Reducing the blocking in two-phase commit protocol employing backup sites

P.Krishna Reddy and Masaru Kitsuregawa
Institute of Industrial Science
The University of Tokyo
7-22-1, Roppongi, Minato-ku
Tokyo 106, Japan
{reddy, kitsure}@tkl.iis.u-tokyo.ac.jp

Abstract

In distributed data base systems (DDBSs), a transaction blocks during two-phase commit (2PC) processing if the coordinator site fails and at the same time some participant site has declared itself ready to commit the transaction. The blocking phenomena reduces availability of the system since the blocked transactions keep all the resources until they receive the final command from the coordinator after its recovery. To remove the blocking problem in 2PC protocol, three phase commit (3PC) protocol was proposed. Although 3PC protocol eliminates the blocking problem, it involves an extra round of message transmission, which further degrades the performance of DDBSs. In this paper, we propose a backup commit (BC) protocol by including backup phase to 2PC protocol. In this, one backup site is attached to each coordinator site. After receiving responses from all participants in the first phase, the coordinator communicates its decision only to its backup site in the backup phase. Afterwards, it sends final decision to participants. When blocking occurs due to the failure of the coordinator site, the participant sites consult coordinator's backup site and follow termination protocols. In this way, BC protocol achieves non-blocking property in most of the coordinator site failures. However, in the worst case, the blocking can occur in BC protocol when both the coordinator and its backup site fail simultaneously. If such a rare case occurs, the participants wait until the recovery of either the coordinator site or the backup site. BC protocol suits best for DDBS environments in which sites fail frequently and messages take longer delivery time. Through simulation experiments it has been shown that BC protocol exhibits superior throughput and response time performance over 3PC protocol and performs closely with 2PC protocol.

Keywords atomicity, reliability, recovery, distributed databases, distributed algorithms, non-blocking protocols.

1. Introduction

In distributed database systems (DDBSs), a transaction blocks during two-phase commit (2PC) processing [8] if the coordinator site fails and at the same time some participant site has declared itself ready to commit the transaction. In this situation, to terminate the blocked transaction, the participant site must wait for the recovery of the coordinator. The blocked transactions keep all the resources until they receive the final command from the coordinator after its recovery. Thus, blocking phenomena reduces the availability of the system. To eliminate this inconvenience, three-phase commit (3PC) protocol [14, 15] was proposed. However, 3PC protocol involves an additional round of message transmission to achieve non-blocking property. If 3PC protocol is employed to eliminate the blocking problem, an extra round of message transmission further reduces the system's performance as compared to 2PC protocol. Especially in DDBS environments, in which frequent site failures and longer message transmission times occur, neither 2PC protocol with blocking problem nor 3PC protocol with performance degradation problem is suitable for the commit processing. In this paper, we propose backup commit (BC) protocol by including backup phase to 2PC protocol. In this, one backup site is attached to each coordinator. After getting the responses from all participants in the first phase, the coordinator communicates the final decision only to backup site in the backup phase. Afterwards, it sends the final decision to participants. When the blocking occurs due to a failure of coordinator site, the participant sites consult coordinator's backup site and then follow termination protocols. Having small amount of overhead (the time required to communicate with the backup site) over 2PC protocol, BC protocol resolves the blocking in most of the coordinator's failures. As compared to 3PC protocol, it reduces both the number of messages and the latency that occurs during second phase, thus exhibits superior performance. However, in the worst case, blocking still occurs in BC protocol if both coordinator

and its backup site fail simultaneously. If such a rare case happens, the participants wait until the recovery of either coordinator or its backup site. The BC protocol suits best for DDBS environments in which the longer transmission delays and frequent site failures occur.

Recently commit processing has attracted strong attention due to its effect on the performance of the transaction processing. In [11], using simulation model, it has been shown that distributed commit processing can have more influence than distributed data processing on the throughput performance. It has been shown in [13] that the time to commit accounts for one third of transaction duration in a general purpose database. In [3], experimental studies have been reported on the behavior of concurrency control and commit algorithms in the wide area network environments. It has been shown that the time to commit can be as high as 80 percent of the transaction time in the wide area network (Internet) environments. In [11, 3] it has been reported that as compared to 2PC protocol, the performance is further degraded with 3PC protocol due to an extra round of message transmission. To reduce the extent of blocking, quorum-based 3PC protocol [15] was proposed that maintains the consistency in spite of network partitions. In [9], enhanced 3PC protocol is proposed which is more resilient to network partitioning failures than quorum based 3PC protocol. In order to deal with the failure of the coordinator, backup processes are used in SDD-1[7]. These processes are initiated by the coordinator before initiating the commit protocol and substitute the coordinator in case of its failure. In order to ensure that only one process will substitute for the coordinator, backups are linearly ordered, so that the first one ''looks'' at the coordinator, the second ''looks'' at the first one, and so on. If one backup fails, say k, backup k+1 starts looking at backup k-1. ''Looking'' in this means periodically sending control messages. In this, the commit protocol with backups consists of four phases. In first phase, the coordinator establishes n linearly ordered backups and each backup is informed of participants identity. In second phase, the coordinator sends the updates to participants. In third phase, the coordinator communicates its decision to backups. And in fourth phase the coordinator sends its decision to participants.

In this paper we use the notion of backup site similar to the notion of backup process in [7]. However, in BC protocol, in case of coordinator's failure the backup site does not assume the role of coordinator. Also, there is no periodic exchange of control messages between the coordinator and corresponding backup site. Instead, the sites themselves resolve the blocking by contacting the backup site. Both coordinator and corresponding backup site are failure independent. The work is motivated by the fact that 2PC protocol is widely applied protocol in commercial database systems. Even though 3PC protocol eliminates blocking, it has not entered into commercial database systems. In this situation, we have made an effort to resolve the blocking problem of 2PC protocol by employing extra hardware. In addition, the proposed protocol can easily be integrated with existing 2PC protocol implementations.

The paper is organized as follows. In section 2, we explain the system model. In section 3, we briefly explain both 2PC and 3PC protocols. In section 4, we propose BC protocol. In section 5, we discuss the performance issues. In section 6, we present the results of simulation experiments. The last section finally consists of summary and conclusions.

2. System model

The DDBS consists of a set of data objects. A data object is the smallest accessible unit of data. Each data object is stored at one site only. Transactions are represented by T_i, T_j,....; and sites are represented by S_i, S_j,. . .; where, i, j . . . are integer values. The data objects are stored at database sites connected by a computer network. Each database site S_i has the backup site and is represented by BS_i. The originating site of T_i acts as a coordinator in T_i's commit processing. The sites which are involved in the processing of T_i, are called participants of T_i. The coordinator site of T_i acts also as a participant site.

Each site works as both transaction manager and a data manager. The transaction manager supervises the processing of transactions, while the data managers manage individual databases. We assume that the network is prone to both link and site failures. When a site fails, it simply stops running and other sites detect this fact. Also, we assume that the network partitioning can occur. That is, in the event of network partitioning failure, the operational sites are divided into two or more groups where every two sites with in a group can communicate with each other, but the sites in different groups can not. The communication medium is assumed to provide the facility of message transfer between sites. When a site has to send a message to some other site, it hands over the message to the communication medium, which delivers it to the destination site in finite time. We assume that, for any pair of sites S_i and S_j, the communication medium always delivers the messages to S_j in the same order in which they were handed to the medium by S_i.

3. Distributed commit protocols : 2PC and quorum based 3PC

In the literature, a variety of commit protocols have been proposed, most of which are based on 2PC protocol. The most popular variants of 2PC protocol are presumed abort and presumed commit protocols [10]. The other protocols include early prepare [13], coordinator log [17], and implicit yes vote [2] protocols. Also, different communication paradigms can be used to implement the 2PC protocol. The one described below is called the centralized 2PC, since the communication is between the coordinator and the participants only. In this process, the participants do not communicate among themselves. In this paper, we do not discuss other protocols since these are not concerned with the blocking problem.

The detailed explanation of termination and recovery protocols for both 2PC and 3PC protocols against failures such as coordinator timeouts, participant timeouts, and participant failures can be found in references [18, 5, 4]. In this section, we briefly explain 2PC and quorum based 3PC protocols.

3.1. Two-phase commit protocol

In DDBSs, 2PC protocol extends the affects of local atomic commit actions to distributed transactions by insisting that all sites involved in the execution of a distributed transaction agree to commit the transaction before its effects are made permanent. A brief description of the 2PC protocol that does not consider failures is as follows. Initially, the coordinator (originating site of a transaction) writes a begin-commit record in the log, sends a $PREPARE$ message to all participating sites, and enters the **wait** state. When a participant receives $PREPARE$ message, it checks if it can commit the transaction. If so, the participant writes a ready record in the log, sends a $VOTE\_COMMIT$ message to the coordinator, and enters the **ready** state. Otherwise, the participant writes an abort record and sends a $VOTE\_ABORT$ message to the coordinator. If the decision of the site is to abort, it can forget about that transaction. The coordinator aborts the transaction globally, even it receives $VOTE\_ABORT$ message from one participant. Then, it writes an abort record, sends a $GLOBAL\_ABORT$ message to all participant sites, and enters the **abort** state; Otherwise, it writes a commit record, sends a $GLOBAL\_COMMIT$ message to all participants, and enters the **commit** state. The participants either commit or abort the transaction according to the coordinators' instructions and sends back ACK (acknowledgment) message at which point the coordinator terminates the transaction by writing an end-of-transaction record in the log.

Blocking problem

In 2PC protocol, consider a situation that a participant has sent $VOTE\_COMMIT$ message to the coordinator and has not received either $GLOBAL\_COMMIT$ or $GLOBAL\_ABORT$ message due to the coordinator's failure. In this case, all such participants are blocked until the recovery of the coordinator to get the termination decision.

Partitioning failure

Consider that a simple partition occurs, dividing the sites into two groups; the group which contains the coordinator is called coordinator group; the other is participant group. In 2PC protocol, for the coordinator this case is equivalent to the participants' failure. After the time-out period, the coordinator terminates the transactions by following termination protocols. However, for the participants in the participant group it is equivalent to the coordinator's failure. So, they wait until the partition is repaired to know the termination decision. Thus, 2PC protocol terminates the transactions consistently in case of partitioning failure.

3.2. Quorum based 3PC protocol

The blocking problem is eliminated in 3PC protocol. The brief description of quorum based 3PC protocol is as follows.

Every site in the system is assigned a vote V_i. Let us assume that the total number of votes in the system is V, and abort and commit quorum are V_a and V_c, respectively. The following rules must be obeyed by the protocol.

1. $V_a + V_c > V$, where $V_a > 0, V_c > 0$.
2. Before a transaction commits, it must obtain a Commit quorum V_c.
3. Before a transaction aborts, it must obtain an Abort quorum V_a.

The abort case is similar to 2PC protocol; the coordinator aborts the transaction globally, even if the coordinator receives $VOTE\_ABORT$ message from one participant. However, the commit case is different. If the coordinator receives all $VOTE\_COMMIT$ messages, it writes a $prepare\_to\_commit$ record, sends $PREPARE\_TO\_COMMIT$ message to all the participants, and enters a new **pre_commit** state. On receiving this message, the participant writes a $prepare\_to\_commit$ record, sends $READY\_TO\_COMMIT$ message, and enters **pre_commit** state. Finally, when the coordinator receives $READY\_TO\_COMMIT$ messages, if the sum of the votes of responding sites equals to or exceeds V_c, after

writing a commit record, it sends $GLOBAL\_COMMIT$ message to all participants, and enters the **commit** state.

Elimination of blocking

We briefly explain how 3PC protocol eliminates blocking problem by dividing the situation into two cases. First, a participant has sent the $VOTE\_COMMIT$ message but has not received $PREPARE\_TO\_COMMIT$ message due to the coordinator's failure. Second, a participant has received $PREPARE\_TO\_COMMIT$ message but has not received $GLOBAL\_COMMIT$ message due to the coordinator's failure.

In both cases, the operational participants elect a new coordinator. The new coordinator collects the states from all the sites, and tries to resolve the transaction. If any site has previously committed or aborted, the transaction is immediately committed or aborted accordingly. Otherwise, the coordinator tries to establish a quorum. The coordinator commits the transaction if at least one site is in the pre_commit state and the group of sites in the **wait** state together with the sites in the pre_commit state form a Commit quorum. The coordinator aborts the transaction if the group of sites in the **wait** state together with the sites in the pre_abort state form an Abort quorum.

Network partitioning

Quorum based 3PC protocol is resilient to partitioning failure. When a network partitioning occurs, each partition elects a new coordinator assuming that other sites are down. In order to ensure that the same decision is reached by all coordinators, a coordinator must explicitly establish a Commit quorum (V_c) to commit, or an Abort quorum (V_a) to abort. Otherwise, they wait until the merger of partitions. In this way consistency is achieved.

4. Backup commit protocol

In this section, we first propose BC protocol. For BC protocol, same termination and recovery protocols of 2PC protocol against different types of failures (coordinator timeouts, participant timeouts, and participant failures) can be employed. However, the blocking case is different. In next two subsections, we explain the termination and recovery protocols in case of blocking. Subsequently, we discuss the behavior of BC protocol in case of partitioning failures.

4.1. Protocol

Suppose there are n sites in DDBS. In this approach, for each site S_i, the backup site BS_i is attached. Both should be failure independent. Let $backup\_set$ be a set of identity of all backup sites. Backup_set is stored at each S_i. The first and third phases of BC protocol are similar to first and second phases of 2PC protocol, respectively. The BC protocol is as follows.

- **First phase**

  Coordinator : Initially, the coordinator writes a begin-commit record in the log, sends $PREPARE$ messages to all participating sites, and enters the **wait** state.

  Participant : When a participant receives $PREPARE$ message, it checks if it can commit the transaction. If so, the participant writes a ready record in the log, sends $VOTE\_COMMIT$ message to the coordinator, and enters **ready** state. Otherwise, the participant writes an abort record and sends a $VOTE\_ABORT$ message to the coordinator.

- **Second phase**

  Coordinator : If the coordinator (S_i) receives $VOTE\_COMMIT$ messages from all participants, after writing $decided\_to\_commit$ record on its stable storage, sends $DECIDED\_TO\_COMMIT$ message to corresponding backup site (BS_i). Otherwise, even if it receives $VOTE\_ABORT$ message from one participant, it writes an abort record on the stable storage and sends $GLOBAL\_ABORT$ message to all participants.

  Backup site :
  On receiving $DECIDED\_TO\_COMMIT$ message, BS_i writes $decided\_to\_commit$ record on the stable storage, sends back $RECORDED\_COMMIT$ message to S_i.

- **Third phase**

  Coordinator : On receiving $RECORDED\_COMMIT$ message from the backup site, the coordinator writes commit record on the stable storage and sends $GLOBAL\_COMMIT$ message to all participants.

  Participant : The participant follows the coordinator's instructions, and sends back acknowledgment message to the coordinator.

Coordinator : After receiving acknowledgment messages from all participants, the coordinator writes $end\_of\_transaction$ record on the stable storage.

4.2. Termination protocol

If blocking occurs due to the failure of the coordinator site (S_i), the blocked participant checks the identity of BS_i in its $backup\_set$.

1. If $BS_i \in backup\_set$, the participant contacts BS_i. If no information exists at BS_i about the transaction, the participant aborts the transaction. The coordinator will abort the transaction at restart. Otherwise, if the participant finds $decided\_to\_commit$ record at BS_i, it commits the transaction. The coordinator will commit the transaction at restart. Otherwise, if the participant site is unable to contact BS_i due to its failure, it waits until the recovery of either the coordinator or the backup site.
2. If $BS_i \notin backup\_set$, the participant waits until the recovery of the coordinator.

4.3. Recovery protocol

When coordinator recovers from failure, it may be in one of the following states.

I The coordinator finds $begin\_commit$ **record but no** $decided\_to\_commit$ **record**
In this case, it can safely abort the transaction without contacting participants. Because, the participants either might have aborted the transaction by contacting the backup site or are in the **ready** state.

II The coordinator finds $decided\_to\_commit$ **record but no commit record**
In this case, there exist three possibilities. First, the backup site might have failed before receiving $DECIDED\_TO\_COMMIT$ message from the coordinator. Second, the coordinator might have failed after writing $decided\_to\_commit$ record but before sending $DECIDED\_TO\_COMMIT$ message to backup site. And third, the coordinator might have failed after the backup site has received $DECIDED\_TO\_COMMIT$ message.

After recovery, the coordinator contacts the backup site. If no information exists about the transaction, the coordinator sends $GLOBAL\_ABORT$ messages to all the participants. At most a participant either might have aborted the transaction by contacting the backup site or is in the **ready** state. Otherwise, if the coordinator finds $decided\_to\_commit$ record at the backup site, it sends $GLOBAL\_COMMIT$ messages to all the participants. At most, a participant either might have committed the transaction by contacting backup site or is in the **ready** state.

After recovery, if the coordinator is unable to contact the backup site, there exist two options. First, it waits for the recovery of the backup site and follow the above recovery protocol. Second, it will ask all the participants to report the transaction's state. In this case, we assume that the participant distinguishes the recovery messages from normal messages. That is, after responding to recovery messages of coordinator, the participant will not contact the backup site in future about corresponding transaction. Thus, even though the coordinator fails during recovery, the re-execution of the recovery protocol makes no difference.

1. If at least one participant has aborted the transaction, the coordinator aborts the transaction.
2. If at least one participant has committed the transaction, the coordinator commits the transaction.
3. If all the participants are in **ready** state, the coordinator aborts the transaction. [1]

III The coordinator finds commit record but no $end\_of\_transaction$ **record**
In this case, it can safely re-send $GLOBAL\_COMMIT$ message to all participants. Because, at most, the participants either might have committed the transaction by contacting the backup site or are in the **ready** state.

4.4. Backup site failure and network partitioning

As an option, to minimize the latency we choose the nearest site to the coordinator as corresponding backup site. For example, both coordinator site and backup site are connected to same local area network (Ethernet). When partition failure occurs in wide area network, both the coordinator and corresponding backup site fall in same partitioning group. As a result, the participants of other groups wait until the connection is repaired. In this way, BC protocol ensures consistent termination of transactions in case of partitioning failures.

Now, we explain termination protocols by considering a generalized case where the coordinator and the backup site

[1] Since the backup site is down, we conservatively abort the transaction

are separated due to partition failure.

Consider a situation that the partition has occurred after sending the $DECIDED\_TO\_COMMIT$ message to backup site. As a result, the coordinator does not receive the reply to $DECIDED\_TO\_COMMIT$ message from its backup site. In this case, there exist two possibilities: either the backup site may be down or the network has partitioned such that the backup site and the coordinator fall in different groups. In this case, if coordinator unilaterally either aborts or commits the transaction, inconsistency may occur. Because, if partition occurs after receiving $DECIDED\_TO\_COMMIT$ message by backup site, the participants in the backup site group commit the transaction by contacting the backup site. Otherwise, if the partition occurs before receiving the $DECIDED\_TO\_COMMIT$ message by backup site, participants in the backup site group abort the transaction by contacting backup site. Therefore, when the coordinator fails to get response to $DECIDED\_TO\_COMMIT$ message from backup site, it asks the participants and backup site to report the transaction's state. (In this case also, we assume that the participant distinguishes the recovery messages from normal messages. That is, after responding to recovery messages of the coordinator, the participant will not contact the backup site in future. Thus, even though the coordinator fails during recovery, the re-execution of the recovery protocol makes no difference.) Based on the responses received, the coordinator proceeds as follows.

1. If the coordinator receives response from the backup site, it can terminate the transaction as follows. If no information exist about the transaction at the backup site, the coordinator aborts the transaction by sending $GOBAL\_ABORT$ messages to all participants. Otherwise, if $decided\_to\_commit$ record exist at the backup site, the coordinator commits the transaction by sending $GLOBAL\_COMMIT$ messages to all participants.

2. If at least one participant has aborted the transaction, the coordinator aborts the transaction.

3. If at least one participant has committed the transaction, the coordinator commits the transaction.

4. If all the participants are in **ready** state, the coordinator aborts the transaction.

After terminating the transaction, if the backup site is down, the coordinator can follow one of the following options.

i The coordinator suspends the processing of transactions until the backup site is up.

ii The coordinator selects another backup site and updates the $backup\_set$. Next, it sends the new $backup\_set$ information to all the backup sites. On receiving this information, the site updates its $backup\_set$. Next, the coordinator follows BC protocol for the commit processing.

iii If the coordinator is unable to select a new backup site, it deletes the identity of its backup site information from the $backup\_set$ and sends this information to all the sites. On receiving this information, the site updates its $backup\_set$. Next, the coordinator can follow 2PC protocol at the risk of blocking.

5. Performance discussion

In this section, we first analyze the non-blocking behavior of BC protocol. Next, we discuss overheads and benefits of BC protocol.

5.1. Failure probability of backup site while coordinator is down

Reliability of a module is statistically quantified as mean-time-to-failure (MTTF). The service interruption of a module is statistically quantified as mean-time-to-repair (MTTR). The module availability is statistically quantified as

$$\frac{MTTF}{MTTF + MTTR}$$

.

Let $MTTF_c$ and $MTTR_c$ represent MTTF and MTTR of the coordinator site respectively. Also, $MTTF_b$ represents MTTF of corresponding backup site. Since the backup site and the coordinator are failure independent, the probability that backup site fails when the corresponding coordinator is down is calculated as below.

The probability that the coordinator site is unavailable is:

$$P_c = \frac{MTTR_c}{MTTF_c + MTTR_c}$$

$$\simeq \frac{MTTR_c}{MTTF_c} \ since \ MTTR_c \ll MTTF_c$$

The probability that the backup site fails is:

$$P_b = \frac{1}{MTTF_b}$$

The probability that backup site fails and the corresponding coordinator is down is:

$$P_b \times P_c = \frac{1}{MTTF_b} \times \frac{MTTR_c}{MTTF_c} = \frac{MTTR_c}{MTTF_b \times MTTF_c}$$

From above equation, it can be observed that the probability that backup site fails while corresponding coordinator is down is reduced significantly. Thus, in case of coordinator site failure, with the introduction of the backup site, blocking probability is considerably reduced as compared to 2PC protocol. Further, it can be observed that the purpose of the backup site is to terminate the blocked transactions at the participant sites when the corresponding coordinator is down. After the termination of the blocked transactions, even though the backup site fails, it does not effect the consistency of the database. Let $term\_time$ be the time duration required to terminate the blocked transactions by contacting the backup site when the coordinator is down. The above equation denotes the probability that the backup site fails during the entire period ($MTTR_c$) when the coordinator is down. However, in the worst case the blocked transactions are consistently terminated even if the backup site is up only during $term\_time$ and then fails. As $term\_time$ (few minutes) is much less than the down time (few hours) of the coordinator, the probability that the backup site fails during the $term\_time$ while the coordinator is down is further reduced.

5.2. Message overhead, latency and failures

As compared to 2PC protocol, to commit a transaction, BC protocol requires extra messages and time duration (to communicate with the backup site). However, as compared to 3PC protocol, independent of number of participants, BC protocol requires only two messages and fixed time duration during the second phase. In BC protocol the latency during the second phase is considerably reduced as compared to 3PC protocol. Also, by making the nearby site to the coordinator as the backup site, the latency can be minimized. This brings the performance of BC protocol close to 2PC protocol by achieving non-blocking property in most of the coordinator failures.

Also, in BC protocol, the overhead during the recovery is considerably reduced. Because, after recovery, the coordinator terminates the transactions consistently by only contacting the backup site. However, in a rare case, if it is unsuccessful in contacting the backup site, it has to demand the state information from all the participants. Also, in case of partition failures, BC protocol terminates the transactions consistently with blocking problem, similar to 2PC and 3PC protocols.

6. Simulation experiments

We have carried out the simulation experiments to compare the performance of BC protocol with 2PC and 3PC protocols.

Parameter	**Meaning**	**Value**
db_size	Number of objects in the database	1000 objects
num_sites	Number of sites in the system	5 sites
trans_size	Mean_size of transaction	8 objects
max_size	Size of largest transaction	12 objects
min_size	Size of smallest transaction	4 objects
write_prob	Pr (write X/read X)	1
local_to_total	local requests / total requests	0.6
res_cpu	CPU time	15 msec
res_io	I/O time	35 msec
MPL	Multiprogramming level	Simulation variable
trans_time	Transmission time between two sites	Simulation variable
backup_trans _time	Trans_time to contact backup site	0

Table 1. Model parameters with settings

The meaning of each model parameter for simulation is given in Table 1. The size of the database is assumed to be db_size data objects. The database is uniformly distributed across the all num_sites sites. The new transaction is assigned the arrival site which is chosen randomly over num_sites. The parameter $trans\_size$ is the average number of data objects requested by the transaction. It is computed as the mean of a uniform distribution between max_size and min_size (inclusive). The probability that an object read by a transaction will also be written is determined by the parameter $write\_prob$. The parameter $trans\_time$ is the time required to transmit a message between sites. The parameter $backup\_trans\_time$ is the time required to transmit a message between the coordinator and the corresponding backup site. The parameter $local\_to\_total$ is the ratio of the number of local requests to the number of total requests for a transaction. The parameter res_io is the amount of time taken to carry out i/o request and the parameter res_cpu is the amount of time taken to carry out CPU request. Accessing a data object requires res_cpu and res_io. Also, each message of 2PC protocol requires res_cpu and res_io for processing. The total number of

concurrent transactions active in the system at any time is specified by the multiprogramming level (MPL).

The communication network is simply modeled as a fully-connected network. Any site can send messages to all the sites at the same time. The wide area network behavior is realized by varying $trans\_time$. This is true because, in DDBSs, even though the difference in the delay to receive responses for remote requests varies considerably, the transaction does not complete its execution, unless it receives responses from all the remote sites. We employ static distributed two-phase locking algorithm for concurrency control.

The setting of res_io, res_cpu, db_size, $trans\_size$, min_size and max_size values are given in Table 1 which are adopted in [1]. The parameter $local\_to\_total$ ratio for a transaction is fixed at 0.6 [6]. Thus, 60 percent of the data objects are randomly chosen from the local database and the 40 percent of the data objects are randomly chosen from the remaining database sites. A transaction writes all the data objects it reads ($write\_prob$ is set to 1). With these settings, by varying MPL values, sufficient variation in the data contention is realized.

In the simulation experiments, we have evaluated the two performance metrics : throughput and response time. The throughput metric is evaluated as the number of transactions completed per second by the system. The response time is the time spent by the transaction in the system. This is the difference between when the transaction was first submitted and when the transaction decides to commit. In all simulation experiments we have set $backup\_trans\_time$ to 0. That is, we consider that both the coordinator and its backup site are connected to the high speed local area network.

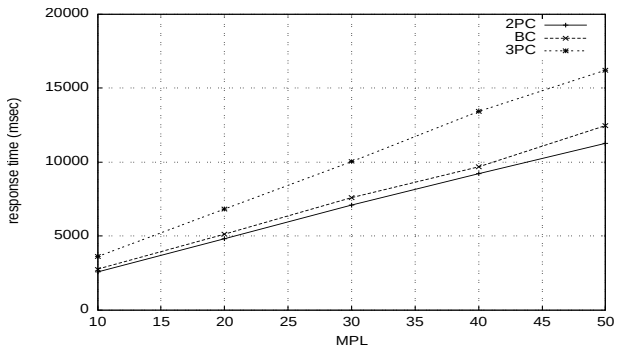

Figure 1. MPL versus response time at trans_time=0

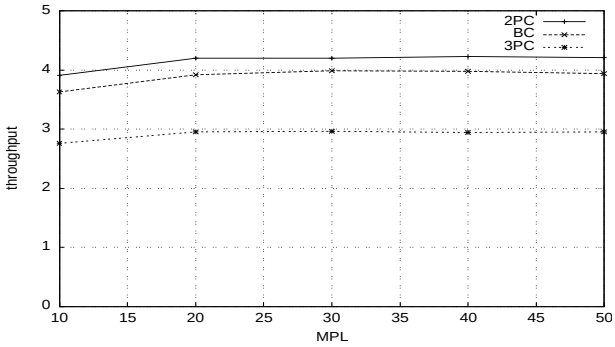

Figure 2. MPL versus throughput at trans_time=0

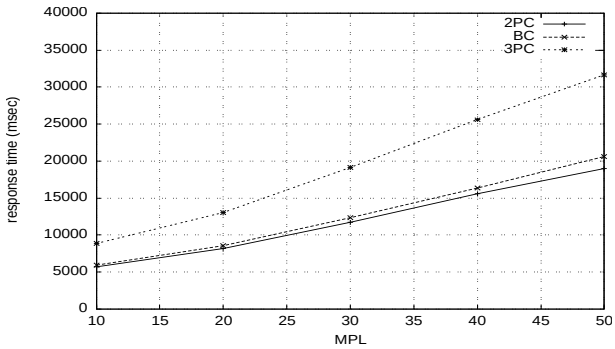

Figure 3. MPL versus response time at trans_time=1000

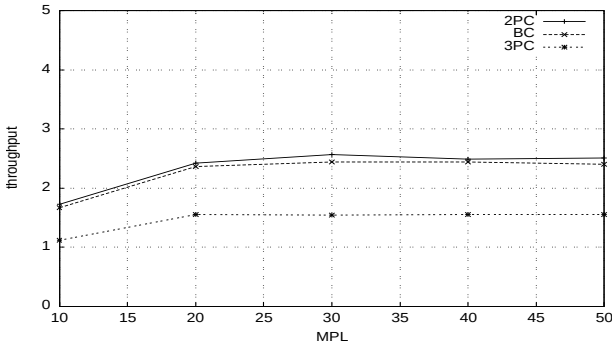

Figure 4. MPL versus throughput at trans_time=1000

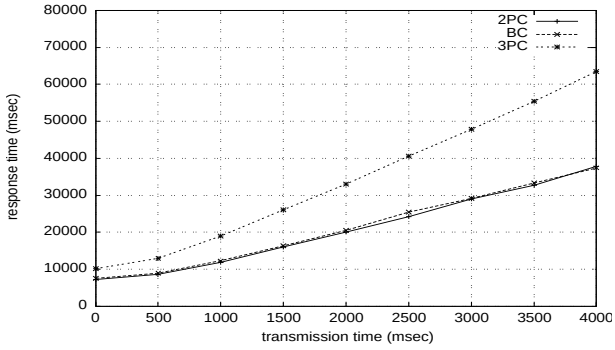

Figure 5. Transmission time versus response time at MPL=30

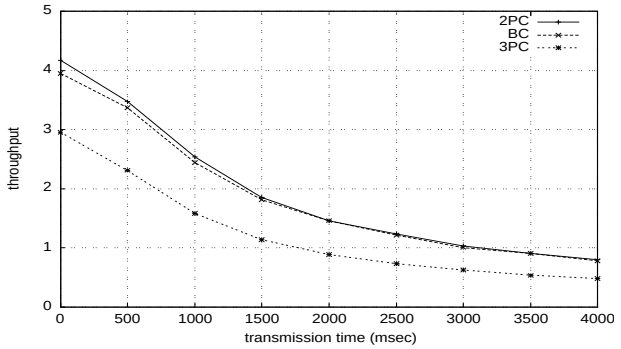

Figure 6. Transmission time versus throughput at MPL=30

At different MPL values, Figure 1 shows the response time results and Figure 2 shows the throughput results by setting transmission time (trans_time) to 0. These graphs show the behavior of BC protocol in the local area network based environments. In DDBS, the increase in MPL results in higher resource contention. As a result, more number of transactions wait for resources. So, the response time of three protocols increases with MPL. From Figure 1, it can be observed that, the response time performance using BC protocol is little higher than 2PC protocol. Also, in Figure 2, the throughput is little less than 2PC protocol. This indicates the overhead of BC protocol over 2PC protocol in local area network based environments.

At different MPL values, Figure 3 shows the response time results and Figure 4 shows the throughput results by setting transmission time to 1000 msec. Here, wide area network environment is assumed. It can be observed that, with the increase in transmission time, the effect of communication with the backup site is nullified. As a result, both response time and throughput curves of BC protocol are close to 2PC protocol.

At different transmission time values, Figure 5 shows the response time results and Figure 6 shows the throughput results by setting MPL to 30. As the transmission time increases, the transaction spends longer duration in the commit processing. Consequently, more number of transactions wait for the data objects for longer duration. As a result response time is increases and throughput decreases. It can be observed that due to longer transmission times, the overhead of the BC protocol over 2PC protocol is nullified. As a result, both response time and throughput curves of BC and 2PC protocols coincide.

7. Summary and conclusions

In this paper we have proposed BC protocol for the commit processing in DDBSs that exhibits non-blocking behavior in most of the coordinator failures. In this protocol, one backup site is attached with each operational site. The proposed protocol differs with 3PC protocol in the second phase.After receiving responses from all participants in the first phase, the coordinator communicates its decision only to its backup site in the backup phase. In case of blocking, the participants consult the backup site and follow termination protocols. This protocol incurs little overhead (messages and time required to write to the backup site) over 2PC protocol. By selecting nearby site to the coordinator as the backup site, overhead can be nullified. This brings the performance of BC protocol close to 2PC protocol. Also, BC protocol preserves the consistency of the database in case of partitioning failures. We have analytically shown that the probability that both the coordinator and the corresponding backup site are down at the same time is reduced significantly. Also, the simulation results show that the performance of BC protocol is very close to 2PC protocol. The proposed BC protocol has following merits. First, it eliminates the blocking of transactions in most of the coordinator failures. Second, it ensures consistency of the database in case of partitioning failures. And third, the performance of BC protocol is close to 2PC protocol. With these merits, BC protocol becomes a good choice for commit processing in DDBS environments where frequent site failures occur and messages take longer delivery time.

Acknowledgments
This work is partially supported by Grant-in-Aid for Creative Basic Research # 09NP1401: ‘‘Research on Multimedia Mediation Mechanism for Realization of Human-oriented Information Environments’’ by the Ministry of Education, Science, Sports and Culture, Japan and Japan Society for the Promotion of Science, Japan.

References

[1] R.Agrawal, M.J.Carey and M.Livny, ‘‘Concurrency control performance modeling: alternatives and implications’’, *ACM Transactions on Database Systems*, vol.12, no.4, December 1987, pp. 609-654.

[2] Y.Al-Houmaily and P.Chrysanthis, ‘‘Two-phase commit in giga-bit networked distributed databases’’, *proceedings of 8th International Conference on Parallel and Distributed Computing Systems*, September 1995.

[3] B.Bhargava, Y.Zhang, S. Goel, ‘‘A study of distributed transaction processing in an internetwork’’, *Volume 1006 of Lecture Notes in Computer Science*, Springer-Verlag, 1995, pp. 135-152.

[4] P.A.Bernstein, V.Hadzilacos and N.Goodman, *Concurrency control and recovery in database systems*, Addison-Wesley, 1987.

[5] S.Ceri and P.Pelagatti, *Distributed databases: principles and systems*, New York:McGraw-Hill, 1984.

[6] Alok N.Choudhary, Cost of distributed deadlock detection:A performance study, *in 1990 proc. of IEEE Conference on Data Engineering*, 1990, pp.174-181.

[7] Michael Hammer and David Shipman, ''Reliability mechanisms for SDD-1: A system for distributed databases'', *ACM Transactions on Database Systems*, vol.5, no.4, December 1980, pp.431-466.

[8] J.N.Gray, ''Notes on database operating systems: in operating systems an advanced course'', *Volume 60 of Lecture Notes in Computer Science*, 1978, pp. 393-481.

[9] Idit Keider and Danny Dolev, ''Increasing the resilience of atomic commit, at no additional cost'', *in proc. of the Fourteenth ACM SIGACT-SIGMOD-SIGAR Symposium on Principles of Database Systems*, 1995, pp.245-254.

[10] C.Mohan, B.Lindsay and R.Obermark, ''Transaction management in the R^* distributed database management system'', *ACM Transactions on Database Systems*, vol.11, no.4, 1994.

[11] Ramesh Gupta, Jayant Haritsa and Kirti Ramamritam, ''Revisiting commit processing in distributed database systems'', *ACM SIGMOD*, 1997, pp. 486-497.

[12] G.Samaras, K.Britton, A.Citron, and C.Mohan, ''Two-phase commit optimizations in a commercial distributed environment'', *Journal of Distributed and Parallel Databases*, vol.3, no.4, 1995.

[13] P.Spiro, A.Joshi, and T.K.Rangarajan, ''Designing an optimized transaction commit protocol'', *Digital Technical Journal*, 1991.

[14] D.Skeen, ''Nonblocking commit protocols'', *ACM SIGMOD*, June 1981.

[15] D.Skeen, ''A quorum-based commit protocol'', *in proc. of 6th Berkeley Workshop on Distributed Data Management and Computer Networks*, February 1982, pp. 69-80.

[16] D.Skeen and M.Stonebraker, ''A formal model of crash recovery in a distributed system'', *IEEE Transactions on Software Engineering*, vol.SE-9, no.3, 1983, pp.219-227.

[17] J.Stamos and F.Cristian, ''Coordinator log transaction execution protocol'', *Journal of Distributed and Parallel Databases*, 1993, pp.383-408.

[18] M.Tamer Ozsu and Patrick Valduriez, *Principles of distributed database systems*, Prentice-Hall, 1991.