Distributed Real-Time Transaction Commit Processing*

QIN Biao, LIU Yun-sheng

(*College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan* 430074, *China*)

E-mail: biaoqin@263.net

Received June 4, 2001; accepted May 10, 2002

Abstract: It is difficult for a distributed real-time transaction to satisfy its deadline because of the complexities of its commitment processing. A new commit protocol called A2SC (active double space commit) is proposed, which cater for the distributed real-time transaction commitment processing. All kinds of dependencies caused from data conflicts access are analyzed. When data conflicts between the prepared transaction and the execution transactions occur, A2SC allows the execution transactions to access the locked data optimistically in a controlled manner. When the prepared transaction aborts, only transactions in its abort dependency set are aborted. Furthermore, a notion of fruitless run is proposed. When a transaction finds fruitless run, it will actively abort. Extensive simulation experiments have been performed to compare the performance of the A2SC with other protocols such as the base protocol, the PROMPT and the DDCR. The simulation results show that A2SC is highly successful in minimizing the number of missed transactions deadlines. So A2SC caters for high-performance distributed real-time transaction.

Key words: distributed real-time transaction; deadline; commit protocol; abort dependency; commit dependency

Researchers had proposed some real-time commit protocol in the literatures. Gupta[1] has proposed *Opt* protocol, which allowed requesters to access data held by committing transactions. Further Gupta[2] proposed *Healthy-Opt* protocol. In Healthy-Opt, a health factor H_t is associated with each transaction T and a transaction is allowed to lend its data only if its health factor is greater than a minimum value M. Haritsa proposed PROMPT[3]. The protocol allows transactions to "optimistically" borrow, in a controlled manner, the updated data of transactions currently in their commit phase. This controlled borrowing reduces the data inaccessibility and the priority inversion that is inherent in distributed real-time commit processing.

Lam proposed DDCR[4]. Conflict resolution in the DDCR is divided into two parts: (a) resolving conflicts at the conflict time; and (b) reversing the commit dependency when a transaction, which depends on a committing transaction, wants to enter the decision phase and its deadline is approaching.

This paper proposes a new real-time commit protocol, called Active double space commit (A2SC). The main contributions of this paper are the followings:

First, A2SC proposes the notion of fruitless run and extends the property of "active abort"[3]. Second, A2SC

* Supported by the National Natural Science Foundation of China under Grant No.60073045 (国家自然科学基金); the Defence Pre-Research Project of the 'Ninth Five-Year-Plan' of China under grant No.00J15.3.3.JW0529 (国家"九五"国防预研基金); the National Doctoral Fundation of China (博士点基金)

QIN Biao was born in November 1972. He is a Ph.D. candidate at the College of Computer Science and Technology, HUST. His research interests are distributed real-time database, active database and electronic commerce. **LIU Yun-sheng** was born in 1940. He is a professor and doctoral supervisor at the College of Computer Science and Technology, HUST. His current research areas include advanced databases, database and information system development, real-time data engineering, and software methodology and engineering technology.

allows a non-healthy transaction to lend its locked data to the transactions in its commit dependency set. Third, when a prepared transaction aborts, only the transactions in its abort dependency set are aborted.

The rest of this paper is organized as follows. Section 1 describes a distributed real-time database system model. Section 2 proposes A2SC protocol. Section 3 reports on the performance evaluation of the new protocol. Finally, in Section 4 we present the conclusions of our study.

1 Distributed Real-Time Transaction Model

The transaction model used in this paper is firm real-time transaction. We modify the firm deadline semantics in the distributed environment as follows[3].

Definition 1. A distributed firm-deadline real-time transaction is said to be committed if the master has reached the commit decision before the expiry of the deadline at its site. This definition applies irrespective of whether the cohorts have also received and recorded the commit decision by the deadline.

To ensure transaction atomicity with the above definition, the prepared cohorts that receive the final decision after the local expiry of the deadline still implement this decision. And the transactions, which would normally expect the data to be released by the deadline, only experience a delay. Sharing of data items in conflicting modes creates dependencies among the conflicting transactions and constrains their commit orders. Basically there are two kinds of dependencies, commit dependency and abort dependency[5].

Definition 2. The set, called abort dependency set, is made up of those transactions which are abort dependent on transaction T, which is denoted by $ADS(T)$. So $ADS(T)=\{T_i|T_iADT\}$.

Definition 3. The set, called commit dependency set, is made up of those transactions which are commit dependent on transaction T, which is denoted by $CDS(T)$. So $CDS(T)=\{T_i|T_iCDT\}$.

Definition 4. That a transaction's run will miss its deadline is called fruitless run.

A health factor is defined as follows[3], HF_T(health factor=$TimeLeft/MT_T$), where $TimeLeft$ is the time left until the transaction's deadline, and MT_T is the minimum time required for commit processing. The health factor is computed at the point of time when the master is ready to send the prepare messages. $MinHF$ is the threshold that allows the data held the by committing transaction T to be accessed. The fruitless run lies in three cases: First, a cohort T is in executing state and $D(T)-MT_T<0$; Second, $HF_T<0$; Third, $D(T)<0$ and T is in the voting phase.

2 The A2SC Protocol

The communication delays caused by message exchanges constitute substantial overheads to the response time of a distributed transaction. Thus, it becomes more difficult to satisfy the timing constraint of transactions in DRTDBS than in a centralized one[6]. In order to cater for the deadline, the A2SC protocol cannot require many message passings in processing the local commitment. Thus a transaction table at each site maintains the followings information for each local active transaction or cohort T_i:

$CDS(T_i)$: the set of transactions which are commit dependent on transaction T_i;

$ADS(T_i)$: the set of transactions which are abort dependent on transaction T_i.

2.1 "Active Abort" in global transaction space

In global transaction space, a cohort will be "active abort" because of the following two reasons. First, a cohort will abort whenever it finds that its run is fruitless run. Second, a cohort that is not yet in its commit phase can be aborted due to conflicts with higher priority transactions. If the first case happens, there is no need for the master to invoke the abort protocol since the cohorts of the transaction can independently realize the fruitless run. If the second case happens, it may be better for an aborting cohort to immediately inform the master so that the abort of the transaction at the sibling sites can be done earlier.

```
If a cohort $T_i$ finds its fruitless run
        It will abort silently without exchanging messages with its master
else
        if a cohort $T_i$ conflicts with a higher transaction && $T_i$ is not in commit phase {
                $T_i$ tells its master that it will abort
                $T_i$ aborts }
```

2.2 "Optimistically Processing" in local transaction space

In the local transaction space, let T_1 be a local transaction or cohort holding a lock on data item x and be in committing state, and let T_2 be a local transaction or cohort requesting the same data item x in the executing state. The essences of A2SC protocol are (1) concurrency control is based on lock scheme; (2) each transaction has a health factor and T_2 is allowed to access the data held by the committing transactions in a controlled manner.

When data conflicts occur, there are three possible cases of conflict: (1) write→read conflict; (2) write→write conflict; (3) read→write conflict. The details in resolving data conflict are processing as follows:

```
If   $T_1$ is fruitless run
        Active abort $T_1$
Else {
        If   $T_2$ is fruitless run
                Active  abort $T_2$
        Else {
                If   $T_2 CD\ T_1$ {
                        $CDS(T_1) = CDS(T_1) \cup \{T_2\}$ ;
                        $T_2$ is granted to write-lock;}
Else {
        If  $HF_{T_1} \geq MinHF$  {
                $ADS(T_1) = ADS(T_1) \cup \{T_2\}$
                $T_2$ is granted to read-lock;}
        Else
                $T_2$ will be blocked; }
                }
}
```

When T_2 had accessed the locked data, three situations may arise. (1) T_1 receives decision before T_2 has completed local data processing; (2) T_2 completes data processing before T_1 receives its global decision; (3) T_2 aborts before T_1 receives its global decision. So the protocol will process as following:

```
One: if $T_1$ receives global decision before $T_2$ ends
        execution
    {if   $T_1$'s global decision is commitment
        {$T_1$ enters the decision phase
            all transactions in $CDS(T_1) \cup ADS(T_1)$ will
              be released and execute as usual }
    else
        { $T_1$ aborts
          aborts all transactions in $ADS(T_1)$
          all transactions in $CDS(T_1)$ will be released
            and execute as usual}}
else if $T_2$ ends execution before $T_1$ receives global
            decision
{ $T_2$ waits until $T_1$ receive global decision ||
            $T_2$ is fruitless run
    if   $T_1$  receives global decision
            goto One
    else {
            $T_2$ active aborts
            Deletes $T_2$ from $CDS(T_1) \cup ADS(T_1)$}}
else { undoes and aborts $T_2$
            deletes $T_2$ from $CDS(T_1) \cup ADS(T_1)$}
```

2.3 Discussion on the proposed protocol

The A2SC is an improvement for PROMPT. The improvement lies in three aspects. (1) A2SC proposes the notion of fruitless run and extends the property of "active abort". So it will actively abort fruitless run transactions, which will minimize the wastage of both logical and physical system resources. (2) When data conflict occurs, Non-healthy transaction can lend locked data to its commit dependency transactions in A2SC. Because any borrower has to be blocked if it completes data processing before the lender receives global decision, A2SC cannot influence the serialization order. (3) If the prepared transaction aborts, the transaction in its commit dependency set can execute as usual in A2SC.

And we don't change other properties of PROMPT. So A2SC cannot bring other negative influences. Thus A2SC is an improvement for PROMPT and has the properties of PROMPT.

3 Performance Evaluation

3.1 Simulation model

The simulation experiments are based on ARTs-II, which is a distributed real-time database system test bed developed by our lab. The global database (GDB) is modeled as a collection of DBSize pages that are uniformly distributed across all the sites. At each node, transactions arrive in an independent Poisson rate. The baseline setting of the values for the parameters is shown in Table 1. The base protocol is 2PC.

Table 1 Baseline values for the model parameters

Parameter	Meaning	Default setting
N_{site}	Number of sites	4
SiteID	Site Ids	0,1,2,3
AR	Arrival rate	3 transactions/s
T_{com}	Communication delay	100 ms (constant)
SF	Slack factor	1~4 (uniform distribution)
P_{write}	Write operation probability	0.0~1.0
PageCPU	CPU page processing time	5ms
PageDisk	Disk page processing time	20ms
DBSize	Database size	200 data objects/site
N_{oper}	Number of operations in a transaction	3~20 uniformly distributed

Upon arrival, each transaction T is assigned a deadline using the formula $D_T=A_T+SF*R_T$, where D_T, A_T and R_T are its deadline, arrival time and resource time, respectively, while SF is a slack factor. The transaction priority assignment policy used is the widely used Earliest Deadline First (EDF). All the cohorts of a transaction inherit their master's priority. Messages also retain their sending transaction's priority. For concurrency control, the simulation system adopts an extended 2PL High Priority (E2PL-HP) protocol.

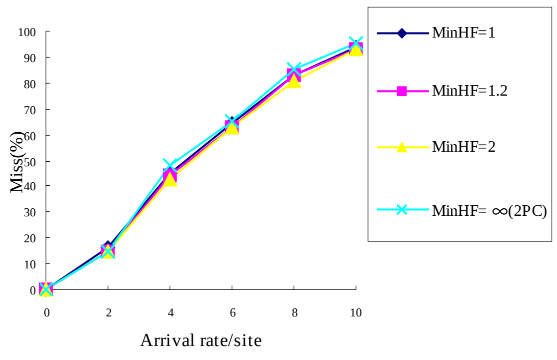
In DRTDBS, the major performance measure is the missing rate, which is defined as: $MissPercent=T_{\text{miss}}/T_{\text{Total}}$, where, T_{miss} and T_{Total} are the number of transactions missing their deadlines and the total number of transactions processed respectively. The way to evaluate the system workload is as following: (1) if $0<MissPercent<20$, system is under normal loads; (2) if $20<MissPercent<100$, system is under heavy loads.

3.2 Analysis the result of experiments

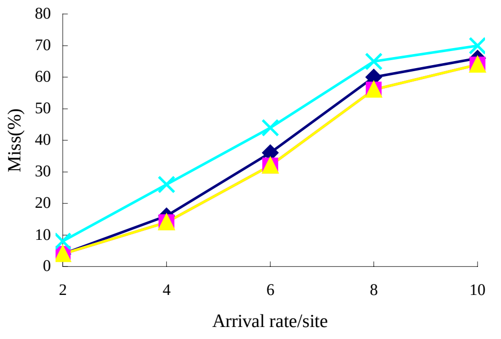
Using the firm-deadline DRTDBS model described in the previous section, we conducted an extensive set of simulation experiments comparing the performance of A2SC with that of the 2PC, PA, DDCR and PROMPT. Our experiment was conducted using default settings for all model parameters (Table 1), resulting in significant levels of both resource contention (RC) and data contention (DC).

3.2.1 Determination of the value for Min*HF*

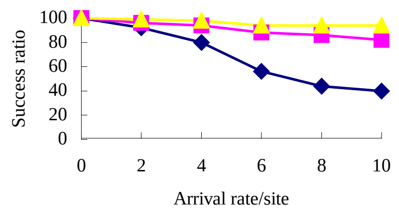
A range of Min*HF* values is considered in our experiments, including Min*HF*=1, Min*HF*=1.2, Min*HF*=2, Min*HF* =∞, which is equivalent to the 2PC protocol. The results for these various settings are shown in Fig.2. From the Figs.1(a),(b), we can see that their *MissPercent* is approximately equal when Min*HF*=1.2 and Min*HF*=2. We can see that Min*HF* =1.2 is in general better than Min*HF* =1 and Min*HF* =∞ and especially so under heavy loads in the pure DC environment. From Figs.1(c)(d), we can see the success ratio of Min*HF*=1.2 is seen to be much higher than that of Min*HF*=1. And the success ratio of Min*HF*=2 is only slightly better than that of Min*HF*=1.2. We observe that the performance of 2SC for Min*HF*=1.2 and Min*HF*=2 is almost identical. So Min*HF*=1.2 is efficient to control optimistically data access. Thus the threshold of Min*HF* is 1.2. The remaining experiments on the 2SC will use this value for Min*HF*.



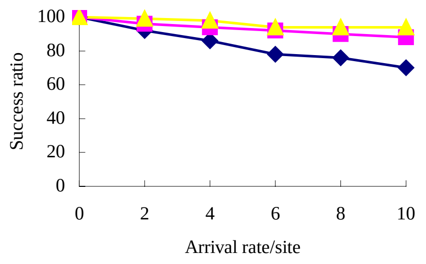
(a) MissPercent (RC+DC)



(b) MissPercent (pure DC)



(c) Success ratio (RC+DC)



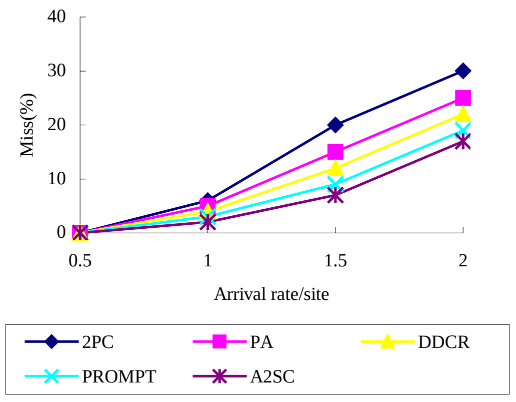
(d) Success ratio (pure DC)

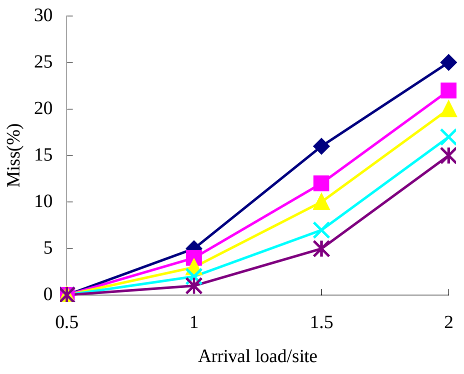
Fig.1

3.2.2 Performance analysis in different situation

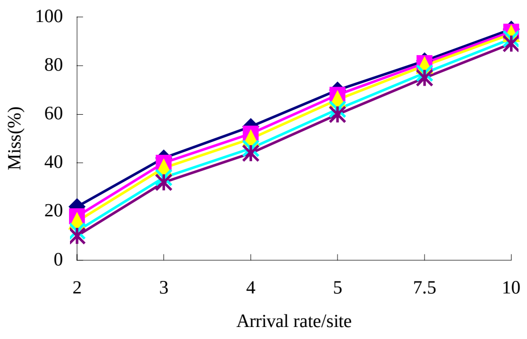
From the Fig.2, we can see the performance between A2SC, PA[7], DDCR, PROMPT and 2PC has little difference at normal workload. Resources and data conflicts access seldom take place in local site. With workload increasing, conflict access to resources and data between transactions occurs frequently. In these figure we see PA always does slightly better than 2PC because PA has lower communication overheads when a transaction aborts.

In DDCR, a higher degree of concurrency can be achieved, as executing transactions can access data items held by committing transactions in conflicting modes. Also, the impact of temporary failures on transactions in dependencies set is much reduced by reversing the dependencies between the transactions. So DDCR has consistently better performance than PA. Three copies of the data may be temporarily created for a data item in the database in DDCR, which needs much more system resources and serializability checking is difficult. Reversing the dependency may induce a committing transaction abort. Although PROMPT has properties of healthy lending, it only needs one copy of data. The property of active abort can reduce system resources wasted by the aborting transactions. And the property of silent kill can save system resources by eliminating a round of messages when the transaction misses its deadline. So the performance of PROMPT is considerably better than that of DDCR over most of the loading range. Finally, we compare the performance between A2SC and PROMPT. From the Fig,2, we can see the performance of A2SC is in general slightly better than that of PROMPT. This is because A2SC has the following new properties (1) it can actively abort the fruitless run transaction, (2) non-healthy transaction can lend locked data to its commit dependency transactions, (3) if the prepared transaction aborts, the transaction in its commit dependency set can execute as usual. So A2SC has better performance than that of PROMPT. Therefore, the A2SC has the best performance.





(b) Normal load (pure DC)



(d) Heavy Load (pure DC)

Fig.2

4 Conclusions

This paper proposes a distributed transaction model and A2SC protocol. In this model, it is the master who makes decision about the transaction fate. If the master can give the decision before its deadline, the transaction will not miss its deadline. In A2SC, we propose the notion of fruitless run and extend the property of “active abort”. A2SC has two properties. One is actively abort in global transaction space. The other is optimistically processing in local transaction space. Finally we integrate A2SC with E2PL-HP. In this way we can ensure transaction serializability and atomicity. We compare the performance of A2SC with that of 2PC, PA, DDCR and PROMPT. The results of simulation experiments have shown that some improvement can be obtained with the use of the A2SC as compared with the base and other optimistic protocols.

References:

[1] Gupta, R., Haritsa, J., Ramamritha, K., *et al*. Commit processing in distributed real-time database systems. In: Proceedings of the 17th IEEE Real-time Systems Symposium. 1996. 220～229.

[2] Gupta, R., Haritsa, J., Ramamritha, K. More optimistic about real-time distributed commit processing. In: Proceedings of the 18th IEEE Real-Time Systems Symposium. 1997. 123～133.

[3] Haritsa, J., Ramamritham, K., Gupta, R. The PROMPT real-time commit protocol. IEEE Transactions on Parallel and Distributed Systems, 2000,11(2):160～181.

[4] Lam, K., Pang, C., Son, S.H., *et al*. Resolving executing-committing conflicts in distributed real-time database systems. The Computer Journal, 1999,42(8):674～692.

[5] Ramamritham, K., Chrysanthis, P.K. A taxonomy of correctness criteria in database applications. VLDB Journal, 1996,5(1):85～97.

[6] Franaszek, P.A., Robinson, J.T., Thomasian, A. Concurrency control for high contention environments. ACM Transactions on Database Systems, 1992,17(2):304～345.

[7] Mohan, C., Lindsay, B., Obermarck, R. Transaction management in the R* distributed database management system. ACM Transactions on Database Systems, 1986,11(4):378～396.

分布式实时事务提交处理

覃 飙, 刘云生

(华中科技大学 计算机科学与工程学院,湖北 武汉 430074)

摘要: 由于提交处理的复杂性,分布式实时事务很难满足其截止期.提出了一种新的提交协议 A2SC(主动的双空间提交),它适合于分布式实时事务提交处理的需要.分析了由于数据冲突访问而形成的各种依赖关系.当处于准备状态的事务和处于提交状态的事务发生数据冲突访问时,A2SC 允许处于执行状态的事务在一种控制的方式下乐观地访问锁住的数据.当处于准备状态的事务夭折时,仅仅只有其夭折依赖集中的事务夭折.进一步提出了“没有结果的运行”的观念.当一个事务发现它是没有结果的允许时,它将主动夭折.进行了广泛的模拟实验比较 A2SC 和其它协议比如基准协议、PROMPT 和 DDCR 的性能.模拟结果表明 A2SC 在最小化错过截止期的事务数方面较成功,因此 A2SC 适合于高性能分布式实时事务.

关键词: 分布式实时事务;截止期;提交协议;夭折依赖;提交依赖

中图法分类号: TP311 文献标识码: A