Probabilistic Concurrency Control for Mobile Application

Kailash Prasad Dewangan[#] , Somesh Dewangan*

#Department of Computer Science & Engineering
Disha Institute of Management and Technology
Raipur (C.G.), India
** Department of Computer Science & Engineering*
Disha Institute of Management and Technology
Raipur (C.G.), India

[1]kaishapt@yahoo.com,
[2]somesh_4@gmail.com

***Abstract*- As we already know, the distributed transactions in mobile peer-to-peer systems are expected to be long-lived. So the shared data items would be locked and other transactions would be blocked for a long time if we adopt this protocol in mobile peer-to-peer systems.**

In order to improve transaction throughput in the system, we need reduce the constraint of the two-phase lock protocol. At the same time, we want to control cascading aborts. We need to find a proper balance between long blocking and cascading aborts. We propose a probabilistic approach to concurrency control. Instead of locks, we use references to record the shared access to data.

***Keywords*- Distributed Transaction, Peer to Peer System, Concurrency Control, Probabilistic Protocol, Commit Protocol**

I. INTRODUCTION

A. *Distributed Transactions*

One logical transaction may include multiple components, which execute at different sites. As we discussed before, this logical transaction could be a traditional database transaction or a collaborative application with transactional properties. A distributed transaction can be represented as T (P1, P2, …,Pn), in which P1, P2, …, Pn are participants of the transaction.

B. *Transaction Group*

There are two kinds of groups in a distributed transaction: logical group and physical group. Logical groups are formed on the basis of collaboration relationships among participants. Nevertheless, all participants belong to one group.

There are sub-groups within the group. All these logical groups form the group hierarchy.Physical groups are formed on the basis of physical connectivity among participants. For the discussion of transaction commit protocols in this chapter, we will assume that a transaction group usually refers to a physical group.

1. *Write-SetT:* **T**he set of data items updated by the transaction T.
2. *Read-SetT*: The set of data items read by the transaction T.

Based on the above definition, the system model could be defined as follows. There are N participants for a distributed transaction. There are M partitions among these N participants. A transaction could be represented as <V, G>. V is the vertex set and each vertex is a participant. G is the group set and each group represents a connected partition.

II. PROBABILISTIC CONCURRENCY CONTROL

Current currency control mechanism utilizes locks to control the access to shared data. There are two kinds of locks: read lock (shared lock) and write lock (exclusivelock). Two locks from different transactions accessing to the same data are compatible if both locks could co-exist without compromising the serializability. The compatibility matrix in Table I demonstrates the compatibility between these access modes.

TABLE I
THE COMPATIBILITY MATRIX FOR ACCESS MODES

		Access Mode Requested	
		Read	Write
Access Mode Existed	Read	Yes	No
	Write	No	No

The two-phase locking protocol is the main concurrency control mechanism to enforce serializability and conflict-free. There are a variety of two-phase locking protocols depending on when locks are released. In all varieties of two-phase locking

protocols, all lock requests precede all unlock requests for every transaction.

"Cascading abort" occurs when abort of one transaction leads to abort of other transactions. In order to avoid cascading abort, usually all locks are released after the transaction commits. (This is also called the strict two-phase locking protocol.) Hence,other transactions cannot access data items locked by uncommitted transactions.

Instead of locks, we use references to record the shared access to data. A reference table is created to record the list of current transactions referencing each data item. Each mobile device has a reference table for the data items that it owns. All these referencing transactions could be regarded as staying at the second phase of two-phase locking protocols.

A reference table described in Table II is an inverted index where each data item is followed by current transactions referencing the data item and transactions waiting for accessing the data item. Referencing transactions are ordered based on the order of accessing a data item.Waiting transactions are also ordered based on the order of initially waiting for the data item. Transaction commit probability is used to describe the possibility of a transaction's being committed.

TABLE II
REFERENCE TABLE

Data Item	Ordered List of Referencing Transactions	Ordered List of Waiting Transactions
….	….	….

***Probabilistic protocol*:** When a transaction T tries to access the data items in its write set and read-set, the scheduler will check the reference table. If the access mode requested is compatible with the previous referencing access modes, the transaction T can access this data item and it will be added into the ordered list of referencing transactions.Otherwise, instead of blocking the transaction T in traditional concurrency control mechanism, the scheduler will compute the transaction commit probability $P_c(T)$ of the transaction T. If $P_c(T)$ is less than the probability threshold P_t, the transaction T will be blocked and added into the waiting list of the data item.

How to compute transaction commit probability of a transaction and how to set the probability threshold are two keys to the probabilistic approach. We will discuss the details in the following.

Fig I describes the dependency graph among transactions and data items. Thecommit probability of a transaction needs to consider two aspects: vertical aspect and horizontal aspect. The vertical aspect describes the commit probability based on the reference hierarchical levels of a single data item being accessed by a transaction. The horizontal aspect describes the commit probability based on multiple data items begin accessed by a transaction.Data item ordered list of referencing Transactions Ordered list of waiting transactions

In the following, we discuss transaction commit probability based on the vertical aspect, i.e., probability based on a single data item. At the first level, a transaction can access this data item since no other transactions reference it. The commit probability of atransaction T,

$$\mathbf{Pc(T)} = \frac{\boldsymbol{Ng}}{\boldsymbol{Nt}} \qquad (2.1)$$

in which *Ng* is the number of transaction participants in the group and *Nt* is the total number of participants in this transaction. If the commit probability *Pc*(*T*) of the transaction T >*Pt*, the data item could be accessed by the transaction T.

At the lower level, if the access modes are not compatible, whether we could grant the access to the data item is based on the resulting commit probability.Let X be the dataitem being accessed. Let *Pt* be the probability threshold of the transaction T*k*+1 at level k+1.

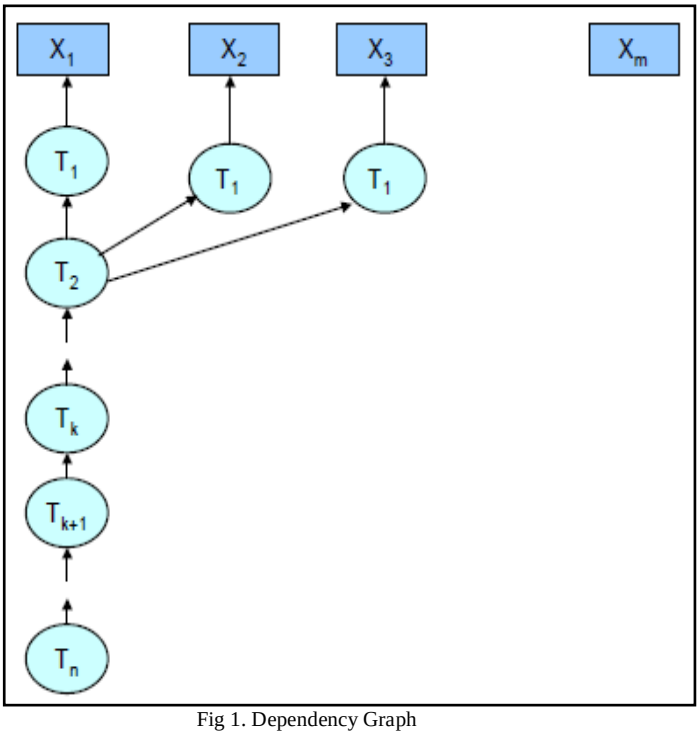

Fig 1. Dependency Graph

Let *Tk* be the transaction at level k. (The level stands for the referencing level of X.)The commit probability of Tk+1,

$$\mathbf{Pc(T_{k+1})=\frac{Ng(k+1)}{Nt(k+1)} * Pc(Tk) * \alpha()} \quad (2.2)$$

Ng(*k*+1) is the number of transaction participants in the group for T*k*+1,N*t*(*k*+1) is the total number of participants in T*k*+1 and α is a reduction factor and 0<α <1. If the commit probability *Pc(Tk*+1) of the transaction T*k*+1 >*Pt*, the data item X referenced by *Tk* could be accessed by the transaction T*k*+1.

What is the status of referencing transactions in the multi-state transaction model?Usually a referencing transaction is staying at the *tentatively committed* state. There is not enough information from all participants of the transaction to decide whether the transaction will finally commit. So they could commit or abort. When a referencing transaction at the top level commits, this transaction will be removed from the reference table. At the same time, we will adjust the status of transactions at the lower level and decide to commit them or continue to wait for more information. When a referencing transaction at the top level aborts, all transactions at the lower level will abort. This causes cascading abort.

Fortunately, the degree of cascading abort is controlled. When a referencing transaction at the middle level decides to commit, it remains at the tentatively commit state. When its high level transaction commits, it will finally commit. When a referencing transaction at the middle level decides to abort, it will also abort the transactions that are at the lower level.

Now, let us look at the horizontal aspect. A transaction T may try to access multiple data items X1, X2, …,Xm. So we can compute multiple transaction commit probabilities (Pc(T,X1),Pc(T,X2),……Pc(T,Xm))respectively for data items (X1, X2, …, Xm). The commit probability

Pc(T)=MIN(Pc(T,X1),Pc(T,X2),…Pc(T,Xm)) **(2.3)**

The other problem is to choose the proper probability threshold P_t. The choice of *Pt* decides the level of transaction commit ratio and the degree of cascading abort. Obviously, 0<= P_t<=1. When *Pt* increases, the possibility of a transaction being blocked increases and the degree of cascading abort decreases. We can adjust P_t based on the dynamic environment and the feedback from previous results.

When P_t= 0, transactions can access the data items anytime. There is no blocking. The serilizability may be maintained through other ways; however it will cause unlimited cascading abort.

Lemma 2.1: When P_t= 1, the probabilistic approach is a strict two-phase lockingprotocol.
Proof: From (2.1), we have

$$\mathbf{Pc(T)=\frac{Ng}{Nt}}$$

When *Pt*= 1, so *Pc*(*T)*>=1. We can deduce that Ng=*Nt.*

That means all participants agree to commit.
From (2.2), we have

$$\mathbf{Pc(Tk+1)=\frac{Ng(k+1)}{Nt(k+1)} * Pc(Tk) * \alpha()}$$

Since *Pt*=1, *Pc*(*T)* =1.

$$\mathbf{As\ 0<\alpha<1\ and \frac{Ng(k+1)}{Nt(k+1)} \leq 1}$$

We will have Pc(T*k*+1)<1. So Pc(T*k*+1) will not be able to access the data items until P*c(Tk)* commits.

As the minimum commit probability is chosen to be the commit probability of a transaction when the transaction needs to access multiple data items. The conclusion still stands.

So the probabilistic approach is a strict two-phase locking protocol when the commit probability threshold is 1.

Another important problem for the probabilistic approach is to limit the effect of cascading abort. Lemma 2.2 gives the upper bound of the level of cascading abort.

Lemma 2.2*: The upper bound of the level of "cascading abort"* is $\mathbf{\frac{ln(Pt)}{ln(\alpha} + 1}$

Proof: From(2.2), we have

$$\mathbf{P_c(T_k)=\frac{Ngk}{Ntk} * Pc(Tk-1) * \alpha}$$

Recursively applying (2.2), we have

$$P_c(T_k)=\frac{Ngk}{Ntk} * \frac{Ng(k-1)}{Nt(k-1)} *..* \frac{Ng2}{Nt2} * Pc(T1) * \alpha^{k-1} \quad (2.4)$$

From (2.1), we have

$$\mathbf{P_c(T1)=\frac{Ng1}{Nt1}}$$

Replacing Pc(T1)in (2.4), we have

$$\mathbf{P_c(Tk)=\frac{Ngk}{Ntk} * \frac{Ng(k-1)}{Nt(k-1)} * ... * \frac{Ng2}{Nt2} * \frac{Ng1}{Nt1} * \alpha^{k-1}} \quad (2.5)$$

Let

$$\mathbf{\beta=\frac{Ngk}{Ntk} * \frac{Ng(k-1)}{Nt(k-1)} * ... * \frac{Ng2}{Nt2} * \frac{Ng1}{Nt1} 1}$$

Obviously β is a constant and 0<β <=1
So we have

$$\mathbf{P_c(T_k)= \beta * \alpha^{k-1}} \quad (2.6)$$

As we know, a transaction can be allowed to access the data items only if it satisfies the following condition: ($Pc(T_k)$ is the minimum commit probability from multiple data items)

$$\mathbf{P_c(T_k) >= P_t}$$

That means $\mathbf{\beta * \alpha^{k-1} >= Pt}$

It could translate to $\mathbf{\alpha^{k-1} >= \frac{Pt}{\beta}}$

Take logarithm at both sides, we have

$$\mathbf{(k-1) * \ln(\alpha) >= \ln(P_t) - \ln(\beta)} \quad (2.7)$$

Since 0<α <1, so ln(α) < 0
Divided ln(α) from both sides in (2.7), we have

$$\mathbf{(k-1) <= \frac{\ln(Pt)}{\ln(\alpha)} - \frac{\ln(\beta)}{\ln(\alpha)}}$$

This is equal to

$$\mathbf{k <= \frac{\ln(Pt)}{\ln(\alpha)} - \frac{\ln(\beta)}{\ln(\alpha)} + 1} \quad (2.8)$$

As 0 <Pt<=1and 0 < β <= 1, we know ln(α) < 0 . So we can know

$$\mathbf{\frac{\ln(Pt)}{\ln(\alpha)} > 0 \text{ and } \frac{\ln(\beta)}{\ln(\alpha)} > 0}$$

From (2.8), we could deduce that

$$\mathbf{K <= \frac{\ln(Pt)}{\ln(\alpha)} + 1} \quad (2.9)$$

So this gives the upper bound of the level of cascading abort.

Discussion: α is the reduction factor that is used to control the commit probability inthe vertical aspect. As 0<α <1, a transaction T2 in the lower level should have smaller commit probability than a transaction T1 in the higher level as T2 accesses the data item after T2. When α increases, the priority that T1 has over T2 is smaller; the possibility of a transaction being blocked decreases and the degree of possibly cascading abort increases.So α is also called *Vertical Control Factor*. In contrast, *Pt* is called *Horizontal Control Factor* as it controls the threshold for the commit probability of a transaction accessing multiple data items. When *Pt* increases, the possibility of a transaction being blocked increases and the degree of cascading abort decreases.

III. A GROUP BASED TRANSACTION COMMIT PROTOCOL

Although the transaction commit protocol is a well-researched database topic, it has been studied mostly in the context of distributed computing. The two-phase commit protocol is the most widely used commit protocol. In the first phase, a coordinator polls all participants whether they want to commit or abort the transaction. In the second phase, the coordinator sends the decision to all components. Usually the decision is to commit if and only if all participants agree to commit.

In the traditional two-phase commit protocol and the three-phase commit protocol, they assume that there is a central commit coordinator and all participants of the transaction could communicate with this commit coordinator.

When these commit protocols moves to mobile and possibly ad-hoc computing, we find that frequent disconnections and partitions among transaction participants render current protocols in effective and that no strategies exist for this setting yet. In turn, we devised a group based transaction commit protocol, which is based on our multi-state transaction model, in the presence of dynamic partitions.

A transaction process can be divided into two stages: execution and commit. So each participant of a transaction has these two stages. To simplify the discussion of the commit protocol in this section, we ignore the execution stage in the participant.

(**Definition 3.1**) *Synchronous Path*: A communication path existing between two sites that are located in the same partition at the same time.

(**Definition 3.2**) *Asynchronous Path*: If two sites do NOT have a synchronous path, but they are able to communicate through other sites passing the information between them at different time. Such a communication path is called an asynchronous path. Delay tolerant networks describe the architectures for forming such an asynchronous path.

The assumptions for the group based commit protocol are described as follows:
i) Partitions can be detected.
ii) An asynchronous path exists between any two members.
iii) Each site maintains a log.
iv) A central coordinator is not always available.
v) Participants of a transaction are decided at the beginning of the transactionand will not be changed.

As we do not assume a central coordinator, we need to find a group coordinator for each group. The process of electing a group coordinator is described as follows

Step 1: Randomly pick any participants of the group

Step 2: Apply utility functions and find the optimal one. For example, the utility function could be the cost of communicating with other participants in the group.

TABLE III
GROUP BASED COMMIT PROTOCOL

Input: G(P1, P2, …, Pm, Q1, Q2,…, Qn)
Process:

Step 1: P1 initiates the transaction, P2, ..., Pm and Q1, ... Qn receive it. G is partitioned into two groups G1 (P1,..., Pm) and G2 (Q1,..., Qn). Let G1 and G2 elect the leader sites E1 and E2 for respective partitions.

Step 2: P1, P2… , Pm continue the process (Assume the process in each participant does not need the information from participants in the other partitions. For example, a query may be assigned to multiple distributed sites.). Similarly, Q1, … ,Qn also continue the process. So G1 and G2 maintain group ACID properties and are in the group active state. If there are no conflicts within G1 or G2, G1 or G2 tentatively commits. Otherwise G1 or G2 aborts. Within each group, the two-phase commit protocol could be used to decide whether it should be tentatively committed or aborted.

Step 3: When the membership of G1 or G2 changes (e.g., some participants in G1 join G2 or some participants in G2 join G1, or G1 and G2 are merged into G'), we will know more information to make the decision whether the transaction should be kept tentatively committed, aborted or completely committed.

In this protocol, each site maintains a tentatively committed transaction queue. For each tentatively committed transaction, it stores the transaction ID, states (variables,corresponding values, constraints), and the associated group. Each site also maintains an aborted transaction queue. If the queue is too long, the participant will export it to the disk. All these tentatively committed transactions and aborted transactions are recorded in the local transaction LOG.

Now let's revisit the *correctness criteria* for commit protocols proposed by Bernstein,Hadzilacos and Goodman.

i) All processes that reach a decision, reach the same one.
ii) A process cannot reverse its decision after it has reached one.
iii) The commit decision can only be reached if all processes voted OK.
iv) If there are no failures and all processes voted OK, the decision will be tocommit.
v) Given any execution schedule containing failures (of the type that the algorithm is designed to tolerate), if all failures are repaired and no new failures occur for a sufficiently long time, then all processes will eventually reach a decision.

The group based commit protocol is described as Table III. The input is a distributed transaction with (n+m) participants. Network partitioning divides the transaction participants into different groups. We need to decide whether the transaction should be committed or aborted. In traditional commit protocols, the transaction will abort and restart later.

If these participants are not connected together into a partition again, the transaction will keep aborting and not be able to complete. In the group based commit protocol, the transaction still could make the decision even if these participants are not connected into a partition again. The basic idea is to utilize the frequent membership change within different partitions. When a participant changes the membership from one group to another group, it could carry the transaction states in the old group into the new group. So the participants in the new group will know the decisions made in the old group so it will make the proper decision.

As we can see from the group based commit protocol, a tentatively committed transaction could be promoted to being committed or demoted to being aborted. But acommitted transaction will not be reversed. This satisfies Criterion ii). Based onAssumption ii) and the protocol, eventually all processes (in different groups) will reach the same decision. So this satisfies criterion i) and iii). An aborted transaction will abort the transaction from all participants. For Criterion iv), if there are no transaction state information exchanges among partitions, a transaction may be aborted after a long time even if each participant in every partition votes to commit. However, assumption ii) will make sure the asynchronous path among these partitions, so Criterion iv) is also satisfied. Based on the same reason, Criterion v) will also be met. So the group based commit protocol will guarantee correct results.

If we want to extend the group based transaction commit protocol from abort veto mode to majority vote mode, the above correct criteria should be modified to base on majority processes instead of all processes.

IV. CONCLUSION

Our probabilistic concurrency control mechanism provides a generalized approach to deal with long blocking while controlling the degree of "cascading abort". The commit probability threshold could be dynamically adjusted based on the feedback statistics from previous samples, changes of environment and expectation of concurrency level.

The group based transaction commit protocol utilizes the multi-state transaction model to addresses frequent disconnections and network partitioning. The transaction state information in one partition could be carried over to another partition. A distributed transaction could still complete even if all participants could not be connected at the same time. Combining probabilistic concurrency control mechanism with the group based commit protocol; we could reduce the blocking and improve the transaction throughput and the transaction commit ratio.

REFERENCES

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

[2] P. K. Chrysanthis, "Transaction Processing in a Mobile Computing Environment", in IEEE workshop on Advances in Parallel and Distributed Systems, October 1993.

[3] Dirckze, R. and L. Gruenwald, "A Pre-Serialization Transaction Management Technique for Mobile Multidatabases", ACM Mobile Networks and Applications,Volume 5, Number 4, December 2000, pp. 311-321.

[4] Dirckze, R. and L. Gruenwald, "A Toggle Transaction Management Technique for Mobile Multidatabases", ACM Conference on Information and Knowledge Management (CIKM), November 1998, pp. 371-377.

[5] Eddie Y.M. Chan, Victor C.S. Lee, and Kwok-Wa Lam , "Using Separate Processing for Read-Only Transactions in Mobile Environment", LNCS 2574, pp106-121,2003.

[6] M.H. Eich and A. Helal, "A Mobile Transction Model that Captures Both Data and Movement Behavior", ACM-Baltzer Journal on Special Topics on Mobile Networksand App, 1997.

[7] H. Garcia-Molina, "Using semantic knowledge for transaction processing in adistributed database", ACM Trans. Database System 8, 2 (June 1983), 186-213.

[8] T. Imielinksi and B. R. Badrinath, "Wireless Mobile Computing: Challenges in Data Management", Communications of the ACM, October 1994, 37(10), pp. 18-28.

[9] B. Li and K. Nahrstedt, "A Control-based Middleware Framework for Quality of Service Adaptations", IEEE Journal of Selected Areas in Communication, SpecialIssue on Service Enabling Platforms, vol. 17, num. 9, pp. 1632-1650, September,1999.

[10] E. Pitoura and B. Bhargava.A Framework for Providing Consistent and Recoverable Agent-Based Access to Heterogeneous Mobile Databases. ACMSIGMOD Record, 24(3): 44--49, September 1995

[11] E. Pitoura and B. Bhargava, "Building Information Systems for Mobile Environments", in the 3rd Intenational Conference on Information and KnowledgeManagement (CIKM), 1994

[12] S. K. Prasad, V. Madisetti, R. Sunderraman, et al. "System on Mobile Devices (SyD): Kernel Design and Implementation", in First International Conference onMobile Systems, Applications, and Services (MobiSys), Poster and Demo Presentation, May 5-8, 2003, San Francisco.

[13] S. K. Prasad, A. G. Bourgeois, E. Dogdu, et al. "Enforcing Interdependencies and Executing Transactions Atomically Over Autonomous Mobile Data Stores Using SyD Link Technology", in Mobile Wireless Network Workshop held in conjunctionwith The 23rd International Conference on Distributed Computing Systems (ICDCS), May 19-22, Providence, Rhode Island.

[14] W. Xie , S. B. Navathe, S. K. Prasad., "Supporting QoS-Aware Transaction in the Middleware for a System of Mobile Devices (SyD)," in the 1st InternationalWorkshop on Mobile Distributed Computing in The 23rd International Conference onDistributed Computing Systems (ICDCS), May 19-22, 2003 Providence, RhodeIsland.

[15] W. Xie , S. B. Navathe, "Transaction Adaptation in System on Mobile Devices(SyD): Techniques and Languages," in Symposium of Database Management inWireless Network Environments in the 58th IEEE Vehicular Technology Conference(VTC03 Fall), Oct 7-10, 2003, Orlando, Florida.