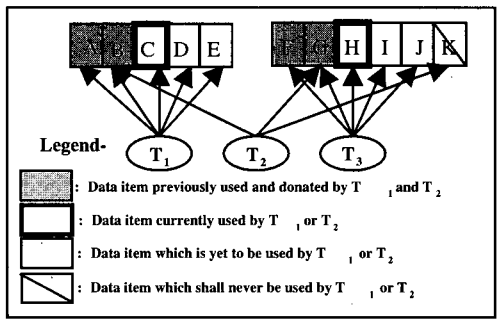

Fig. 4 Execution of T3 with Two Concurrent Long-Lived Transactions

If we apply *XAL* for these transactions, a lock request for *B* by *T3* 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 *T2* commits, *G* can be tossed to *T3*.

In case *2DL* is adopted rather than *XAL*, *T3* could fortunately be allowed to access without any delay. This is made possible by simply including the wake of *T2* into the wake of *T1*. End of Example 3.

Ⅳ. Multiple Donation Locking

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
```

V. Performance Evaluation

In this chapter, 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.

5.1 Simulation Model

5.1.1 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.

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.

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

5.1.2 Simulation Parameters

The simulation input parameters used, as follows, are classified into two categories: the parameters of which values are fixed throughout simulation and those of which values vary.

- Number of data items in database(*db_size*)
- 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*)

Table 1 Parameters Setting for Simulation

Parameters	Values
db_size	100
num_cpus	2
num_disks	4
short_tran_size	2, 3, 4, 5, 6
long_tran_size	4, 5, 6, 7, 8
tran_creation_time	5
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.

Values for parameters were chosen by reflecting real world computing practices. Database size matters if it affects the degree of conflict. If *db_size* is much larger than *short_tran_size* and *long_tran_size*, conflicts rarely occur. To see performance tradeoff between *2DL* and *2PL*, average transaction length represented by number of operation in transaction were treated to vary. The shortest one is assumed to access 2 percent of the entire database, while it is 80 percent for the longest one.

5.2 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.

5.2.1 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. For this experiment, all the other simulation parameters are the same as those that we already used before at Figures 5 and 6 except size of long 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.

2PL performs best in terms of throughput however, *2DL* performs best in terms of average

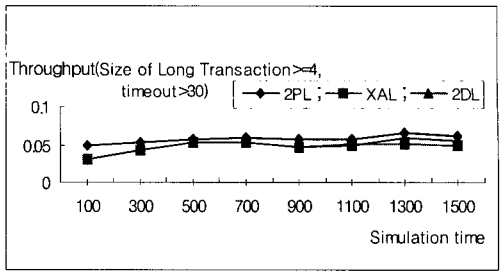

Fig. 5 Throughput with Larger Long-Lived Transaction

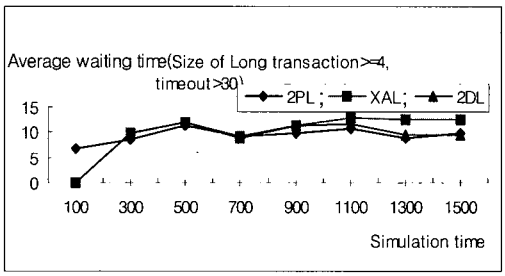

Fig. 6 Average waiting time with Larger Long-Lived Transaction

waiting time owing to two-way donation which contributes to give transactions more chance to use the objects than one-way donation. 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.

5.2.2 Effect of Timeout

At a higher range of *timeout*, *2PL* shows a higher throughput and a lower transaction waiting time.

Performances of both schemes appear to be downgrade as the number of outstanding transactions increases, thereby causing congestion. Throughputs of *2PL* and *XAL* show the same value from simulation time 100 through 900. Throughput of *2DL* tends in particular to degrade from the point of 100 in Figure 7 and *2DL* outperforms *XAL* and *2PL* in terms of transaction throughput until simulation time 900. However, *2PL* rapidly increases its throughput from 1100 to 1300 in Figure 8 and we can also observe that average waiting time curve of 2PL rapidly comes down from 1100 to 1300 in Figure 8. This phenomenon shows us higher throughput gives lower average waiting time also.

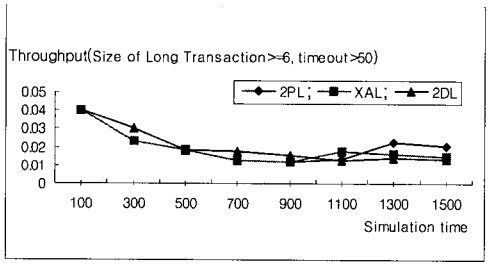

Fig. 7 Throughput with Longer Timeout

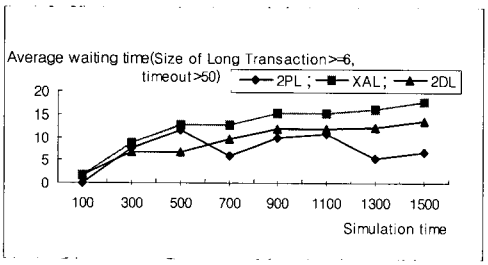

Fig. 8 Average Waiting Time with Longer Timeout

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.

References

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

[4] R. Agrawal, M. J. Carey and M. Linvy, "Concurrency Control Performance Modeling: Alternative and Implications," ACM Transactions on Database Systems, Vol. 12, No. 4, pp. 609-654, December 1987.

[5] A. Law, and W. Kelton, Simulation Modeling & Analysis, Second Edition, McGraw-Hill, 1991.

[6] P. Welch, The Statistical Analysis of Simulation Results, Computer Performance Modeling Handbook, Academic Press, pp. 267-329, 1983.

[7] A. Pritsker, Introduction to Simulation and SLAM II, Third Edition, Systems Publishing Corporation, 1986.

[8] J. Lee, and S. Son, Performance of Concurrency Control Algorithms for Real-Time Database Systems, Performance of Concurrency Control Mechanisms in Centralized Database Systems, Prentice Hall, pp. 429-460, 1996.

[9] S. C. Moon and G. G. Belford, Performance Measurement of Concurrency Control Methods in Distributed Database Techniques and Tools for Performance Analysis, Sophia Antipolis, France, pp. 279-296, 1985.

이 혜 경(Hae-kyung Rhee) 정회원

1979년 2월 : 숭실대학교 전자계산학과 졸업

1985년 4월 : University of Illinois(Urbana-Champaign) 전산학과 석사

1996년 3월~현재 : 성균관대학교 전기전자컴퓨터공학부 박사과정

1988년 3월~1989년 2월:국립천안공업전문대학 전자계산과 전임강사

1992년 3월~현재:경인여자대학 멀티미디어정보전산학부 조교수

<주관심 분야> 온라인 거래처리, 분산데이터베이스, 이동데이터베이스

김 응 모(Ung-mo Kim) 정회원

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

<주관심 분야> 데이타마이닝,Web데이타베이스, 객체지향DB, 트랜잭션관리