Multiway Merge Recovery in Optimistic Partitioned Operation of Distributed Databases.

Shobha Prabakar
B.E., Mysore University, India, 1985

A thesis submitted to the faculty
of the Oregon Graduate Center
in partial fulfillment of the
requirements for the degree
Master of Science
in
Computer Science

August 1989

The thesis "Multiway Merge Recovery in Optimistic Partitioned Operation of Distributed Databases" by Shobha Prabakar has been examined and approved by the following Examination Committee:

Anil Garg
Adjunct Assistant Professor
Thesis Research Advisor

David E. Maier
Professor

David G. Novick
Assistant Professor

Multiway Merge Recovery in Optimistic Partitioned Operation of Distributed Databases

ABSTRACT

Network Partitioning is a serious problem in distributed databases because it threatens the reliability and availability of replicated data. The loss of communication between different sites may lead to destruction of mutual consistency of the database, unless restrictions are imposed on transaction processing or the database is repaired after communication is reestablished. Optimistic partitioned operation allows independent and unrestricted transaction processing to occur during partitioning and makes the database consistent after reconnection of the partitions by backing out the transactions that cause inconsistencies.

Previous research using a graph theoretic approach to optimistic partitioned operation restricts partitions to be merged only two at a time. Our research is concerned with generalizing this approach by allowing more than two partitions to merge at a time. Further, we simulated the multiway merge algorithms to evaluate their performance and analyze the advantages and disadvantages as compared to two-way merge recovery. Our results show that when conflicts between transactions are low, multiway merging reduces the number of transactions that are backed out.

Table of Contents

CHAPTER 1

INTRODUCTION

A *distributed database system* (DDBS) is a database implemented on a network of computers located at different sites and interconnected by a communication subsystem. A user at one site has the ability to access data that is stored at any other site. In case of a site failure in a distributed database, the remaining sites can still continue with their normal operation. In particular, if the data is replicated at several sites, then a transaction in need of a particular data item may still be able to access another site having a copy of that item. Thus, failure of a site need not necessarily lead to the shutdown of the entire system.

In a DDBS, query processing can be speeded up by decomposing a query into many subqueries and executing them in parallel at several sites. In the case of a replicated database, some subqueries may be processed at the sites which are found to be less busy.

However, there is a price to be paid for this increased availability of data and fast query processing capability. It is much more complex to achieve intersite coordination. There is increased processing overhead

because of exchange of messages and the update propagation. In a replicated database, all the copies of a data item must have the same value when all the update activities end. This requirement is called *mutual consistency.* To maintain mutual consistency, an update operation on a data item requires the update to be performed on all the copies of that data item [ASC 85], [Garc 81], [Ram 89]. Algorithms for replicated data management also become more complex, especially in the face of site or communication link failures or both.

1.1. Partitioning in DDBS

The following description of network partitioning and mutual consistency is from [Ma 86].

1.1.1. Network Partitioning

Network Partitioning is a serious problem in distributed databases because it threatens the reliability and availability of replicated data. Network Partitioning divides a network into groups of sites, called *partitions.* Sites within a particular partition can communicate with each other but not with the sites in any other partition. This loss of communication between

different sites may lead to destruction of mutual consistency of the database, unless restrictions are imposed on transaction processing. Mutual inconsistency may result if the copies of a data item are updated to different values by different transactions in different partitions.

1.1.2. Maintenance of Mutual Consistency

To maintain mutual consistency in a DDBS in the face of partitioning, one of the following three approaches can be taken [BeGo 84], [Ma 86].

(1) Do not allow any updates until the network is united. However, queries or read-only transactions are allowed.

(2) The *pessimistic* approach, which allows at most one partition to process new update transactions.

(3) The *optimistic* approach, which allows all the partitions to process new transactions independently.

The first approach is too restrictive because no update can be performed anywhere even if only a single site is partitioned from the rest of the network. Previous work on the pessimistic and optimistic approaches is described in the next section.

1.2. Previous Work Pessimistic approach:

This approach is also called "conservative" since it never allows the database to become inconsistent [Wrig 83]. Many protocols have been suggested for transaction processing during partitioning that guarantee mutual consistency throughout the system by limiting the availability of replicated data: rules are given that guarantee that each replicated data item is accessible in at most one partition. Updates are simply forwarded to the other partitions at recovery. Thus, an important problem using this approach is how to guarantee that each data item is updated in at most one partition. The methods or solutions proposed to solve this problem include voting [EaSe 83], tokens [Lela 78], and the primary copy [WiLa 84]. Since this approach restricts data availability, it is appropriate for environments where data consistency is of paramount importance.

Optimistic approach:

This approach allows independent and unrestricted transaction processing to occur during partitioning and assumes that transactions can be undone. As each partition updates the database, the database may diverge; i.e., copies of data items may have different values at different sites. At network reconnection time the database has to be repaired. This approach is

called *optimistic* because it is hoped that the independent updates will not conflict and hence will not have to be undone. Studies have shown that, for applications where data availability (as opposed to temporary loss of consistency and the resultant cost of transaction backout) is important, optimistic partitioned operation can be attractive [Davi 82], [Garc 81], [AbTo 89].

One way to make the database consistent after partitioning is to use syntactic information to backout or undo the transactions that cause inconsistencies. For example, the graph theoretic method, described in ([Daga 81], [Davi 82], [Davi 84], [Wrig 83], and [WrSk 83]) uses a serialization graph to perform conflict detection and resolution. This method is described in detail in the next chapter.

1.3. Problem Statement and Thesis Overview

Antony Vu Ma ([Ma 86]) used the graph theoretic approach to investigate optimistic partitioned operation in distributed database systems. His approach has the restriction that partitions can only be merged two at a time.

Our research is concerned with generalizing Ma's design of a transaction processing mechanism using graph theoretic approach to optimistic partitioned operation in replicated distributed database systems. We perform *multiway* merging by allowing more than two partitions to merge at a time. We generalize the reconnection detection protocol and merge recovery algorithms to incorporate multiple partitions. Further, we simulate the multiway merge algorithms to evaluate their performance and analyze the advantages and disadvantages as compared to the two-way-only merging.

The thesis is organized as follows. Chapter 2 contains a description of the optimistic partitioned operation and two-way merge recovery. In Chapter 3, we describe multiway reconnection detection protocol and algorithms for multiway merge recovery. Chapter 4 contains an overview of the simulation program and the results of the simulation of multiway merging and recovery. Finally, Chapter 5 contains the conclusions of our research and suggestions for further work in this and related areas.

CHAPTER 2

OPTIMISTIC PARTITIONED OPERATIONS

In this chapter, basic definitions and assumptions used in this thesis are stated. The necessary theoretical background for this thesis is provided and previous work on the optimistic partitioned operation using graph-theoretic method for conflict detection and resolution is reviewed.

2.1. Assumptions and Basic Definitions

We assume that the database is fully replicated, i.e., a copy of each item of the database is stored at every site in the network. A transaction uses READ operations to access data items and WRITE operations to possibly modify them. The set of data items read by a transaction is called its READSET. The set of data items inserted or modified by a transaction is called its WRITESET. We assume that the writeset of a transaction is a subset of its readset. A transaction is executed atomically, i.e., the system executes either all or none of its operations.

A transaction first tries to obtain locks on all the data items it needs to read or write. The locking protocol used is distributed two phase locking,

CHAPTER 2

OPTIMISTIC PARTITIONED OPERATIONS

In this chapter, basic definitions and assumptions used in this thesis are stated. The necessary theoretical background for this thesis is provided and previous work on the optimistic partitioned operation using graph-theoretic method for conflict detection and resolution is reviewed.

2.1. Assumptions and Basic Definitions

We assume that the database is fully replicated, i.e., a copy of each item of the database is stored at every site in the network. A transaction uses READ operations to access data items and WRITE operations to possibly modify them. The set of data items read by a transaction is called its READSET. The set of data items inserted or modified by a transaction is called its WRITESET. We assume that the writeset of a transaction is a subset of its readset. A transaction is executed atomically, i.e., the system executes either all or none of its operations.

A transaction first tries to obtain locks on all the data items it needs to read or write. The locking protocol used is distributed two phase locking,

called *distributed 2PL* [TCB 83]. If the transaction cannot obtain all the locks, it must wait until the transactions that currently hold the required locks are resolved. The transaction starts execution once it obtains all the locks it needs. When the transaction is committed at the site of its origination (its *originator*), update messages are sent to the other sites (its *subordinates*) in the rest of the system that are participating in that transaction. Note that in the case of a fully replicated database, all the sites participate in a non-read-only transaction.

The two-phase commit protocol (2PC) is used to achieve atomic commitment ([Gray 79], [Kohl 81]). The originator sends a message to all its subordinate sites asking them if the transaction can be committed. If so, the subordinate site precommits the transaction and sends a 'yes' message back to the originator. If the transaction cannot be committed (for instance due to the data items not being available for update at that site), the subordinate sends a 'no' message to the originator. If the originator receives only 'yes' messages from all the subordinate sites, it commits the transaction and sends commit messages to all the sites. If, however, any subordinate sends a 'no' message, the originator aborts the transaction and sends abort messages to all the subordinate sites that sent a 'yes' message. The write

locks held by a transaction are not released at a subordinate until the subordinate receives an abort or commit message for the transaction from its originator.

Certain failures during the 2PC protocol can leave transactions in the precommitted state. The resolution of such transactions is described in the next section.

2.2. Network Partitioning

Partitioning occurs because of link or site failures. It may take more than one link failure to cause partitioning, depending on how the sites in the network are connected to each other. However, a site failure always results in partitioning: an inactive partition containing the failed site, and another partition containing the rest of the sites. Since it is not possible to distinguish between a failure and a site that is taking too long to respond, in practice the availability of a site is determined by the use of message timeouts. The Partition Detection protocol described in [Ma 86] and reviewed later in this chapter uses this mechanism to detect partitioning.

If a system partitioning occurs after a transaction reaches the precommit stage, the subordinate may not receive a commit or abort

message, leaving the precommit dangling and hence the locks unreleased. The dangling precommit (DP) algorithms described in [Ma 86] release the locks held by such a precommitted transaction by tentatively committing or aborting the transaction. When the partitions merge, it is possible that one partition decided to commit the transaction whereas another aborted the transaction. To remove such inconsistencies, some of the conflicting transactions along with their dependents[1] are rolled back. The strategies used to select the transactions to be rolled back are described in the next chapter.

The partitions can be merged when the failed links or sites recover. When a reconnection between partitions is detected, a merge algorithm is used to repair the database using a graph theoretic method [Davi 84].

In the subsequent sections, we summarize the partition detection protocol, the reconnection detection protocol, and the graph theoretic method for conflict detection and resolution in optimistic partitioned operation [Ma 86].

---

[1]The *dependents* of a transaction T are recursively defined as the transactions that read a data item after it is written by T, and in turn, their dependents. Also refer to the definition of a serialization graph in section 3.2.2 for the definition of transaction dependency.

2.3. Partition Management Protocol

Partitioning occurs as a result of a site or communication link failure. In the following discussion, the term *failure* is used to describe both forms of failures.

We assume that the network partitionings and partition reconnections are discrete, i.e., there is a finite non-zero interval between partitionings and reconnections. The protocol requires that all the sites in a partition agree on the set of available sites and execute the proper recovery algorithms whenever a partition or a reconnection occurs. If a failure is encountered while a merge is taking place, the merge is aborted.

The Partition Management (PM) protocol consists of two separate protocols: the Partition Detection (PD) and Reconnection Detection (RD) protocols.

To perform the PD and the RD protocols, some data structures are required at each site. Three flags, *R-flag*, *A-flag*, and *M-flag* are used to indicate the status of the node with respect to partitioning or merging activity. A site accepts transactions for processing only when the *A-flag* is true. The *R-flag* is set to true while the site's current partition is undergoing reconfiguration. The *M-flag* is set to true while a site is undergoing a merge.

Each site also maintains its logical view of the network using a table called the *Network Status Table* (*NST*). The *NST* contains entries for each site in the network. A site's entry is marked DOWN if that site is perceived to be down; otherwise, it is marked UP.

2.3.1. Partition Detection Protocol

We now give an informal description of the PD protocol. For a detailed formal description of the protocol, refer to [Ma 86]. The PD protocol consists of two parts : normal operation and operation during reconfiguration.

For each site, another site in the network is assigned as a guardian. If the network is partitioned, the guardian of a site is selected to be within the partition to which the site belongs. If a partition is a single site partition, the site has no guardian.

During normal operation, every site must periodically send a HI message to its guardian. The only purpose of this message is to inform the guardian of its availability. If the guardian of a site does not receive the HI message for a specified time period, the PD protocol is switched to the reconfiguration mode.

In the reconfiguration mode, the guardian broadcasts REORG messages to all the other sites in its partition. A site issuing REORG messages is called an *initiator* for the reconfiguration. It is possible that multiple sites send REORG messages at the same time since there can be multiple failures or multiple detections of the same failure. Such concurrent reconfiguration attempts are resolved using the *R-flag* and by checking whether the receiver of a message is in the same partition as indicated by the Partition identifier (*Pid*) of the intended recipient contained in the message. The REORG messages also contain the proposed *Pid* for the new partition. The other sites in the partition that receive the REORG messages send responses to the initiator indicating their willingness to join the partition proposed by the initiator. The initiator then becomes the coordinator for the new partition. The coordinator then sends the new *NST* to the sites that agreed to join the partition. The transmission of the new *NST* marks the end of the reconfiguration and resumption of the normal operation at each site, but within a limited partition.

For a more detailed explanation of the PD protocol, refer to [Ma 86].

2.3.2. Reconnection Detection Protocol

The purpose of the RD protocol is to detect reconnection of different partitions. [Ma 86] describes the RD protocol that is specific for two-way reconnection and merging of partitions. The RD protocol for the generalized multiway reconnection and merge recovery is explained in detail with a formal algorithm in the next chapter.

2.4. Serialization Graph and Merge Recovery

When a reconnection between partitions is detected, the coordinators for the partitions initiate the merge recovery algorithm. Processing of new transactions in the merging partitions is suspended until the recovery is complete. Each coordinator derives a global serial history of the transactions that were executed in its partition. The individual global serial histories are used to construct a serialization graph following a set of rules as described in [Ma 86]. Presence of cycles in the serialization graph indicates conflicts between transactions. Transactions are backed out until the graph is acyclic. When a transaction is backed out, all its dependents are also backed out. Backing out a transaction involves resetting the value of each data item in the write set of the transaction to the values that were read by

that transaction.

After the cycles are removed, the database is repaired by forwarding the recovery information such as the list of backed out transactions to the sites in the merging partitions. The merge recovery algorithm is described in greater detail in the next chapter.

CHAPTER 3

MULTIWAY MERGING AND RECOVERY

3.1. Reconnection Detection Protocol

[Ma 86] has a description of the two-way reconnection detection (RD) protocol. In this section we describe how it can be generalized to the multiway reconnection detection (MRD) protocol.

3.1.1. Data Structures

In addition to the data structures used by the PD protocol, the multiway RD protocol uses the following data structures. The data structures are meaningful only at coordinator sites. Each partition participating in a merge has its own coordinator. Among them, the coordinator with the largest site number is chosen to be merge coordinator.

Merge_info: A record (one at each partition) that keeps track of the information about partitions involved in a particular merge. The up-to-date merge information is kept track of in the current merge coordinator for the merging partitions. The *merge_info* records at coordinators other than the

merge coordinator are invalid.

Each *merge_info* record contains the following fields.

nummerge : number of partitions that are taking part in the merge. Initial value is 0.

mrgtimer : The remaining time for which the merging partitions wait for the other partitions to join the merge. Initial value is the *maximum_merge_timer*, which is a parameter that can be controlled by the database administrator. It indicates the maximum time for which a merge recovery will be delayed to allow more partitions to participate in the merge.

partition_list : List of partitions that are going to merge. Each item in the list is a pair consisting of the *Pid* of the partition and the *site-id* of the coordinator for the partition.

NST : Holds the merge *NST*, i.e., the *NST* that will be applicable to the partition formed as the result of the merge.

coord_num: The coordinator for all the merging partitions. The initial value is the site's own *site-id*.

3.1.2. Description of MRD Protocol

The description of the MRD protocol is strutured along the lines of the description of the RD protocol in [Ma 86].

As long as the *A-flag* is true, *M-flag* is false and there is at least one DOWN site in the *NST*, periodically the coordinator of a partition broadcasts MERGE_INVITATION (MI) messages carrying a copy of its *NST* to the DOWN sites with site numbers greater than its own site number. The reason for sending MI messages only to greater site numbers is to avoid indefinite postponement of merging [Ma 86]. The transmission of the MI message is called a *probe*. The interval between two consecutive probes (*tprobe*) can be chosen randomly.

MI messages received by the non-coordinators are always ignored. Also, upon receiving an MI from a coordinator *B*, a coordinator *A* will ignore the message if its *A-flag* is false (it is not active) or *M-flag* is true (it is already merging or is in the middle of accepting an MI from some other partition coordinator, in which case it will not be ready at that time to accept another MI).

Next, partition *A* checks whether $\mathrm{UP}(NST_A) \cap \mathrm{UP}(NST_B)$ is empty. If the intersection is not empty, then *A* will ignore the MI because a failure has

not been detected by one of the partitions (for an example, see [Ma 86]). Otherwise, A will set its *M-flag* to true to block receiving further MI's. A sets a timer called *cancel_merge_timer*, with the timer value equal to maximum communication delay for round trip message ($2d_{max}$), and sends back MERGE_REQUEST (MR) to B containing A's and B's *Pid*'s and stops sending MI's to other DOWN sites. If site A does not hear from site B within *cancel_merge_timer* after it sent the MR to site B, then A aborts the merge attempt with B, resetting its *M-flag* to false. A, then, will possibly try to merge with some other partition(s) by sending or accepting MI's.

When B receives an MR message from A, it checks whether its *A-flag* is true, *M-flag* is false and whether the *Pid* contained in MR is its current *Pid* (this check is to ensure that the MR that it received is for an MI sent by itself) and whether it is still a merge coordinator. B might no longer be the merge coordinator because, after sending the MI message, but before receiving the MR message, B's partition might have agreed to join the merge being coordinated by some other merge coordinator.

If the message is not acceptable, due to any of the conditions mentioned above being false, then the message is ignored. Otherwise B agrees to merge with A by setting its *M-flag* to true. *A-flag* is not set to false indicating that

it continues to be an active partition accepting new transactions. In this case A will be the merge coordinator because MI reaches A from B only if A's site number is greater than B's site number. The partition coordinator with bigger site number is chosen to be the merge coordinator. Thus, B sets the merge coordinator to be A.

B also stops its *mrgtimer* because it is no longer the merge coordinator. Then it sends an ACK_MERGE_REQUEST (AMR) message back to A. The AMR contains B's merge information. Since B is no longer the merge coordinator, after sending the AMR, B sets its *M-flag* to false, stops sending MI's to other DOWN sites and stops accepting any MR's from other coordinators. The responsibility for sending MI's and accepting MR's is taken over by the new merge coordinator, A.

Next, B sets *cancel_merge_timer* to the *mrgtimer* value in its *merge_info* plus $2d_{\max}$ round trip delay of the message. If some merge coordinator does not send a PREPARE_TO_MERGE message (to be explained later) by the time B's *cancel_merge_timer* expires, B goes back to its original state and tries to merge with some other partition(s).[1]

---

[1]Note that B remains the coordinator for its partition even though it has agreed for merge with A.

When a merge coordinator receives an AMR, it checks whether its *A-flag* is true and whether its *Pid* is same as the one that it sent with the MR. If these two conditions are satisfied, the *cancel_merge_timer* is stopped. If either condition is not satisfied, some failure must have occurred in the partition after the MR message was sent out and hence the AMR message is ignored.

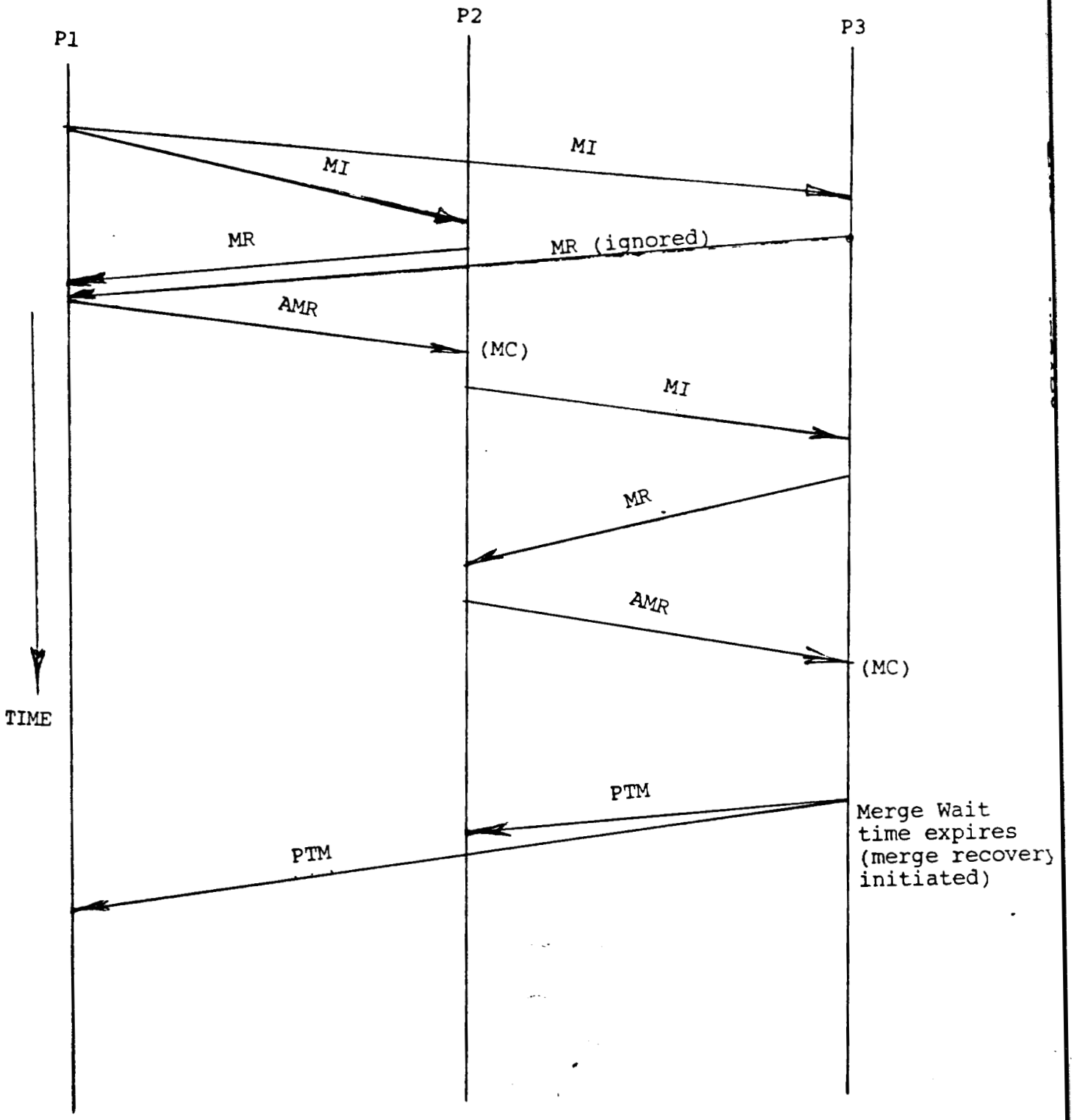
If *nummerge* in *merge_info* record equals the maximum number of partitions allowed to merge or the merge *NST* indicates a fully reconnected network, the merge coordinator sets its *mrgtimer* to zero and the procedure followed for *mrgtimer* expiration (described in the next paragraph) is invoked immediately. Otherwise, the *mrgtimer* is set to the smaller of the two values in the *merge_info* record at the merge coordinator and the record received with the AMR message. The reason for this action is to avoid the merge getting delayed indefinitely. Then the merge coordinator starts sending MI's to other DOWN sites whose site id's are greater than its own *site-id.*

When the *mrgtimer* expires, the merge coordinator sets its *A-flag* to false and *M-flag* to true and determines the new *Pid.* The new *Pid* is the maximum *Pid* seen so far by the merging partitions plus one [Ma 86] (The maximum *Pid* seen by each merging partition is stored in the *max_pid*

variable at the coordinator for the partition). Next, it sends a PREPARE_TO_MERGE (PTM) message to all the sites of all the partitions that have agreed to merge. The merge coordinator sends its *merge_info* record with the PTM message. It updates its own *NST*. If a failure takes place after the merge is initiated, the merge is aborted.

When a PTM arrives at a site, the site checks whether its *A-flag* is true and its *Pid* is the one of the *Pid*'s in the partition list sent by the merge coordinator. This check ensures that the site only accepts the PTM message sent by its own merge coordinator and not some other merge coordinator. If the conditions given above are satisfied, *M-flag* is set to true and *A-flag* is set to false and the *NST* is updated to new *NST* by all the sites. Then, the merge recovery algorithm is executed. When the merge recovery is completed, *M-flag* is reset to false and *A-flag* is set to true.

The illustration in Figure 3.1 shows how a 3-way merge is initiated. Consider P_1, P_2, and P_3 as three partitions of a network with C_1, C_2, and C_3 as their respective coordinators.



MRD PROTOCOL

Figure 3.1

Assume that site-id(C_1) < site-id(C_2) < site-id(C_3). First C_1 sends merge invitations (MI) to C_2 and C_3. C_2 responds with a merge request (MR). C_1 then sends an AMR message to C_2. C_2 is now the merge coordinator for P_1 and P_2. Now, since C_1 is no longer a merge coordinator it will ignore any merge requests from C_3. The merge coordinator C_2 sends an MI message to C_3, which then responds with a MR message. C_2 then sends an AMR as the response. The AMR carries the *merge_info* record maintained at C_2. The *partition_list* in the merge_info record contains the *Pid*s for P_1 and P_2, and the remaining merge timer value at C_2. C_3 becomes the new merge coordinator. The partition list in its merge_info record contains the *Pid*s of P_1, P_2, and P_3. The merge timer is set to the value in the AMR message from C_2 since C_3 was previously not a merge coordinator. If the network consists of only the partitions P_1, P_2, and P_3 then C_3 would immediately initiate a merge. Otherwise, it initiates a merge when its merge timer expires.

In the next section we give a formal algorithm for the MRD protocol described above.

3.1.3. Algorithm for Multiway Reconnection Detection Protocol

The following algorithm is in a notation similar to the syntax of the programming language C. The data structures required are declared as struture types and variables. The algorithm is described using a combination of comments and pseudo-code.

Data Structures:

```
/* The following record defines the structure of a Pid */
struct pid_t
     {
      int seqnum;    /* counter */
      int orig;      /* originator site */
     };

/* The following record defines the structure of the NST */
struct nst_t
     {
      struct pid_t Pid; /* Pid of the site's partition */
      int status[MAX_NO_SITES]; /* indicates whether sites are Up or Down */
     };


/* The following record defines the structure of the partition list
   in the merge_info record at the merge coordinator */
struct part_list_t
     {
      struct pid_t Pid;  /*Pid of the merging partition*/
      int coord;         /* Coordinator for the partition */
     }


/* The following record defines the structure of the merge information
   record maintained at the merge coordinator */
struct merge_info
     {
      int nummerge;        /* no of partitions that are
```

```
                        participating in the merge */
        int mrgtimer;     /* The time for which merge is delayed
                          to accept more partitions */
        struct part_list_t partition_list[];
                        /* The list of coordinators of the
                          partitions that are merging */
        int coord_num;    /* the merge coordinator */
        struct nst_t NST; /* new NST that gets updated whenever
                             a new partition joins the merge */
      };

int   myid;           /* site number of local site */
int   coord();        /* return site number of coordinator */
int   Aflag;          /* active flag */
int   Rflag;          /* reconfigure flag */
int   Mflag;          /* Merging flag */
struct nst_t NST; /* network status table */
```

case (tprobe expires):

```
  /* Now it is time to send the merge invitations; but first
     make sure that this site is eligible to transmit MI */

  if (myid == my_merge_info.coord_num && Aflag && !Mflag
  {
    /* Send MI to each site with greater site-id that is marked as
       Down in the current NST */
    for p ∈ {p|p>myid && NST.status[p] == Down}
    {
      MERGE_INVITATION.sender = myid;
      MERGE_INVITATION.NST = NST;
      send MERGE_INVITATION to p;
    }
  }

  /* Set timer for the next probe */
  tprobe.set(2dmax);
  break;
```

case (MERGE_INVITATION received):

```
  /* On receipt of the merge invitation message, check whether
     this site is a coordinator, whether it is "active" with respect
     to the rest of the network. Also, the sending site's NST and
     this site's NST must not have any UP sites in common. If all
     conditions are satisfied, then send an MR (merge request)
```

```
    message to the sender. */
  if (myid == my_merge_info.coord_num  &&  Aflag  &&  !Mflag &&
     Up(MERGE_INVITATION.NST) ∩ Up(NST) == NULL)
  {
   /* Set the M-flag to prevent any further merges with this coord */
   Mflag = TRUE;

   /* Build the MR message */
   MERGE_REQUEST.sender = myid;
   MERGE_REQUEST.pid1 = MERGE_INVITATION.NST.pid;
   MERGE_REQUEST.pid2 = NST.pid;

   send MERGE_REQUEST to MERGE_INVITATION.sender;

   /* Start the cancel_merge_timer.  If the originator of the
      merge invitation does not respond before the timer expires,
      this site will cancel the current merge and start accepting
      MIs again.  Till then this site will not accept any more MIs */
   cancel_merge_timer.set(2dmax);

   tprobe.stop; /* restarted when merge request is acknowledged
                   or cancel_merge_timer expires */
  }
  break;


case (MERGE_REQUEST message received):
  /* On receipt of an MR message, stop the merge timer at this
  coordinator (if one has been started), and send the AMR message
  to the sender of the MR message.  The sender is the merge
  coordinator */
  if (MERGE_REQUEST.Pid1 == NST.pid && Aflag && !Mflag &&
            (myid == my_merge_info.coord_num))
  {
   /* Set the M-flag to prevent any further merges with this coordinator */
   Mflag = TRUE;

   mrgtimer.stop;

   /* The larger of the node-ids of the sender and the receiver is
      selected to be the coordinator.  Since the merge request is
      in response to a merge invitation, and merge_invitation is always
      sent from smaller node-ids to larger node-ids,
      MERGE_REQUEST.sender must be the new merge coordinator */

   my_merge_info.coord_num = MERGE_REQUEST.sender;
```

```
    /* Build and send the AMR */
    ACK_MERGE_REQ.merge_info = my_merge_info; /* includes NST */
    ACK_MERGE_REQ.sender = myid;
    ACK_MERGE_REQ.pid = MERGE_REQUEST.pid2;
    send ACK_MERGE_REQ to MERGE_REQUEST.sender;

    /* Start the merge timer.  If the PTM message does not arrive
       before it expires, this site resets its merge status and starts
       sending MIs again */
    cancel_merge_timer.set(mrgtimer + 2dmax);

    Mflag = FALSE;
    tprobe.stop;
   }
   break;


case (ACK_MERGE_REQ message):
   if (Aflag && (ACK_MERGE_REQ.pid == NST.pid))
   {
    /* The AMR has been received from the node to which the MR message
       was sent.  Stop the merge timer.  Start it again later with
       the lower of the two values. */
    cancel_merge_timer.stop;

    /* Integrate the information in the AMR message into the merge_info
       record at this site */
    my_merge_info.mrgtimer = minimum(my_merge_info.mrgtimer, ACK_MERGE_REQ.mrgtimer);
    my_merge_info.partition_list = union(my_merge_info.partition_list,
                                         ACK_MERGE_REQ.partition_list);
    my_merge_info.nummerge = my_merge_info.nummerge + ACK_MERGE_REQ.nummerge;
    /* my_merge_info.coord_num is already set to myid correctly */
    my_merge_info.NST = union(my_merge_info.NST, ACK_MERGE_REQ.NST);


    /*  Initiate the merge immediately if the network is fully
        reconnected, or the limit on the merge-degree has been
        reached;  otherwise, start the merge timer to wait for
        more partitions to join the merge.*/
    if ((my_merge_info.nummerge >= MAX_NUM_MERGE) ||
       (Merge NST indicates network is fully reconnected))
      mrgtimer.set(0);  /* initiate the merge immediately */
    else
    {
      mrgtimer.set(my_merge_info.mrgtimer);  /* as updated above */
      tprobe.set(2dmax);  /* Set probe timer for new round of MIs */
```

```
    }

    Mflag = FALSE;
   }
   break;


case (MRGTIMER expires):
  /* Merge timer has expired.  Initiate the merge */
  if (myid == my_merge_info.coord_num)
    {
     Aflag = FALSE;
     Mflag = TRUE;

     /* Determine the Pid for the new partition */
     max_pid = max(merge_info.partition_list);
     max_pid.seqnum++;
     max_pid.orig = coord;

     my_merge_info.NST.pid = max_pid;

     /* Build the prepare_to_merge message */
     PREPARE_TO_MERGE.NST = my_merge_info.NST;
     PREPARE_TO_MERGE.partition_list = my_merge_info.partition_list;

     send PREPARE_TO_MERGE message to all the sites
          of the partitions that have agreed to merge;
     NST = my_merge_info.NST;
    }

   break;

case (PREPARE_TO_MERGE message):

   /* On receipt of the PTM message, the site sets its M-flag and R-flag
      to indicate that it is merging;  the A-flag is set to false to
      prevent new transactions from being accepted.  Then, the site
      starts sending HI messages to its guardian and waits to receive
      the Update-lists after the merge recovery process is completed
      at the merge coordinator */

   /* First check whether the site is still in the intended partition */
   if (AFlag && (NST.pid in PREPARE_TO_MERGE.partition_list))
   {
     Mflag = Rflag = TRUE;
     Aflag = FALSE;
```

```
    NST = PREPARE_TO_MERGE.NST;
    max_pid = NST.pid;

    HI.sender = myid;
    HI.pid = NST.pid;
    send HI to my guardian();
    trec.set(tbh + dmax - dmin);
    tsend.set(tbh);

  }
  break;
```

case (CANCEL_MERGE_TIMER expires):

```
  /* A PTM or a AMR message that was expected has not arrived. Cancel
     the merge. If this site is the merge coordinator, just set the
     Mflag to false. If it is not the merge coordinator reset the
     merge coordinator to be this site's site-id. */
  if (myid == my_merge_info.coordnum)
     MFlag = false;
  else
     my_merge_info.coordnum = myid;
  tprobe.set(2dmax);
  break;
```

3.2. Multiway Merge Recovery

In this section we discuss the multiway merge recovery algorithm. This algorithm is an extension of Ma's two-way merge recovery algorithm [Ma 86], which is based on the graph-theoretic method for conflict detection and resolution. After the MRD protocol has initiated a merge, all the sites in the partitions that are taking part in the merge must finish their in-progress transactions and then execute the merge recovery algorithm. We assume that no new transactions are accepted by the merging partitions until the

recovery is completed.

Before we describe the merge algorithm, we first define the concepts of the *partition history graph*, the *serialization graph*, and the *merge set.*

3.2.1. Partition History Graphs

The data structure used to record the history of a partition is called a *partition history graph* (PHG). Each site in a network has a copy of the PHG of the partition of which it is a member.

The PHG is defined below along with some related terms. The following definitions are reproduced here from [Ma 86] to set the stage for the subsequent description of the multiway merge recovery algorithm. For a detailed explanation of how a PHG is constructed and maintained, refer to [Ma 86]. An example of a PHG is given in next chapter.

A partition P_i is said to be an *immediate predecessor* of a partition P_j if and only if 1) $P_i = P_j$ or 2) there exists a site s in P_j such that s was a member of P_i and after s leaves P_i, the first active partition it joins is P_j. The transitive closure of the *immediate predecessor* relation is called the *predecessor of* relation and is denoted by the symbol "|--" in the following.

Definition 3.1. A *partition history graph* (rooted at O) of a partition P is a directed graph $PHG_o(P) = (V, E)$, where

$V = \{O\} \cup \{ P_i \mid P_i \vdash P \}$

$E = \{ P_i \rightarrow P_j \mid P_i \neq P_j$ and P_i is an immediate predecessor of $P_j \} \cup \{ O \rightarrow P_i$ if P_i does not have a predecessor $\}$.

If P_i and P_j are two (not necessarily distinct) vertices in a $PHG_o(P)$, then P_i *dominates* P_j (or P_i is a *dominator* of P_j) if and only if every path in $PHG_o(P)$ from O to P_j contains P_i. P_i *properly dominates* P_j if and only if $P_i \neq P_j$ and P_i dominates P_j. Also, for each vertex P_j,

$$DOM(P_j) = \{ P_i \mid P_i \text{ dominates } P_j \}.$$

A vertex P is the *nearest common dominator* of n vertices $P_1, P_2, \ldots P_n$ in a partition history graph, if and only if

(1) P dominates each of the n vertices and

(2) if P' dominates each of the n vertices, and P' $\neq$ P, then P' properly dominates P.

The nearest common dominator of $P_1, \ldots P_n$ will be denoted by $NCD(P_1, P_2, \ldots P_n)$ For a discussion of how to compute dominators of a set of vertices in a directed acyclic graph, see [AHU 74].

3.2.2. Serialization Graph

Let GH_i denote the *global serial history*[1] of transactions executed by the sites in P_i since a consistent logical database state DB.[2]

Definition 3.2 :

Let a, b, c, ... x, y denote the number of transactions executed in the various partitions being merged.

Let $GH_1 = T_{11}.Tid \;\; T_{12}.Tid \;\; T_{13}.Tid \;\; ... \;\; T_{1a}.Tid$
Let $GH_2 = T_{21}.Tid \;\; T_{22}.Tid \;\; T_{23}.Tid \;\; ... \;\; T_{2b}.Tid$
.
.
.
Let $GH_n = T_{n1}.Tid \;\; T_{n2}.Tid \;\; T_{n3}.Tid \;\; ... \;\; T_{ny}.Tid$

be the global histories of P_1, P_2, ... P_n as defined above. The serialization graph $G(GH_1, GH_2, ... GH_n) = (V, E)$ is the graph defined by:

$$V = \{T_{11}.Tid, T_{12}.Tid, ..., T_{1a}.Tid\}$$
$$\cup \{T_{21}.Tid, T_{22}.Tid, ..., T_{2b}.Tid\}$$
$$.$$
$$.$$
$$.$$
$$\cup \{T_{n1}.Tid, T_{n2}.Tid, ..., T_{ny}.Tid\}$$

[1]The global serial history of a set of transaction is a serialization order of the transactions. For a formal definition, see [Ma 86].

[2]As indicated by Lemma 5.5 in [Ma 86], the natural choice for the state DB is the initial logical database state in *NCD* of the merging partitions.

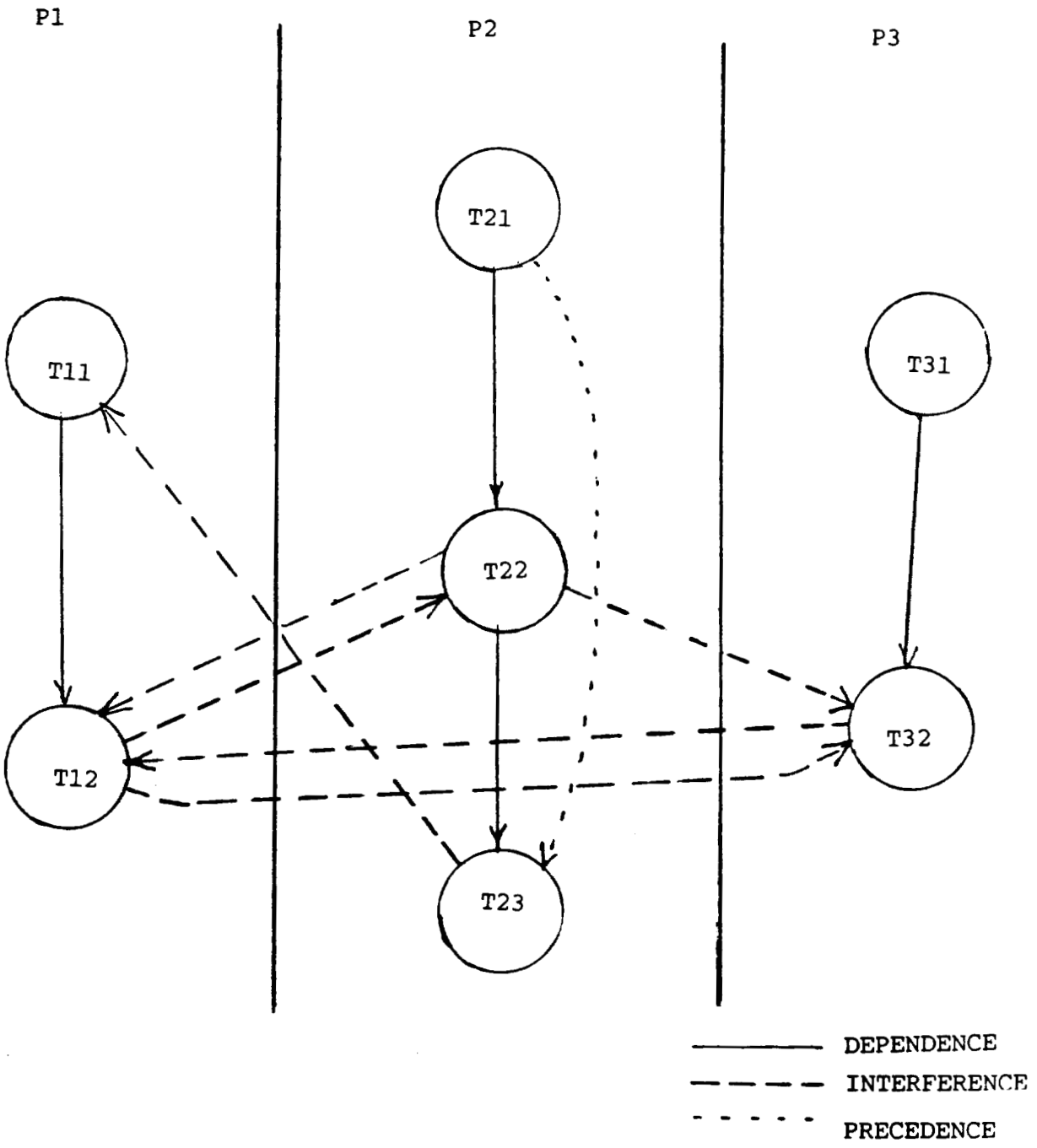
E = {Dependency Edges} ∪ {Precedence Edges} ∪ {Interference Edges}

(1) there exists a dependency edge T_{ci}.Tid → T_{ck}.Tid if and only if $i < k$ and there exists $X \in Ws(T_{ci}) \cap Rs(T_{ck})$ and forall j: $i < j < k$, X NOT $\in Ws(T_{cj})$. The dependency edges indicate that one transaction read a value produced by another transaction in the same partition.

(2) there exists a precedence edge T_{ci}.Tid → T_{ck}.Tid if and only if $i < k$ and there exists $X \in Rs(T_{ci}) \cap Ws(T_{ck})$ and forall j: $i < j < k$, X NOT $\in Ws(T_{cj})$ and there is no dependency edge T_{ci}.Tid → T_{ck}.Tid. The precedence edges indicate that a transaction read a value that was later changed by a second transaction in the same partition.

(3) there exists an interference edge T_{ci}.Tid → T_{dj}.Tid ($c = 1,..,n$; $d = 1,..,n$; and $c \neq d$) if and only if there exists $X \in Rs(T_{ci}) \cap Ws(T_{dj})$ and T_{ci}.Tid. Thus if there are P partitions, P(P-1)/2 combinations must be examined to construct interference edges. The interference edges represent the fact that if a transaction in one partition read a data item, it must precede any transaction that updated the same data item in some other partition.

Figure 3.2 shows a serialization graph of transactions from three partitions.



SERIALIZATION GRAPH

Figure 3.2

P_1, P_2, and P_3 are three partitions of a network.

- transactions T_{11} and T_{12} belong to P_1,
- transactions T_{21}, T_{22}, and T_{23} belong to P_2, and
- transactions T_{31} and T_{32} belong to P_3.

The solid arrows represent dependency edges. The dashed arrows represent interference edges.The dotted arrows represent precedence edges.

T_{23} interferes with T_{11}, T_{12} interferes with T_{22}, T_{22} interferes with T_{32}, and T_{32} interferes with T_{12}. It can be seen that there are several cycles in the serialization graph, indicating that the databases in the three partitions are mutually inconsistent. One of the cycles is:

$$T_{11} \text{ --> } T_{12} \text{ --> } T_{22} \text{ --> } T_{23} \text{ --> } T_{11}$$

The strategies for removing the cycles in this graph are explained in a later section in this chapter.

3.2.3. Determining the Merge Set

Let P_i (i= 1, 2, ..., n) be the n partitions being merged. Let P_{new} be the merged partition. Let P_{ncd} be the NCD (P_1, P_1, ..., P_n) in PHG(P_{new}).

The set of transactions that are committed or retained in each partition is called the *retained set (RS)* for that partition. Let $RS_i(P)$ be the set of transactions executed in the partition P (i.e., for a transaction T that is executed in a partition P, T.Pid = P.Pid) and retained (i.e., not backed out) in P_i. Obviously, $P \vdash P_i$.

The following lemma and its proof are generalizations of its counterparts from [Ma 86].

LEMMA: Assuming that the initial database state in any partition is consistent, if $T \in RS_i(P)$ where $P \neq P_{ncd}$ and $P \vdash P_{ncd}$, then T cannot be in a serialization conflict with any transaction retained in partition P_j where $j \neq i$.

PROOF: The proof straightforwardly follows because, every site in the merging partitions was in P_{ncd} (from the definition of NCD) and from the assumption that the initial logical database state in P_{ncd} is consistent.

It is possible for a transaction that reached the precommit stage but was not resolved before a partitioning occured, to be committed in one partition and aborted in another as result of the dangling-precommit algorithms discussed in [Ma 86]. The first step in determining the merge set for P_1, P_2, ... ,

P_n is to back out transactions that are retained in one partition but backed out in any other partition to achieve a consistent resolution for them.

After the backing out of transactions as described above, the union of the retained sets of all the merging partitions is formed. This union is called the *merge set.* The transactions in the merge set are the nodes of the serialization graph for the merge. The following algorithm describes how the merge set is constructed.

Let $BS_i(P)$ be the set of transactions executed in partition P and backed out in P_i ($P \vdash P_i$). Let $DEP_i(T)$ be the dependency set of a transaction T in partition P_i. In other words, $DEP_i(T)$ is the set of transactions T' such that T' is retained in P_i and there is a path of dependency edges from T to T'.

Procedure for constructing Merge Set:

let I_1 = {P such that $P_{ncd} \vdash P \vdash P_1$ and $P \in V_1$ of $PHG(P_1)$};
let I_2 = {P such that $P_{ncd} \vdash P \vdash P_2$ and $P \in V_2$ of $PHG(P_2)$};
.
.
.
let I_n = {P such that $P_{ncd} \vdash P \vdash P_n$ and $P \in V_n$ of $PHG(P_n)$};

for i = 1 to n do {

$U = \emptyset$;
for each $P \in (I_1 \cap I_2 \cap I_3 \ldots \cap I_n)$ do

$$U = U \cup (\bigcup^{n} (RS_i(P) \cap BS_j(P)))$$

```
                    j≠i

  for each T ∈ DEP_i(U) do {

        let P be such that P.Pid = T.Pid;
        add T to BS_i(P);
        remove T from RS_i(P);
  }
}
```

$$MS = \left(\bigcup_{P_{ncd} \mid\!- P}^{P \mid\!- P_1} RS_1(P) \right) \cup \left(\bigcup_{P_{ncd} \mid\!- P}^{P \mid\!- P_2} RS_2(P) \right) \cup \cdots \cup \left(\bigcup_{P_{ncd} \mid\!- P}^{P \mid\!- P_{n-1}} RS_{n-1}(P) \right) \cup \left(\bigcup_{P_{ncd} \mid\!- P}^{P \mid\!- P_n} RS_n(P) \right)$$

3.2.4. Merge Recovery Algorithm

The multiway merge recovery algorithm is a straightforward extension of the two-way merge recovery algorithm described in [Ma 86]. Instead of constructing a serialization graph of transactions from two partitions, the algorithm involves constructing a graph consisting of transactions from the N partitions that are merging. The algorithm is outlined below.

As described in the MRD protocol, the merge coordinator is the site with the highest site-id among the coordinators for the partitions participating in a merge. Let C_m denote the merge coordinator. First, C_m uses the algorithm described in Section 3.2.3 to determine the merge set from its own transaction information and the information it receives from the other coordinators participating in the merge. The information to be sent by each of the other

coordinators are:

(1) The retained transactions set (RS) and the backed out transactions set (BS).

(2) For each transaction retained in the coordinator's partition, the associated readset, writeset, the before values of the writeset and the read-from set.[1]

After finding the merge-set, the merge coordinator constructs the multiway serialization graph according to Definition 3.2 and then detects and resolves conflicts by detecting and eliminating cycles in the graph using strategies described in the next section. The rest of the algorithm is mostly as described in Section 5.7.2 of [Ma 86]. The modification is that the Update List[2] must be constructed and distributed by each coordinator to sites in all other partitions. Thus, in Ma's algorithm only two update lists are constructed — one for each partition being merged, to be distributed to the sites in the other partition. In the multiway case, if there are N partitions participating in the merge, N update lists are constructed and each list is

[1]The read-from set is used to construct a serialization order of the transactions executed in a partition. The read-from set of a transaction T contains the transactions that last modified each of the data items read by T. The read-from set is formally defined in [Ma 86].

[2]The Update List contains values for logical data items that must be broadcast by each coordinator C_i participating in a merge to the sites that were not in C_i's partition.

distributed to the sites in the N-1 other partitions being merged.

3.2.5. Graph Reduction Techniques

While eliminating cycles from the serialization graph, we must strive to minimize the number of transactions that must be backed out as a result. Absolute minimization is an NP-complete problem ([Davi 82], [Davi 84], [Wrig 83]). Hence, we must employ some heuristics that usually give good results. It would be prohibitively expensive to enumerate and then break all cycles since the total number of cycles potentially grows exponentially with the the number of nodes in the graph. Thus, we must find other more cost effective strategies for finding and eliminating all cycles. The basic idea is to break each cycle as soon as it is found; this removes several transactions and their edges from the serialization graph, potentially removing several other cycles as well. Thus, we might eliminate all cycles by breaking just a small proportion of them.

The strategies used in the simulation program to break cycles are the ones suggested in [Ma 86]. [Wrig 83], [Davi 82], [WrSk 83] and [Dave 84] provide performance studies supporting this approach. We iteratively break all 2-cycles (cycles involving only two nodes) by removing the node with lower

weight. (Each node is given a weight defined as the number of nodes connected to it via outgoing dependency edges.) If the graph is still cyclic then long cycles are found before short cycles and are broken by deleting the node with the lowest weight together with its dependency set.

In the example shown in Figure 3.2, T_{12} can be deleted since it does not have any dependents. Similarly, T_{32} can be deleted. However, since deleting T_{12} eliminates all cycles, T_{12} is the better candidate for deletion. This example is further discussed in the next chapter to illustrate how multiway merging reduces transactions back-outs.

The results of the simulation of the MRD protocol and the multiway merge recovery algorithm are presented in the next chapter.

3.3. Message Cost of Multiway Merge

The multiway reconnection algorithm requires a minimum of 4(N-1) messages to be exchanged to initiate an N way merge. This is because at least 3 messages (Merge Invitation, Merge Request and Acknowledge Merge Request) are required to include each of the N-1 other participants in the merge-info of the merge coordinator, and N-1 Prepare_To_Merge messages are sent

when the merge is started.

The original two-way reconnection algorithm [Ma 86] requires at least 3 messages to initiate a two-way merge: Merge Invitation, Merge Request and Prepare To Merge.[3] Since it takes (N-1) merges to merge N partitions 2 at a time, the number of messages required is at least 3(N-1). Thus, the multiway reconnection algorithm requires about (N-1) extra messages to be sent.

3.4. Run Time Cost of Multiway Merging

There are three major costs to performing a merge recovery:

(1) The cost to compute the merge set. To compute the merge set we compare the retained set of each partition against the backed out set of all the other partitions.

(2) The cost of constructing the serialization graph by finding the dependency, precedence, and the interference edges. Finding the interference edges takes the longest time, since the read and write sets of every retained transaction in one partition have to be compared to the read and write sets of every retained transaction in each of the

[3]Ma's RD protocol does not require the Acknowledge Merge Request message because only two partitions are involved, and sending the merge request is sufficient to initiate a merge.

other partitions.

(3) The cost to find the conflicts by finding cycles in the serialization graph and backing out some transactions and their dependents to resolve the conflicts.

If X_1 and X_2 are the number of transactions in two partitions being merged, the cost of the merge depends on $X_1 * X_2$ since every transaction of one partition has to be compared against every transaction of the other partition.

To perform two-way merges of N partitions, we need (N-1) merges. Each time, we analyze transactions from two partitions. If we assume that on an average, each partition has x transactions, the first merge involves (x & x) transactions. The second merge involves ($2x-b_1$ & x) transactions where b_1 is the number of transactions backed out as the result of the first merge. Thus in the (N-1)th merge there are ((N-1)x - B, x) transactions where B is the number of transactions backed out in the previous N-2 merges. Thus, total cost of the (N-1) merges of 2 partitions at a time depends on:

$$C(N,2)=x*x + x*(2x-b_1) + \cdots + x*((N-1)x-B)$$

$$= x^2 \left[\frac{N(N-1)}{2} - \frac{(b_1 + (b_1 + b_2) + \cdots + B)}{x} \right]$$

In an N way merge of N partitions, a total of Nx transactions is involved, x in each partition. The transactions in each partition have to be compared against the transactions from (N-1) partitions. Thus, the total cost is depends on:

$$C(N,N) = x*(N-1)x + x*(N-2)x + \cdots + x*x$$

$$= \frac{N(N-1)}{2} x^2$$

Note that the cost also depends on the actual number of dependency, precedence and interference edges between the transactions. C(N,2) appears to be lower than C(N,N). How much lower, depends on how many transactions are backed out during the two-way merges.

Thus, doing the all-way merge seems to be more expensive than doing (N-1) two-way merges. However, the time increase is not exponential with N and is acceptable in cases where multiway merging reduces the number of transaction backouts. This point is discussed further in the next chapter.

CHAPTER 4

SIMULATION AND RESULTS

We simulated the algorithms described in Chapter 3 to evaluate their performance vis-a-vis the algorithms described in [Ma 86]. In this chapter, we give an overview of the simulation program and analyze the results of a number of simulation runs.

4.1. Purpose of Simulation

The purpose of the simulation was to evaluate whether multiway merging improves upon two-way merging with respect to the percentage of transactions that are completed and the percentage of transactions that have to be backed-out.

In the MRD protocol, the merge timer controls how long a merge recovery is delayed to permit more partitions to join the merge. As the merge timer value increases, the chance of further site or link recoveries increases and hence the number of partitions that participate in a merge increases. Our expectation was that there is a trade-off between the benefits

of increased degree of merging and the increased conflicts due to waiting longer before initiating a merge.[1] Our investigation focussed on finding an optimal value for the merge timer, i.e., the value that yields the least percentage of backed-out transactions or highest percentage of completed transactions.

4.2. Description of the Simulation Program

The simulator is a generalized version of Bahra's simulator [Bahr 87] for optimistic partitioned operation of DDBS as described in [Ma 86]. We rewrote major portions of the simulator to incorporate the multiway reconnection detection protocol and the multiway merge recovery algorithm. [Bahr 87] presents a detailed description of the simulator program structure, data structures and the algorithms. Here we present an overview of the modified simulation program.

The program simulates optimistic partitioned operation and multiway merge recovery in distributed database system. It is an event driven simulation program. Every action of the distributed database system is

[1]This trade-off is described in more detail in section 4.5.

represented by an event. Some of the events are: transaction arrival, link failure, link recovery.

The events are processed chronologically and new events are generated as necessary during execution by the program. The time when an event occurs is generated randomly, except for events that need to occur at fixed or predetermined times as dictated by the PD and the MRD protocols. The mean values are different for each random event depending upon the nature of the event. These mean values can be changed to simulate different environments for the distributed system.

The simulator consists of four major modules:

(1) *PD protocol*: This module is a complete implementation of the events and messages that comprise the partition detection protocol.

(2) *MRD protocol*: This module is a complete implementation of the multiway reconnection detection protocol described in the previous chapter.

(3) *Transaction Execution*: This module simulates the events such as locking, commit and abort that are associated with transaction execution.

(4) *Merge Recovery*: This module is a faithful implementation of the multiway merge recovery algorithm. Determination of the merge set, construction of the serialization graph, and cycle detection and elimination are all implemented by this module.

Thus, only network partitioning and transaction execution are simulated. The PD and MRD protocols and the merge recovery algorithms are implemented and run as described in the previous chapters.

4.3. Assumptions

To simplify the simulation program the following assumptions are made.

• *Communication subsystem*:

(1) The network is 'point-to-point', i.e., the sites are connected pairwise and not by a bus-like link such as Ethernet.

(2) If a receiver A receives two messages from a sender B, then the messages are received in the same order in which they are sent.

(3) The messages arriving at any site are uncorrupted.

(4) The network partitionings and reconnections are discrete, i.e., there is a non-zero interval between any two partitionings, any two

reconnections, and between a partitioning and a reconnection.

(5) All the sites within a partition agree on the set of available sites.

(6) There are no failures during recovery.

(7) The time delay for communication between any two sites depends upon the relative distance between the sites. For sites closer together the communication time is less than for sites which are farther apart. The relative distance is measured in terms of the number of sites a message must pass through before it reaches its final destination.

- *Database*:

(1) The database is fully replicated.

(2) The writeset of a transaction is a subset of its readset i.e., a transaction has to read a data item before it can write or update the item.

4.4. Simulation Parameters

The parameters given below can be changed to study the behavior of the transaction execution and transaction back-out in the system.

— number of sites.

— database size.

— transaction arrival rate.

— READSET/WRITESET size.

— time between link failure per link.

— link recovery time per link.

— time between site failure.

— site recovery time.

— merge wait time.

Not all the parameters mentioned above have a strong influence on the performance of two-way or multiway merging. For instance, site-failure interval and site-recovery interval have no effect on our investigation, since no transactions are backed-out as the result of a recovering site rejoining the network. Further, early simulation runs indicated that the number of sites, the transaction arrival rate, and the link failure and recovery intervals do not have much impact on the pattern of the results.

The probability of conflicts (as indicated by the interference edges in a serialization graph) between transactions run in the different partitions participating in the merge has the greatest impact. Conflicts increase as the

ratio of the writeset size to the readset size of transactions increases. Conflicts decrease if the percentage of read-only transactions increases. Conflicts increase if the sizes of the readset and writeset increase in proportion to the database size.

Hence, we concentrated on varying the merge timer values for different combinations of the following parameters:

- database size
- Readset/Writeset size
- percentage of read-only transactions

The degree of merge is varied automatically as the merge timer value is varied. As the merge timer value is increased, the merge recovery is delayed for a longer time and more links recover before the merge recovery is initiated, leading to more partitions participating in the merge.

4.5. Merge Wait Timer and Merge Degree Trade-Off

We now describe the trade-off between the expected benefits of an increased degree of merging and the expected increase in conflicts due to waiting longer before completing a merge.

In optimistic partitioned operation, transactions executed in different partitions potentially conflict with each other. These conflicts are represented by cycles involving interference edges in the serialization graph built during the merge recovery. As described earlier, the cycles must be eliminated by backing out some transactions to derive a global serialization order for the transactions.

4.5.1. Effect of Multiway Merge on Back-outs

As the degree of merge increases the number of transactions that are backed-out tends to decrease. This effect can best be explained by means of an example. Suppose a distributed database is divided into the partitions P_1, P_2, P_3 and as reconnection takes place, they can be merged two at a time, or all three at once. Further suppose transactions T_1, T_2, and T_3 are executed in the partitions P_1, P_2, and P_3 respectively. T_1 conflicts with T_2 and T_3. There are no other conflicts.

Suppose P_1 is first merged with P_2. The cycle elimination algorithm might back out T_2 since it conflicts with T_1. Later when P_3 is merged with the merged partition, either T_1 or T_3 has to be backed-out to eliminate the

cycle in the serialization graph for the merge. However, if all three partitions are merged at once, the cycle elimination algorithm can detect that backing out T_1 is sufficient to remove all cycles from the serialization graph. Thus, the number of backed-out transactions has been reduced due to the increased visibility of interference provided by the increased degree of merge. This aspect is hereafter referred to as *visibility*.

In the example shown in Figure 3.2 also, we can see how multiway merging improves upon two way merging. If P_1 and P_3 merge first and then the resulting partition merges with P_2, the two two-way merge recoveries might result in both T_{32} and T_{12} being backed-out. However, when all three partitions are merged together, the merge recovery algorithm can detect that backing out T_{12} removes all cycles, thereby saving T_{32}.

4.5.2. Effect of Waiting Longer on Back-outs

The longer the partitions remain separate, the greater the number of transactions that are executed independently in the partitions. As the number of transactions in partitioned operation increases, the possibility of conflicts between transactions executed in different partitions increases. This

increase is because there is a greater chance of overlap in the data items accessed and modified in the different partitions. Increased conflicts result in more cycles in the serialization graph and, in general, result in a greater number of transactions being backed-out to eliminate cycles in the graph.

Since increasing the merge timer value delays the merging of partitions, it tends to increase the number of transactions that are backed-out.

4.5.3. Simulation Focus

From the above discussion, we can see that increasing the merge timer value yields two conflicting effects. It potentially increases the degree of merge by delaying the merge to allow more partitions to be reconnected; this effect would tend to reduce the number of transactions that are backed-out. At the same time, it also tends to increase the back out rate due to increased conflicts.

Our goal was to find out how the variation of the merge timer value would affect the overall back out rate, and whether there are some optimal values that minimize the back out rate. We performed several sets of simulation runs to study the behavior. For each set of runs, all the simulation parameters, except the merge timer value were set to the same

value. Different sets of runs were obtained by varying the level of conflicts by changing the database size, readset size, the writeset-to-readset ratio, and the percentage of read-only transactions. In the following sections, we present the results of the different simulation runs and then summarize the results.

4.6. Discussion of Results

4.6.1. Partitioning and Reconnection Profile

The database is fully connected at the beginning and at the end. As simulation progresses, partitions are formed via link failures and are reconnected after link recoveries. All the cases given below have the same link failure mean-time and link recovery mean-time. The link failures and recoveries are distributed randomly in the simulation.

The constant parameters for the simulation were:

Number of Sites = 8

Total simulation time = 65

Link Failure mean = 25

Link Recovery mean = 15

The number of sites was chosen to be 8 for the following reasons:

(1) Choosing a smaller number would limit the degree of merge to too small a number, limiting the comparisons we could make on the variation of the back-out rate with the merge degree.

(2) Choosing a larger number would have increased the simulation time without yielding much additional information.

The link failure mean and the link recovery mean were chosen after many trials to provide a non-trivial combination of the partitionings and reconnections. Some of the criteria used were:

(1) The number of sites in partitions should vary, e.g., not all should be single site partitions for every merge.

(2) It should not always be the case that partitions that are isolated earlier are reconnected earlier.

(3) It should be possible to increase the degree of the merge up to the maximum possible (8 in this case) by increasing the merge wait timer value appropriately.

The merge-degree (the number of partitions that participate in a merge) for successive merges in a particular run for different merge wait timer values are as follows[2]:

merge-timer	merge-degrees of successive merges
2	2, 2, 2, 2, 2, 2, 2.
4	2, 2, 3, 2, 2, 2.
6	2, 2, 3, 2, 3.
8	2, 2, 4, 3.
10	2, 2, 4, 3.
12	2, 2, 6.
14	2, 2, 6.
16	3, 4, 3.
18	3, 3, 4.
20	3, 2, 5.
22	7, 2.
24	7, 2.
26	8.
28	8.

The partition history graph for the merge timer value 14 is shown in figure 4.1 as an example:

[2]For instance, in the following table, for the run with a merge timer value of 14, there are 2 two-way merges followed by a six-way merge.

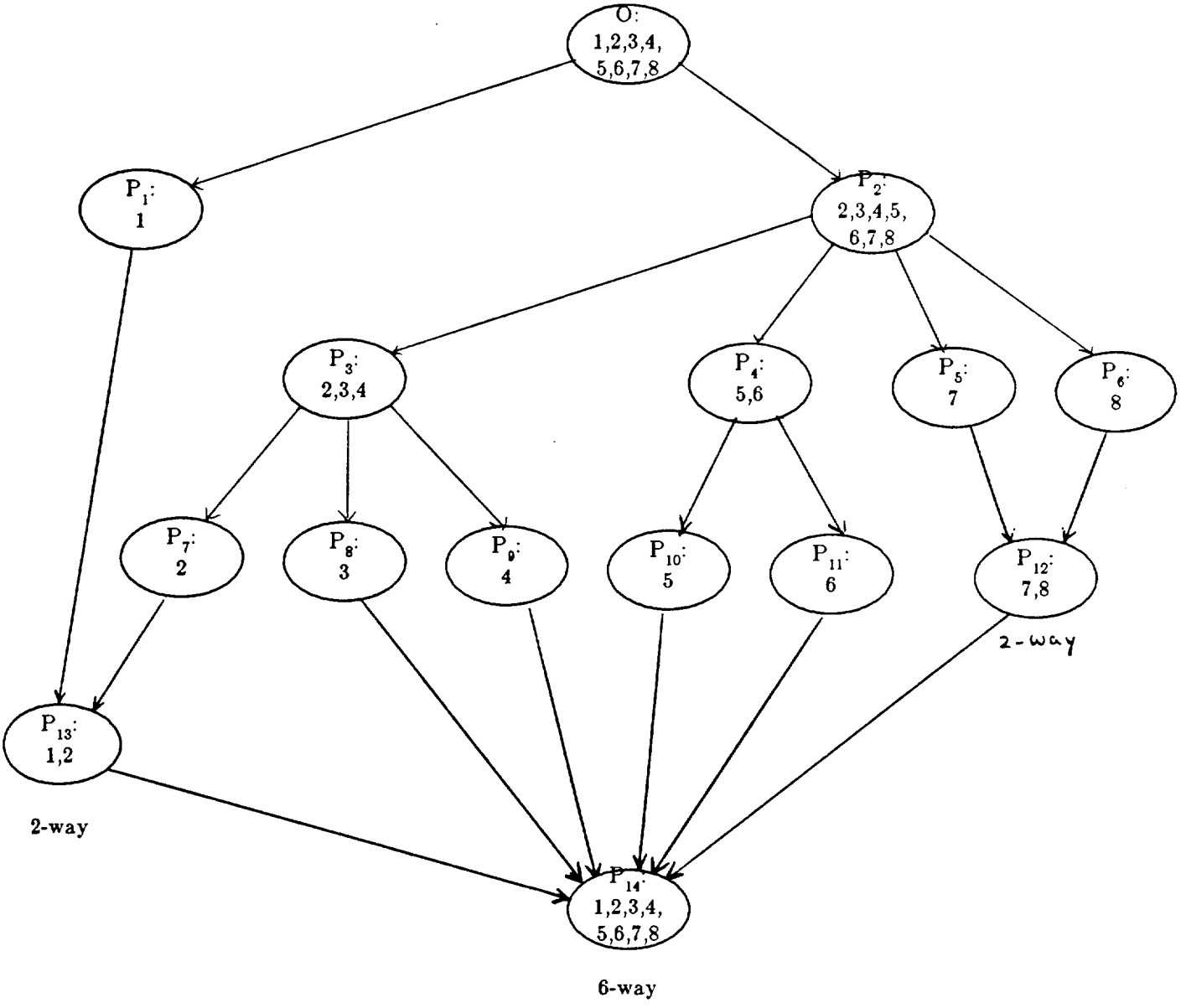

O:
1,2,3,4,
5,6,7,8
P1:
1
P2:
2,3,4,5,
6,7,8
P3:
2,3,4
P4:
5,6
P5:
7
P6:
8
P7:
2
P8:
3
P9:
4
P10:
5
P11:
6
P12:
7,8
2-way
P13:
1,2
2-way
P14:
1,2,3,4,
5,6,7,8
6-way

Figure 4.1

4.6.2. Result Sets

The tables and graphs in Appendix A are representative of the simulation runs we performed. Some runs were repeated multiple times with different seeds for the random number generator. These runs are gathered together, and the mean values are shown in a separate table and graph. An analysis of the results presented in Appendix A is presented in the next section.

4.6.3. Analysis of Simulation Results

In the following discussion, we refer to the tables and graphs in Appendix A. The result sets are grouped into three major categories. The value of the variable parameters for each category are indicated.

Category 1: Large Database, Small Read and Write Sets:

DBSIZE = 1000;

Number of data-items accessed and modified = 5 to 20[1];

[1]The transactions are divided into 4 sets, each containing about 25% of the total transactions. The sets have 5, 10, 15 and 20 items respectively in their readset. To which set a particular transaction belongs is determined using a random number.

Thus only 0.5 to 2% of the items are accessed by a transaction and, hence, there is a relatively low possibility of conflicts. This case is represented by the Tables 4.1 through 4.5.

(1)(a) Low write-set percentage (20%, 40%, 60%):

When the write-set of transactions is a relatively small percentage (20%, 40%, 60%) of the read-set, represented by Tables 4.1 to 4.3, there are two optimal points that result in the least percentage of transactions being backed-out. The primary one, with 26% back-out rate, is when the merge wait time is such that the degree of merge is 3 or 4. For the above set of link failure and recovery parameters, the merge wait time is about 8 to 10. The secondary optimal point, with 28% back-out rate, is when all the partitions merge together. In this simulation set, this point is represented by the merge wait time of 26 to 28.

The primary optimal point is reached because initially the increased visibility (over the two-way case) of transactions that run in different partitions reduces the back-out rate. After that, with increasing merge wait time, conflicts increase as the number of transactions that are processed in the partitioned state increases. This leads to increasing number of backed-out

transactions, peaking at the merge wait time of 22 to 24, when a 7 way followed by a two-way merge is performed to reunite all the partitions. After that, we reach the case where the merge wait time is long enough for all the partitions to merge together. In this case, there is maximum visibility over the conflicts between transactions. The increased visibility counteracts the effects of the increased conflicts, thereby greatly reducing the number of backed-out transactions and thus results in the secondary optimal point.

(1)(b) High Write Set Percentage (80%, 100%):

When the write-set is a high percentage of the read-set, the conflicts between transactions are very high. In this case, the back-out rate keeps on increasing as the merge wait time increases. This category is represented by Tables 4.4 and 4.5. The increased visibility provided by merging multiple transactions cannot adequately compensate for the higher rate of increase of conflicts. Even the complete visibility provided by merging all the partitions fails to be as effective as in the lower conflict cases described (a). Thus, in these cases it is better to proceed with a merge as soon as two partitions merge.

Category 2: Larger Read and Write Set Size:

DBSIZE = 200; Number of data items = 5 to 20;

Thus, the number of data items is 2.5% to 10% of the database. This percentage is larger than the previous case, and hence causing more conflicts. This case is represented by the tables 4.6 through 4.9.

All these cases generate high conflicts amongst the transactions. The increase in conflicts results in the back-out rate increasing as the merge wait time increases. Thus in this case, as in case (1b), it is better to merge as soon as two partitions merge.

Category 3: 50% Read Only transactions plus 50% update transactions:

DBSIZE = 1000; Number of data items = 5 to 20;

Since half the transactions are read only, the conflicts are even lower than case (1). This case is represented by the tables 4.11 through 4.15.

In this case, when the write-sets of the update transactions is a low percentage of the read-sets (20%, 40%), the conflicts are very low. Thus, the behavior described in (1a) holds except that the merge wait time of 2 yields fewer back-outs than the all-merge case (merge wait time = 26). The reduction in back-outs might be because with so many read-only transactions, the

two-way merge sees much fewer conflicts and having a long merge wait time increases conflicts to the level where the all way merge cannot compensate enough.

However, for write-set percentages of 60, 80, and 100, it is clear that we should merge as soon as possible since the back-out rate keeps increasing as the merge wait time increases.

Testing for Statistical Significance of Results:

To validate that the difference between the mean values of the back-out rates for the low conflict cases and the high conflict cases is statistically significant, we performed the *two tailed pairwise t-test* [Mid 76] on a representative selection from the simulation results. The details of the test are presented in Appendix B. This analysis indicates that we can have high levels of confidence in the effect of varying the merge wait timer value, the size of database, and the writeset percentage. For writeset percentage of 60 and 100 however, we note that the null hypothesis cannot be rejected with confidence. This suggests that the high conflict cases might yield back-out rates that are close to each other.

Testing the Goodness of the Random Number Generator:

It was indicated earlier that the sequence of events that are simulated is generated based on random numbers. The size of the readset and the particular data items accessed by a particular transaction are also determined using random numbers.

The simulator generates the random numbers using the Berkeley Pascal random number generator function, *random*. This generator is a linear congruential random number generator. To test the randomness of the random number sequence, we performed the *maximum-of-t/Kolmogorov-Smirnov* [Knu 69] test on a sequence of 10000 random numbers generated using the generator mentioned above.

The test was performed as follows: The 10000 numbers were divided into sets of 5 each (thus the test was maximum-of-t with t = 5). The maximum value of each set was determined, giving 2000 values. These 2000 maximum values were divided into sets of 20, giving 100 sets. Each set was sorted and the values K_n^+ and K_n^- were computed as given on page 49 of [Knu 69]. The resulting values of K_n^+ and K_n^- were found to be between the 25th and 75th percentile of the table shown in page 48 of [Knu 69], thus indicating that the random number generator we used generates a sequence of numbers with satisfactory randomness.

4.7. Simulation Results Summary

As described earlier, there are two counteracting effects:

(1) Increasing the merge wait time increases the number of conflicts and hence the number of back-outs.

(2) Increasing the merge wait time increases the degree of merge, which reduces the back-outs due to the increased visibility of conflicts.

To what extent the second effect overcomes the first effect determines whether or not we should wait for a multi-way merge and for how long. The outcomes in the particular cases have been explained above.

In all the cases, it can be observed that merging seven-way and then two-way yields the maximum back-out rate. This increase in back-outs is because this case suffers the maximum conflicts and yet does not have the benefit of the complete visibility of an all-way merge. We can clearly see that the all-way merge cases always have fewer back-outs than the seven-way plus two-way merge cases even though their merge wait time is longer. This effect shows that increased visibility does reduce back-outs.

CHAPTER 5

CONCLUSIONS

This chapter summarizes our research, states our conclusions and suggests directions for further research.

5.1. Summary of Thesis

This thesis is concerned with optimistic partitioned operation of distributed databases. We have generalized the algorithms in [Ma 86] to permit reconnections and merge recovery of more than two partitions at a time. We have presented the results of our simulation of the multiway algorithms and compared the merging message overhead and runtime costs of two-way vs. multiway merging.

The results of our simulation show that, when the conflict between transactions in different partitions is low, multiway merging does indeed improve upon two-way merging by reducing the number of transactions that are backed out. The reason for low conflicts could be a combination of: (i) a small percentage of data items being accessed by each transaction, or (ii) write-set being a small subset of the read-set, or both. When the conflicts

are high, it is better to merge partitions as soon as they are reconnected.

We found that the benefits of multiway merging come at the cost of increased message overhead and longer time required to carry out the merge recovery. Since the sites participating in the merge recovery do not process transaction for the duration of merge recovery, we can say that multiway merging leads to reduced availability. Thus, whether or not multiway merging is desirable might depend upon whether reduced transaction back-out rate or increased availability is considered to be the more important goal for a given distributed database installation.

It is difficult to give a specific recommendation as to how long a merge should be delayed to allow more partitions to be reconnected, since that would depend upon the actual link failure and recovery intervals in a specific network. It might be possible to derive such a recommendation if the study were to be conducted with data from a real distributed database.

5.2. Future Research

It would be interesting to investigate the advantages and disadvantages of multiway merging in the context of a real distributed database. It may then be possible to derive realistic estimates of the costs and to find a

specific optimal value for the *merge_wait_timer* for that database, given its mean link-failure and link-recovery intervals.

In the MRD protocol, the merge recovery is not performed if a failure occurs before the merge is initiated. It is possible to add messages to confirm the participation of partitions that have not suffered a failure and proceed with the merge recovery for only those partitions. We have not incorporated this feature since it seems to add more complexity to the protocol. If future studies show it to be worthwhile, the MRD protocol can be enhanced to recover from pre-merge failures.

The reconnection and merge recovery algorithms could be modified so that some of the actions taken during merge recovery are done as soon as two partitions decide to merge. For instance, the merge-set computation during merge recovery compares the backed-out and retained transaction sets of each partition. This task can be performed during reconnection itself so that the merge recovery algorithm takes less time.

APPENDIX A

SIMULATION RESULTS

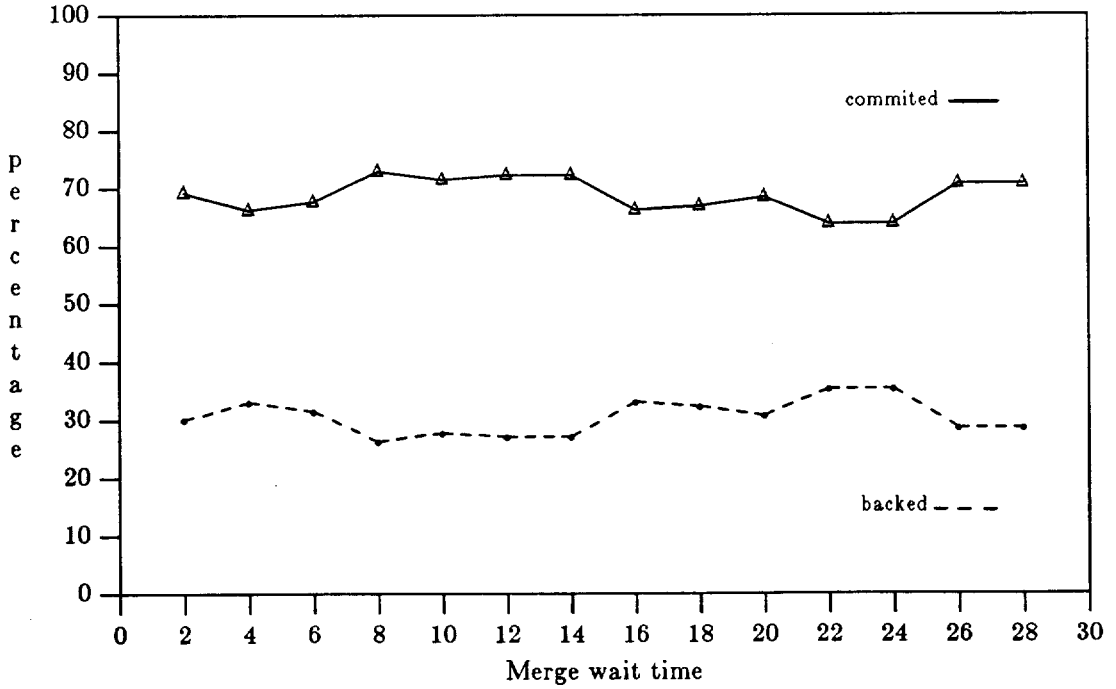
The following tables and graphs are representative of the simulation runs we performed. All the rows in a table represent the same transactions, readsets and writesets — only the merge wait timer value is different for different rows of a table. In each table, the first column gives the merge wait timer values. The other columns give the number of committed, aborted[1] and backed out transactions, and their percentages of the total number of transactions.

[1]Transactions can be aborted either because they cannot obtain some required locks, or because they are in progress when a partitioning occurs.

Database Size = 1000 items. Seed = 7774755
Number of Data items read = 5 to 20.
Percentage of Write Set = 20 % of read set.

Table 4.1.0

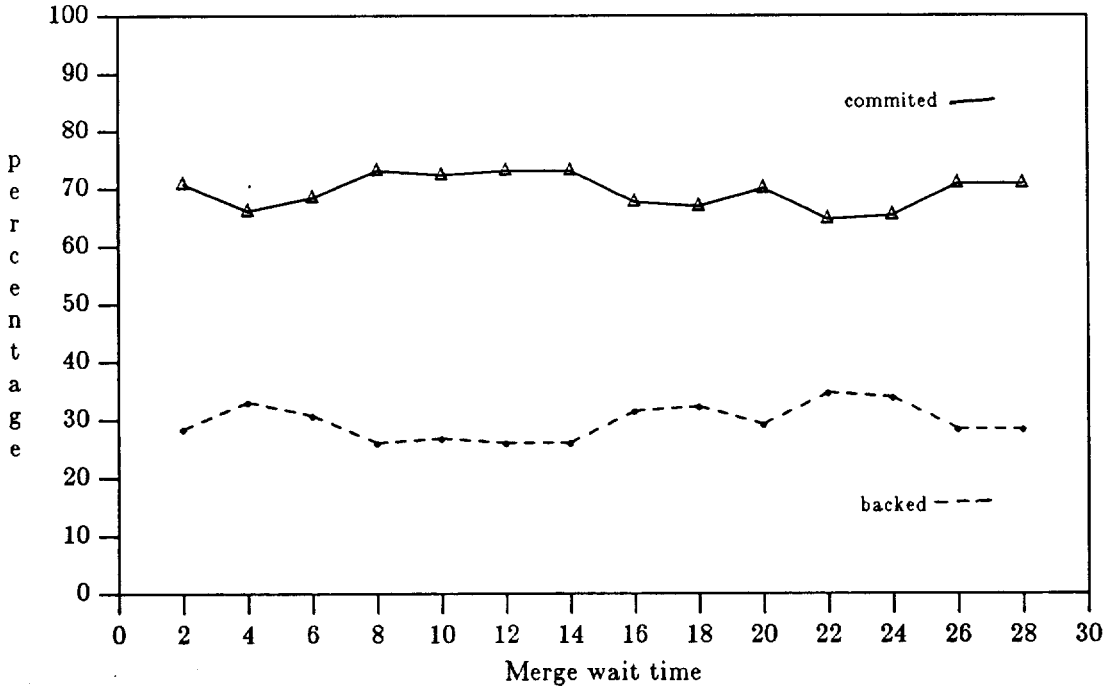
merge wait time	committed transactions	commit%	aborted transactions	abort%	backed out transactions	backed out%	total transactions
2	92	69.17	1	0.75	40	30.08	133
4	88	66.17	1	0.75	44	33.08	133
6	90	67.67	1	0.75	42	31.58	133
8	97	72.93	1	0.75	35	26.32	133
10	95	71.43	1	0.75	37	27.82	133
12	96	72.18	1	0.75	36	27.07	133
14	96	72.18	1	0.75	36	27.07	133
16	88	66.17	1	0.75	44	33.08	133
18	89	66.92	1	0.75	43	32.33	133
20	91	68.42	1	0.75	41	30.83	133
22	85	63.91	1	0.75	47	35.34	133
24	85	63.91	1	0.75	47	35.34	133
26	94	70.68	1	0.75	38	28.57	133
28	94	70.68	1	0.75	38	28.57	133



Database Size = 1000 items. Seed = 56743
Number of Data items read = 5 to 20.
Percentage of Write Set = 20 % of read set.

Table 4.1.1

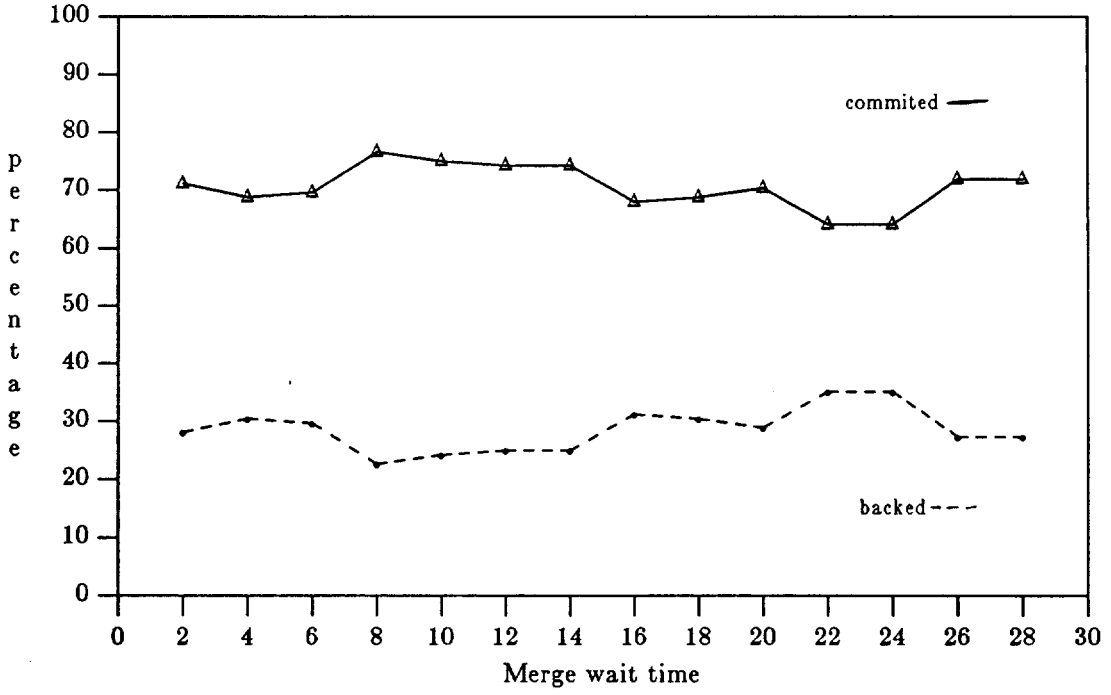
merge wait time	committed transactions	commit%	aborted transactions	abort%	backed out transactions	backed out%	total transactions
2	92	70.77	1	0.77	37	28.46	130
4	86	66.15	1	0.77	43	33.08	130
6	89	68.46	1	0.77	40	30.77	130
8	95	73.08	1	0.77	34	26.15	130
10	94	72.31	1	0.77	35	26.92	130
12	95	73.08	1	0.77	34	26.15	130
14	95	73.08	1	0.77	34	26.15	130
16	88	67.69	1	0.77	41	31.54	130
18	87	66.92	1	0.77	42	32.31	130
20	91	70.00	1	0.77	38	29.23	130
22	84	64.62	1	0.77	45	34.62	130
24	85	65.38	1	0.77	44	33.85	130
26	92	70.77	1	0.77	37	28.46	130
28	92	70.77	1	0.77	37	28.46	130



Database Size = 1000 items. Seed = 101489
Number of Data items read = 5 to 20.
Percentage of Write Set = 20 % of read set.

Table 4.1.2

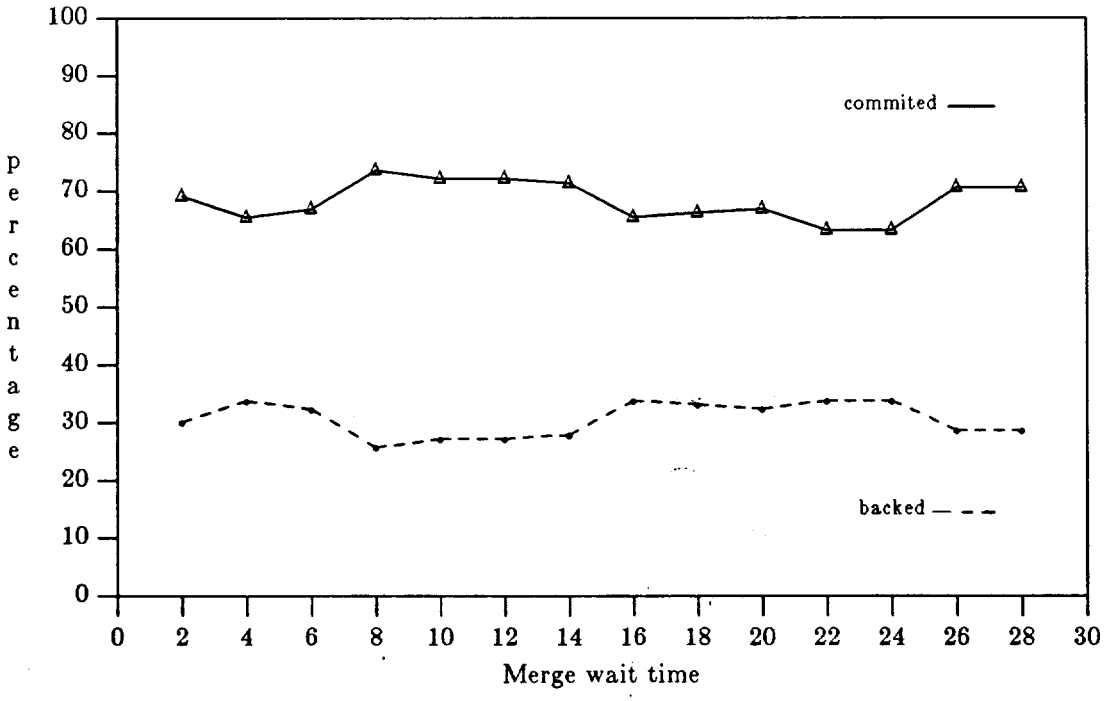
merge wait time	committed transactions	commit%	aborted transactions	abort%	backed out transactions	backed out%	total transactions
2	91	71.09	1	0.78	36	28.13	128
4	88	68.75	1	0.78	39	30.47	128
6	89	69.53	1	0.78	38	29.69	128
8	98	76.56	1	0.78	29	22.66	128
10	96	75.00	1	0.78	31	24.22	128
12	95	74.22	1	0.78	32	25.00	128
14	95	74.22	1	0.78	32	25.00	128
16	87	67.97	1	0.78	40	31.25	128
18	88	68.75	1	0.78	39	30.47	128
20	90	70.31	1	0.78	37	28.91	128
22	82	64.06	1	0.78	45	35.16	128
24	82	64.06	1	0.78	45	35.16	128
26	92	71.88	1	0.78	35	27.34	128
28	92	71.88	1	0.78	35	27.34	128



Database Size = 1000 items. Seed = 4235761
Number of Data items read = 5 to 20.
Percentage of Write Set = 20 % of read set.

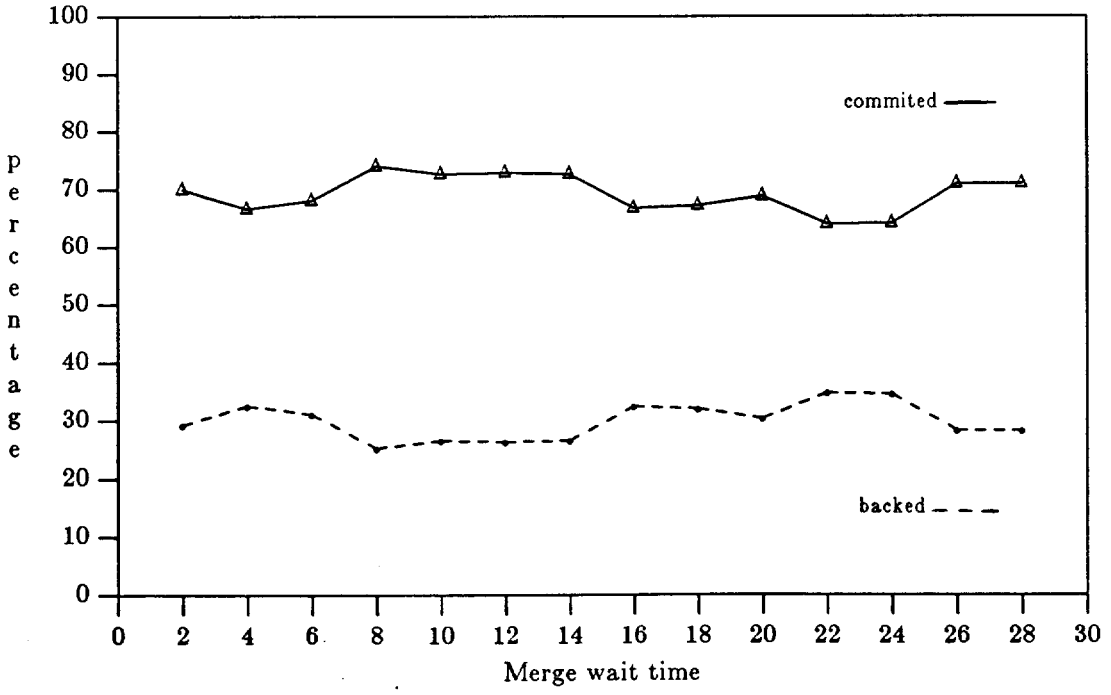
Table 4.1.3

merge wait time	committed transactions	commit%	aborted transactions	abort%	backed out transactions	backed out%	total transactions
2	94	69.12	1	0.74	41	30.15	136
4	89	65.44	1	0.74	46	33.82	136
6	91	66.91	1	0.74	44	32.35	136
8	100	73.53	1	0.74	35	25.74	136
10	98	72.06	1	0.74	37	27.21	136
12	98	72.06	1	0.74	37	27.21	136
14	97	71.32	1	0.74	38	27.94	136
16	89	65.44	1	0.74	46	33.82	136
18	90	66.18	1	0.74	45	33.09	136
20	91	66.91	1	0.74	44	32.35	136
22	86	63.24	2	1.47	46	33.82	136
24	86	63.24	2	1.47	46	33.82	136
26	96	70.59	1	0.74	39	28.68	136
28	96	70.59	1	0.74	39	28.68	136



Database Size = 1000 items.
Number of Data items read = 5 to 20.
Percentage of Write Set = 20 % of read set.

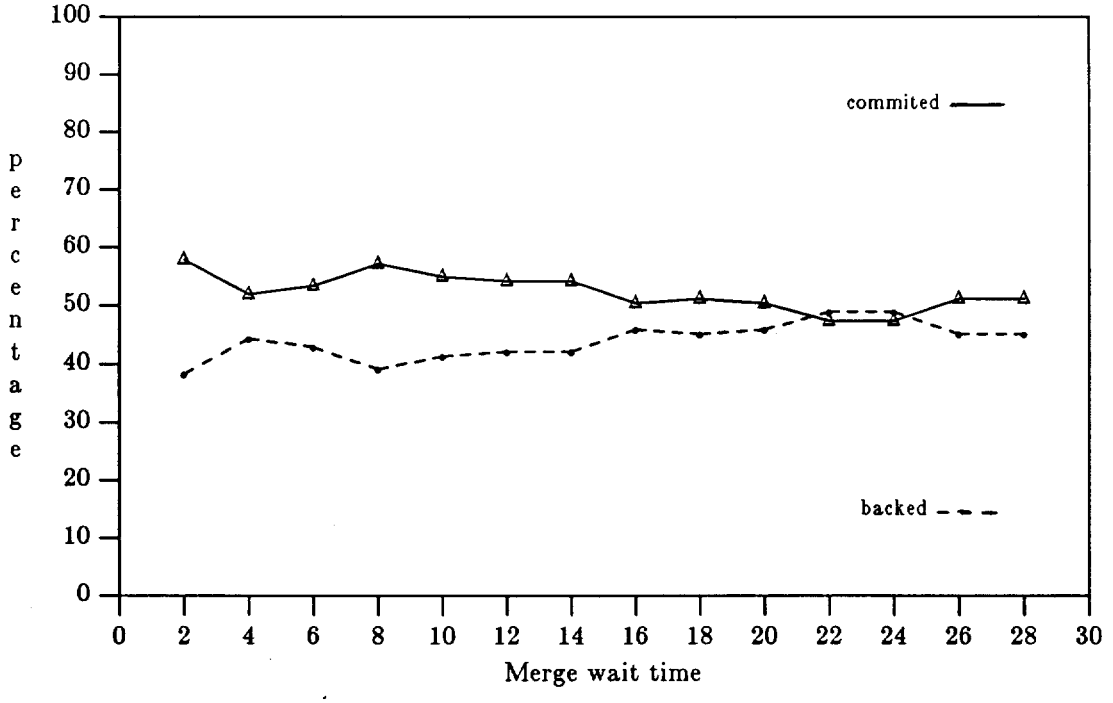
Mean Table - 4.1			
merge wait time	commit%	abort%	backed%
2	70.04	0.76	29.20
4	66.63	0.76	32.61
6	68.14	0.76	31.10
8	74.03	0.76	25.22
10	72.70	0.76	26.54
12	72.89	0.76	26.36
14	72.70	0.76	26.54
16	66.82	0.76	32.42
18	67.19	0.76	32.05
20	68.91	0.76	30.33
22	63.96	0.94	34.74
24	64.15	0.94	34.54
26	70.98	0.76	28.26
28	70.98	0.76	28.26



Database Size = 1000 items. Seed = 7774755
Number of Data items read = 5 to 20.
Percentage of Write Set = 40 % of read set.

Table 4.2

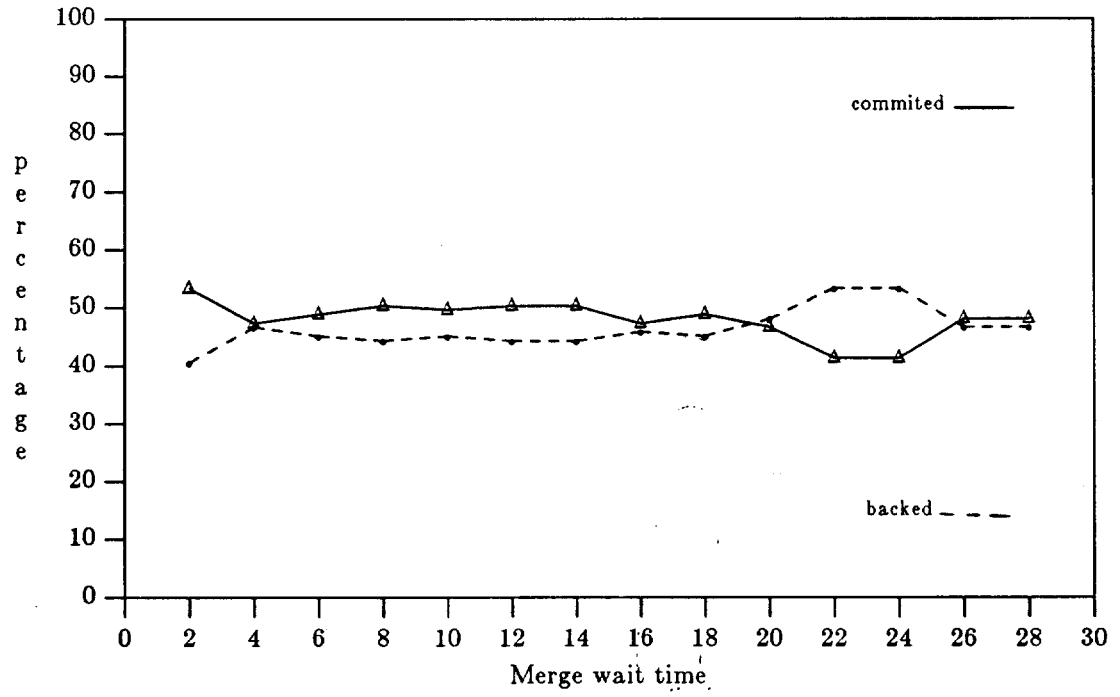
merge wait time	committed transactions	commit%	aborted transactions	abort%	backed out transactions	backed out%	total transactions
2	77	57.89	5	3.76	51	38.35	133
4	69	51.88	5	3.76	59	44.36	133
6	71	53.38	5	3.76	57	42.86	133
8	76	57.14	5	3.76	52	39.10	133
10	73	54.89	5	3.76	55	41.35	133
12	72	54.14	5	3.76	56	42.11	133
14	72	54.14	5	3.76	56	42.11	133
16	67	50.38	5	3.76	61	45.86	133
18	68	51.13	5	3.76	60	45.11	133
20	67	50.38	5	3.76	61	45.86	133
22	63	47.37	5	3.76	65	48.87	133
24	63	47.37	5	3.76	65	48.87	133
26	68	51.13	5	3.76	60	45.11	133
28	68	51.13	5	3.76	60	45.11	133



Database Size = 1000 items. Seed = 7774755
Number of Data items read = 5 to 20.
Percentage of Write Set = 60 % of read set.

Table 4.3.0

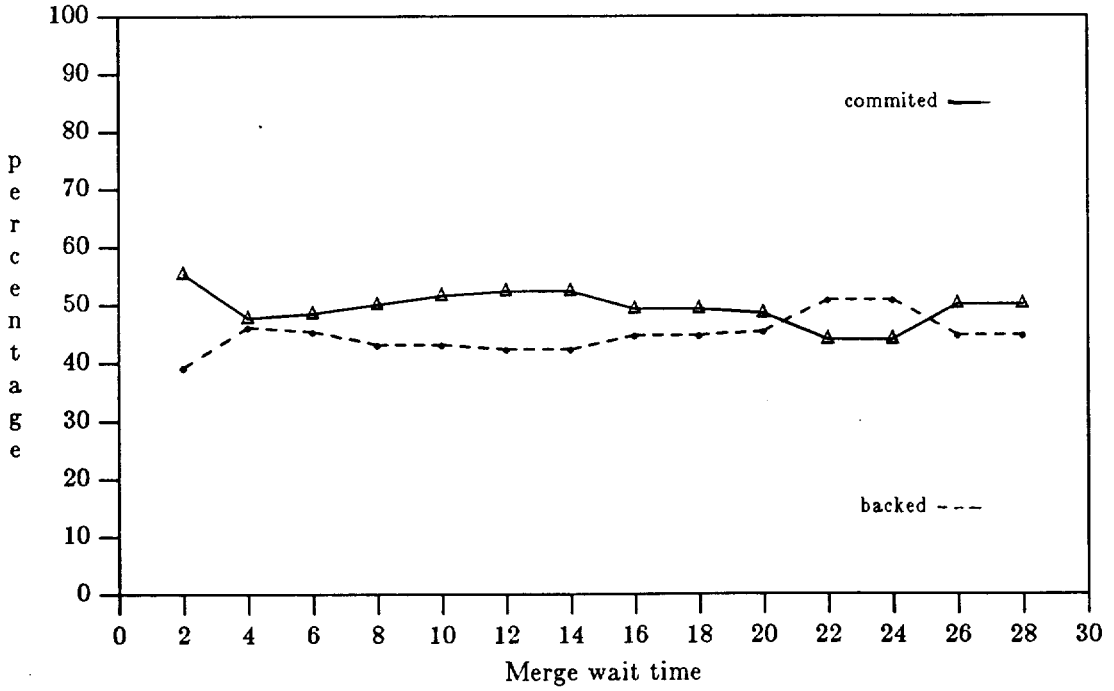
merge wait time	committed transactions	commit%	aborted transactions	abort%	backed out transactions	backed out%	total transactions
2	71	53.38	8	6.02	54	40.60	133
4	63	47.37	8	6.02	62	46.62	133
6	65	48.87	8	6.02	60	45.11	133
8	67	50.38	7	5.26	59	44.36	133
10	66	49.62	7	5.26	60	45.11	133
12	67	50.38	7	5.26	59	44.36	133
14	67	50.38	7	5.26	59	44.36	133
16	63	47.37	9	6.77	61	45.86	133
18	65	48.87	8	6.02	60	45.11	133
20	62	46.62	7	5.26	64	48.12	133
22	55	41.35	7	5.26	71	53.38	133
24	55	41.35	7	5.26	71	53.38	133
26	64	48.12	7	5.26	62	46.62	133
28	64	48.12	7	5.26	62	46.62	133



Database Size = 1000 items. Seed = 56743
Number of Data items read = 5 to 20.
Percentage of Write Set = 60 % of read set.

Table 4.3.1

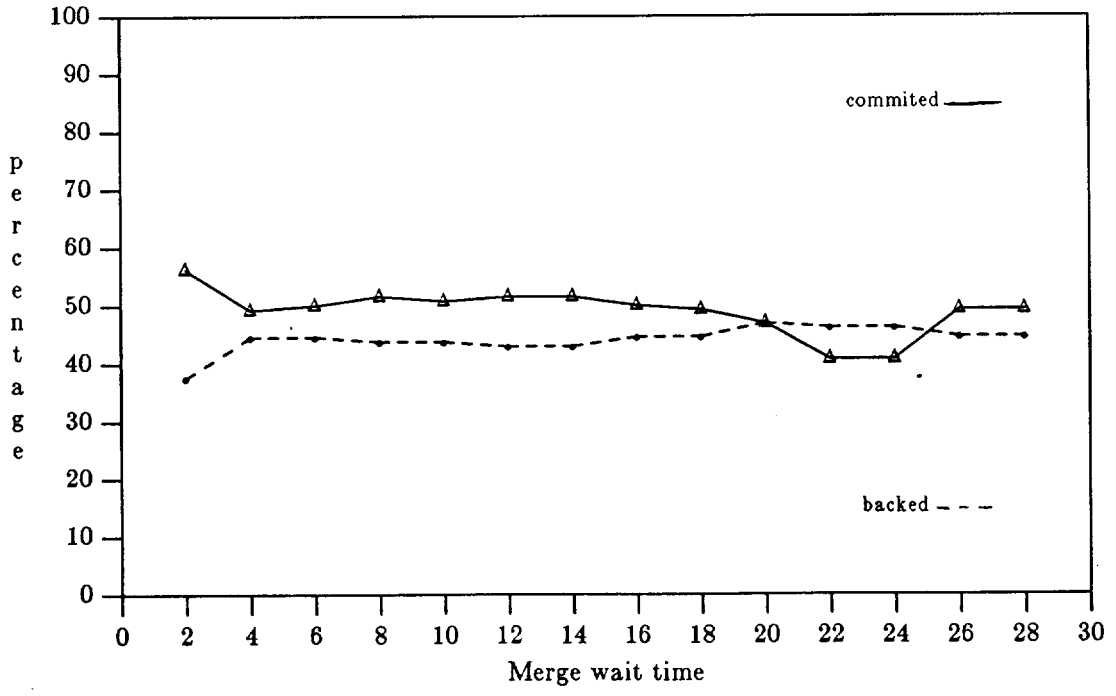
merge wait time	committed transactions	commit%	aborted transactions	abort%	backed out transactions	backed out%	total transactions
2	72	55.38	7	5.38	51	39.23	130
4	62	47.69	8	6.15	60	46.15	130
6	63	48.46	8	6.15	59	45.38	130
8	65	50.00	9	6.92	56	43.08	130
10	67	51.54	7	5.38	56	43.08	130
12	68	52.31	7	5.38	55	42.31	130
14	68	52.31	7	5.38	55	42.31	130
16	64	49.23	8	6.15	58	44.62	130
18	64	49.23	8	6.15	58	44.62	130
20	63	48.46	9	6.92	59	45.38	130
22	57	43.85	7	5.38	66	50.77	130
24	57	43.85	7	5.38	66	50.77	130
26	65	50.00	7	5.38	58	44.62	130
28	65	50.00	7	5.38	58	44.62	130



Database Size = 1000 items. Seed = 101489
Number of Data items read = 5 to 20.
Percentage of Write Set = 60 % of read set.

Table 4.3.2

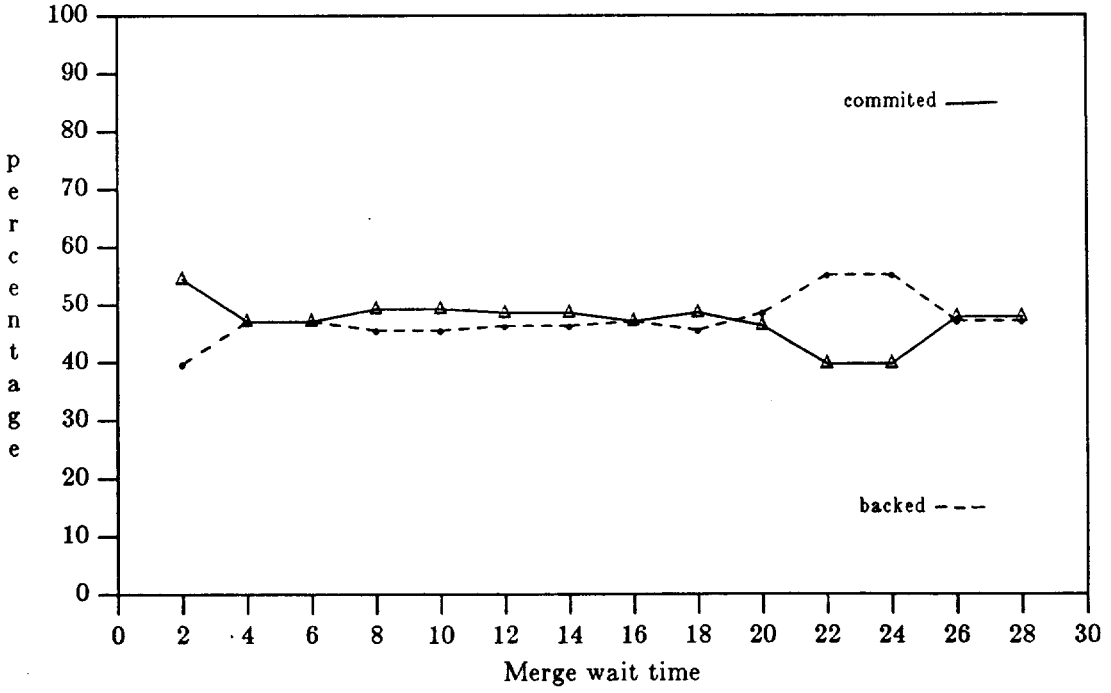
merge wait time	committed transactions	commit%	aborted transactions	abort%	backed out transactions	backed out%	total transactions
2	72	56.25	8	6.25	48	37.50	128
4	63	49.22	8	6.25	57	44.53	128
6	64	50.00	7	5.47	57	44.53	128
8	66	51.56	6	4.69	56	43.75	128
10	65	50.78	7	5.47	56	43.75	128
12	66	51.56	7	5.47	55	42.97	128
14	66	51.56	7	5.47	55	42.97	128
16	64	50.00	7	5.47	57	44.53	128
18	63	49.22	8	6.25	57	44.53	128
20	60	46.88	8	6.25	60	46.88	128
22	52	40.63	7	5.47	59	46.09	128
24	52	40.63	7	5.47	59	46.09	128
26	63	49.22	8	6.25	57	44.53	128
28	63	49.22	8	6.25	57	44.53	128



Database Size = 1000 items. Seed = 4235761
Number of Data items read = 5 to 20.
Percentage of Write Set = 60 % of read set.

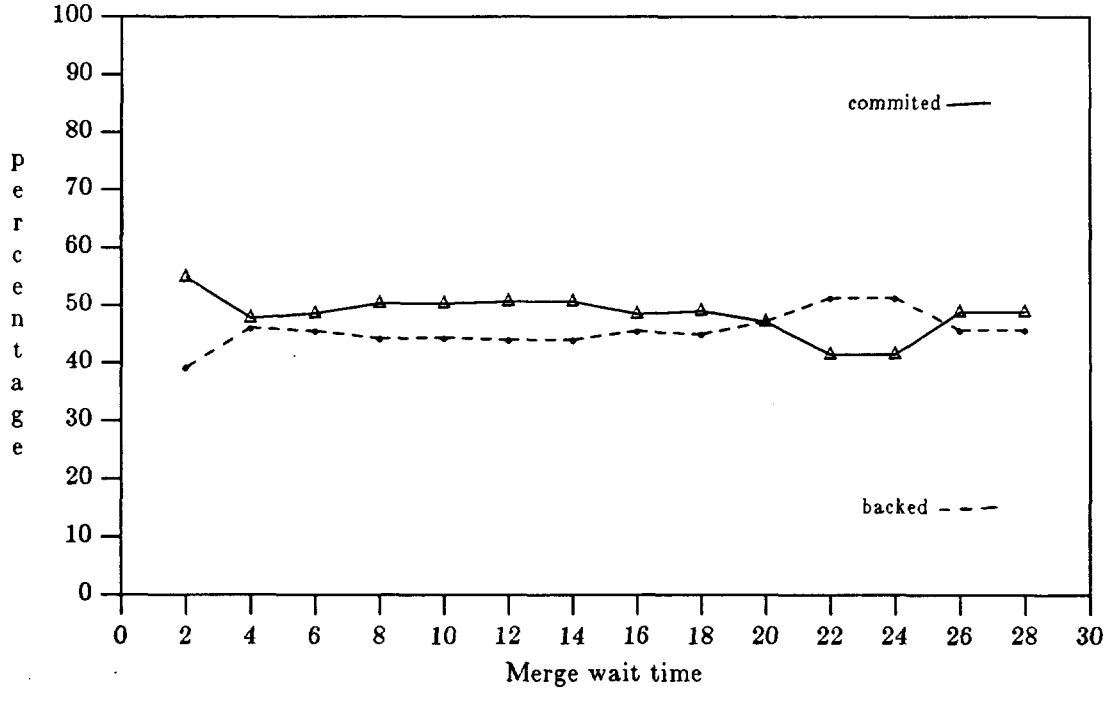
Table 4.3.3

merge wait time	committed transactions	commit%	aborted transactions	abort%	backed out transactions	backed out%	total transactions
2	74	54.41	8	5.88	54	39.71	136
4	64	47.06	8	5.88	64	47.06	136
6	64	47.06	8	5.88	64	47.06	136
8	67	49.26	7	5.15	62	45.59	136
10	67	49.26	7	5.15	62	45.59	136
12	66	48.53	7	5.15	63	46.32	136
14	66	48.53	7	5.15	63	46.32	136
16	64	47.06	8	5.88	64	47.06	136
18	66	48.53	8	5.88	62	45.59	136
20	63	46.32	7	5.15	66	48.53	136
22	54	39.71	7	5.15	75	55.15	136
24	54	39.71	7	5.15	75	55.15	136
26	65	47.79	7	5.15	64	47.06	136
28	65	47.79	7	5.15	64	47.06	136



Database Size = 1000 items.
Number of Data items read = 5 to 20.
Percentage of Write Set = 60 % of read set.

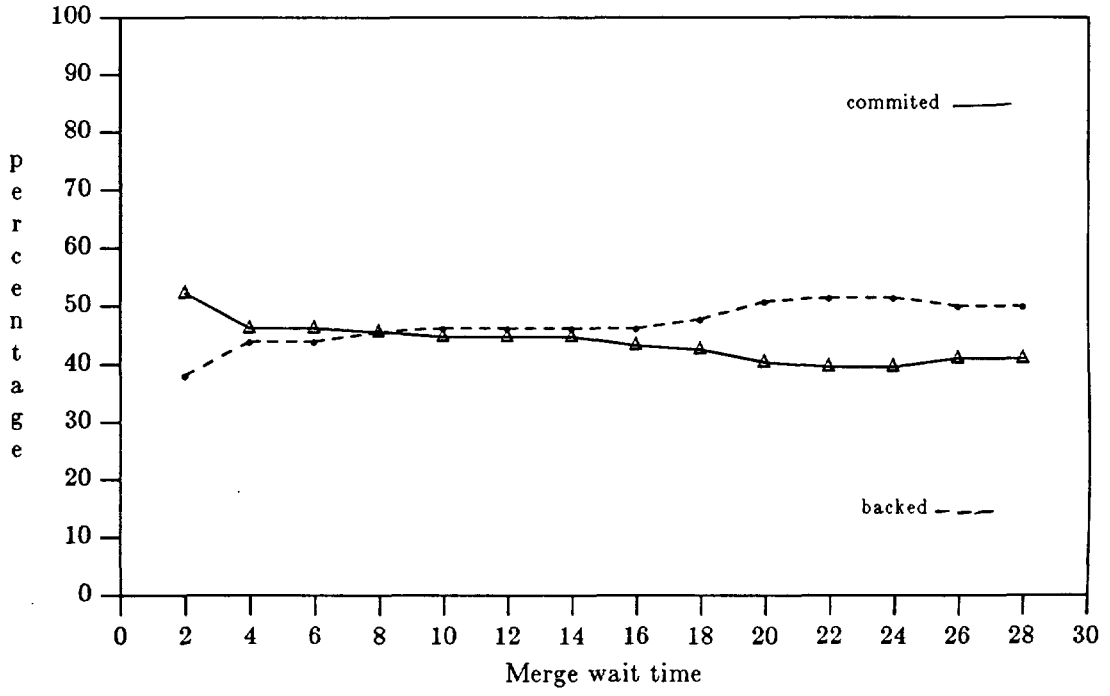
Mean Table - 4.3			
merge wait time	commit%	abort%	backed%
2	54.86	5.88	39.26
4	47.83	6.07	46.09
6	48.60	5.88	45.52
8	50.30	5.51	44.19
10	50.30	5.32	44.38
12	50.69	5.32	43.99
14	50.69	5.32	43.99
16	48.42	6.07	45.52
18	48.96	6.07	44.96
20	47.07	5.89	47.23
22	41.39	5.32	51.35
24	41.39	5.32	51.35
26	48.78	5.51	45.71
28	48.78	5.51	45.71



Database Size = 1000 items. Seed = 7774755
Number of Data items read = 5 to 20.
Percentage of Write Set = 80 % of read set.

Table 4.4

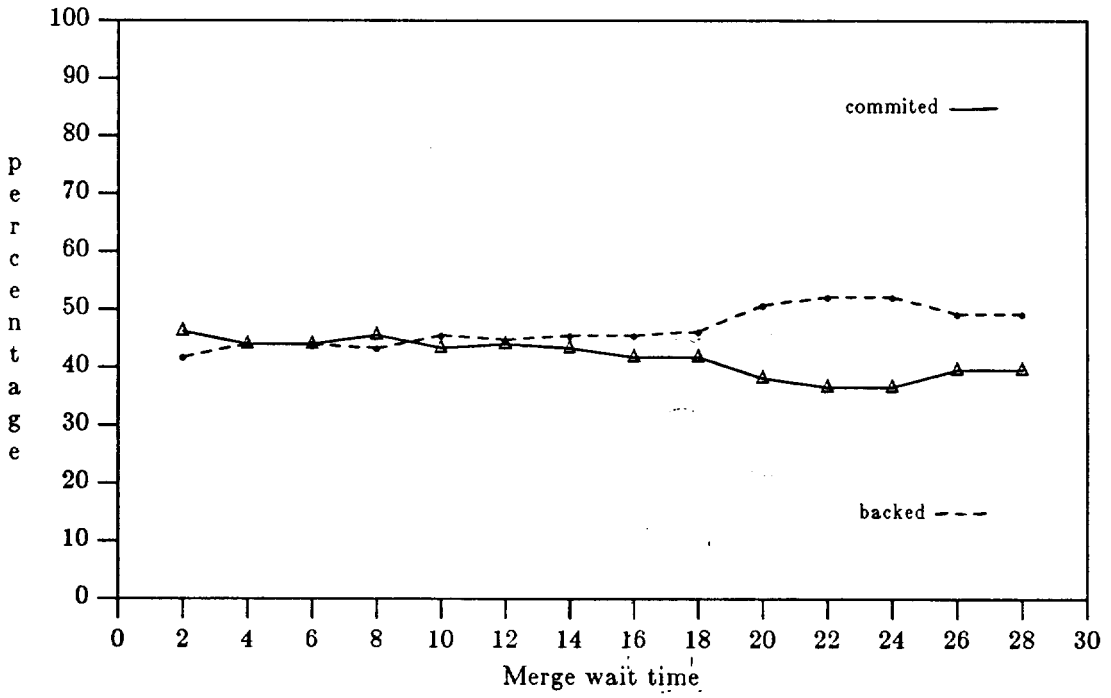
merge wait time	committed transactions	commit%	aborted transactions	abort%	backed out transactions	backed out%	total transactions
2	70	52.24	13	9.70	51	38.06	134
4	62	46.27	13	9.70	59	44.03	134
6	62	46.27	13	9.70	59	44.03	134
8	61	45.52	12	8.96	61	45.52	134
10	60	44.78	12	8.96	62	46.27	134
12	60	44.78	12	8.96	62	46.27	134
14	60	44.78	12	8.96	62	46.27	134
16	58	43.28	14	10.45	62	46.27	134
18	57	42.54	13	9.70	64	47.76	134
20	54	40.30	12	8.96	68	50.75	134
22	53	39.55	12	8.96	69	51.49	134
24	53	39.55	12	8.96	69	51.49	134
26	55	41.04	12	8.96	67	50.00	134
28	55	41.04	12	8.96	67	50.00	134



Database Size = 1000 items. Seed = 7774755
Number of Data items read = 5 to 20.
Percentage of Write Set = 100 % of read set.

Table 4.5.0

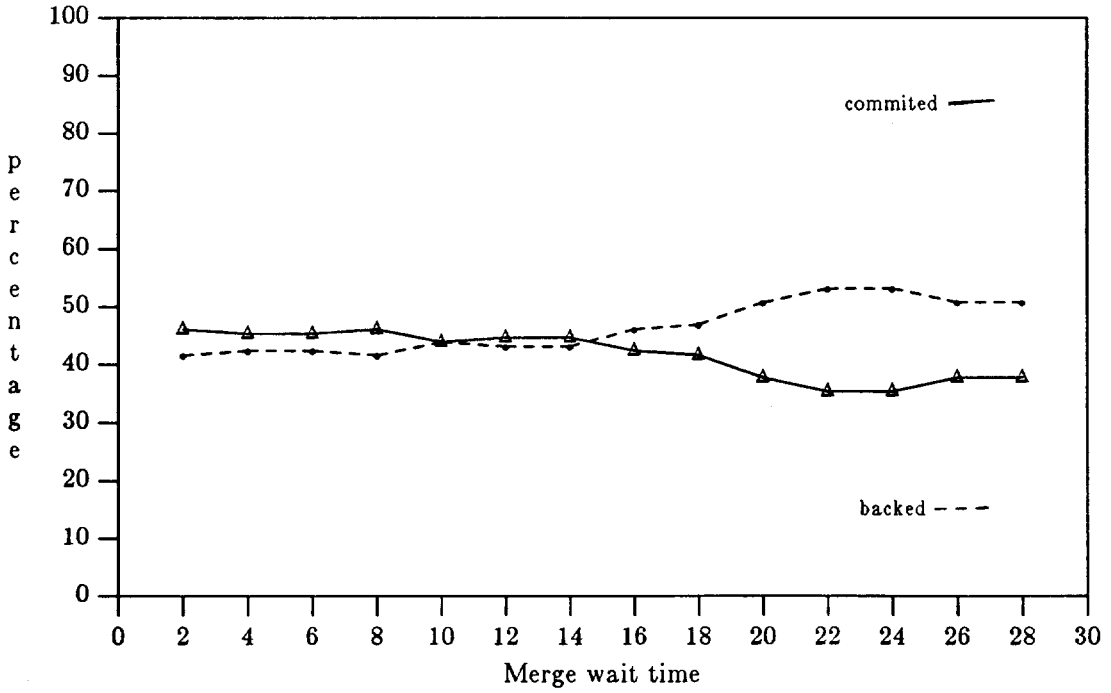
merge wait time	committed transactions	commit%	aborted transactions	abort%	backed out transactions	backed out%	total transactions
2	62	46.27	16	11.94	56	41.79	134
4	59	44.03	16	11.94	59	44.03	134
6	59	44.03	16	11.94	59	44.03	134
8	61	45.52	15	11.19	58	43.28	134
10	58	43.28	15	11.19	61	45.52	134
12	59	44.03	15	11.19	60	44.78	134
14	58	43.28	15	11.19	61	45.52	134
16	56	41.79	17	12.69	61	45.52	134
18	56	41.79	16	11.94	62	46.27	134
20	51	38.06	15	11.19	68	50.75	134
22	49	36.57	15	11.19	70	52.24	134
24	49	36.57	15	11.19	70	52.24	134
26	53	39.55	15	11.19	66	49.25	134
28	53	39.55	15	11.19	66	49.25	134



Database Size = 1000 items. Seed = 56743
Number of Data items read = 5 to 20.
Percentage of Write Set = 100 % of read set.

Table 4.5.1

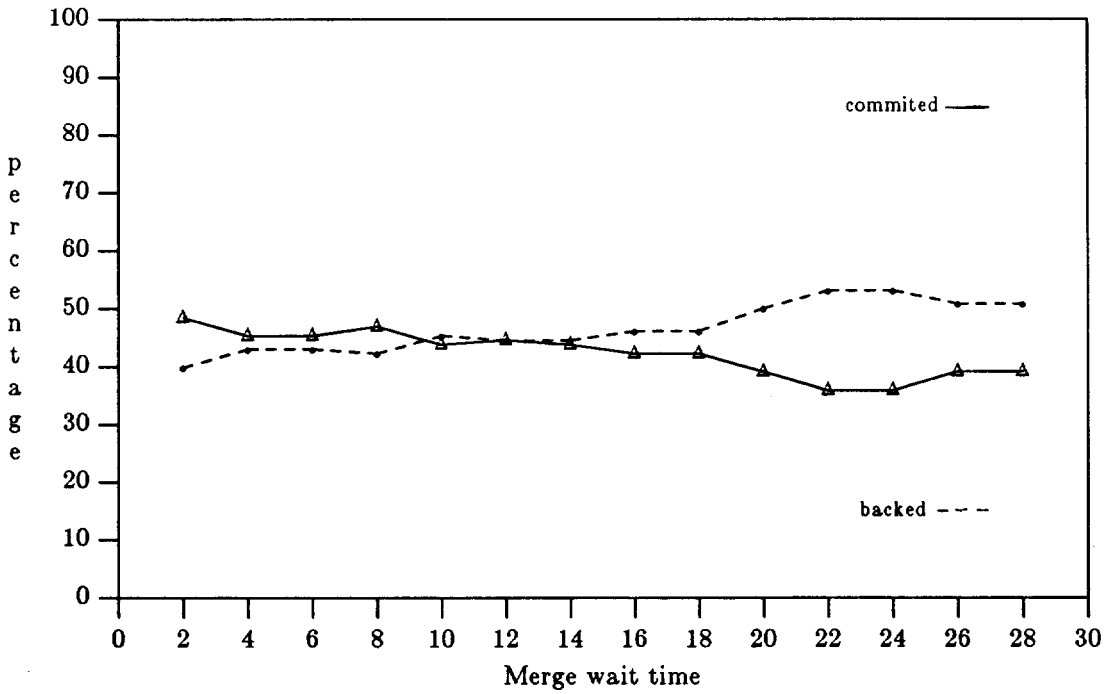
merge wait time	committed transactions	commit%	aborted transactions	abort%	backed out transactions	backed out%	total transactions
2	60	46.15	16	12.31	54	41.54	130
4	59	45.38	16	12.31	55	42.31	130
6	59	45.38	16	12.31	55	42.31	130
8	60	46.15	16	12.31	54	41.54	130
10	57	43.85	16	12.31	57	43.85	130
12	58	44.62	16	12.31	56	43.08	130
14	58	44.62	16	12.31	56	43.08	130
16	55	42.31	15	11.54	60	46.15	130
18	54	41.54	15	11.54	61	46.92	130
20	49	37.69	15	11.54	66	50.77	130
22	46	35.38	15	11.54	69	53.08	130
24	46	35.38	15	11.54	69	53.08	130
26	49	37.69	15	11.54	66	50.77	130
28	49	37.69	15	11.54	66	50.77	130



Database Size = 1000 items. Seed = 101489
Number of Data items read = 5 to 20.
Percentage of Write Set = 100 % of read set.

Table 4.5.2

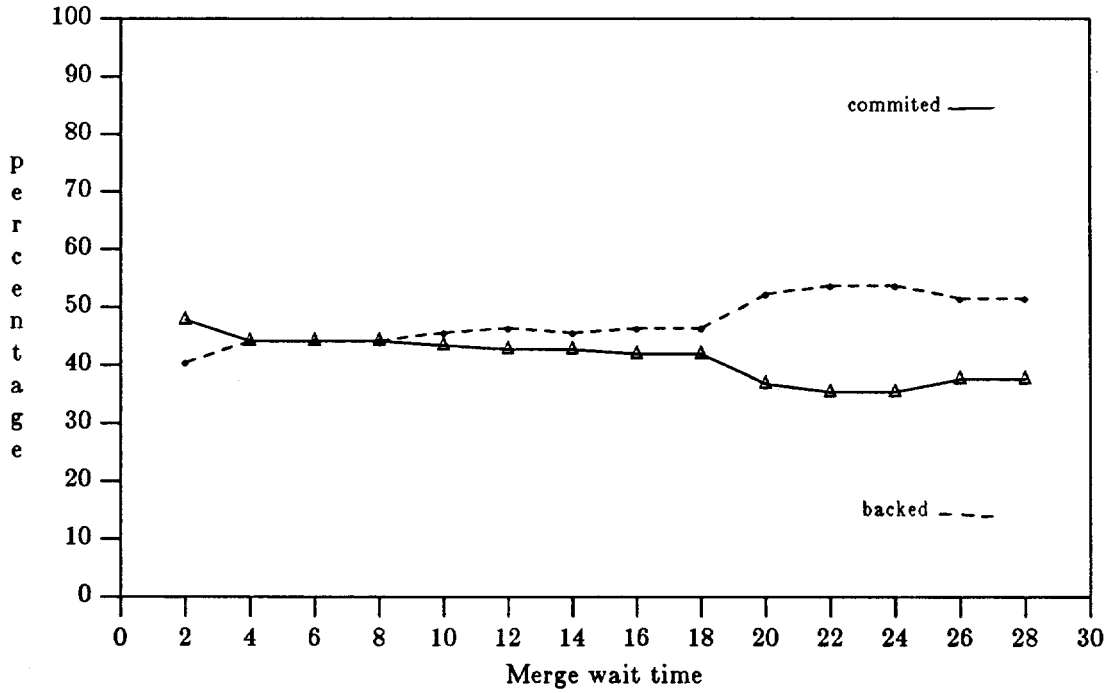
merge wait time	committed transactions	commit%	aborted transactions	abort%	backed out transactions	backed out%	total transactions
2	62	48.44	15	11.72	51	39.84	128
4	58	45.31	15	11.72	55	42.97	128
6	58	45.31	15	11.72	55	42.97	128
8	60	46.88	14	10.94	54	42.19	128
10	56	43.75	14	10.94	58	45.31	128
12	57	44.53	14	10.94	57	44.53	128
14	56	43.75	15	11.72	57	44.53	128
16	54	42.19	15	11.72	59	46.09	128
18	54	42.19	15	11.72	59	46.09	128
20	50	39.06	14	10.94	64	50.00	128
22	46	35.94	14	10.94	68	53.13	128
24	46	35.94	14	10.94	68	53.13	128
26	50	39.06	13	10.16	65	50.78	128
28	50	39.06	13	10.16	65	50.78	128



Database Size = 1000 items. Seed = 4235761
Number of Data items read = 5 to 20.
Percentage of Write Set = 100 % of read set.

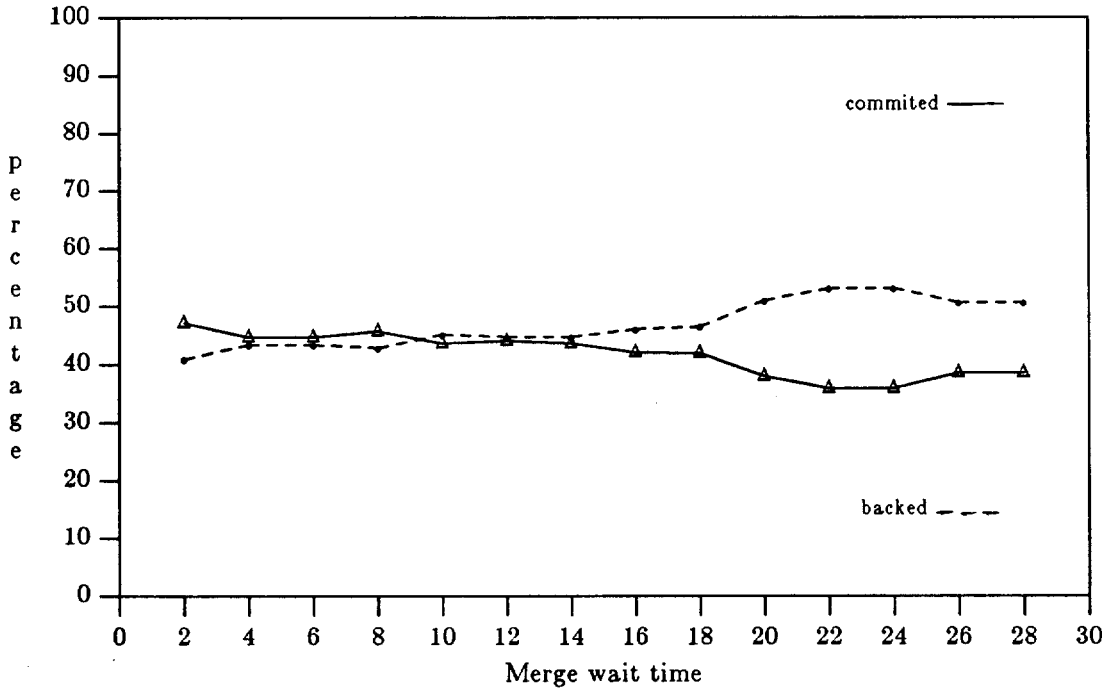
Table 4.5.3

merge wait time	committed transactions	commit%	aborted transactions	abort%	backed out transactions	backed out%	total transactions
2	65	47.79	16	11.76	55	40.44	136
4	60	44.12	16	11.76	60	44.12	136
6	60	44.12	16	11.76	60	44.12	136
8	60	44.12	16	11.76	60	44.12	136
10	59	43.38	15	11.03	62	45.59	136
12	58	42.65	15	11.03	63	46.32	136
14	58	42.65	16	11.76	62	45.59	136
16	57	41.91	16	11.76	63	46.32	136
18	57	41.91	16	11.76	63	46.32	136
20	50	36.76	15	11.03	71	52.20	136
22	48	35.29	15	11.03	73	53.68	136
24	48	35.29	15	11.03	73	53.68	136
26	51	37.50	15	11.03	70	51.47	136
28	51	37.50	15	11.03	70	51.47	136



Database Size = 1000 items.
Number of Data items read = 5 to 20.
Percentage of Write Set = 100 % of read set.

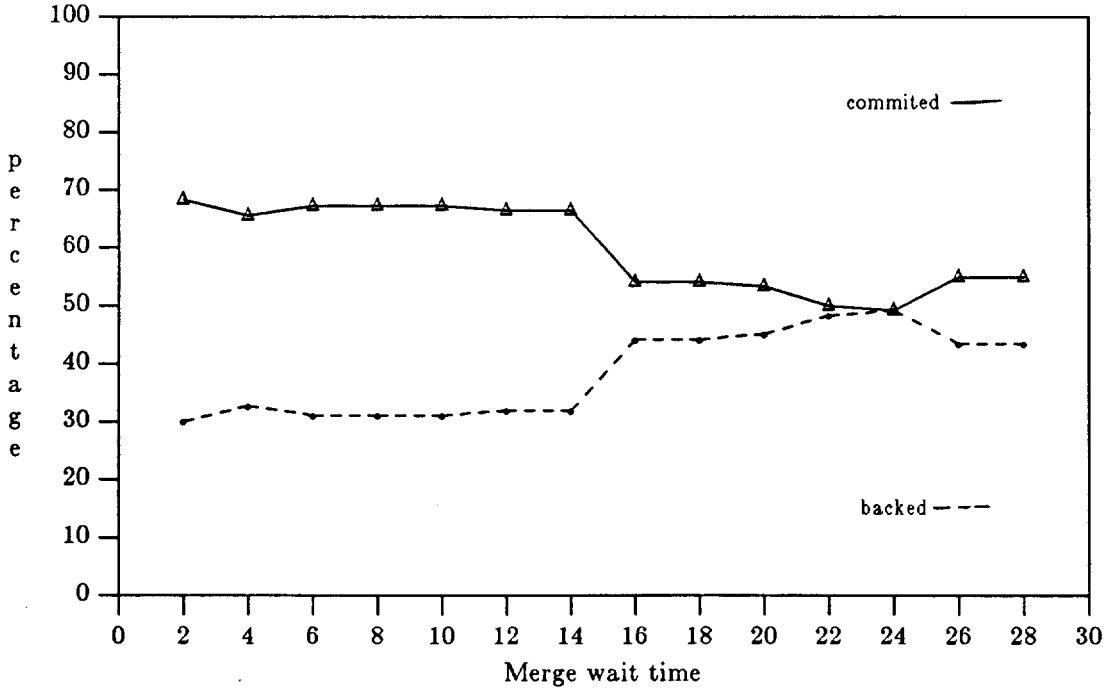
Mean Table - 4.5			
merge wait time	commit%	abort%	backed%
2	47.16	11.93	40.90
4	44.71	11.93	43.36
6	44.71	11.93	43.36
8	45.67	11.55	42.78
10	43.57	11.37	45.07
12	43.96	11.37	44.68
14	43.57	11.75	44.68
16	42.05	11.93	46.02
18	41.86	11.74	46.40
20	37.89	11.17	50.93
22	35.79	11.17	53.03
24	35.79	11.17	53.03
26	38.45	10.98	50.57
28	38.45	10.98	50.57



Database Size = 200 items. Seed = 7774755
Number of Data items read = 5 to 20.
Percentage of Write Set = 20 % of read set.

Table 4.6.0

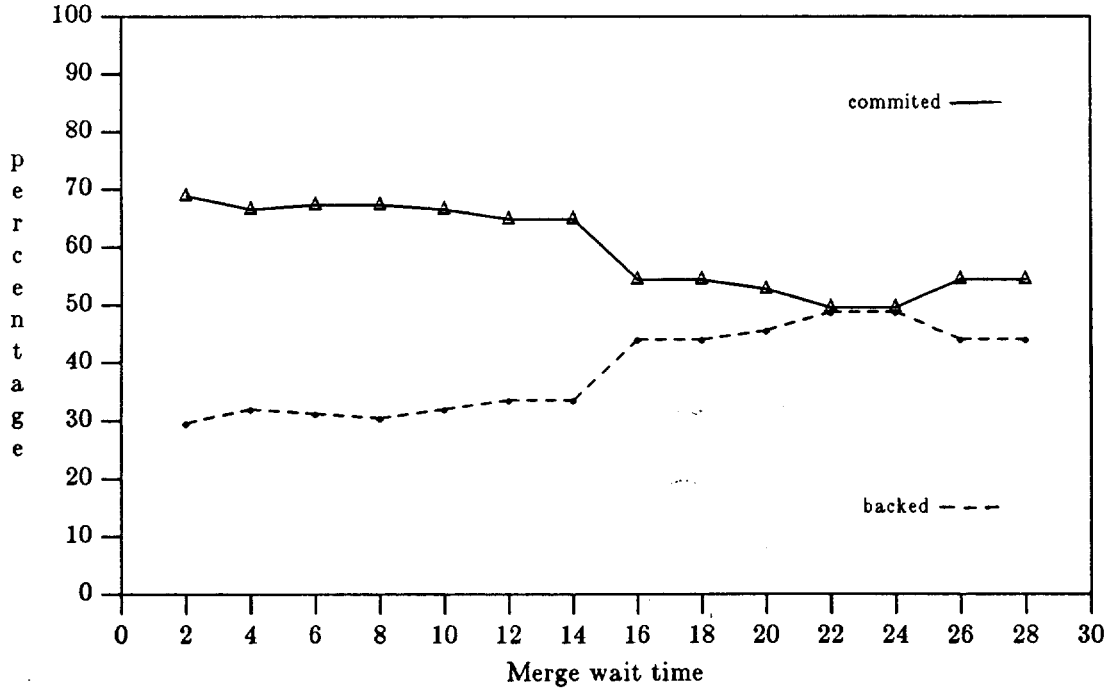
merge wait time	committed transactions	commit%	aborted transactions	abort%	backed out transactions	backed out%	total transactions
2	84	68.29	2	1.63	37	30.08	123
4	80	65.57	2	1.64	40	32.79	122
6	82	67.21	2	1.64	38	31.15	122
8	82	67.21	2	1.64	38	31.15	122
10	82	67.21	2	1.64	38	31.15	122
12	81	66.39	2	1.64	39	31.97	122
14	81	66.39	2	1.64	39	31.97	122
16	66	54.10	2	1.64	54	44.26	122
18	66	54.10	2	1.64	54	44.26	122
20	65	53.28	2	1.64	55	45.08	122
22	61	50.00	2	1.64	59	48.36	122
24	60	49.18	2	1.64	60	49.18	122
26	67	54.92	2	1.64	53	43.44	122
28	67	54.92	2	1.64	53	43.44	122



Database Size = 200 items. Seed = 56743
Number of Data items read = 5 to 20.
Percentage of Write Set = 20 % of read set.

Table 4.6.1

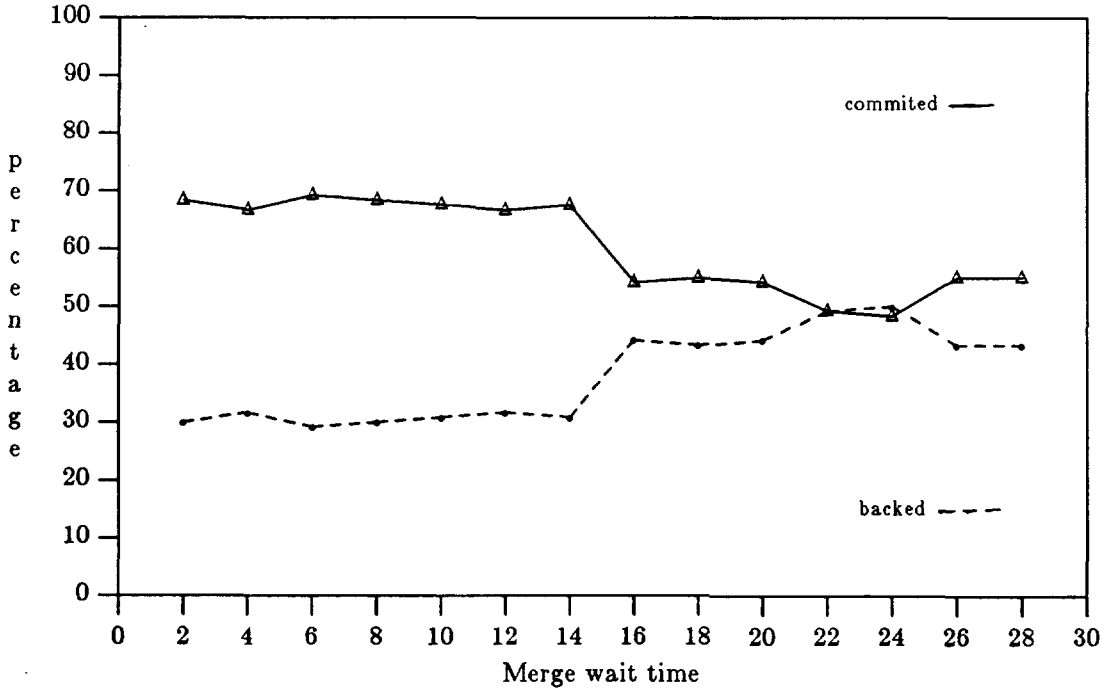
merge wait time	committed transactions	commit%	aborted transactions	abort%	backed out transactions	backed out%	total transactions
2	86	68.80	2	1.60	37	29.60	125
4	83	66.40	2	1.60	40	32.00	125
6	84	67.20	2	1.60	39	31.20	125
8	84	67.20	3	2.40	38	30.40	125
10	83	66.40	2	1.60	40	32.00	125
12	81	64.80	2	1.60	42	33.60	125
14	81	64.80	2	1.60	42	33.60	125
16	68	54.40	2	1.60	55	44.00	125
18	68	54.40	2	1.60	55	44.00	125
20	66	52.80	2	1.60	57	45.60	125
22	62	49.60	2	1.60	61	48.80	125
24	62	49.60	2	1.60	61	48.80	125
26	68	54.40	2	1.60	55	44.00	125
28	68	54.40	2	1.60	55	44.00	125



Database Size = 200 items. Seed = 101489
Number of Data items read = 5 to 20.
Percentage of Write Set = 20 % of read set.

Table 4.6.2

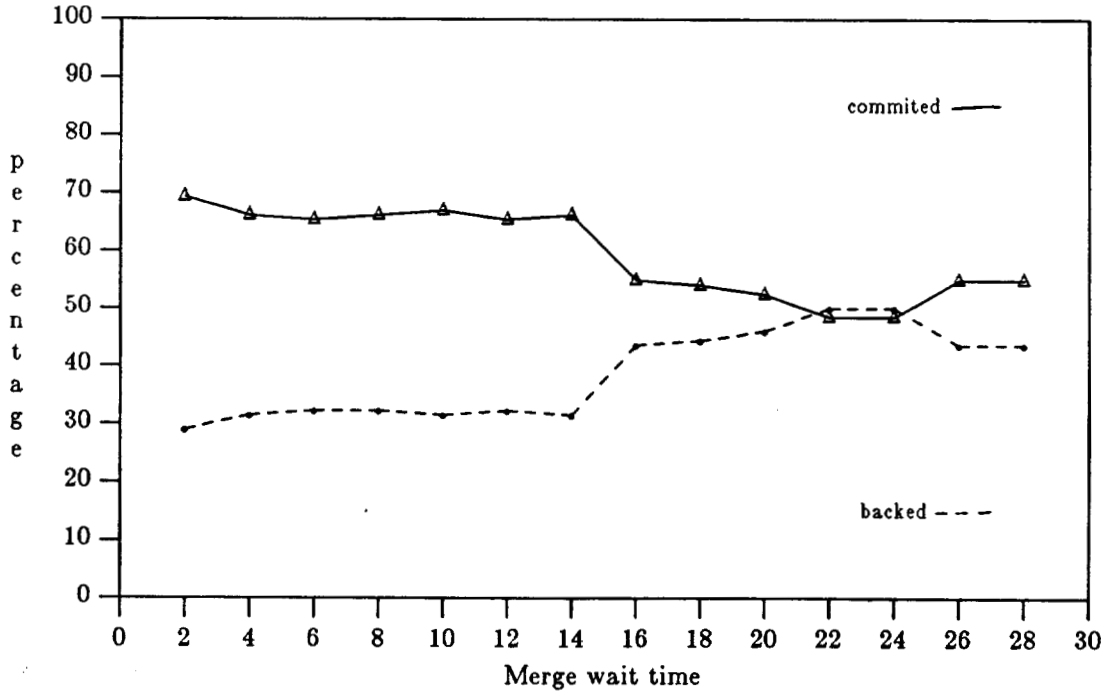
merge wait time	committed transactions	commit%	aborted transactions	abort%	backed out transactions	backed out%	total transactions
2	82	68.33	2	1.67	36	30.00	120
4	80	66.67	2	1.67	38	31.67	120
6	83	69.17	2	1.67	35	29.17	120
8	82	68.33	2	1.67	36	30.00	120
10	81	67.50	2	1.67	37	30.83	120
12	80	66.67	2	1.67	38	31.67	120
14	81	67.50	2	1.67	37	30.83	120
16	65	54.17	2	1.67	53	44.17	120
18	66	55.00	2	1.67	52	43.33	120
20	65	54.17	2	1.67	53	44.17	120
22	59	49.17	2	1.67	59	49.17	120
24	58	48.33	2	1.67	60	50.00	120
26	66	55.00	2	1.67	52	43.33	120
28	66	55.00	2	1.67	52	43.33	120



Database Size = 200 items. Seed = 4235761
Number of Data items read = 5 to 20.
Percentage of Write Set = 20 % of read set.

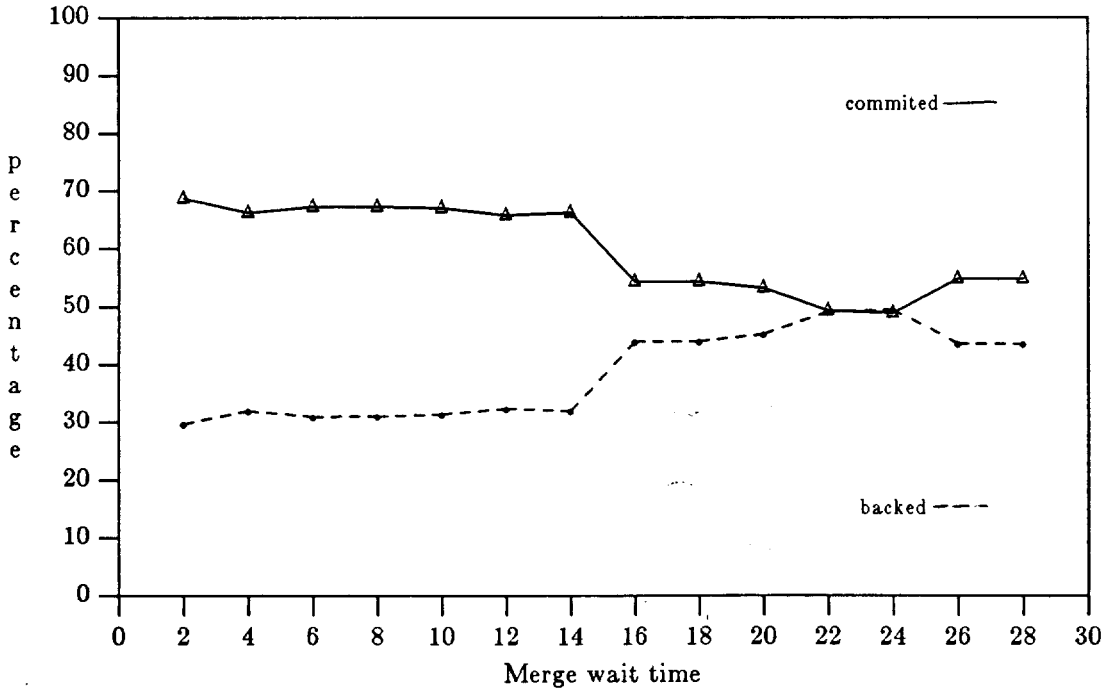
Table 4.6.3

merge wait time	committed transactions	commit%	aborted transactions	abort%	backed out transactions	backed out%	total transactions
2	86	69.35	2	1.61	36	29.03	124
4	82	66.13	3	2.42	39	31.45	124
6	81	65.32	3	2.42	40	32.26	124
8	82	66.13	2	1.61	40	32.26	124
10	83	66.94	2	1.61	39	31.45	124
12	81	65.32	3	2.42	40	32.26	124
14	82	66.13	3	2.42	39	31.45	124
16	68	54.84	2	1.61	54	43.55	124
18	67	54.03	2	1.61	55	44.35	124
20	65	52.42	2	1.61	57	45.97	124
22	60	48.39	2	1.61	62	50.00	124
24	60	48.39	2	1.61	62	50.00	124
26	68	54.84	2	1.61	54	43.55	124
28	68	54.84	2	1.61	54	43.55	124



Database Size = 200 items.
Number of Data items read = 5 to 20.
Percentage of Write Set = 20 % of read set.

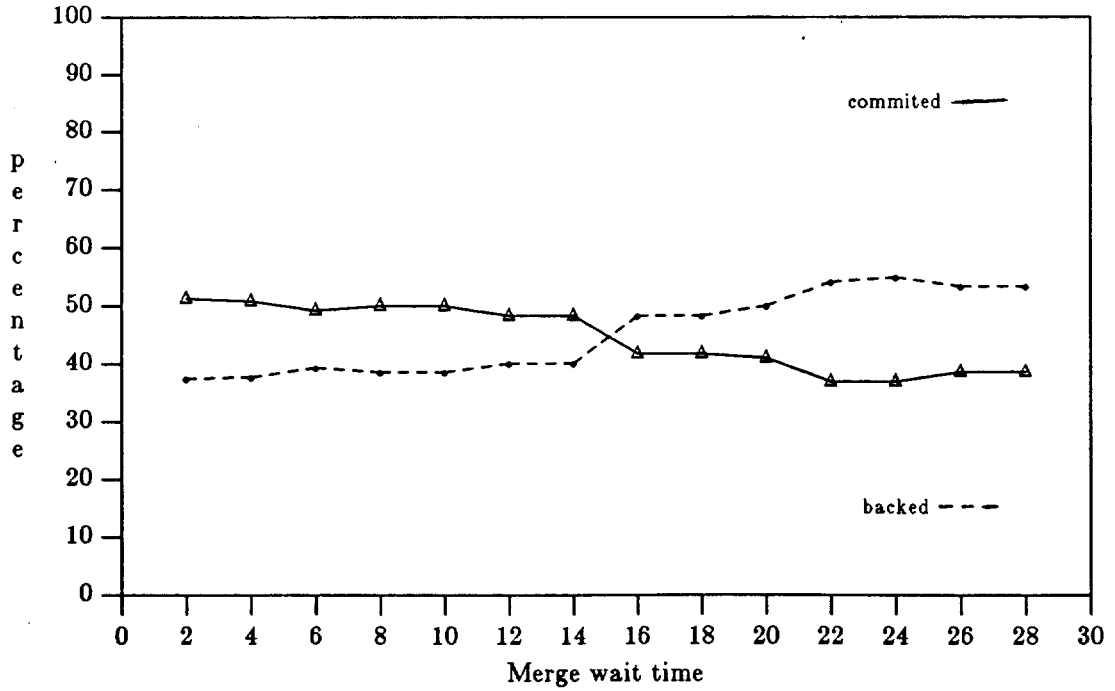
Mean Table - 4.6			
merge wait time	commit%	abort%	backed%
2	68.69	1.63	29.68
4	66.19	1.83	31.98
6	67.22	1.83	30.94
8	67.22	1.83	30.95
10	67.01	1.63	31.36
12	65.79	1.83	32.38
14	66.21	1.83	31.96
16	54.38	1.63	43.99
18	54.38	1.63	43.99
20	53.17	1.63	45.21
22	49.29	1.63	49.08
24	48.88	1.63	49.49
26	54.79	1.63	43.58
28	54.79	1.63	43.58



Database Size = 200 items. Seed = 7774755
Number of Data items read = 5 to 20.
Percentage of Write Set = 40 % of read set.

Table 4.7

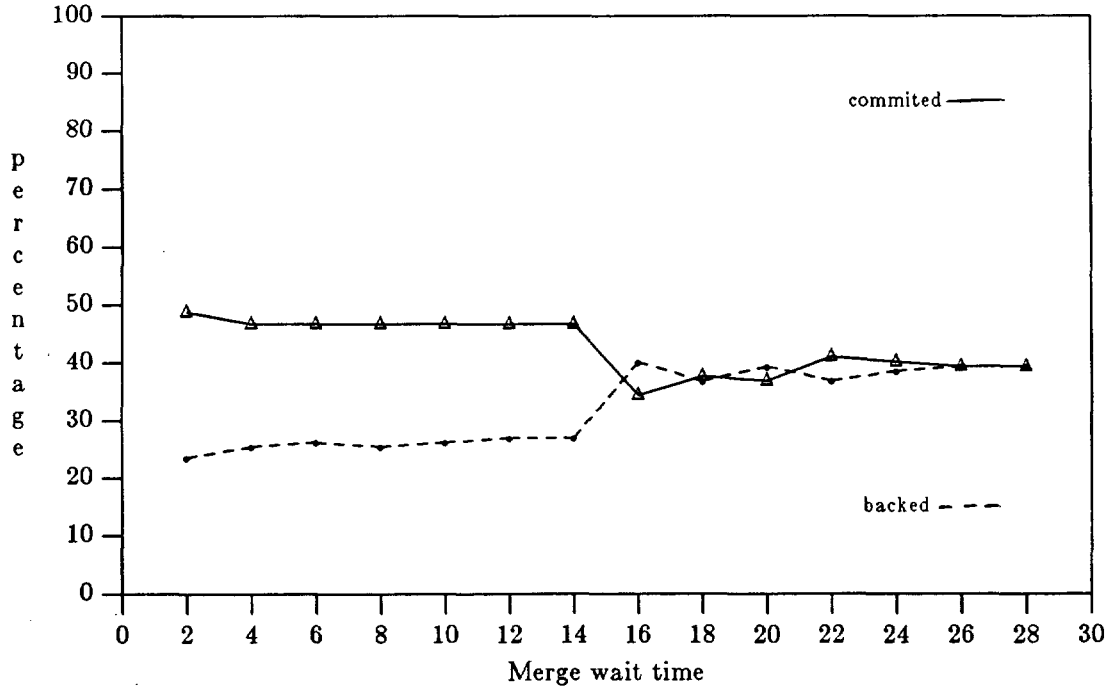
merge wait time	committed transactions	commit%	aborted transactions	abort%	backed out transactions	backed out%	total transactions
2	63	51.22	14	11.38	46	37.40	123
4	62	50.82	14	11.48	46	37.70	122
6	60	49.18	14	11.48	48	39.34	122
8	61	50.00	14	11.48	47	38.52	122
10	61	50.00	14	11.48	47	38.52	122
12	59	48.36	14	11.48	49	40.16	122
14	59	48.36	14	11.48	49	40.16	122
16	51	41.80	12	9.84	59	48.36	122
18	51	41.80	12	9.84	59	48.36	122
20	50	40.98	11	9.02	61	50.00	122
22	45	36.89	11	9.02	66	54.10	122
24	45	36.89	10	8.20	67	54.92	122
26	47	38.52	10	8.20	65	53.28	122
28	47	38.52	10	8.20	65	53.28	122



Database Size = 200 items. Seed = 7774755
Number of Data items read = 5 to 20.
Percentage of Write Set = 60 % of read set.

Table 4.8.0

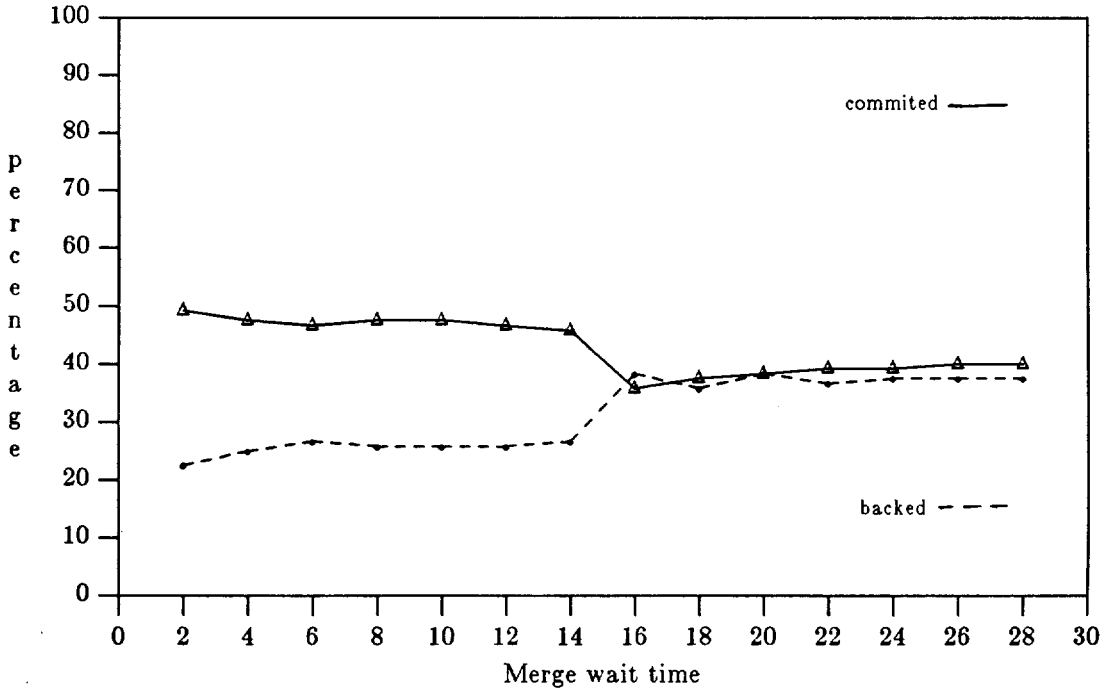
merge wait time	committed transactions	commit%	aborted transactions	abort%	backed out transactions	backed out%	total transactions
2	60	48.78	34	27.64	29	23.58	123
4	57	46.72	34	27.87	31	25.41	122
6	57	46.72	33	27.05	32	26.23	122
8	57	46.72	34	27.87	31	25.41	122
10	57	46.72	33	27.05	32	26.23	122
12	57	46.72	32	26.23	33	27.05	122
14	57	46.72	32	26.23	33	27.05	122
16	42	34.43	31	25.41	49	40.16	122
18	46	37.70	31	25.41	45	36.89	122
20	45	36.89	29	23.77	48	39.34	122
22	50	40.98	27	22.13	45	36.89	122
24	49	40.16	26	21.31	47	38.52	122
26	48	39.34	26	21.31	48	39.34	122
28	48	39.34	26	21.31	48	39.34	122



Database Size = 200 items. Seed = 56743
Number of Data items read = 5 to 20.
Percentage of Write Set = 60 % of read set.

Table 4.8.1

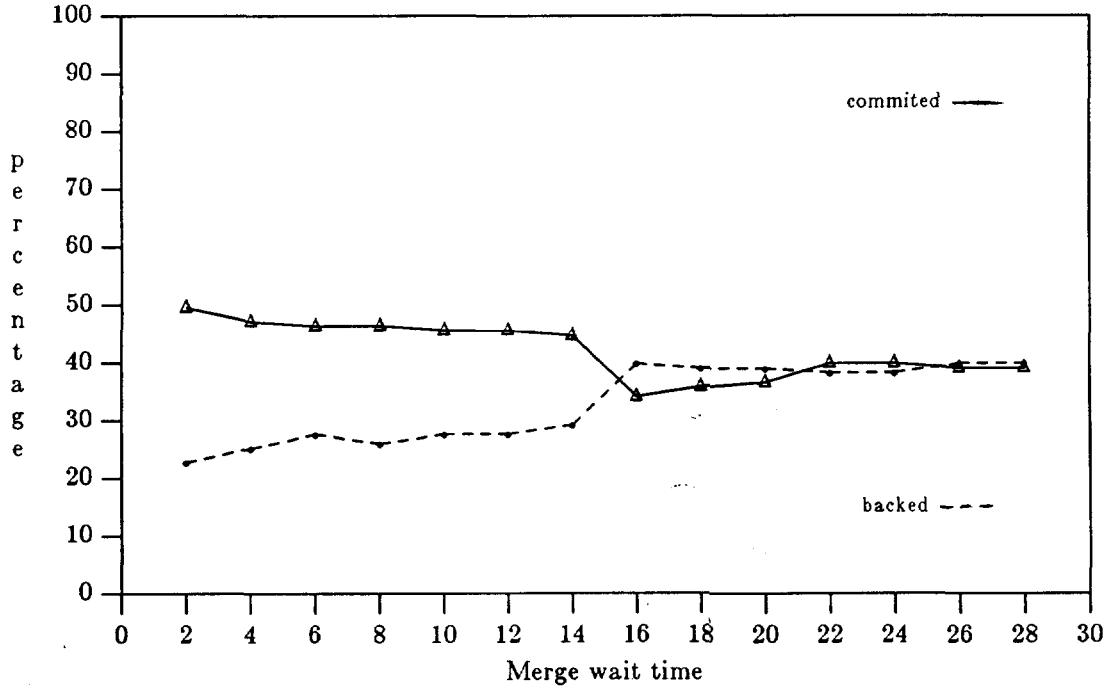
merge wait time	committed transactions	commit%	aborted transactions	abort%	backed out transactions	backed out%	total transactions
2	59	49.17	34	28.33	27	22.50	120
4	57	47.50	30	25.00	30	25.00	120
6	56	46.67	32	26.67	32	26.67	120
8	57	47.50	32	26.67	31	25.83	120
10	57	47.50	32	26.67	31	25.83	120
12	56	46.67	33	27.50	31	25.83	120
14	55	45.83	33	27.50	32	26.67	120
16	43	35.83	31	25.83	46	38.33	120
18	45	37.50	32	26.67	43	35.83	120
20	46	38.33	28	23.33	46	38.33	120
22	47	39.17	29	24.17	44	36.67	120
24	47	39.17	28	23.33	45	37.50	120
26	48	40.00	27	22.50	45	37.50	120
28	48	40.00	27	22.50	45	37.50	120



Database Size = 200 items. Seed = 101489
Number of Data items read = 5 to 20.
Percentage of Write Set = 60 % of read set.

Table 4.8.2

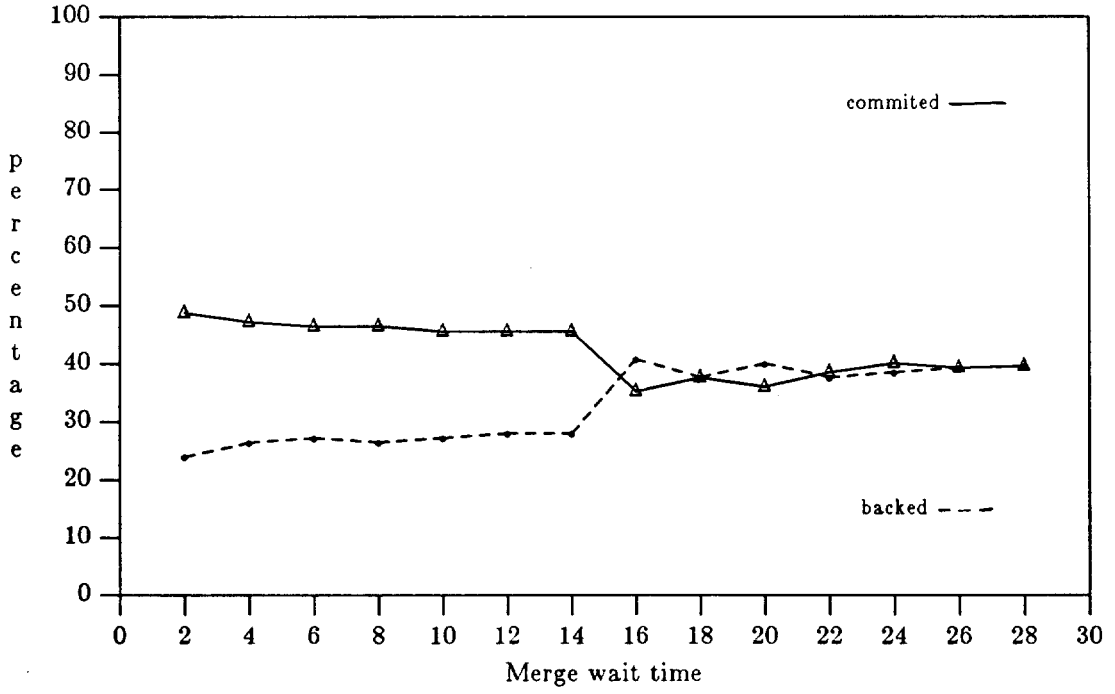
merge wait time	committed transactions	commit%	aborted transactions	abort%	backed out transactions	backed out%	total transactions
2	61	49.59	34	27.64	28	22.76	123
4	58	47.15	34	27.64	31	25.20	123
6	57	46.34	32	26.02	34	27.64	123
8	57	46.34	34	27.64	32	26.02	123
10	56	45.53	33	26.83	34	27.64	123
12	56	45.53	33	26.83	34	27.64	123
14	55	44.72	32	26.02	36	29.27	123
16	42	34.15	32	26.02	49	39.84	123
18	44	35.77	31	25.20	48	39.02	123
20	45	36.59	30	24.39	48	39.02	123
22	49	39.84	27	21.95	47	38.21	123
24	49	39.84	27	21.95	47	38.21	123
26	48	39.02	26	21.14	49	39.84	123
28	48	39.02	26	21.14	49	39.84	123



Database Size = 200 items. Seed = 4235761
Number of Data items read = 5 to 20.
Percentage of Write Set = 60 % of read set.

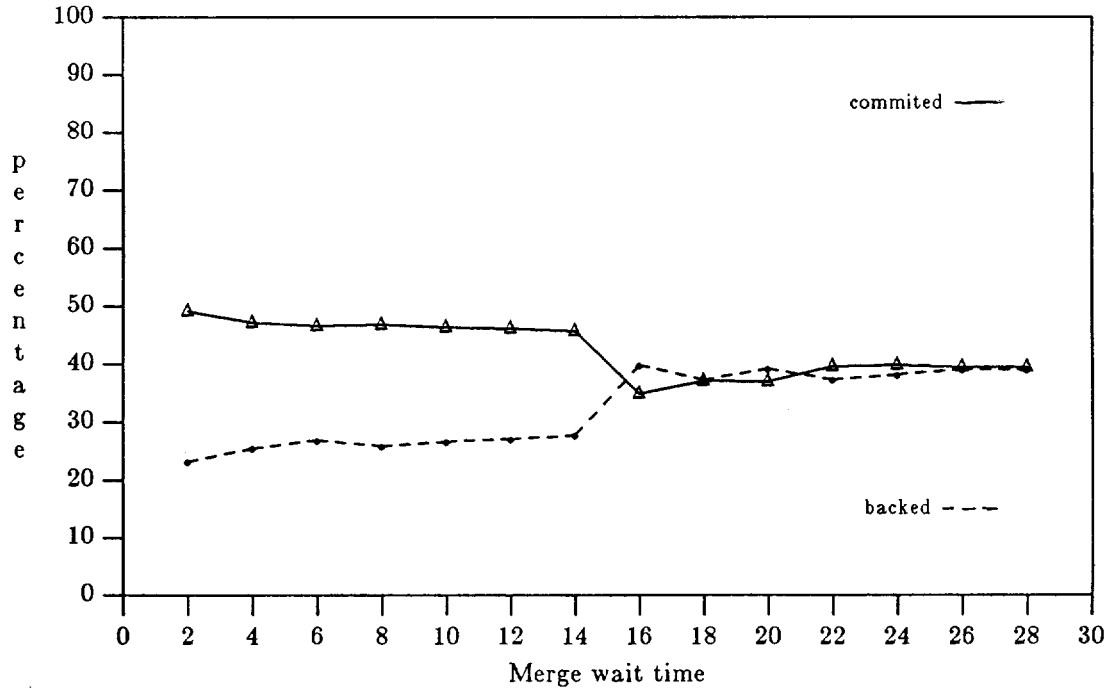
Table 4.8.3

merge wait time	committed transactions	commit%	aborted transactions	abort%	backed out transactions	backed out%	total transactions
2	61	48.80	34	27.20	30	24.00	125
4	59	47.20	33	26.40	33	26.40	125
6	58	46.40	33	26.40	34	27.20	125
8	58	46.40	34	27.20	33	26.40	125
10	57	45.60	34	27.20	34	27.20	125
12	57	45.60	33	26.40	35	28.00	125
14	57	45.60	33	26.40	35	28.00	125
16	44	35.20	30	24.00	51	40.80	125
18	47	37.60	31	24.80	47	37.60	125
20	45	36.00	30	24.00	50	40.00	125
22	48	38.40	29	23.20	47	37.60	125
24	50	40.00	27	21.60	48	38.40	125
26	49	39.20	27	21.60	49	39.20	125
28	49	39.20	27	21.60	49	39.20	125



Database Size = 200 items.
Number of Data items read = 5 to 20.
Percentage of Write Set = 60 % of read set.

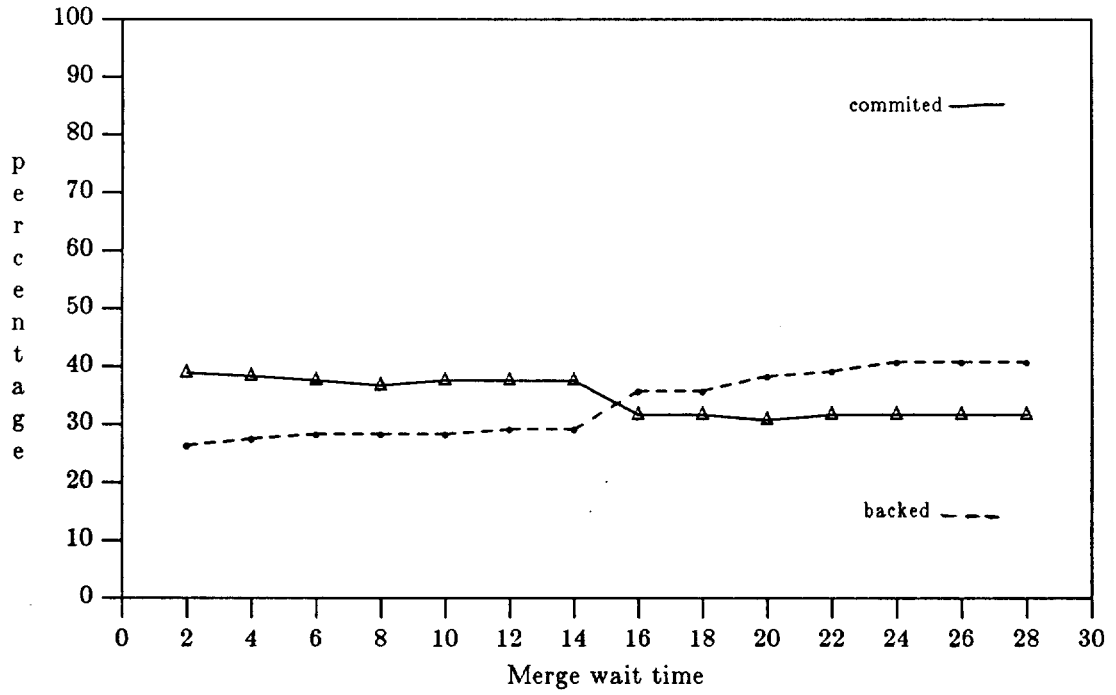
Mean Table - 4.8			
merge wait time	commit%	abort%	backed%
2	49.08	27.70	23.21
4	47.14	26.73	25.50
6	46.53	26.54	26.94
8	46.74	27.35	25.91
10	46.34	26.94	26.72
12	46.13	26.74	27.13
14	45.72	26.54	27.75
16	34.90	25.31	39.78
18	37.14	25.52	37.33
20	36.95	23.87	39.17
22	39.60	22.86	37.34
24	39.79	22.05	38.16
26	39.39	21.64	38.97
28	39.47	21.68	39.05



Database Size = 200 items. Seed = 7774755
Number of Data items read = 5 to 20.
Percentage of Write Set = 80 % of read set.

Table 4.9

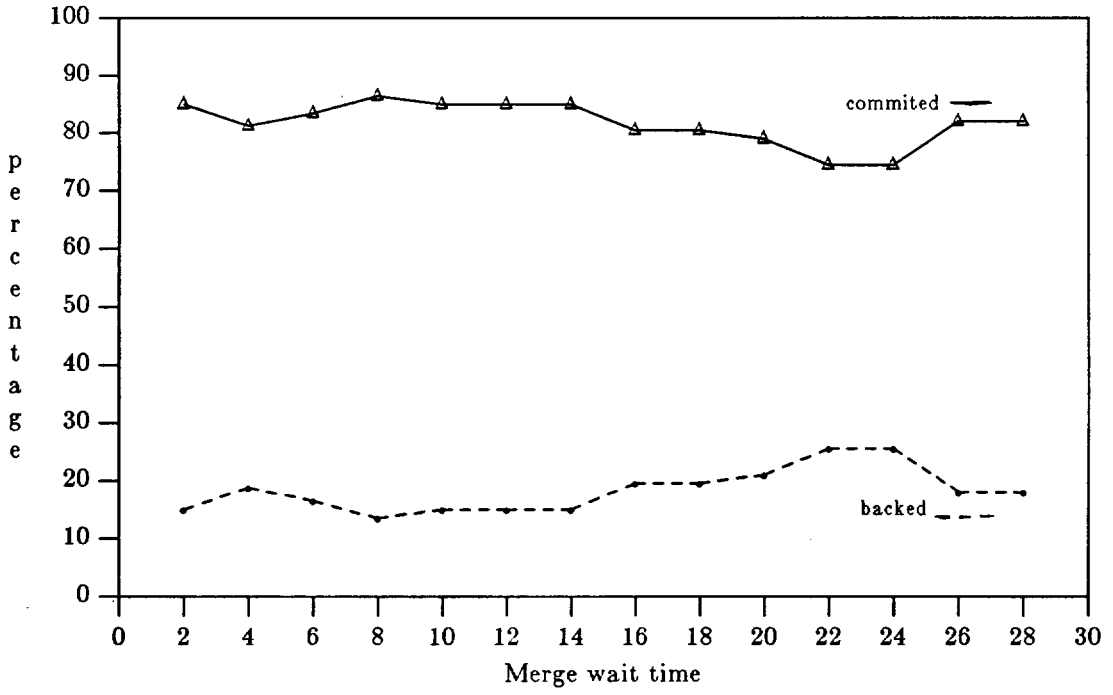
timer	committed	commit%	aborted	abort%	backed	backed%	total TR
2	47	38.84	42	34.71	32	26.45	121
4	46	38.33	41	34.17	33	27.50	120
6	45	37.50	41	34.17	34	28.33	120
8	44	36.67	42	35.00	34	28.33	120
10	45	37.50	41	34.17	34	28.33	120
12	45	37.50	40	33.33	35	29.17	120
14	45	37.50	40	33.33	35	29.17	120
16	38	31.67	39	32.50	43	35.83	120
18	38	31.67	39	32.50	43	35.83	120
20	37	30.83	37	30.83	46	38.33	120
22	38	31.67	35	29.17	47	39.17	120
24	38	31.67	33	27.50	49	40.83	120
26	38	31.67	33	27.50	49	40.83	120
28	38	31.67	33	27.50	49	40.83	120



Database Size = 1000 items. Seed = 7774755
Number of Data items read = 5 to 20.
50% read-only & 50% transactions have Write Set = 20 % of read set.

Table 4.11

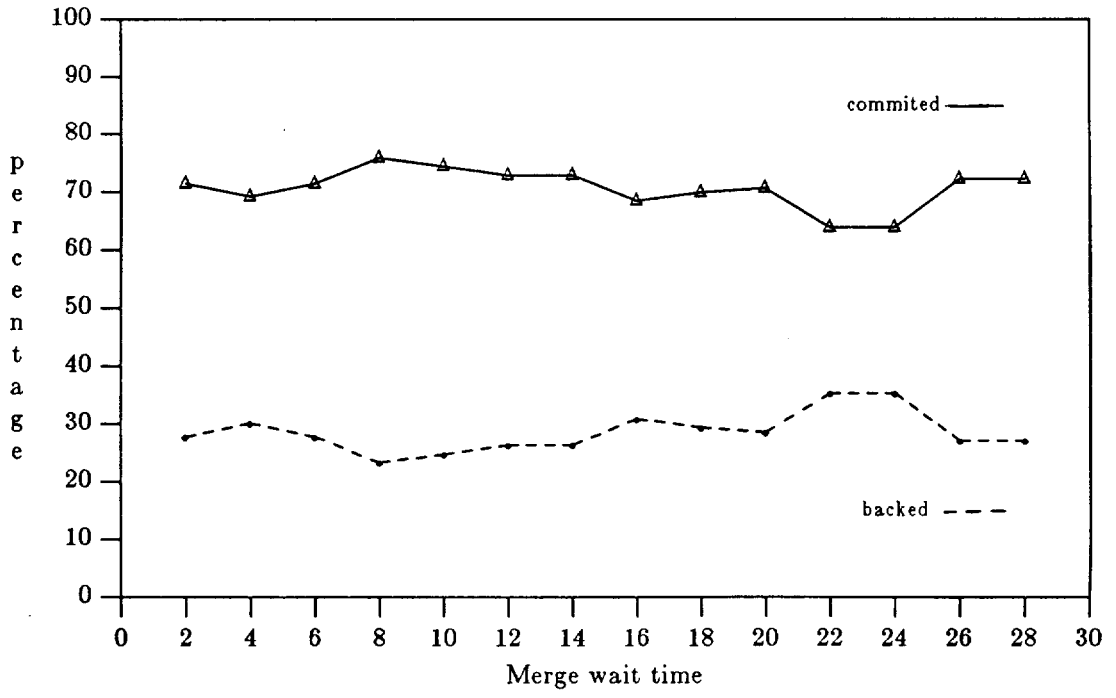
merge wait time	committed transactions	commit%	aborted transactions	abort%	backed out transactions	backed out%	total transactions
2	113	84.96	0	0.00	20	15.04	133
4	108	81.20	0	0.00	25	18.80	133
6	111	83.46	0	0.00	22	16.54	133
8	115	86.47	0	0.00	18	13.53	133
10	113	84.96	0	0.00	20	15.04	133
12	113	84.96	0	0.00	20	15.04	133
14	113	84.96	0	0.00	20	15.04	133
16	107	80.45	0	0.00	26	19.55	133
18	107	80.45	0	0.00	26	19.55	133
20	105	78.95	0	0.00	28	21.05	133
22	99	74.44	0	0.00	34	25.56	133
24	99	74.44	0	0.00	34	25.56	133
26	109	81.95	0	0.00	24	18.05	133
28	109	81.95	0	0.00	24	18.05	133



Database Size = 1000 items. Seed = 7774755
Number of Data items read = 5 to 20.
50% read-only & 50% transactions have Write Set = 40 % of read set.

Table 4.12

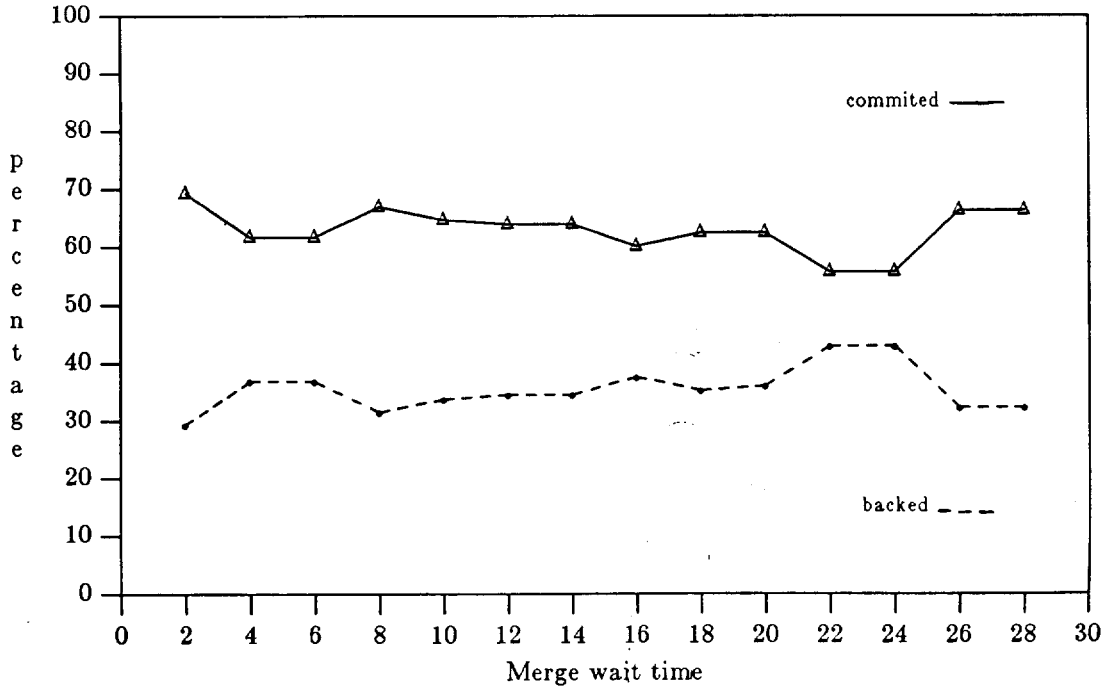
merge wait time	committed transactions	commit%	aborted transactions	abort%	backed out transactions	backed out%	total transactions
2	95	71.43	1	0.75	37	27.82	133
4	92	69.17	1	0.75	40	30.08	133
6	95	71.43	1	0.75	37	27.82	133
8	101	75.94	1	0.75	31	23.31	133
10	99	74.44	1	0.75	33	24.81	133
12	97	72.93	1	0.75	35	26.32	133
14	97	72.93	1	0.75	35	26.32	133
16	91	68.42	1	0.75	41	30.83	133
18	93	69.92	1	0.75	39	29.32	133
20	94	70.68	1	0.75	38	28.57	133
22	85	63.91	1	0.75	47	35.34	133
24	85	63.91	1	0.75	47	35.34	133
26	96	72.18	1	0.75	36	27.07	133
28	96	72.18	1	0.75	36	27.07	133



Database Size = 1000 items. Seed = 7774755
Number of Data items read = 5 to 20.
50% read-only & 50% transactions have Write Set = 60 % of read set.

Table 4.13

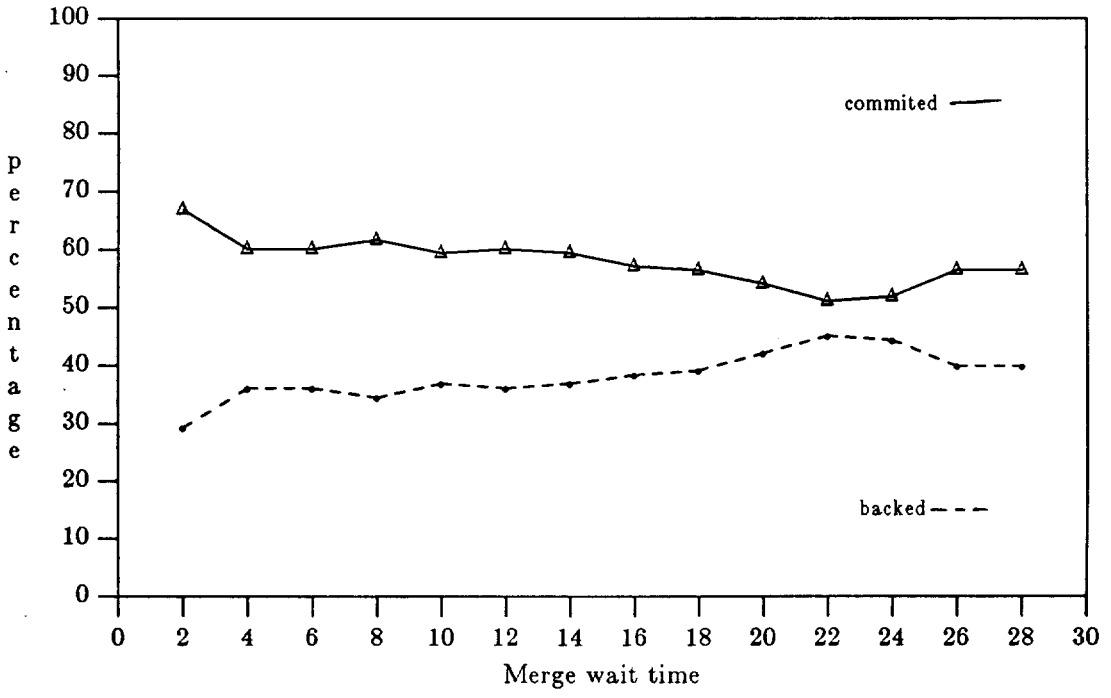
merge wait time	committed transactions	commit%	aborted transactions	abort%	backed out transactions	backed out%	total transactions
2	92	69.17	2	1.50	39	29.32	133
4	82	61.65	2	1.50	49	36.84	133
6	82	61.65	2	1.50	49	36.84	133
8	89	66.92	2	1.50	42	31.58	133
10	86	64.66	2	1.50	45	33.83	133
12	85	63.91	2	1.50	46	34.59	133
14	85	63.91	2	1.50	46	34.59	133
16	80	60.15	3	2.26	50	37.59	133
18	83	62.41	3	2.26	47	35.34	133
20	83	62.41	2	1.50	48	36.09	133
22	74	55.64	2	1.50	57	42.86	133
24	74	55.64	2	1.50	57	42.86	133
26	88	66.17	2	1.50	43	32.33	133
28	88	66.17	2	1.50	43	32.33	133



Database Size = 1000 items. Seed = 7774755
Number of Data items read = 5 to 20.
50% read-only & 50% transactions have Write Set = 80 % of read set.

Table 4.14

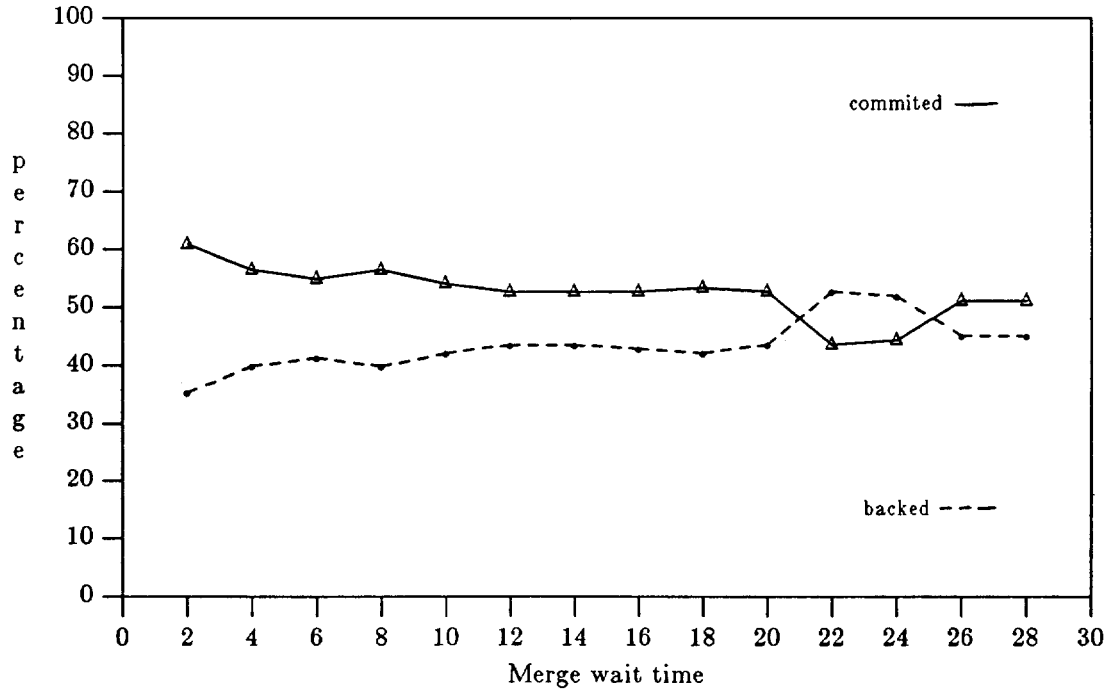
merge wait time	committed transactions	commit%	aborted transactions	abort%	backed out transactions	backed out%	total transactions
2	89	66.92	5	3.76	39	29.32	133
4	80	60.15	5	3.76	48	36.09	133
6	80	60.15	5	3.76	48	36.09	133
8	82	61.65	5	3.76	46	34.59	133
10	79	59.40	5	3.76	49	36.84	133
12	80	60.15	5	3.76	48	36.09	133
14	79	59.40	5	3.76	49	36.84	133
16	76	57.14	6	4.51	51	38.35	133
18	75	56.39	6	4.51	52	39.10	133
20	72	54.14	5	3.76	56	42.11	133
22	68	51.13	5	3.76	60	45.11	133
24	69	51.88	5	3.76	59	44.36	133
26	75	56.39	5	3.76	53	39.85	133
28	75	56.39	5	3.76	53	39.85	133



Database Size = 1000 items. Seed = 7774755
Number of Data items read = 5 to 20.
50% read-only & 50% transactions have Write Set = 100 % of read set.

Table 4.15

merge wait time	committed transactions	commit%	aborted transactions	abort%	backed out transactions	backed out%	total transactions
2	81	60.90	5	3.76	47	35.34	133
4	75	56.39	5	3.76	53	39.85	133
6	73	54.89	5	3.76	55	41.35	133
8	75	56.39	5	3.76	53	39.85	133
10	72	54.14	5	3.76	56	42.11	133
12	70	52.63	5	3.76	58	43.61	133
14	70	52.63	5	3.76	58	43.61	133
16	70	52.63	6	4.51	57	42.86	133
18	71	53.38	6	4.51	56	42.11	133
20	70	52.63	5	3.76	58	43.61	133
22	58	43.61	5	3.76	70	52.63	133
24	59	44.36	5	3.76	69	51.88	133
26	68	51.13	5	3.76	60	45.11	133
28	68	51.13	5	3.76	60	45.11	133



APPENDIX B

TEST FOR STATISTICAL SIGNIFICANCE

To check that the simulation results were significant, we conducted multiple simulation runs for different sequences of random numbers (i.e., with different seeds). The results for the repeated runs are in Tables 4.1.1 through 4.8.3 of Appendix A. The Tables 4.1.1, 4.1.2 and 4.1.3 correspond to Table 4.1.0 and so on. It can be seen that the results change very little for the different runs. The mean values for the percentages of committed, aborted and backed out transactions are tabulated in Tables 4.1.mean thru 4.8.mean. It can be seen from the corresponding graphs that the values for the different repeated runs are close to the corresponding mean values. Thus, the standard deviations across the runs would be extremely small.

To verify that the various sets of simulation results are statistically significant, we performed the *two tailed pairwise t-test* [Mid 76] on them. The t-test was performed by pairing the percentage of backed out transactions from four sets of results of two different runs for the same merge wait timer value and computing the mean and standard errors of the differences between the paired values. The value of t is the mean value of the differences divided by the standard error of the differences. We had 4

samples[1] (from 4 runs) of the back-out percentages. From Table A-1 in [Mid 76], the value of t must be greater than 3.182 for the difference between the means for two different back-out percentages to be considered statistically significant to the 95 percent confidence level.

The values of t for a representative selection of the Tables 4.1 through 4.15 are given below. For instance, the value of t for the comparison between the back-out percentages in Tables 4.1 through 4.1.3 and 4.3 through 4.3.3 for a merge wait timer value of 2 is 29.009. Thus the mean values for the above two collections of runs is considered to be statistically significant above the 99.999 confidence level. The other values that were computed were:

Comparison of Tables 4.1 through 4.1.3
Merge Wait Timer values: 2 and 12

mean1: 29.205000
mean2: 26.357500
mean of differences: 2.847500
standard error: 0.183411
value of t is: 15.525241
The means are different at more than 99.999% confidence level.

Comparison of Tables 4.1 through 4.1.3 and 4.3 through 4.3.3
Merge Wait Timer value: 2

mean1: 39.260002
mean2: 29.205000

[1]In statistics terminology, this implies that the *degrees-of-freedom* is 3.

mean of differences: 10.055000
standard error: 0.346610
value of t is: 29.009512
The means are different at more than 99.999% confidence level.

Comparison of Tables 4.1 through 4.1.3 and 4.3 through 4.3.3
Merge Wait Timer value: 12

mean1: 43.990005
mean2: 26.357500
mean of differences: 17.632502
standard error: 0.617956
value of t is: 28.533605
The means are different at more than 99.999% confidence level.

Comparison of Tables 4.1 through 4.1.3 and 4.3 through 4.3.3
Merge Wait Timer value: 2

mean1: 40.902500
mean2: 29.205000
mean of differences: 11.697501
standard error: 0.569548
value of t is: 20.538231
The means are different at more than 99.999% confidence level.

Comparison of Tables 4.1 through 4.1.3 and 4.5 through 4.5.3
Merge Wait Timer value: 12

mean1: 44.677502
mean2: 26.357500
mean of differences: 18.320002
standard error: 0.604988
value of t is: 30.281605
The means are different at more than 99.999% confidence level.

Comparison of Tables 4.1 through 4.1.3 and 4.6 through 4.6.3
Merge Wait Timer value: 26

mean1: 43.580002
mean2: 28.262501
mean of differences: 15.317500
standard error: 0.274208

value of t is: 55.860933
The means are different at more than 99.999% confidence level.

Comparison of Tables 4.3 through 4.3.3 and 4.5 through 4.5.3
Merge Wait Timer value: 2

mean1: 40.902500
mean2: 39.260002
mean of differences: 1.642501
standard error: 0.405121
value of t is: 4.054348
The means are different at more than 95% confidence level.

Comparison of Tables 4.3 through 4.3.3 and 4.5 through 4.5.3
Merge Wait Timer value: 12

mean1: 44.677502
mean2: 43.990005
mean of differences: 0.687499
standard error: 0.330690
value of t is: 2.078982
The means are different at more than 90% confidence level.

In the last case, the difference between the means is not as strong as in the other cases that are shown. However, since both are high conflict cases, it is possible that both give back-out percentages close to each other.

From the above analysis, we conclude that the results obtained by varying the parameters are significantly different from each other and are not dependent on the choice of the seed or the sequence of random numbers used to generate simulated events.

REFERENCES

[AHU 74] Aho,A.V., Hopcroft,J.E., and Ullman,J.D. *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974, pp. 209-217.

[ASC 85] Abbadi,A.E., Skeen,D., and Cristian,F. "An Efficiant, Fault-tolerant Protocol For Replicated Databases", proc 4th ACM SIGACT/SIGMOD Symp. Principles of Database Systems, March 85.

[AbTo 89] Abbadi,A.E., Toueg, S. "Maintaining Availability in Partitioned Replicated Databases", ACM Transactions on Database Systems, 14, 2, June 1989, pp. 264-290.

[Bahr 87] Bahra, R.S. "Performance Simulation of Optimistic Operations in Partitioned Distributed Database Systems" M.S. Thesis, Dept. of Computer Science, Southern Illinois University at Carbondale, Dec 1987.

[DaGa 81] Davidson,S.B. and Garcia-Molina,H. "Protocols for Partitioned Distributed Database Systems", Proc. Symp. Reliability in Distributed Software and Database Systems, July 1981.

[Davi 82] Davidson,S.B. "Evaluation of an Optimistic Protocol for Partitioned Distributed Database Systems", Tech. Rep. #299, EECS Dept., Princeton Univ., 1982.

[Davi 84] Davidson,S.B. "Optimism and Consistency In Partitioned Distributed Database Systems", ACM Trans. Database Systems 9, 3, Sept. 1984.

[EaSe 83] Eager,D.L., and Sevcik,K.C. "Achieving Robostness in Distributed Database Systems", ACM Trans. Database Systems 8, 3, Sept. 1983.

[Garc 81] Garcia-Molina,H. *Performance of Update Algorithms for Replicated Data*, UMI Research Press, 1981.

[Gray 79] Gray,J.N. "Notes on Database Operating Systems", in *Operating Systems: An Advanced Course*, Eds. R. Bayer, R.M. Graham, and G. Seegmuller, Springer Verlag, 1979. pp. 394-381.

[Kohl 81] Kohler,W.H. "A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems", ACM Computing Surveys, Vol 13, No. 2, June 81.

[Knu 69] Knuth, D.E. *The Art of Computer Programming, Vol 2: Seminumerical Algorithms*, Second Edition, pp 45-68.

[Lela 78] Lelann,G. "Algorithms for Distributed Data-sharing Systems which Use Tickets", Proc. 3rd Berkeley Workshop on Distributed Data Management and Computer Networks, 1978.

[Ma 86] Ma, Antony Vu. "Optimistic Partitioned Operation in Distributed Database Systems" Ph.D. Thesis, Computer Science Dept., University of Illinois at Urbana-Champaign, 1986.

[Mid 76] Middlebrooks, J.E. *Statistical Calculations, How to Solve Statistical Problems*, 1976.

[Ram 89] Ramarao, K.V.S. "Detection of Mutual Inconsistency in Distributed Databases", Journal of Parallel and Distributed Computing 6, pp. 498-514 (1989)

[TCB 83] Thanos, C., Carlesi, C., and Bertino, E. "Performance Evaluation of Two-Phase Locking Algorithms in a System for Distributed Databases", Proc. 3rd Symp. on Reliability in Distributed Software and Database Systems, 1983.

[Wrig 83] Wright,D.D. *Managing Distributed Databases in Partitioned Networks*, Ph.D. Thesis, Dept. of Computer Science, Cornell University, Sept. 1983.

[WiLa 84] Wilkinson,W.K., and Lai,M.-Y. " Managing Replicate Data in JASMIN", Proc. 4th Symp. on Reliability in Distributed Software and Database Systems, 1984.

[WrSk 83] Wright,D.D., and Skeen,D. "Merging Partitioned Databases", Tecnical Report 83-547, Dept of Computer Science, Cornell University, April 1983.