분산환경에서 거래관리를 위한 두단계 기부 잡금규약

이 혜경*, 김용모**

요 약

테이터베이스 기술의 적용분야가 점차 확대되어감에 따라 작업처리율을 증대시키기 위한 다양한 형태의 거래 처리 모형들은 필요로 하는 추세이다. 그러나 기존의 syntax위주의 작업성 이론만 가지고서는 거래의 실행시간 상 차별화 특성을 수용하면
서 다수의 거래에 대한 단위시간당 처리 생성성을 높이는 힘든 혼란이다. 이러한 상황을 해결하기 위하여 이타적 잡금기법
(altruistic locking: AL)은 거래가 객제를 사용한 다음 다음 그 객제를 요구하지 않음을 때 다른 거래들이 그 객제를 로크할 수 있도록 두려 객제에 대한 로크를 해제함으로써 거래들의 대기시간을 줄이기 위한 취지에서 제안된 것이다. 확장형 이타적 잡
금(extended altruistic locking: XL)기법은 AL을 지배의 환경 축면에서 개선한 잡금기법으로서 AL이 근본적으로 안전하고 있는
반드시 기부된 객제만을 처리해야 한다는 부담을 보다 유화한 기법이다. 본 논문에서는 우선 잡기기법에 의한 단기거래의 장
기적 대기현상 완화 축면에서의 AL와 XL의 공통적 한계점을 분석하였다. 분산 환경에서 잡기거래로 인한 대기거래의 장
기적 대기현상을 최소화하도록 향후에 동시성 제어의 적도를 높이는 반면, 거래간의 평균 대기시간을 줄일 수 있는 새로운
확장형 이타적 잡금법인 전후간방식의 신형 확장 기법인 2DL(two-way donation locking)을 제안하였다. 기법의 적용 광
범위성을 위해 분산 계산 환경에서도 각선될 수 있게 설계하였다. 도의실형에 의한 성능평가 결과 잡기거래의 갱이 5이상, 9이하인 상황에서 2DL은 2PL보다 작업 처리율과 거래의 평균 대기시간 면에서 우수한 결과를 나타내었다.

Two-Way Donation Locking for Transaction Management in Distributed Database Systems

Hae-Kyung Rhee†· Ung-Mo Kim**

ABSTRACT

Database correctness is guaranteed by standard transaction scheduling schemes like two-phase locking for the context of concurrent execution environment in which short-lived ones are normally mixed with long-lived ones. Traditional syntax-oriented serializability notions are considered to be not enough to handle in particular various types of transactions 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 under the circumstances that the size of long-transaction is in between 5 and 9.

† 정 회원 : 경인지리대학 컴퓨터정보전산학과 교수
†† 정 회원 : 성균관대학교 전기전자 및 컴퓨터공학부 교수
논문접수 : 1999년 10월 11일, 심사완료 : 1999년 11월 12일
1. Introduction

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.

As long as long transactions and short transactions live 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.

2. Related Work

2.1 Altruistic Locking

The basic idea of AL is to allow long transactions to release their locks early, once it is determined that the data which the locks protect will no longer be accessed. Unlike other early-released protocols, the AL guarantees serializable executions and places no restrictions on the way data must be accessed. Unlike other early-release approaches, the AL strategy guarantees serializable executions and places no restrictions on the way data must be accessed.

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.

2.2 Extended Altruistic Locking

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 1 shows this.
Example 1 (Delay Effect Caused by Donation Extension): Suppose that $T_1$ 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 $T_1$ at all. Assume that $T_1$ has already locked and successfully donated $A$, $B$, and $C$. $T_1$ now is supposed in the stage of accessing D. Suppose also that there are three more transactions concurrently in execution along with $T_1$: $T_2$ wishing for $B$ and $E$, $T_3$ wishing for $E$ and $F$, and $T_4$ wishing for $F$ and $H$ (Figure 1).

(Figure 1) Four Transactions, T1 through T4, Competing for Same Data Donated

If we apply XAL for this situation, $T_2$ could in some circumstances fortunately be allowed to access both $B$ and $E$ without experiencing any delay.

In case $T_2$ initially requests $B$ first rather than $E$, $T_2$ is able to access not only $B$ but $E$ as well, since $T_2$ is fully in the wake of $T_1$. $T_2$ therefore succeeds to commit. $T_3$ then could acquire $E$ released by $T_2$. $T_4$ could thereafter acquire $F$ released by $T_3$.

In case, however, $T_2$ initially requests $E$ first rather than $B$, $T_2$ 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, $T_2$ can be allowed to access $B$ only after it is finally released by $T_1$. $T_2$ in this case is therefore blocked. $T_3$ must then be blocked for $E$ to be released by $T_2$. $T_4$ as well must be blocked for $F$ to be released by $T_3$, forging a chain of blockage.

End of Example 1.

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 unidonation 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.

3. Transaction Processing Model

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).

3.1 Transaction Processing Model
TM receives transactions from clients and passes them SCH queue(Figure 2). TM could receive a message informing abortion from SCH or an acknowledgment informing completion of a requested operation from DM. DM analyzes an operation from SCH to determine which data item the operation is intended to access, and then sends the operation to the disk where the requested data item is stored. The server in each disk executes operations in its own FIFO queue one at a time. Whenever an operation is completed at the server, it sends to TM the message informing that the requested operation has been completed successfully.

(Figure 2) Transaction Processing Model

3.2 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. In Example 2 shows this.

Example 2(Allowing Proceeding for Short-Lived Transaction with Multiple Concurrent Long-Lived Ones): Suppose that $T_i$, a long transaction, attempts to access data items $A$, $B$, $C$, $D$ and $E$ in an orderly manner. Note that data items $F$, $G$, $H$, $I$ and $J$ shall never be accessed by $T_i$ at all. Presume that $T_i$ has already locked and thereafter donated $A$ and $B$. $T_i$ now is supposed in the stage of accessing $C$. Suppose also that there are two more transactions concurrently in execution along with $T_2$: $T_2$ wishing for $J$, $I$ and $B$, and $T_3$ wishing for $J$ and $I$ in an orderly manner(Figure 3).

(Figure 3) Execution of $T_1$ with Two More Concurrent Transactions

If we apply XAL for these transactions, lock request for $B$ by $T_2$ would be rejected due to donation extension. $T_2$ unfortunately cannot be included in the wake of $T_i$. While $T_2$ experiences delay, $T_3$ would not be permitted to access $J$ and $I$ because they were still locked by $T_2$.

In case 2DL is adopted rather than XAL, $T_2$ could fortunately be allowed to access $B$ without any delay. The scheduler checks whether $T_i$ will later access $J$ and $I$. The scheduler then includes $J$ and $I$ into the wake of $T_i$, since we presumed that $T_i$ will not request those data items. When $T_3$ attempts to request $J$ and $I$, $T_3$ is as well allowed to continue to acquire locks without delay because data item $J$ and $I$ have already been donated by $T_2$. All of data items that accessed by $T_3$ could in turn be included into
the wake of \(T_1\). If there are many transactions like \(T_3\), the scheduler has a burden to maintain enlarged wakes. This sort of deficiency would fortunately not incur a substantial burden to the system because the access time of short transactions usually do not take too long.

End of Example 2.

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 \(T_1\), 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 4).

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, \(G\) 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 3.

4. Two-Way 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 = Donation)

THEN

/* Locks being requested by ST already donated by long trans other than LT1 and LT2 */

FOREACH \(\in LT1 \cup LT2\)

IF (ST.wake = LT1) THEN

/* Donation conducted by LT1 */

IF (ST.data \(\in LT1\).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 \(\in LT2\).marking-set) THEN

/* Data being requested by ST to be later accessed by
5. 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.

5.1 Simulation Model

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).

5.2 Simulation Parameters

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):

- 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 ($tran_{creation}_{time}$)
- Mean time for deadlock time out ($timeout$)
- Simulation length ($sim_{leng}$)

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$db_{size}$</td>
<td>100</td>
</tr>
<tr>
<td>$num_{cpus}$</td>
<td>2</td>
</tr>
<tr>
<td>$num_{disks}$</td>
<td>4</td>
</tr>
<tr>
<td>$short_{tran}_{size}$</td>
<td>2, 3, 4, 5</td>
</tr>
<tr>
<td>$long_{tran}_{size}$</td>
<td>4, 5, 6, 7, 8, 9, 10</td>
</tr>
<tr>
<td>$tran_{creation}_{time}$</td>
<td>5 units</td>
</tr>
<tr>
<td>$timeout$</td>
<td>30, 50</td>
</tr>
<tr>
<td>$sim_{leng}$</td>
<td>100, 300, ..., 1500</td>
</tr>
</tbody>
</table>

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.

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.

5.3 Simulation Results and Their Interpretations

Our simulation experiments were focused on the
effects of sensitive parameters in the performance indices to measure their performance behaviors. We now discuss the results of simulation experiments performed for the three different replication control schemes: 2PL, XAL, and 2DL. Overall behaviors have been revealed that as the simulation length gets longer, in terms of throughput 2PL and 2DL perform similarly.

5.3.1 Effect of Multiprogramming Level

The major force behind the prevalence of 2DL mainly comes from capitalizing advantage from maintaining two different transaction wakes. Performance gain of 2DL against 2PL is about 135 per cent in terms of average waiting time (Figure 6). 2PL however outperforms XAL and 2DL with 105 per cent of performance at transaction throughput (Figure 5). This is because both XAL and 2DL have a certain overheads to reserve data objects to be accessed.

We can observe that the waiting time curve of 2DL tends to be flat as the simulation length is getting longer. 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.

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.

5.3.2 Effect of Transaction Size

This experiment is used to investigate the effect of the size of transaction on the performance of concurrency control schemes, as the degree of donations varies.

Overall simulation result shows that 2DL performs best in terms of throughput and its average waiting time since as the size of transaction gets longer, two-way donation which contributes to give transactions more chance to use the objects than one-way donation. We can observe that the throughput curves of 2DL inclines rapidly down from simulation time 300 (Figure 7). The throughput curve of 2DL tends to be flat as the simulation length is getting longer.

2DL and 2PL eventually perform similarly in terms of average waiting time, at simulation time 700 (Figure 8). It seems that the performance between two schemes appear to be about the same, however the average waiting time of 2PL exhibits slightly better behavior than 2DL. As the transaction size gets longer, 2DL outperforms the other schemes due mainly to enhanced
degree of freedom given to 2DL in accessing donated data by extending to multiple donation.

(Figure 8) Average Waiting Time with Larger Transactions

6. Conclusions

The performances of 2DL, XAL, and 2PL have been evaluated through simulation approach under various workloads in order to probe their performance tradeoffs. The simulation results indicate that 2DL is capable of providing a moderate performance across a wide range of workloads. 2DL leads to its superior performance in the property of average waiting time in most cases. We have also observed that XAL exhibits the worst performance among the three with respect to an average waiting time, due to wake expansion overhead.

2DL is considered to be a practical solution to take in real world environments where long-lived transactions naturally coexist with short-lived ones. 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 traditional standard transaction scheduling schemes, such as two-phase locking, the degree of concurrency might be hampered by selfishness associated with lock retention in which short-lived ones are normally mixed with long-lived ones. Lazy release in turn could aggravate the fate of long-lived ones in that they are more vulnerable to get involved in deadlock situations. This could the other way around contribute to exacerbate the fate of short-lived ones as well in a way that they suffer from starvation or livelock affected by long-lived ones.

References

이 혜경
e-mail: rhee@k.dove.kyungin.ac.kr
1979년 숭실대학교 전자계산학과
졸업
1985년 University of Illinois
(Urbana-Champaign)
전산학과 석사
1995년 현재 성균관대학교 전기전자컴퓨터공학부 박사과정
1988년 ~ 1989년 국립천안공업전문대학 전자계산과 전임
강사
1992년 ~ 현재 경인여자대학 멀티미디어정보 전산학부
교授
 관심분야: 온라인 거래처리, 분산데이터베이스, 이동대
이터베이스

김응모
e-mail: umkim@yurim.skku.ac.kr
1981년 성균관대학교 수학과 학사
1986년 Old Dominion University
전산학과 석사
1990년 Northwestern University
전산학과 박사
1997년 ~ 1998년 University of California, Irvine
전산학과
방문교수
1991년 ~ 현재 한국정보처리학회 논문지 편집부 위원장
1999년현재 성균관대학교 전기전자 및 컴퓨터공학부
교授
 관심분야: 데이터마이닝, Web데이터베이스, 객체지향
DB, 트랜잭션관리