EVALUATION OF CPM - A COMMIT PROTOCOL FOR MOBILE DISTRIBUTED DATABASE SYSTEMS

Le Huu Lap

Posts Telecommunications Institute of Technology

Nguyen Khac Lich, Tran The Truyen

Research Institute of Posts and Telecommunications

1. Introduction

Recent explosion of *mobile computing* has promoted the research and development on mobile transactions, especially on distributed database systems (MDDBS) [2, 3, 8, 10]. The MDDBS are typically built on top of wired-cum-wireless network, which consists of mobile devices (such as PDA, laptop, etc) called *Mobile Units* (MU) communicating with fixed network via *Base Stations* (BS) or *Mobile Support Stations* (MSS) (Figure 1). Each BS has number of wireless communication channels for MUs, which move and handoff from cell to cell. A distributed transaction generally involves participation of several sites, which usually distribute over the fixed network, but some of them may be in wireless network. For the purpose of transaction study, the network can be viewed as a logical mess (Figure 2).

Such MDDBS must execute commit protocols to ensure atomicity after the transaction executions on data. Typically the commit protocols are based on message passing between participants in the transaction, in one or more different phases such as *1PC* (One-Phase Commit), *2PC* (Two-Phase Commit) and *3PC* (Three-Phase Commit) [4, 5, 10]. Unlike with fixed network, those protocols may not work well in wireless medium and with the flexibility of user mobility for various reasons [8, 10]:

- wireless connection is prone to failure due to the nature of radio propagation like fading, obstacle, and interference,
- MU has limited battery power, processing speed, and stable storage,
- handoff rate is unpredictable,
- limited wireless channels for sharing between MUs,
- typically the communication delay is higher due to unreliable physical connections, and
- for database Participating sites are not usually in the same network partition, they are connecting for a short duration.

In this paper we evaluate a new commit protocol - *CPM* (Commit Protocol for Mobile) that is designed to over come some limitation of the most widely used 2PC and to allow user-enabled disconnections and offline processing. Section 2 provides a short review of several commit protocols and points out drawbacks of 2PC in mobile computing. Section 3 describes and analyzes the theoretical aspects of the CPM. The next two sections present the modeling and simulation of the CPM.

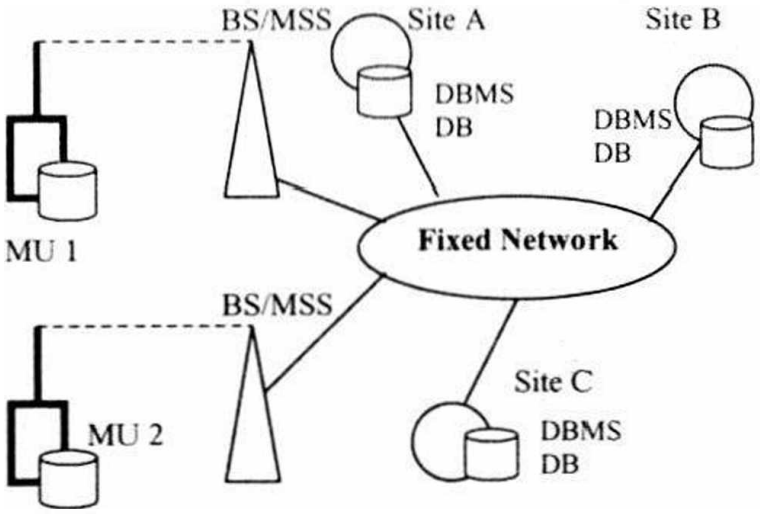

Figure 1: Generic physical model of MDDBS

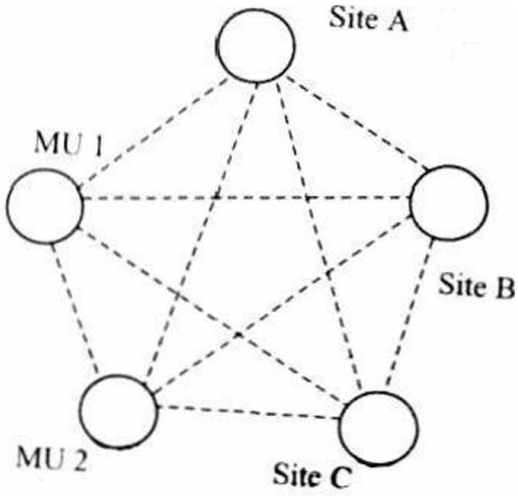

Figure 2: Logical view of DDBS

2. Commopitment in distributed transaction processing

2.1. Transaction Models

There have been many transaction models proposed for mobile computing [2, 3, 12] that capture classical transaction management features such as concurrency control and recovery [1] and mobility. For the purpose of this paper, they are assumed to already exist. A distributed transaction T_i initiated at a site consists of number of fragments $\{T_{ij}\}$ which are distributed to participating sites to execute. The generic transaction model considered in this paper consists of a set of applications $\{APP^k / \text{ for arbitrary } k \in J\}$ originating transaction T_i at MUs. A T_i requires the involvement of a set of participants $\{P^j / j \in J, J=\{1,2,...,m\}\}$ and a set of corresponding transaction controller $\{TC^j\}$. Each transaction controller TC^j handles commands related to transaction on behalf of participant P^j. One of the TC^j is chosen to be the *coordinator* (CO) for coordinating the transaction in *Master/Slave* fashion. There are no requirements on the location of TC^j in the network and with respect to P^j but the coordinator should be located in reliable wired network to ensure satisfactory performance.

2.2. Commit protocols

It is a strict requirement of atomicity in database transactions, that a transaction T_i must either fully committed or aborted at all participating sites. To ensure this property, a *commit protocol* is executed by the transaction coordinator,

after all sites inform their completion of T_i. Among the most widely deployed is the *Two-Phase Commit* (2PC) protocol, which we will describe in more detail in the next sub section. Even though, 2PC is considered to be insufficient because of high commit cost and potential of I/O *blocking*. There have been many attempts to improve the 2PC and to propose alternatives to solve these issues. Some aim at minimizing costs of log writing and communications, in which *PrC* (Presumed Commit) and *Early Prepare* [9, 11] are of those noticeable. Other research includes 1PC and variants such as *CLP* (Coordinator Log Protocol), and *IYVP* (Implicit Yes-Vote Protocol). 1PC ignores the PREPARE phase of 2PC but its assumptions are too strict to be useful in commercial systems. In contrast to those approaches, *3PC* (three-phase commit) introduces more cost and complexity by an additional phase to help avoid the I/O blocking in 2PC.

2.3. Two phase commit protocol

The commit protocol, as its name indicates, has two phases: the first asks for voting from all sites ready to commit, and the second carry out the commitment.

Phase 1: CO adds record <<PREPARE T_i>> to the log, writes the log to stable storage and starts a timer (PREPARE_TIMEOUT) then sends PREPARE message to all TC^j. On receiving such a message, each TC^j decides whether or not to commit. If the answer is NO, TC^j adds record <<NO T_{ij}>> to the log and responses ABORT T_{ij} to CO. Otherwise, TC^j adds record <<READY T_{ij}>> to the log and forces the log to the stable storage. TC^j replies READY T_{ij} to CO.

Phase 2: Upon receiving READY T_{ij} from all TC^j within timeout period, CO force-writes T_i log to stable storage. At this point, the fate of the transaction is determined. CO then sends COMMIT T_i to all TC^j. In the bad case, the PREPARE phase timeouts, CO logs T_i and sends ABORT T_i to all TC^j. When TC^j receives a COMMIT or ABORT from CO, it writes the message to the log, then acknowledges back to CO.

Despite the potential of *blocking*, the two-phase commit protocol is currently the most widely used in practice for distributed database systems [10]. However, it may be difficult to apply in mobile distributed environment. For example, the mobile users may temporarily disconnect to process its transaction fragment offline for saving power, and this action can lead to transaction abort, resulting in the high abort frequency. This is because of a critical point in 2PC, that after transaction executions finish, the decision to commit relies entirely on the chosen coordinator, which may not be aware of mobile user preference. Once the CO decides to commit, the connections need to be kept to avoid possible abort due to timeout. This can be done reliably in high quality wired network, but there are no guarantees in the wireless.

In addition, 2PC may not be suitable for MUs with small memory because they have to maintain a local log to handle CO's demands at any time. Another weakness of 2PC is due to its commit nature of 2 phases requiring $4p$ messages (where p is number of participants), which are rather expensive in mobile communications in term of time (and possibly money since the mobile cost is usually determined on the basis of connection time).

3. CPM - Commit protocol for mobile transaction models

3.1. The CPM Commit Protocol

In the light of above discussion, it is reasonable to minimize the need to communicate over wireless link and the number of messages passing in the network per transaction. We proposed a novel commit protocol called *CPM* (*C*ommit *P*rotocol for *M*obile Transactions) that avoids the possible disconnections during the commit process. In addition CPM enables user-proactive disconnection and offline processing for arbitrary amount of time. The disconnection may be initiated by MU's transaction controller before decision to commit is made.

Our commit protocol CPM operates under generic model presented in Section 2.1. Although each TC^j does not need to be at the same site as corresponding P^j, TC^k is set at MU site to avoid possible complexity in local communication via wireless medium. In CPM, the transaction controller of MU (TC^k) where a transaction T_i is originated plays the role of coordinator in 2PC during transaction execution. It divides the transaction T_i into fragment set $\{T_{ij}\}$, extracts its fragment T_{ik} and force-writes the transaction log to stable storage. Instead of sending the rest of transaction $T_{ij} - T_{ik}$ to coordinator as in 2PC, TC^k distributes those fragments to corresponding participants and starts a TRANS_EXEC timeout. Each participating site executes its part and send completion message back to the requester.

If the TC^k receives all such messages, it starts the commit process by sending COMMIT_REQUEST message and transaction log to CO. Upon receiving commit request and transaction log from TC^k, CO fore-writes the log to stable storage and sends COMMIT message to all participating sites. Each site then commits its transaction portion T_{ij} and acks back to CO once finished. If all such acknowledgements are received at CO, the CO logs the transaction and sends COMMIT_ACK message back to TC^k. The transaction is completed from now on and CO can forget about it. Note that there is no need to send COMMIT to TC^k because the COMMIT_ACK is always embedded in COMMIT_REQUEST beforehand. Thus the number of messages passing wireless medium is minimized.

Otherwise the TRANS_EXEC timeout is triggered before TC^k gets all TRANS_ACK from P^j, TC^k sends ABORT message to CO and TRANS_NOT_OK to APP^k. In this case, CO has not logged the transaction yet, so the ABORT message

needs to carry all necessary information about the aborted transaction (i.e. ID of participants $\{P^j\}$). Using given information, CO multicasts ABORT messages to all TC^j ($j \neq k$). TC^j then tells its corresponding P^j to abort.

As mentioned above, the participant P^j and the agent TC^j acting on its behalf are not necessarily at the same site, so during the commit process, P^j may fail and may not verify the commit with TC^j. So TC^j needs to use a COMMIT_EXEC timeout when commanding P^j. Upon receiving such command, P^j finishes its transaction portion and then log-writes a "commit". The transaction portion is done. If the timeout is passed and P^j has not confirmed the commit, TC^j assumes the failure of P^j. TC^j then checks the existence of "commit" log at P^j. If the log does not exist, the TC^j tell P^j to redo the transaction execution and commit procedures.

Once receiving COMMIT_ACK from CO, TC^k sends TRANS_OK to APP^k, the whole transaction is fully completed. The CPM can be expressed semi-formally in term of pseudo code as in Figure 3-Figure 6.

Activities at APP:

```
begin
send transaction T_i to TC^k
wait until receiving instruction from TC^k
if(the instruction is TRANS_OK)
      start new transaction after a time interval
else //the transaction is aborted
      restart transaction after a time interval
end
```

Figure 3: CPM algorithm at application APP

Activities at TC:

```
Begin
if(TC^k receive a transaction T_i) //this plays the role of TC^k
      begin
      TC^k send T_i to the scheduler and then get fragments {T_ij}
      TC^k write all <<LOG T_ij>> to RAM
      TC^k extract its portion T_ik
      TC^k send the rest {T_i - T_ik} to all corresponding P^j
      TC^k process its portion T_ik
      TC^k start timer and wait for confirmation from {P^j}
      if(timeout && TC^k not receive all TRANS_ACK_ij of T_i)
            begin
            TCk send ABORT T_i to CO
            TCk send TRANS_NOT_OK T_i to APP^k
            end
      else //TC^k receive all TRANS_ACK_ij
```

```
                begin
                TCk send COMMIT T_ij, all <<LOG T_ij>>
                and its ack of fragment T_ik to CO
                TCk wait for response from CO
                if(TCk receive COMMIT_ACK T_i from CO)
                        TC^k send TRANS_OK to APP^k
                end
        end
    //From now on TC plays the role of TC^j
    else if(TC^j receive COMMIT T_ij from CO)
        begin
        TC^j send record <<COMMIT T_ij>> and command COMMIT T_ij to P^j
        TC^j start timer and wait for ack of T_ij from P^j
        if(TC^j receive commit confirmation of P^j)
                TC^j send COMMIT_ACK T_ij to CO
        else //timeout
                begin
                //P^j fails
                TC^j check log at P^j
                if(exist the log <<COMMIT T_ij>>)
                        send COMMIT_ACK T_ij to CO
                else
                        TC^j ask P^j to redo the transaction portion T_ij
                        based on log from CO
                end
        end
    else //TCj receive ABORT T_ij from CO
        send ABORT T_ij to P^j
    end
```

Figure 4: CPM algorithm at transaction controllers (TC^k & TC^j)

Activities at P:

```
    begin
    if(P^j receive transaction fragment T_ij from TC^k)
        begin
        execute T_ij
        send TRANS_ACK T_ij back to TC^k
        wait for commit from corresponding TC^j
        end
    else if(P^j receive COMMIT T_ij from TC^j)
        begin
        commit the portion T_ij
        fore-write log <<COMMIT T_ij>> to stable storage
        end
    else if(P^j receive ABORT T_ij from TC^j)
        abort its portion T_ij
    end
```

Figure 5: CPM algorithm at participant P

Activities at CO:

```
begin
if(CO receive ABORT T_i from TC')
      if(no <<LOG T_i>> exist)
            extract participant ID from message ABORT T_i
      else
            examine <<LOG T_i>> to get TC'
      send ABORT T_i to all TC'
else if(CO receive all <<LOG T_i>>,
      COMMIT T_i and COMMIT_ACK T_i from TC')
      begin
      force-write <<LOG T_i>> to stable storage
      send COMMIT T_i to all TC'
      wait for acks from TC'
      if(CO receive all COMMIT_ACK T_i from TC')
            complete the transaction T_i and
            send COMMIT_ACK T_i to TC'
      end
end
```

Figure 6: CPM algorithm at coordinator CO

3.2. Cost analysis of CPM

The CPM is designed with the cost minimization in mind, as shown in Table 1.

Table 1: Cost comparison of commit protocols

Protocol	Number of messages	Log force-writes	Communication delay
2PC	4(n-1)	1+2n	4
PrC	3(n-1)	1+2n	4
EP	n-1	2+n	2
CL	n-1	1+(n + *n_op*)	2
IYV	2(n-1)	1	2
CPM	***2(n-1)***	***1+n***	***2***

where *n_op* is number of transaction fragments.

Although CPM allows flexible offline processing and disconnection during committing, but when too many transactions waiting to in the coordinator queue, the coordinator is forced to cutoff some transactions to avoid process overflow. The cutoff mechanism can be as simple as round-robin to remove the oldest item in the queue, but this is open to later implementations

4. Simulation modeling

In addition to qualitative analysis presented in previous sections, we have quantitatively investigated the performance of CPM in different scenarios and in

comparison with the 2PC. We developed a Discrete Event Simulator (DES) that runs the model of the generic wired-cum-wireless network topology (see Figure 1). The main idea behind the DES approach to simulation is based on concept of event, an occurrence in the system at discrete time. Events are often created in advance ahead of current time and scheduled and activated at predefined time-stamp associated with them.

The main flow unit in the system model is message (data & signaling), that is appropriate for current model of mobile communications. The commit protocols are operating on the application layer where the underlying network provides the transparent communication channels. Although our simulator supports arbitrary network topology, but for the purposes of this paper, the fixed network is implemented as a generic node with primary functionality of routing. For simplicity we do not implement any mechanism for checking for error and message ordering at Transport layer as usual in communication network, the loss of message and disconnection and reconnection in wireless link are modeled as sufficiently long delay. This is because it is usual for transport layer to assume message loss based on timeout when network congestions occur. Similarly, handoff is also modeled as long delay with the average delay time for handoff is 1s. Taking into account the fact that average velocity of mobile users is 3m/s, average cell diameter is around 300m, so the handoff rate is about one every 100s.

Each site consists of an application APP, a transaction controller TC and a database (or participant). Transactions are always initiated at MU and the new one is created 4s after the completion or abort of the previous. Each portion T_{ij} of transaction T_i is assigned to a participant in the network randomly, except for the originating site (MU). Because there are multiple transactions being concurrently requested at CO and participating sites, each site needs to maintain a transaction queue. For simplicity, the queue is implemented on First Come First Serve (FCFS) basis. To avoid possible processing overload, the transaction queue at CO operates in round-robin strategy, where the oldest awaiting transaction will be aborted when the queue is full.

At each site, CPU execution and I/O disk access are implemented in mutual exclusive fashion taking a certain amount of time depending on nature of operations. Other requests have to wait until the completion of the previous. Currently no parallel processing is modeled, but it can be simulated by adjusting the average processing time.

We use the mean delay of 10ms (V. Kumar, 2002) when sending message in wireless environment, and in wired network, the average transmission time is 5ms. Parameters used in simulation models are shown in **Table 2**:

Table 2: Parameters setting in simulation

Log Force-write	200ms
Transaction Segment Execution	34ms
Time to set one lock	1ms
Time to unlock	1ms
Data object update time	6ms
Transaction Degree of distribution	7-10 fragments
Message handling time	1ms
Average wireless transmission delay	10ms
Average wired transmission delay	5ms
Database system degree of distribution	>= 10 sites
Timeout	60s

The simulation software is developed in standard C++ to promote portability. Currently it supports 2PC and CPM with similar working conditions so that they can be compared with each other.

5. Experiments and Results

We have conducted a comprehensive experiment set to evaluate the performance of two commit protocols of interest: 2PC and CPM. The performance is primarily measured using metric of *transaction throughput* per unit of time. Other factors include *portion of successful transaction*, *average number of messages per transaction* and *transaction turnaround time* (Figure 10). The transaction turnaround time is defined as the duration between the start and the end of the successful transaction at originating application (at MUs in this study). The results presented in this paper are only those of the most interest.

For the system architecture setting, each transaction T_i can optionally selects a new CO, but for purposes of performance study under heavy loads, there is only one CO is set for the whole simulation process, for both cases of 2PC and CPM. For most studies, the network consists of 10 DBS but the simulator allows arbitrary number of DBS.

5.1. Performance comparison under normal condition

This experiment investigates throughput of 2PC and CPM under load of multiple parallel active MUs. The system runs under the disconnection probability on wireless link of 0.5% - in normal situation. The queue size is 10 set for both 2PC and CPM.

Under this condition, the CPM performs better than 2PC until it reaches maximum (Figure 7). After that point, the system throughput of CPM decreases

rapidly while the 2PC performance degrades rather gradually. Following the point where number of active MUs is roughly 35, the CPM works worse than 2PC does. Figure 9 shows the average number of message of the two protocols. This reassures previous theoretical analysis (**Table 1**). Figure 8 presents the same study as in Figure 7, but under weak wireless environment (disconnection rate 4%). This will be studied in the next sub section.

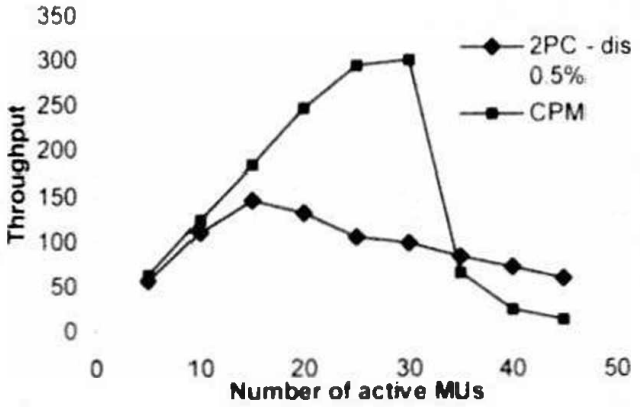

Figure 7: Performance of 2PC and CPM under load with normal disconnection rate (0.5%)

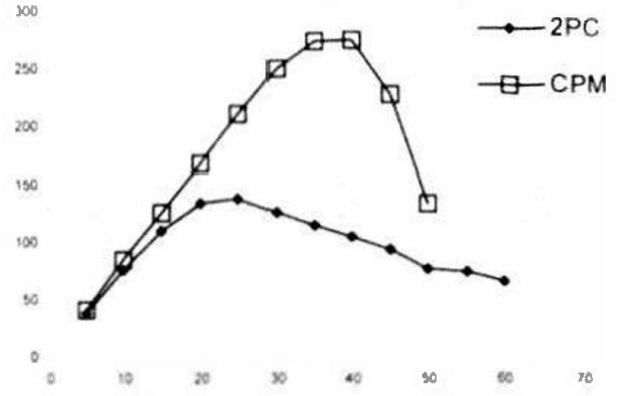

Figure 8: Performance of 2PC and CPM under load with high disconnection rate (4%)

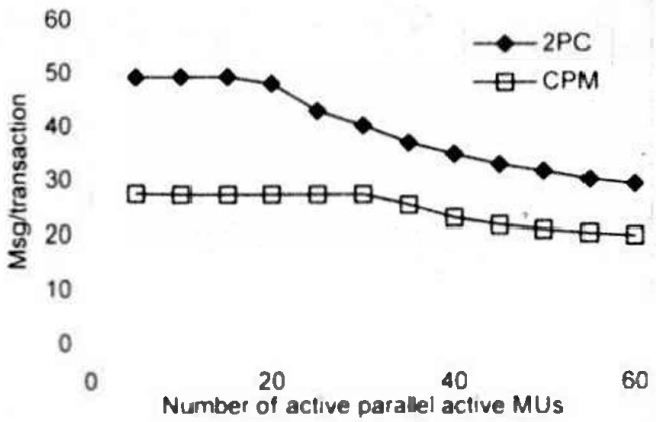

Figure 9: Average number of messages per transaction

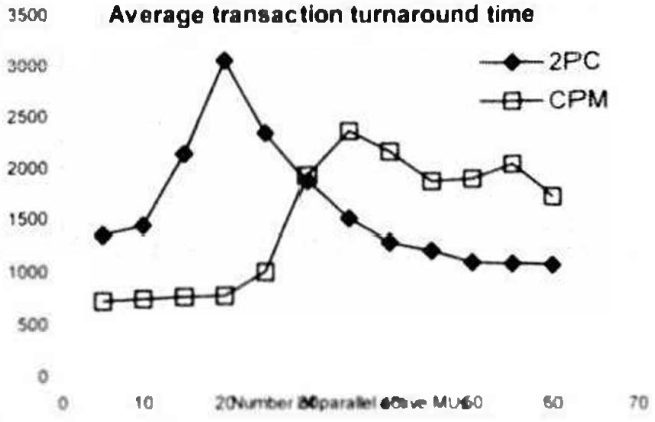

Figure 10: Average transaction turnaround time

5.2. Effect of handoffs and wireless link failures

The common property between handoffs and wireless link failures are long delay (ranging from seconds to hours) compared to normal transmission time (1-10ms). However, the average handoff time using in this study is around 1s, which is much less than typical timeout value (minutes to hours). We run the simulation of various average handoff rates ranging from no handoffs to 2 handoffs/minute, or with the speed of 10m/s (or 36km/h) across the cell diameter of 300m. The results confirm our intuition as there is no significant degrade in overall system throughput as handoff rate grows.

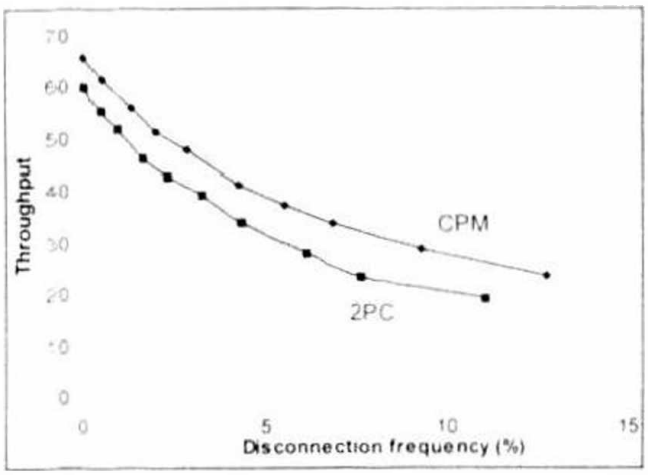

Figure 11: Effect of disconnection frequency on overall throughput (handoff rate is 0.01)

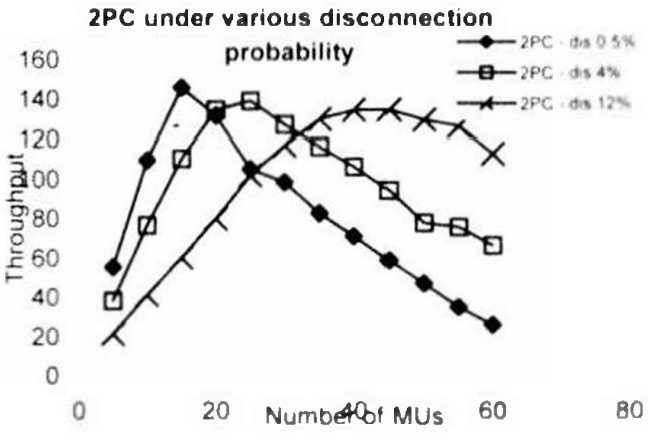

Figure 12: Performance of 2PC with different transaction disconnection rates (0.5%, 4%, 12%)

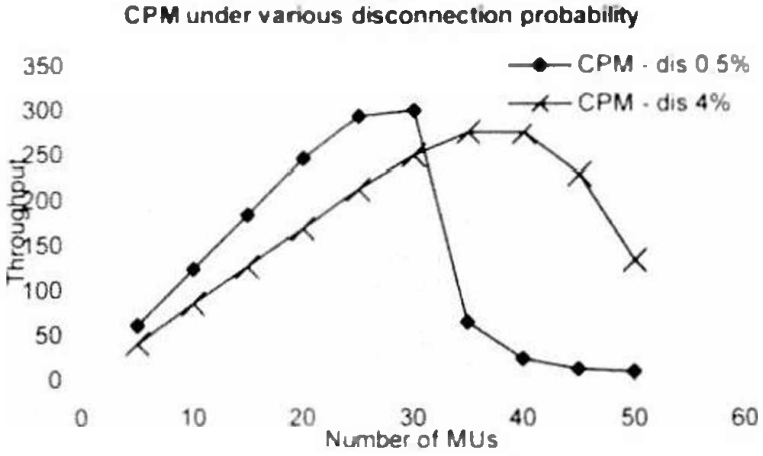

Figure 13: Performance of CPM with different transaction disconnection rates (0.5%, 4%)

The next experiment (Figure 11) is to investigate how frequent disconnection changes overall system throughput. The disconnection frequency is the probability that a transaction can be disconnected during its lifetime. Although the probability being studied is rather artificial (up to 12%), the simulation demonstrates that both 2PC and CPM performance drops rapidly as the probability of wireless link failure increases. This is due to timeout occurrences, during the commit process in 2PC and during the transaction execution in CPM. In case of 2PC, the decision to abort due to timeout in PREPARE phase is made entirely by CO. This abort message may take significant amount of time to reach TC^k to schedule the transaction restart. In CPM, for most of the time, the timeout occurs at TC^k, and the transaction restart is immediately scheduled. Together with the fact that the number of messages for each successful transaction in 2PC is lager than that in CPM, one can expect that CPM performs consistently better on overall.

The performance of CPM under load with different transaction disconnection probability (Figure 13) tends to have the same characteristics as that of 2PC

(Figure 12). That is the weak communication medium makes the time to reach maximum throughput longer and also decreases the maximum value.

5.3. System performance in relation with queue size at CO

It is intuitive that the queue size at CO influences the overall system performance because of the round-robin operation. A set of experiments has been conducted to confirm this conclusion (Figure 14). The overall throughput does not change with queue size when the number of parallel active MUs is small. However, it appears to determine when the system degrades under heavy load. The larger queue size, the better it can handle multiple awaiting transactions, which is clearly roughly proportional to number of active MUs operational in the network. Similarly, the average transaction turnaround time is directly related to queue size in closed queue theory (Figure 15).

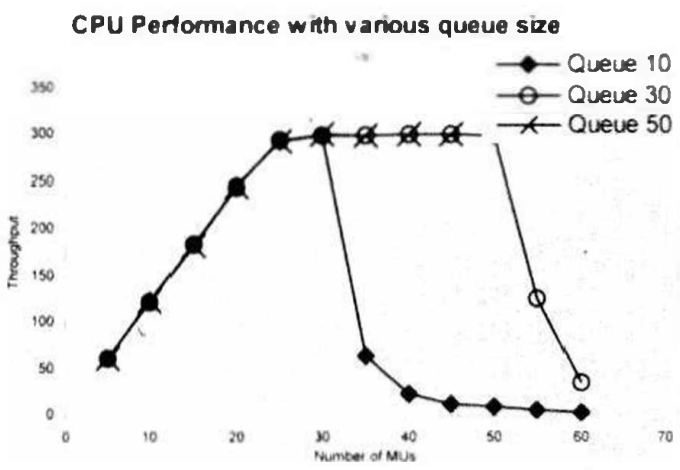

Figure 14: Influence of queue size at CO on system throughput

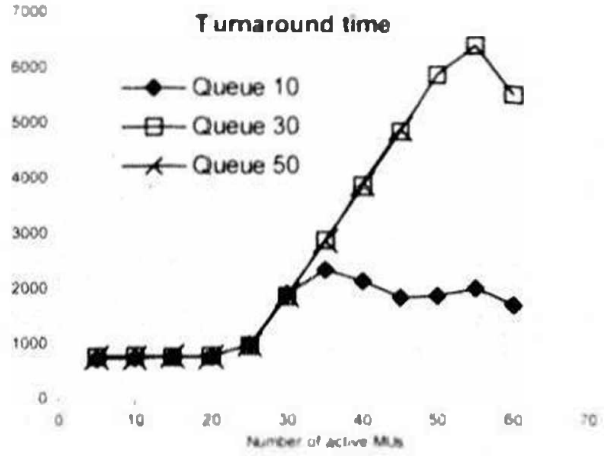

Figure 15: Average transaction turnaround time with different queue size (10,30,50)

5.4. Other experiments

A significant number of other experiments have been carried out include the investigations of various strategies to start an restart a transaction, different probabilistic distributions of random handoff and disconnection time, timeout value analyses and CPU speeds. Those results all confirm main conclusions presented in this paper, of which we summarize in next Section.

6. Conclusions and Further work

We have proposed a new commit protocol for mobile transaction models (CPM). From theoretical analysis and quantitative study using simulation, we found that CPM performs consistently better than the classical two-phase commit (2PC) protocol. The simulation approach has proved a great aid to the

understanding of protocol behaviors and correctness under various configurations. The main results from experiments presented in this paper are summarized below:

- Most of the time the CPM performs significantly better than the classical 2PC under mobile configurations. In addition, CPM allows offline processing and MU-enabled disconnection during the commit time.
- Handoff rate is not very critical in system performance
- Disconnections during transaction degrade the performance rapidly.
- Turnaround time tends to be proportional to size of the queue at coordinator. This illustrates that the queuing model can be applicable to this problem. It is inconclusive about this time in case of 2PC and CPM.
- The following extensions to this research are in consideration:
- Using Mobile Agent (MA) for transaction management in mobile distributed database systems
- The software will be re-designed as a framework for simulate commit protocols and to enable the proof of correctness using more powerful techniques such as Colored Petri nets [7].

References

1. P.A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency control and recovery in database systems*, Addison-Wesley, 1987.
2. P.K. Chrysanthis, Transaction processing in mobile computing environment, *Proc. IEEE Workshop Advances in Parallel and Distributed Systems*, Oct. 1993, pp. 77-82,
3. M.H. Dunham, A. Helal, S. Balakrishnan, A mobile transaction model that captures both the data and movement behaviour, *ACM/Baltzer J. Special Topics in Mobile Networks and Applications*, 1997, 149-162.
4. J. Gray, A comparison of the byzantine agreement problem and the transaction commit problem, *LNCS448*, Springer Verlag, 1987.
5. J. Gray, Notes on database operating systems, *Lecture Notes in Computer Science*, Vol. 60 Springer Verlag, 1978.
6. R. Gupta, J. Haritsa, K. Ramamritham, Revisiting commit processing in distributed database systems, *Proceedings of ACM SIGMOD International Conference on Management of Data*, May 13-15, 1997, Tucson, Arizona, USA.
7. L. M. Kristensen, S. Christensen, and K. Jensen, The practitioner guide to coloured petri nets, *Int. J. STTT* (1998) 2:98-132.
8. V. Kumar, N. Prabhu, M. H. Dunham, and A. Y. Seydim, TCOT - A Timeout-Based mobile transaction commitment protocol, *IEEE Transactions on Computers*, Vol. 51, No. 10, October 2002.
9. C. Mohan, B. Lindsay and R. Orbermarck, Transaction management in the distributed database management system. *ACM Transactions on Database Systems*, 1986.

10. A. Siberschatz, H. F. Korth and S. Sudarshan, *Database system concepts*, 3rd ed. Mc Graw-Hill, 1997.

11. J. Stamos and F. Cristian, A Low-Cost Atomic Commit Protocol, *Proc. of 9th synmp. on reliable sistributed systems*, Oct 1990.

12. G.D.Walborn, P.K.Chrysanthis, Transaction processing in PRO-MOTION, *proceedings of the 1999 ACM Symposium on applied computing, SAC '99*, San Antonio, TX, USA, 1999, pp. 389-398.

TẠP CHÍ KHOA HỌC ĐHQGHN, KHTN & CN, T.XIX, N$_0$4, 2003

ĐÁNH GIÁ GIAO THỨC HỢP THỨC CPM CHO CÁC HỆ CƠ SỞ DỮ LIỆU PHÂN TÁN DI ĐỘNG

Lê Hữu Lập

Học viện Công nghệ Bưu chính Viễn thông

Nguyễn Khắc Lịch, Trần Thế Truyền

Viện Khoa học Kỹ thuật Bưu điện

Hệ thống cơ sở dữ liệu phân tán di động (MĐBS – Mobile Distributed Database Systems) gần đây thu hút các nhà nghiên cứu với nhiều các mô hình giao tác khác nhau đã được đề xuất. Các giao thức hợp thức (Commit Protocol) cổ điển như 2PC (Two-Phase Commit) có khá nhiều điểm yếu trong môi trường này do tính chất của thông tin vô tuyến và sự di động của người dùng. Bài báo này đề xuất một giao thức hợp thức mới gọi là CPM (Commit Protocol for Mobile) có thể hoạt động tốt trên các mô hình giao tác khác nhau, trong đó cho phép xử lý ngoại tuyến và ngắt kết nối tạm thời trong khi đang thực hiện một giao dịch dữ liệu qua mạng di động.

Giao thức CPM có số lượng bản tin, số lần viết nhật ký vào bộ nhớ ổn định và chi phí truyền thông ít hơn 2CP, thích hợp với môi trường truyền thông kém tin cậy như di động. CPM cho phép động chủ động cao hơn từ phía di động thay vì lệ thuộc hoàn toàn vào bộ điều phối trung tâm như với 2PC và các giao thức khác.

Chúng tôi xây dựng một phần mềm mô phỏng mạng di động theo hướng sự kiện rời rạc (DES – Discrete Event Simulation) với giao thức hợp thức 2PC và CPM hoạt động trên đó. Các kết quả mô phỏng đã chứng tỏ được hiệu suất của CPM cao hơn 2PC trong môi trường phân tán di động.