Reducing the blocking in two-phase commit with backup sites

P.Krishna Reddy*and Masaru Kitsuregawa
Institute of Industrial Science, The University of Tokyo
4-6-1, Komaba, Meguro-ku, Tokyo- 1538505, Japan

August 21, 2002

Abstract

The blocking phenomena in two-phase commit (2PC) reduces the availability of the system as the blocked transactions keep all the resources until the recovery of the coordinator. The three-phase commit (3PC) protocol involves an extra round of message transmission to resolve the blocking problem. In this paper, we propose a backup commit (BC) protocol to reduce the blocking problem by attaching multiple backup sites to the coordinator site. In BC, after receiving responses from the participants, the coordinator quickly communicates the final decision to the backup sites, before it sends the final decision to the participants. When blocking occurs, the participant sites can terminate the transaction by consulting a backup site of the coordinator. The BC protocol resolves the blocking in most of the coordinator site failures without involving an expensive communication cycle as in 3PC. The simulation experiments indicate that the throughput performance of BC is close to 2PC.

Keywords: Distributed database, Commit protocol, Two-phase commit, Three-phase commit, Non-blocking protocols, Distributed computing.

1 Introduction

The two-phase commit (2PC) [1] protocol (or its variation) is widely employed for commit processing in distributed data base systems (DDBSs). In DDBSs, a blocking phenomena occurs during 2PC if the coordinator site fails and at the same time some participant site has declared itself ready to commit the transaction. In this situation, the participants must wait for the recovery of the coordinator to terminate the blocked transactions. The blocked transactions keep all the resources until the participants receive the final command from the coordinator only after its recovery. Thus, the blocking phenomena reduces the availability of the system. The three-phase commit (3PC) protocol [2] was proposed to remove the blocking problem. However, 3PC requires an extra round of message transmission over 2PC which reduces the performance.

In this paper, we propose a backup commit (BC) protocol to reduce the blocking problem by attaching multiple backup sites to the coordinator site. It is assumed that, at least one backup site is very close to the coordinator so that the network latency between them is negligible as compared to that in a WAN environment. Both the coordinator and corresponding backup sites are failure independent. In BC, after receiving responses from the participants, the coordinator quickly communicates the final decision to the backup sites before it sends the final decision to the participants. When blocking occurs, the participant sites consult the backup sites of the coordinator to resolve the blocking. Having a little overhead over 2PC, the BC protocol resolves blocking in most of the coordinator's failures. As compared to 3PC, it reduces both the number of messages and the latency that occur during the second phase, thus exhibiting a superior throughput performance. However, in the worst case, blocking still occurs with BC if both the coordinator and all of its backup sites fail simultaneously. If such a rare case happens, the participants wait until the recovery of

*With International Institute of Information Technology (IIIT), Hyderabad (India), from April 2002.

either the coordinator or its backup sites.

Recently, commit processing has attracted strong attention due to its effect on the performance of transaction processing. In [3, 4], it has been reported that as compared to 2PC, performance is further degraded with 3PC due to an extra round of message transmission. Also, it has been reported in the literature [5] that the performance of commit processing depends on the message processing latency of the coordinator. This means 3PC incurs a significant overhead over 2PC due to the extra round of message transmission. In [7, 8], an idea of delegating commit processing to other reliable nodes has been exploited, to reduce blocking. In this approach, the nodes are categorized according to their reliability. The blocking problem is reduced by allowing a less reliable coordinator to delegate the job of commit processing to a highly reliable coordinator that fails rarely and recovers quickly.

To reduce the blocking problem, the notion of backup processes are used in SDD-1[6]. 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. "Looking" in this means periodically sending control messages. It can be observed that the backup processes in SDD1 are concerned with the maintenance of uninterrupted service in case of coordinator's failure. For this, the backup process continuously looks to the coordinator and records the coordinator's state to enable the replacement of coordinator's operations, if the coordinator fails. So, the backup processes in SDD1 are employed to increase the reliability of service and needs a complex handling. However, the backup sites in BC are only concerned with the resolution of the transactions that were blocked due to the failure of the coordinator. In BC, the coordinator pushes the information to the backup sites. The operation is a kind of a remote log. The participants pulls the information from the backup site to resolve the blocking.

The proposed approach is motivated by the fact that 2PC is widely applied protocol in commercial database systems. Even though 3PC eliminates blocking, it has not entered into commercial database systems. In this situation, we have made an effort to reduce the blocking problem of 2PC by employing extra hardware. The BC protocol suits best for the Internet environments [9] in which messages take longer delivery times. Also, it can be integrated with existing 2PC implementations with little effort.

The preliminary versions of proposed protocol have appeared in [10, 11]. In [10], a backup commit protocol has been proposed in which only one backup site is attached to a coordinator whereas in [11] multiple backup sites are attached to the coordinator. In this paper, we present a generalized version of the BC protocol in which multiple backup sites can be attached to a coordinator to reduce blocking problem. We have also extended the analysis and discussion.

The paper is organized as follows. In section 2, we briefly explain both 2PC and 3PC. In section 3, we explain the BC protocol. In section 4, we discuss the blocking analysis and performance evaluation. The last section consists of summary and conclusions.

2 Distributed commit protocols

In the literature, a variety of commit protocols have been proposed, most of which are based on 2PC. The most popular variants of 2PC are the presumed abort and presumed commit protocols [12]. The others include the early-prepare [13], the coordinator-log [14], and the implicit-yes-vote [15] protocols. Also, different communication paradigms can be used to implement 2PC. The one described below is called the centralized 2PC, since the communication is between the coordinator and participants only. In this process, the participants do not communicate among themselves. In this paper, we do not discuss the other protocols since these are not concerned with the blocking problem.

The detailed explanation of the termination and recovery protocols for both 2PC and 3PC against failures such as coordinator timeouts, participant timeouts, and participant failures can be found in the literature [16]. Here, we briefly explain 2PC and quorum based 3PC.

2.1 Two-phase commit

In DDBS, the data objects are stored at database sites connected by a computer network. Each site works as both transaction manager and data manager. The transaction manager supervises the processing of transactions, while the data managers manage individual databases [16]. The originating site of a transaction, T_i, acts as a coordinator for T_i's commit processing. The sites which are involved in the processing of T_i, are called the participants of T_i. The coordinator site of T_i acts also as a participant site[1].

A brief description of 2PC that does not consider failures is as follows. The coordinator writes a begin-commit record in the log, sends a *PREPARE* message to all the participating sites, and then enters the **wait** state. When a participant receives the *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 if it receives the *VOTE_ABORT* message from one participant. Then, it writes an abort record, sends a *GLOBAL_ABORT* message to all the participants, and enters the **abort** state. Otherwise, it writes a commit record, sends a *GLOBAL_COMMIT* message to all the participants, and enters the **commit** state. The participants either commit or abort the transaction according to the coordinator's instructions and send back an *ACK* (acknowledgment) messages, at which point the coordinator terminates the transaction by writing an end-of-transaction record in the log.

Blocking problem: Consider a situation in which a participant has sent the *VOTE_COMMIT* message to the coordinator and has not received either the *GLOBAL_COMMIT* or *GLOBAL_ABORT* message in return due to the coordinator's failure. In this case, all such participants are blocked until the recovery of the coordinator to get the coordinator's decision.

[1]So, when the coordinator fails, one of the participant also fails.

2.2 Quorum based three-phase commit

In the case of quorum based 3PC, 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 the number of votes in abort and commit quorum are V_a and V_c, respectively. The following rules must be obeyed by 3PC.

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; the coordinator aborts the transaction globally, if the coordinator receives the *VOTE_ABORT* message even from one participant. However, the commit case is different. If the coordinator receives *VOTE_COMMIT* messages from all the participants, it writes a *prepare_to_commit* record, sends a *PREPARE_TO_COMMIT* message to all the participants, and enters a new **pre_commit** state. On receiving this message, each participant writes a *prepare_to_commit* record, sends a *READY_TO_COMMIT* message, and enters a **pre_commit** state. Finally, when the coordinator receives the *READY_TO_COMMIT* messages, if the sum of the votes of the responding sites equals to or exceeds V_c, after writing a commit record, it sends the *GLOBAL_COMMIT* message to all the participants, and enters the **commit** state.

Blocking elimination: We briefly explain how 3PC eliminates the blocking problem by dividing the situation into two cases. First, the participant has sent the *VOTE_COMMIT* message but has not received the *PREPARE_TO_COMMIT* message due to the coordinator's failure. Second, a participant has received the

PREPARE_TO_COMMIT message but has not received the *GLOBAL_COMMIT* message due to the coordinator's failure. In both cases, the operational participants elect a new coordinator. The new coordinator collects the state information 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.

3 The backup commit protocol

In this section, we first explain the BC protocol. Next, we explain the termination and recovery protocols in case of blocking. Then, we discuss the behavior in case of partitioning failures.

3.1 The BC Protocol

Suppose there are n sites in DDBS. The notation BS(i,j) denotes the j'th backup site of site S_i. In this approach, k backup sites $BS(i,j)(j = 1 \ldots k)$ are attached to each site S_i. A backup site is failure independent and connected with the coordinator in a shared nothing mode. Let $backup\_set_i$ be a set of identity of all the backup sites of S_i. The first and third phases of the BC protocol are similar to the first and second phases of 2PC, respectively. The description of BC is as follows.

1. **First phase**

   1.1 Coordinator : The coordinator writes a *begin-commit* record in the log, sends a *PREPARE* message along with $backup\_set_i$ and the participants information to all the participants, and enters a **wait** state.

   1.2 Participant : When a participant receives a *PREPARE* message, it stores the *backup_set* information, and 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 a **ready** state. Otherwise, the participant writes an *abort* record and sends a *VOTE_ABORT* message to the coordinator.

2. **Second phase**

   2.1. Coordinator : If the coordinator (S_i) receives the *VOTE_COMMIT* messages from all the participants, it sends a *DECIDED_TO_COMMIT* message to all $BS(i,j)$, where j=1 to k, after force-writing a *decided_to_commit* record.

   Otherwise, if it receives a *VOTE_ABORT* message from even one participant, a *GLOBAL_ABORT* message is sent to all $BS(i,j)$ (where, j=1 to k), and participants after force-writing an *abort* record.

   (In case of abort, the coordinator can send the *VOTE_ABORT* messages to the participants without informing the backup sites. However, we inform the backup sites to speedup the recovery of participants in case of blocking. This will not affect the performance as most of the submitted transactions commit than abort.)

   2.2. Backup site BS(i,j) ($1 \leq j \leq k$) : If $BS(i,j)$ receives a *DECIDED_TO_COMMIT* message, it writes a *recorded_commit* record on the stable storage and then sends back a *RECORDED_COMMIT* message to S_i.

   Otherwise, if it receives a *GLOBAL_ABORT* message, $BS(i,j)$ writes an *abort* record on the stable storage.

3. **Third phase**

   3.1. Coordinator : If it receives the *RECORDED_COMMIT* message from any of the BS(i,j), where $1 \leq j \leq k$, the coordinator writes a *commit* record on the stable storage and sends the *GLOBAL_COMMIT* message to all the participants.

Otherwise, if it receives no response from the backup sites, the following actions are followed. If the coordinator is able to confirm the fact that no backup site has received the *RECORDED_COMMIT* message it can safely abort[2] the transaction, after writing the *abort* record on the stable storage. Otherwise, if the coordinator is unable to confirm the receipt of the *RECORDED_COMMIT* message by a backup site, it follows the recovery protocol (see section 3.3) for that transaction.

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

3.3. Coordinator : After receiving the acknowledgment messages from all the participants, the coordinator writes the *end_of_transaction* record on the stable storage.

3.2 Termination protocols

In case of blocking, the participant sends the inquire messages to all the other participants, backup sites and coordinator to resolve the transaction's fate.

1. If the coordinator or any other participant contains a commit or abort record, it follows the same.
2. If any one of the backup site contains *recorded_commit*/abort record, it commits/aborts the transaction, accordingly.
3. If all the backup sites are up and no information exists at all the backup sites, it aborts the transaction.
4. If some backup sites are down and no information exists at the up-backup sites, the participant waits for the recovery of the other backup sites or the coordinator. After the recovery, one of the preceding steps are followed.

[2]Since there is no response from the backup sites, we conservatively abort the transaction.

3.3 Recovery protocol for the coordinator

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

I The coordinator finds the *begin_commit* record but no *decided_to_commit* record
In this case, it can safely abort the transaction without contacting the participants. (In the worst case, 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 coordinator might have failed after writing the *decided_to_commit* record but before sending a *DECIDED_TO_COMMIT* message to the backup sites. Second, the backup sites might have failed before receiving the *DECIDED_TO_COMMIT* message from the coordinator. And third, the coordinator might have failed after the receipt of the *DECIDED_TO_COMMIT* message by the backup site. These possibilities are resolved in BC as follows.

- If the coordinator finds the *decided_to_commit* record at any one of the backup sites, it sends the *GLOBAL_COMMIT* messages to all the participants. (In the worst case, a participant either might have committed the transaction by contacting the backup site or is in the **ready** state.) Otherwise, if the coordinator finds no information about the transaction at the backup sites, the coordinator sends the *GLOBAL_ABORT* messages to all the participants. (In the worst case, a participant either might have aborted the transaction by contacting the backup site or is in the **ready** state.)

- If the coordinator is unable to contact the backup sites, there exist two options. First, it waits for the recovery of all the backup sites and follows the preceding recovery protocols. Second, it will ask all the participants to report the transaction's status. In this case, we assume that the participant distinguishes the recovery messages from the normal messages; i.e., after responding to the recovery messages of the coordinator, the participant will not contact the backup site in future about the corresponding transaction. Thus, even though the coordinator fails during the recovery, the re-execution of the recovery protocol makes no difference.
    1. If at least one participant has committed/aborted the transaction, the coordinator follows suit.
    2. If all the participants are in **ready** state and the coordinator is still unable to resolve the transaction by establishing the contact with the backup sites, the coordinator aborts the transaction[3].

III The coordinator finds the *commit/abort* **record but no** *end_of_transaction* **record**
In this case, it can safely re-send the *GLOBAL_COMMIT/GLOBAL_ABORT* messages to all the participants, accordingly. (In the worst case, the participants either might have committed/aborted the transaction by contacting the backup sites or are in the **ready** state.)

3.4 Backup site failure and network partitioning

So, when a partition occurs, for a coordinator, it is equivalent to backup sites' or participant's failure; for a participant it is equivalent to a coordinator's failure. So, the recovery and termination protocols in case of blocking ensure the consistency in case of partitioning.

[3]Since the backup sites are down, we conservatively abort the transaction.

4 Blocking analysis and performance evaluation

In this section, we divide the performance study into two parts. We first show that the blocking of participants in BC is significantly reduced over 2PC by analyzing the failure probability. Next, through simulation experiments, we show that the performance of BC is close to 2PC.

4.1 Blocking analysis

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, let $MTTF_b$ represents MTTF of the 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.

Let P_c be the probability that the coordinator site is unavailable. Then, $P_c = \frac{MTTR_c}{MTTF_c+MTTR_c} \simeq \frac{MTTR_c}{MTTF_c}$, since $MTTR_c \ll MTTF_c$.

Let P_b be the probability that the backup site fails. Then, $P_b = \frac{1}{MTTF_b}$.

The probability that k backup sites fail and the corresponding coordinator is down is equal to:
$P_b^k \times P_c = \frac{MTTR_c}{(MTTF_b)^k \times MTTF_c}$.

From the preceding analysis, it can be observed that the probability that k backup sites fail while the corresponding coordinator is down is reduced significantly over P_c. Thus, with the introduction of the backup sites, the blocking probability of a participant is significantly reduced as compared to 2PC (i.e., without backup sites).

Further, it can be observed that the purpose of the backup sites 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 sites fail, it does not affect the consistency of the database. Let *term_time* be the time duration required to terminate the blocked transactions by contacting a backup site when the coordinator is down. The above equation denotes the probability that the backup sites fail during the entire period ($MTTR_c$) when the coordinator is down. However, in the worst case the blocked transactions are consistently terminated even if a 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 a backup site fails during the *term_time* while the coordinator is down is further reduced to infinitesimal.

4.2 Simulation experiments

The meaning of each model parameter and the value is given in Table 1. The size of the database is assumed to be *db_size* data objects. The database is uniformly distributed across 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 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 WAN behavior is realized by varying *trans_time*. We employ distributed static two-phase locking algorithm for concurrency control.

The primary performance metric of our experiments is throughput, i.e., the number of transactions completed per second. Each experiment has been run 15 times; each time with 10000 transactions. For each value, we ignored the result of first five experiments; the average of the remaining 10 experiments is taken.

Table 1. Model parameters with settings

Parameter	**Meaning**	**Value**
db_size	Number of objects in the database	1000
num_sites	Number of sites	5 sites
trans_size	Mean_size of transaction	8 objects
max_size	Size of a largest transaction	12 objects
min_size	Size of a 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	5 msec
res_io	I/O time	20 msec
backup_trans _time	Trans_time to contact backup site	1 msec
MPL	Multiprogramming level	Simulation variable
trans_time	Transmission time between two sites	Simulation variable

The setting of the parameters *db_size*, *trans_size*, *min_size* and *max_size* are given in Table 1 which are adopted in [17]. The parameter *local_to_total* ratio for a transaction is fixed at 0.6. Thus, 60 percent of the data objects are randomly chosen from the local database and 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

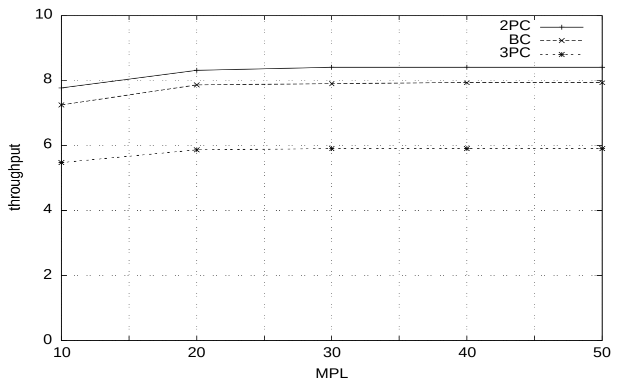

Figure 1: MPL versus throughput at *trans_time* = 1 msec.

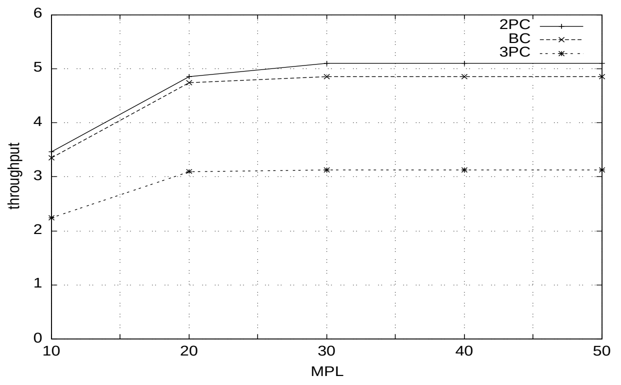

Figure 2: MPL versus throughput at *trans_time* = 500 msec.

to 1). The parameters *res_cpu* and *res_io* are fixed at 5 milliseconds (msec) and 15 msec, respectively [4]. The parameter *backup_trans_time* is fixed at 1 msec. We consider that both the coordinator and at least one backup site are connected to the high speed local area network. With these settings, by varying MPL values, sufficient variation in the data contention is realized.

At different MPL values, both Figure 1 and Figure 2 show the throughput results by setting transmission time (trans_time) to 1 and 500 msec, respectively. From Figure 1, as transmission time is 1 msec (since the *backup_trans_time* is fixed at 1 msec, we fix the minimum value for *trans_time* value at 1 msec), the performance of BC is slightly less than that of 2PC due to the overhead of communicating with the backup sites. However, in Figure 2, with the increase in transmission time, the effect

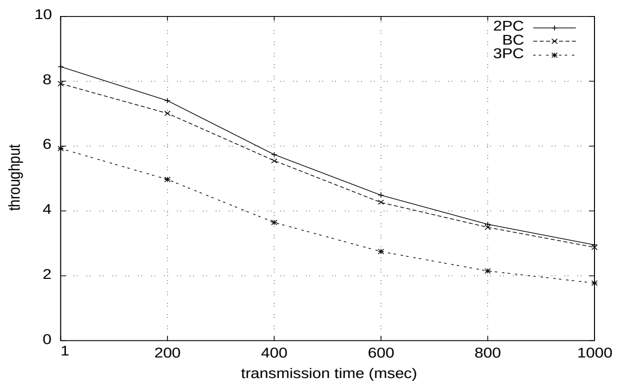

Figure 3. Transmission time versus throughput at MPL=30.

of communication with the backup sites is nullified. As a result, the throughput curve of BC comes close to 2PC. At different transmission time values, Figure 3 shows the throughput results at MPL=30. As transmission time increases, throughput decreases, because more number of transactions wait for the data objects for a longer duration. It can be observed that as transmission time increases, the overhead of communicating with the backup sites in BC is nullified. As a result, the throughput curves of both BC and 2PC coincide.

5 Summary and conclusions

In this paper we have proposed the BC protocol for DDBSs to reduce the blocking. In this protocol, blocking is reduced by attaching multiple backup sites to the coordinator site. Through analysis it has been shown that the probability that both the coordinator and the corresponding backup sites are down at the same time is significantly reduced as compared to the case of coordinator with no backup sites. Also, the simulation results show that the performance of both BC is very close to that of 2PC.

The BC protocol incurs slight overhead over 2PC as BC requires extra messages and time duration (to communicate with the backup sites) to complete a transaction. By selecting nearest site to the coordinator as one of the backup site, the overhead can be nullified. Also, BC requires fixed time duration during the second phase independent of the number of participants. So, BC reduces the blocking phenomena with negligible overhead over 2PC and suits well for commit processing.

Acknowledgments
This work is supported by "Research for the future" (in Japanese Mirai Kaitaku) under the program of Japan Society for the Promotion of Science, Japan.

References

[1] J.N.Gray, Notes on database operating systems: in operating systems an advanced course, Volume 60 of Lecture Notes in Computer Science, (1978) 393-481.

[2] D.Skeen, A quorum-based commit protocol, in proc. of 6th Berkeley Workshop on Distributed Data Management and Computer Networks. (1982) 69-80.

[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) 135-152.

[4] Ramesh Gupta, Jayant Haritsa and Kirti Ramamritam, Revisiting commit processing in distributed database systems, ACM SIGMOD, (1997) 486-497.

[5] M. L. Liu, Divyakant Agrawal, Amr El Abbadi: The Performance of Two Phase Commit Protocols in the Presence of Site Failures. Distributed and Parallel Databases 6(2) (1998) 157-182.

[6] Michael Hammer and David Shipman, Reliability mechanisms for SDD-1: A system for distributed databases, ACM Transactions on Database Systems, 5(4) (1980), 431-466.

[7] Kurt Rothermel and Stefan Pappe, Open commit protocols tolerating commission failures, ACM Transctions on Database Systems, 18 (2) (1993) 289-332.

[8] Yousef J. Al-Houmaily and Panos K.Chrysanthis. Implicit-Yes Vote Commit Protocol with Delegation of Commitment. Proc. of the 9th Int'l Conference on Parallel and Distributed Computing systems, (1996) 804-810.

[9] J.Lyon, K.Evans and J. Klein, Transaction internet protocol, version 3.0, "http://www.faqs.org/rfcs/rfc2371.html", June 2000.

[10] P.Krishna Reddy and Masaru Kitsuregawa, Reducing the blocking in two-phase commit protocol employing backup sites, in Proceedings of 3rd IFCIS International Conference on Cooperative Information Systems (CoopIS'98), (1998) 406-415.

[11] P.Krishna Reddy and Masaru Kitsuregawa, Blocking reduction in two-phase commit protocol with multiple backup sites, in Proceedings of International Workshop in Networked Information Systems (DNIS2000), Japan, (2000).

[12] C.Mohan, B.Lindsay and R.Obermark, Transaction management in the R^* distributed database management system, ACM Transactions on Database Systems, 11(4) (1994).

[13] P.Spiro, A.Joshi, and T.K.Rangarajan, Designing an optimized transaction commit protocol, Digital Technical Journal, (1991).

[14] J.Stamos and F.Cristian, Coordinator log transaction execution protocol, Journal of Distributed and Parallel Databases (1993) 383-408.

[15] 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, (1995).

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

[17] R.Agrawal, M.J.Carey and M.Livny, Concurrency control performance modeling: alternatives and implications, ACM Transactions on Database Systems, 12(4) (1987), 609-654.