A PROTOCOL FOR CONSISTENT CHECKPOINTING RECOVERY FOR TIME-CRITICAL DISTRIBUTED DATABASE SYSTEMS

Junguk L. Kim, Taesoon Park, Prabaharan I. Swarnam
Department of Computer Science
Texas A&M University
and
Myung-Joon Kim
Electronics and Telecommunications Research Institute

ABSTRACT

This paper presents a checkpointing scheme which effectively copes with media failures for a distributed database system (DDBS), which employs the timestamp ordering scheme for concurrency control. In our scheme, normal transactions are executed during the checkpointing process without any interruption. The state of the database taken as a checkpoint by all sites in the system is consistent, so that fast recovery from media failures can be performed. Since our scheme does not interfere with the normal transaction processing, the scheme is essential for time-critical distributed database systems. Simulation results show that our scheme outperforms the existing schemes.

Index Terms — Distributed Databases, Checkpointing, Media Failure, Recovery, Consistency.

1 INTRODUCTION

In a database system, failures in the system might leave the database in an inconsistent state due to abnormal termination of transactions. Three types of failures cause this inconsistency; *transaction failure*, which is the abnormal termination of the transactions, *system failure*, which causes the loss of the contents in volatile storage, such as cache or main memory, and *media failure*, which causes the loss of non-volatile storage, such as the contents of database state stored in a secondary storage.

To restore the database to a correct state after such failures, recovery techniques are used as described in [1] and [2]. The most widely used recovery technique for failures is to reconstruct the correct database state by using the information of the database operations executed by the transactions, which is usually saved in a specified secondary storage space, called a *log*. Using the log information, the transaction operations are undone and/or redone to bring the database up to the most recent consistent state. However, in case of a media failure, both the database and the log in the secondary storage are lost. To recover from this type of failure, the database state is periodically copied into a secure storage, which is assumed not to be affected by any type of failure; we call this copy a *checkpoint*. For the system to resume its computation correctly, the correct and consistent state of the database must be checkpointed.

The data items of a database in a distributed database system (DDBS), are replicated and located over the different sites and a global transaction may access the data items in several sites and update them. Hence, the checkpointing in a distributed database system is quite complicated since the atomicity of the global transaction must be reflected in the checkpoint. That is, if a site S_i takes a checkpoint after a global transaction $T(a)$ has updated the database at S_i, while another site S_j takes the checkpoint before $T(a)$ has updated the database at S_j, then the global checkpoint produced is inconsistent and a recovery with this checkpoint may produce another inconsistent state. Moreover, if $T(a)$ reads one or more data items modified by another transaction, say $T(b)$, then the effects of $T(b)$ at all sites must also be reflected in the checkpoint for the checkpoints taken to be consistent. Hence, to avoid these inconsistencies, a system-wide coordination for checkpointing is required.

The checkpointing for DDBS has been extensively studied and various methods have been proposed and are classified, based on how the coordination among the sites are performed, into non-synchronized and synchronized [3]. In the non-synchronized schemes, each site takes a checkpoint when it finds its local database to be consistent, without respect to the global state of the distributed database. The globally consistent state of the DDB is then constructed during the recovery process by finding the consistent set of checkpoints among the saved ones [4]. In [5] and [6] the checkpointing is performed as part of the transaction-commit process and the recovery is performed by finding the set of transactions and their dependents whose operations have to be discarded and the respective sites, at which these transactions operated on, are informed to discard the effects of these transactions. Possible drawbacks of the above methods are that each site should maintain more than one checkpoint and may take a longer time to recover due to information exchange for finding a globally consistent point.

The checkpointing in the synchronized schemes is performed in coordination between the sites to result in a single checkpoint

which is globally consistent. [7] and [8] suggested schemes which produces globally consistent checkpoints, but the initiation of transactions during the checkpointing process at the sites are delayed until the checkpointing process has been completed. A site initialization method adopted to take checkpoints was proposed in [9], where taking of globally consistent checkpoints may delay transaction processing and may cause unnecessary aborts due to deadlocks. In another synchronized scheme, proposed in [3], the transactions to be included in the checkpoint are identified and the transactions that arrived after the checkpoint initiation at some sites but which should have been included in the checkpoint, are aborted. To avoid this delay and abortion of transactions, the idea to deviate the database during the checkpointing process was proposed by *Fischer, et. al.*, which would minimize interruption of transaction processing [10]. This idea was used by two schemes which tried to minimize the transaction processing interruption [11] and [12].

[11, 13] suggested a scheme where the largest timestamp of the transactions ongoing at the time of checkpoint initiation is found and the updates of transactions which have a lower timestamp than this selected value are included in the checkpoint. However, the transactions which should have been included in the checkpoint but arrive at a site after it takes a checkpoint, are aborted. In the scheme proposed in [14], the transactions carry the last checkpoint number of the originating site and the data items carry the last checkpoint number of the transaction which updated it, to detect consistency among checkpoints and to avoid interruption of transaction processing. However, it is possible that a few transactions may be aborted if they were initiated before the checkpointing process and their subtransactions arrived at sites after the data items at those sites had been checkpointed.

In this paper, we suggest a synchronized scheme for taking globally consistent checkpoints. Our scheme has been designed for a system employing the timestamp ordering policy for concurrency control. Our scheme uniformly identifies, using the largest timestamp value among active transactions, the transactions which are active at the time of checkpointing initiation at all sites, as well as all the transactions on which such active transactions depend identified. After all these identified transactions complete their execution at all sites, the checkpoints are taken including the effects of only the identified transactions [1]. As a result, the global consistency among the checkpoints taken at each site is maintained. Moreover, our checkpointing scheme does not interfere with the normal transaction execution by using the database deviation concept suggested in [10]; that is, no transactions are aborted or interrupted due to the checkpointing process. Our scheme is particularly essential for the systems which executes the time critical transactions. Extensive simulation was performed to compare our scheme with the scheme proposed in [11] and the scheme proposed in [7]. The simulation results show that our scheme outperforms both these scheme in all the cases where the simulation parameters were varied.

The rest of this paper is organized as follows: In Section 2, we explain the system model; The notion of globally consistent checkpointing and the problems in achieving a globally consistent checkpoint are identified in Section 3; We then present our globally consistent and non-intrusive checkpointing scheme in Section 4; In Section 5 we present the simulation results of our scheme and two other existing schemes and we conclude the paper in Section 6.

[1] We reasonably assume no long-lived transactions in the system, which is not quite suitable for the timestamp ordering concurrency control policy

2 SYSTEM MODEL

We consider a database system whose data items, the smallest accessible units by transactions, are physically distributed among N sites in the system and are replicated if necessary. The sites are connected through a communication network and the message exchange between the processes residing on different sites are assumed to be reliable, *i.e.*, the message which is sent by a site is delivered uncorrupted to the destined site within an arbitrary but finite time period. However, the message delivery order need not be maintained.

A transaction is the basic consistent unit of user activity to access the data items and consists of a set of subtransactions, which may be executed at several different sites. The site which is responsible for initiation and termination of a transaction is called the *coordinator* and the participating sites are called *cohorts*. For the termination of a transaction, we assume the two-phase commit protocol; that is, at the time of transaction completion, the coordinator asks the intention of commit or abort of each cohort and based on the intentions collected from all cohorts, the coordinator makes a decision and then all the cohorts follow the decision [15].

In our system model, we do not impose any restriction on when to flush the updates of transactions in the cache or main memory into the database state in the secondary storage. Hence, a recovery from a transaction failure or a site failure (not a media failure) may have to undo and/or redo the log [16].

A set of transactions can be executed concurrently at a site, and for these concurrent transactions to be executed serializably, the basic timestamp ordering concurrency control policy is assumed to be employed at each site [17]. For this, each transaction initiated at a site carries the logical clock value at the time of initiation of that site as its timestamp and all its subtransactions carry the same timestamp. The logical clock value at each site is maintained by the clock rule proposed in [18].

Each transaction can be uniquely identified by its timestamp concatenated with the identifier of its initiating site and all of its subtransactions carry the same identifier. For convenience, we use the following notations in this paper; a transaction with identifier a is denoted by $T(a)$, and a subtransaction of $T(a)$, which is executed at a site S_i, is denoted by $T_i(a)$. Similarly, a set of data items updated by $T(a)$ is denoted by $D(a)$, and a set of data items updated by $T_i(a)$ is denoted by $D_i(a)$. Note that $D_i(a) \subseteq D(a)$.

At site S_k, $T_k(a)$ is said to be *directly dependent on* $T_k(b)$, if $T_k(a)$ reads a data item updated by $T_k(b)$, and this dependency relation is denoted by $T_k(b) \rightarrow T_k(a)$. Similarly, the *transitive dependency*, which is the transitive closure of the direct dependency, is defined as follows: if $T_k(c) \rightarrow T_k(b)$ and $T_k(b) \rightarrow T_k(a)$, then $T_k(a)$ is said to be *transitively dependent on* $T_k(c)$ and this dependency relation is denoted by $T_k(c) \rightarrow^* T_k(a)$. Moreover, if for any k, $T_k(b) \rightarrow T_k(a)$, then $T(b) \rightarrow T(a)$; and if for any k, $T_k(c) \rightarrow^* T_k(a)$, then $T(c) \rightarrow^* T(a)$.

3 CONSISTENT GLOBAL CHECKPOINT

A *global checkpoint* of the system is defined as a collection of local checkpoints, one from each site; that is, $GCP(t) = (L_1(t), L_2(t), ..., L_N(t))$, where $GCP(t)$ is a global checkpoint having a unique identifier t and $L_i(t)$ is a local checkpoint at site S_i. $GCP(t)$ must be consistent in order for the system to restore the correct database state on recovery from a media failure.

One of the conditions for a $GCP(t)$ to be consistent is that the atomicity of each transaction (subtransaction) at a site S_i must be reflected in $L_i(t)$, because a transaction is a basic consistent unit of the database operation. That is, if any update of a transaction is included in $L_i(t)$, then all the updates by that transaction must be included in $L_i(t)$. Similarly, a transaction must also be globally atomic, *i.e.*, if any updates by a subtransaction $T_i(a)$ at site S_i are reflected in $L_i(t)$, then the updates of all other subtransactions of $T(a)$ must also be reflected in the corresponding local checkpoint of $GCP(t)$.

Another consistency constraint is that if the updates of a transaction, say $T_i(a)$, is reflected in $L_i(t)$, then every transaction on which $T_i(a)$ depends at site S_i, say $T_i(b)$, must reflect its updates in $L_i(t)$. This constraint is quite necessary because, by reflecting the updates of $T_i(a)$ in $L_i(t)$, the partial effect of $T_i(b)$ is implicitly reflected on $L_i(t)$. Then, for preserving the atomicity of the transaction $T(b)$, all effects of $T_i(b)$ must be included in $L_i(t)$.

More formally, the consistent global checkpoint can be defined as follows:

Definition 1: A global checkpoint $GCP(t)$ is said to be *consistent* if and only if the following conditions are satisfied :

C1: For any data item $x \in D_i(a)$, if the update of x is reflected in $L_i(t)$, then for every data item $y \in D_i(a)$, the update of y must be reflected in $L_i(t)$.

C2: For any data item $x \in D_i(a)$, if the update of x is reflected in $L_i(t)$, then for every data item $y \in D(a)$, the update of y must must be reflected in $GCP(t)$ at the corresponding sites.

C3: If the updates of $D_i(a)$ are reflected in $L_i(t)$, then for every $T_i(b)$, for which $T_i(b) \rightarrow^* T_i(a)$, the updates of $D_i(b)$ must be reflected in $L_i(t)$. □

The partial state of the database in the cache or main memory is assumed to be flushed into the secondary memory at any time, hence, it is not straightforward to catch the locally consistent state of the database instantly, as required in Condition C1. To cope with this problem, the concept of *database deviation* was introduced in [10], [11] and [12]. That is, after a checkpointing is initiated at S_i, the values of updated data items at the site are recorded in a separate data area with the timestamp of the transactions which updated that data item. We call this data area to store the data items to be deviated at S_i, the *Deviated Database Area* of S_i (DDA_i), which can be in the main and/or secondary memory. Moreover, all read operations must be consistently performed with the data items recorded in DDA_i. Hence, when the transactions to be included in $GCP(t)$ are consistently identified, only the updates made by

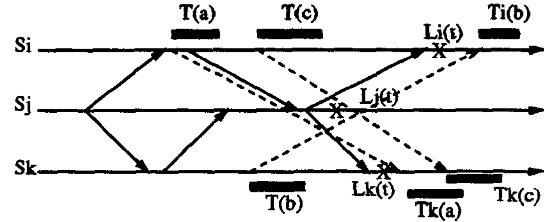


Figure 1: Inconsistent Checkpoint

those transactions can be separated and reflected in $GCP(t)$.

Let OTL_i be the set of transactions which are on-going at site S_i when a global checkpointing for $GCP(t)$ is initiated. Then, every $T_i(a)$ in OTL_i must be included in $GCP(t)$, since their partial effect might already have been reflected in the database state, and to satisfy the condition C2, every subtransaction of $T(a)$ must also be included in $GCP(t)$. Those transactions are uniformly identified by the largest timestamp among the on-going transactions of all sites in the system, in [11]. To collect the largest timestamp, two phases are used: the first phase to collect the largest timestamp among the transactions in OTL_i for every S_i in the system and the second phase to inform the largest timestamp among the collected ones. Then, by including all the transactions having the timestamp lower than the selected one into $GCP(t)$, the condition C2 and also the condition C3 can be satisfied at all sites, because all transactions on which the transactions in OTL_i depend must have lower timestamps than the selected one.

However, there is still some possibility of inconsistency, since some subtransactions of a transaction $T(a) \in OTL_i$ may arrive at another site, S_j, after all the transactions to be included in $L_j(t)$ are already identified, as shown in Figure 1. Then, $T(a)$ can result in an inconsistent $GCP(t)$ since its updates are included in $L_i(t)$ but not in $L_k(t)$. To cope with this possible inconsistency problem, such transactions are identified and aborted in [11], hence this checkpointing scheme does not satisfy the non-intrusive property as claimed. Moreover, to identify such transactions before their executions begin, two-phase initiations are used for each transaction initiation, hence, the initiation of normal transactions are delayed and the response time must be higher.

Hence, we introduce our non-intrusive and globally consistent checkpointing scheme, which consists of three phases: in the first phase, the transactions to be included in $GCP(t)$ are identified, any subtransactions which might cause the inconsistency problem are identified in the second phase, and finally in the third phase, each site actually takes its checkpoint. The detailed description of our checkpointing scheme is shown in the next section.

4 CHECKPOINTING RECOVERY

4.1 Checkpointing Scheme

In this section, we present our non-intrusive checkpointing scheme, in which a designated site, called *checkpointing coordinator*, initiates a checkpointing periodically and all other sites in the system, called *checkpointing cohorts*, participate in the

checkpointing process to produce a consistent global checkpoint.

Our scheme consists of three major phases: In the first phase, the transactions in OTL are identified at all sites in the system, by using the timestamp values of the on-going transactions at each site when the global checkpointing is initiated, and the timestamp value is informed to the coordinator. In the second phase, the coordinator after collecting the timestamp values from all the sites, selects the largest value among them, and forces every site to include the updates of transactions having a lower timestamp value than the selected one in the $GCP(t)$. Since we assume the *timestamp ordering concurrency control* policy, the dependency happens according to the timestamp order. That is, the transactions on which $T_i(a)$ depends must have a lower timestamp value than $T_i(a)$. Hence, when the largest timestamp value is selected, every site updates its local clock value to the one higher than the selected one. By doing so, we can guarantee that the transactions in OTL cannot be dependent on the transactions initiated after this point of time, since the former transactions have a timestamp value lower than the latter ones. However, there are still some active transactions, having a lower timestamp than the selected one, which is spawning the new subtransactions during the checkpointing coordination, and hence, we have to wait until all these transactions complete, for consistent checkpointing. Finally, in the third phase, each site then takes the checkpoint which forms a consistent $GCP(t)$.

For simplicity, we assume page level concurrency control, that is, a data item corresponds to a data page, and we also assume no site failures during the checkpointing coordination process in this section. Our algorithm, however, can be easily extended to handle other levels of concurrency control with slight modification in DDA space management. Handling of site failures will be discussed in the subsection 4.3. We now describe the tasks in the checkpointing process in detail by explaining each checkpointing phase:

Phase I:

(1) The checkpointing coordinator sends the ***checkpointing initiation*** message, $CHCK_{INIT}$, to all the sites in the system.

(2) Upon receipt of $CHCK_{INIT}$, each of the cohorts, say S_j, sets the value of CLT_j as the current local clock value and responds to the coordinator with CLT_j. Note that all the active transactions at site S_j have timestamps lower than or equal to the value of CLT_j.

(3) Each site S_i then creates OTL_i, which is the set of identifiers of the transactions which are on-going at that time.

(4) Each site S_i begins to deviate its database. DDA_i is implemented as a set of linked lists of data pages updated by the transactions with higher timestamps than CLT_i (one list per each data item), and maintained in the main memory until it reaches a certain limit. That is, the deviated data pages in the buffer space are marked and the buffer manager does not select those pages for replacement, unless the buffer space allocated for the DDA_i space reaches a limit. If the limit is reached, those data pages are flushed into the DDA_i residing on disk. For efficient access to the data pages in DDA_i in the disk, *DDA page map table* is maintained in the main memory. The detailed description of database deviation managed by the buffer manager are as follows:

- When a transaction $T_i(a)$ with a higher timestamp than CLT_i wants to update a data page containing x:
  - If that page is not found in the buffer, the DDA page map table is first checked. If an entry for any updated version of x is found in DDA page map table, then the most recent version of x in DDA_i is fetched. Otherwise, the data page in normal database area is fetched. Once the page is updated, it is marked as *Deviated Data Pages*(DDP).
  - If the page is found in the buffer, $T_i(a)$ cannot directly update the page due to the database deviation, unless the page has not been updated yet. If that page has been read only, then $T_i(a)$ updates that page and marked as DDP. Otherwise, the page is copied into another data page in the main memory and the updates is made on the new page, and the link between two pages are maintained if old page has been updated by a transaction with higher timestamp than CLT_i.
- When a transaction $T_i(a)$ with a higher timestamp than CLT_i wants to read a data page containing x:
  - If that page is not found in the buffer, the DDA page map table is first checked. If an entry for any updated version of x is found in DDA page map table, then the most recent version of x in DDA_i is fetched for read. Otherwise, the data page in normal database area is fetched. This fetched page need not to be marked as DDP, because it is read only.
  - If the page is found in the buffer, $T_i(a)$ read from the most recent version of the page.
- When a transaction $T_i(a)$ with a lower timestamp than CLT_i wants to read or update a data page containing x, the operation is performed on the pages in the normal database area.

  Note that if the requesting transaction $T_i(a)$ has timestamp lower than CLT_i and there is any version of x found in DDA_i in the main memory or in the disk, then $T_i(a)$ must be aborted due to the concurrency control.

Phase II:

(1) After receiving CLT_j from every site S_j, the coordinator selects the largest value among CLT_js and CLT_i, and assigns the value as the *globally largest timestamp* (GLT).

(2) It then adjusts its local clock with the value of GLT, and broadcasts GLT to all the checkpoint cohorts.

(3) Upon receipt of GLT, each cohort adjusts its current local clocks with the value of GLT, if its current local clock value is lower than GLT. Note that all the transactions initiated after this point have a larger timestamp than GLT.

(4) Each site S_i then adjusts its OTL_i to include the identifiers of the transactions which are on-going at this time and have a lower timestamp value than GLT. Note that these

transactions are finally included in $GCP(t)$.

(5) For each data item in DDA_i, if the timestamp of an updating transaction is smaller than GLT, the updates are reflected in the normal database space and deleted from DDA_i by unmarking the data page if it is in the main memory, or by discarding the space if it is in the disk. That is, the DDA_i now maintains only the updated versions of data items which are not included in $GCP(t)$. Moreover, for the updated versions of data items remaining in DDA_i, only the most recent version of the data items are maintained from this phase, by deleting the older versions.

(6) S_i then waits until all transactions in OTL complete their execution and then sends a $CHCK_{OK}$ message to the coordinator.

Phase III:

(1) After the checkpoint coordinator receives the $CHCK_{OK}$ from all the cohorts, it broadcasts a $CHCK_{COMMIT}$ message, which informs the sites to begin taking the actual checkpoint.

(2) Upon receiving the $CHCK_{COMMIT}$ message, each site S_i begins taking the checkpoint using the following procedure :

- A new checkpoint is taken at the site by copying the database state in the secondary memory into a secure storage, and associates a new identifier for the global checkpoint. This copying operation is performed concurrently with the normal data base execution not to interfere with the normal transaction processing.
- The current state of the database in the main memory is reflected into the database residing in the secondary memory or into the checkpointed database in the secure storage, according to the timestamp of the transaction which updates the data page most recently, when the page has to be flushed out for the replacement.

4.2 Correctness

We first show that our checkpointing scheme produces a globally consistent checkpoint.

Theorem 1: A global checkpoint $GCP(t)$ created under our scheme is consistent.

Proof : Assume that $GCP(t)$ is not consistent. It then should have violated at least one of the conditions C1, C2 and C3.

Case 1) Suppose that condition C2 is violated. In this case, without loss of generality, we can assume that $T_i(a)$ was included in $L_i(t)$ and $T_j(a)$ was not included in $L_j(t)$, for $T(a)$. Since both of the transactions have the same timestamp, they must be included in $GCP(t)$ in both sites, unless $T_j(a)$ arrives at a site S_j after the checkpointing is terminated. In order for $T_i(a)$ to be included in $L_i(t)$, it should have been in adjusted OTL_i. Since all the transactions in OTL_i complete their execution when informing the $CHCK_{OK}$, $T_i(a)$ should have already completed its execution, before the transmission of $CHCK_{OK}$. Then, under our assumption of two-phase commit protocol, it is not possible for $T_j(a)$ to arrive at site S_j after the termination of the checkpointing. A contradiction.

Case 2) Suppose that C3 is violated. There should have been transactions $T_i(a)$ not included at $L_i(t)$ and $T_i(b)$ included at $L_i(t)$, such that $T_i(a) \rightarrow^* T_i(b)$ and timestamp of $T_i(b)$ is lower than GLT. Since we assume the timestamp ordering concurrency control policy, the timestamp of $T_i(a)$ must be less than $T_i(b)$. Hence, the timestamp of $T_i(a)$ is less than GLT, and it must have been decided that $T_i(a)$ should be in $GCP(t)$. Moreover, the checkpointing is actually taken when all the transactions, having smaller timestamp than GLT, have completed, hence, $T_i(a)$ should have been committed. Then, $T_i(a)$ should have been included in $L_i(t)$. A contradiction.

Case 3) Suppose that condition C1 is violated by $T_i(a)$. The updated versions of the data items accessed by $T_i(a)$ are made in DDA_i, after a checkpointing is initiated. Hence, if $T_i(a)$ is decided to be in $GCP(t)$, all updated version in DDA_i of $T_i(a)$ should have been transferred to the normal database area before $L_i(t)$ was taken. A contradiction.

Therefore, our checkpointing scheme produces a consistent global checkpoint. □

We now show that our checkpointing scheme terminates within a finite period of time.

Theorem 2: A global checkpointing coordination under our scheme will terminate within a finite period of time.

Proof : The first phase of our checkpointing scheme consists of collecting the current local clock values from all sites. Since no site failures are assumed during the checkpointing and the message delay is also assumed to be finite, the clock value can be collected within a finite time. The second phase can finish when all the transactions having a smaller timestamp than GLT are completed at all sites. Since every site changes its clock value, when it receives GLT from the coordinator, with the value larger than GLT, the transactions having a lower timestamp value than GLT can be initiated only before this point of time. Hence, each site can have a finite number of such transactions and the execution time for these transactions is finite. Hence, the second phase can also be completed within a finite time. When the third phase begins, all transactions to be included in the checkpoint should have already committed. The time to deliver the $CHCK_{COMMIT}$ messages and the checkpoint saving time at each site in the third phase are also finite. Therefore, our global checkpointing coordination process can be completed within a finite period of time. □

4.3 Recovery and Failure Handling

When a site S_i recovers from a media failure, it first restores its recent checkpoint, $L_i(t)$, and forces all other sites to restore their checkpoints corresponding to $GCP(t)$. For the consistent recovery from a media failure, it is necessary for any $GCP(t)$ to be consistent. However, if any of the sites fail during the global checkpointing and cannot take a checkpoint, then the global checkpoint produced cannot be used for the recovery since some committed transactions which are included in a $L_i(t)$ are not included in the most recent checkpoint of that failed site (*i.e.* violation of C2 of the definition of consistency).

We here assume that the failure of a site can be detected by other sites within a finite time. Hence, in our scheme, the failures during the global checkpointing are handled as follows:

- If a checkpointing cohort fails during the first or second phase of the checkpointing: Then, the checkpointing coor-

dinator broadcasts the *checkpointing abort* message instead of GLT or $CHCK_{COMMIT}$

- If a checkpointing cohort fails during the third phase of the checkpointing: To handle this problem, we can introduce one more phase in our checkpointing scheme to ensure the completion of the checkpointing. After each site actually saves its checkpoint on a stable storage, it sends the *checkpointing complete* message to the coordinator. The coordinator after collecting this message from all the sites in the system sends the *final checkpointing commit* message to all of them. Upon the receipt of the *final checkpoint commit*, each site discards the old checkpoint and mark the new checkpoint as a globally consistent checkpoint. Hence, each site in the system need to maintain only one checkpoint which is globally consistent.
- If a checkpointing coordinator fails during the checkpointing: Then, all the checkpointing cohorts decide to abort the on-going checkpointing process.

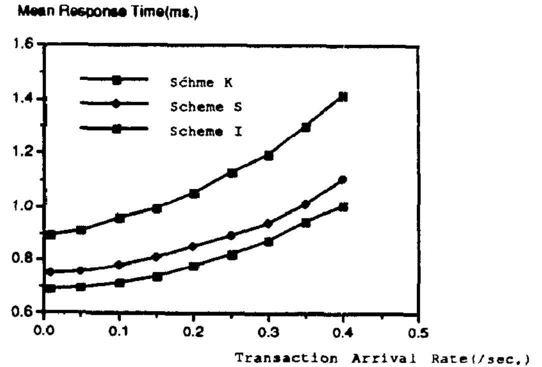
5 DISCUSSION

Extensive simulation was performed to compare the performance of our scheme with two other schemes proposed in [7] and [11]. For simplicity, we denote the scheme suggested in [11] as scheme S, the scheme mentioned in [7] as scheme I and our scheme as scheme K. Scheme I blocks the initiation of new transactions, once the checkpointing is initiated at each site. After the transactions which are active at the time of checkpointing initiation have completed their execution, the checkpoint is taken and the transactions which have arrived during the checkpointing can get serviced. Scheme S uses a two-phase checkpointing mechanism, where it collects the largest timestamp of the ongoing transactions in the system and all updates by transactions with a timestamp lower than this largest value is included in the checkpoint. However, in this scheme, some transactions which initiated before the checkpointing may get aborted if its subtransactions arrive at some remote sites after their checkpointings are completed.

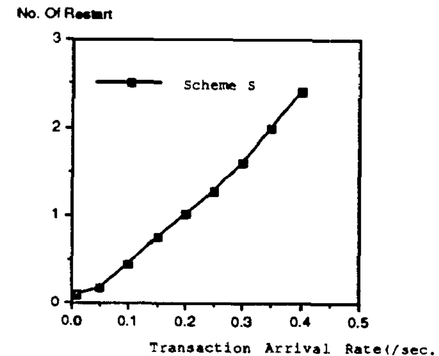
5.1 Simulation Model

A distributed database system with 10 sites connected through a general network was simulated. Each site consists of a CPU, a main memory, a disk, and a secure storage device for the checkpoint, each of which is managed by a scheduler, buffer manager, disk manager, and recovery manager, respectively. In our simulation, it is assumed that CPU and disk processing time for data item is 5 milliseconds and 24 milliseconds respectively. It is also assumed that 128 pages of buffer space is available in the main memory for data caching, and the buffer management policy is LRU. The granularity of the data items accessed by the transactions is assumed to be pages. The database residing at each site is assumed to have 5000 data items.

A global transaction is initiated at each site with an interarrival time which follows an exponential distribution with a rate λ_t and spawns subtransactions at other sites. The number of sites for subtransactions and to which the subtransactions are sent are randomly selected for each transaction. Each subtransaction consists of a sequence of write requests for data items. A normal distribution with a mean N_d and standard deviation



(a)



(b)

Figure 2: Mean Transaction Response Time and Number of Transaction Restart Vs. Transaction Arrival Rate

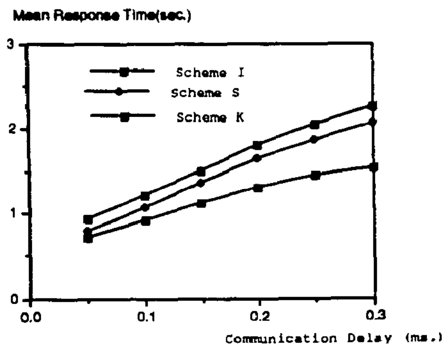
δ_d is used to choose the number of data items that a transaction accesses. We assume the two-phase commit procedure for each global transaction, for all schemes. In scheme S, the two-phase initiation mechanism is also used for transactions during the checkpointing process. However, there is no restriction for initiation of subtransactions in scheme K and scheme I.

Message transmission delay of a link in the network follows an exponential distribution with a rate λ_c, and to be more realistic, the delay is bounded by a certain minimum and maximum values.

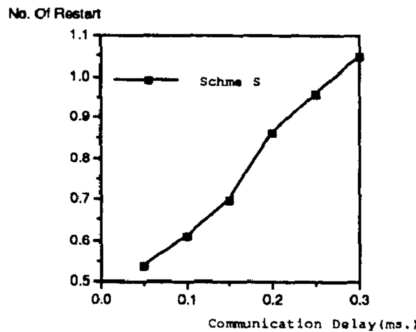
The checkpointing process is initiated by one designated site with a fixed time interval. The DDA space management during the checkpointing coordination exactly follows the algorithm described in section 4, and the same management scheme is also simulated for scheme K and scheme S. No site failure is assumed in the current simulation. The simulation program is written in C language and was executed on Sun Sparc stations under Sun OS 4.1.1.

5.2 Simulation Results

The main performance index used in our simulation is the *mean response time* of the transactions executed during the checkpointing coordination process. This performance index of the three schemes is compared through simulation by varying the parameters, such as transaction arrival rate at each site(λ_t), the



(a)



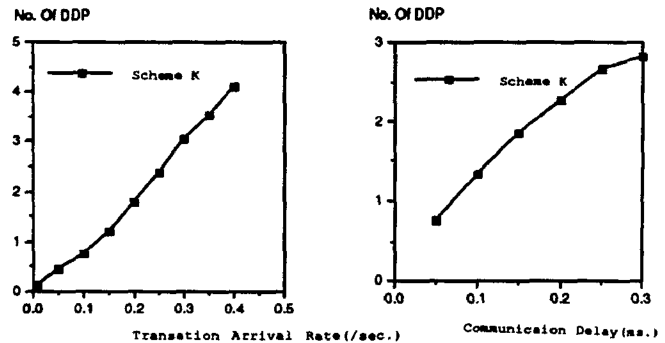
(b)

Figure 3: Mean Transaction Response Time and Number Of Transaction Restart Vs. Communication Delay

mean communication delay of the network($\frac{1}{\lambda_c}$) and the mean number of data items accessed by each subtransaction at a site(N_d). In our simulation, the effect of the actual checkpoint saving time is not considered, since all three schemes have the same overhead for saving the checkpoint.

We first compare the mean response time of the schemes by varying the value of λ_t, from 0.01 transactions/second to 0.4 transactions/second. Figure 2.a shows the mean response times of the schemes where $\frac{1}{\lambda_c}$ is 50 millisecond, N_d is 6 and δ_d is 2. It can be seen that scheme K outperforms the other schemes throughout the variation in λ_t. The relatively poor performance of scheme I is due to the fact that the initiations of new transactions are blocked during the checkpointing coordination. Since the waiting time for the transaction to be initiated is considered in the response time, the transactions under the scheme I experienced the longest response time in all cases. We can also notice that the performance difference becomes larger for scheme I as the transaction arrival rate increases. As the transaction arrival rate becomes higher, there can be more active transactions at the time of checkpointing initiation, which has to complete their execution to take a checkpoint, and hence, the checkpointing coordination time becomes longer, which in turn cause longer delay for the newly arrived transactions.

The performance difference between scheme K and scheme S is mainly due to the communication delay experienced for two-phase initiation in scheme S. Another factor affecting the performance difference between scheme K and scheme S is the restart during the checkpointing coordination in scheme S, be-



(a) (b)

Figure 4: Number Of Deviated Data Pages Vs. Transaction Arrival Rate and Communication Delay

cause the response time of those transactions includes the time spent for two-phase initiation before its abort. Figure 2.b shows the number of restarts in scheme S for each site per checkpoint against the transaction arrival rate. Since the number of restarts increase according to the the transaction arrival rate, the performance difference between scheme K and scheme S widens slightly, as the transaction arrival rate increases.

In Figure 3.a the response time is compared with different communication delay time, when λ_t is 0.1 transactions/second, N_d is 6 and δ_d is 2. Communication delay affects the response time of the transaction because of the two-phase commit, which is common for all three scheme, hence, the increase in communication delay causes the longer response time. In scheme S, the communication delay also affects the time for two-phase initiation, and hence, the performance difference between scheme K and scheme S become larger as the communication delay increases. Moreover, as the communication delay increases, the possibility that subtransaction initiation requests sent by the global transactions arrive at the remote sites after their checkpointings is increased. This increase in the number of transaction restarts per checkpoint at each site can be seen in Figure 3.b. This steep increase in the transaction restarts explains the rapidly widening gap between scheme K and scheme S in Figure 3.a.

From the simulation results, our scheme is found to outperform the other schemes in terms of response time, and the performance difference becomes larger as the transaction arrival rate and the communication delay increases. Such performance difference exists mainly because our scheme never blocks/aborts the transactions and does not requires the two-phase initiation to achieve a consistent checkpoint, thus providing a non-intrusive checkpointing.

Another factors to be considered for performance of the system is the storage overhead and the number of messages exchanged during the checkpointing. Since our checkpointing coordination requires one more phase compared with the scheme S, extra space and messages are required during that phase. To validate our scheme, we also count the average number of deviated data pages maintained for phase III of our scheme throughout the simulation. Figure 4.a shows the number of deviated data pages per each site and each checkpointing against the transaction arrival rate, and Figure 4.b shows the same per-

formance indices against the communication delay. As shown in both figures, the space overhead is not that severe in our simulation environment. Moreover, the number of messages required for phase III of our checkpointing process is 2^*N, where N is the number of sites in the system, and hence the total number of messages exchanged during our checkpointing process is still $O(N)$.

6 CONCLUSIONS

In this paper, we presented a checkpointing scheme for media failure in a distributed database system with timestamp ordering concurrency control, which can be executed concurrently with normal transaction processing, that is, no transactions are aborted or no transaction initiations are delayed due to the checkpointing process. Our scheme always produces a globally consistent checkpoint, so that it can be used for recovery from media failures, where all sites are forced to start from the consistent checkpoints.

Our scheme identifies, in three phases, completion of the transactions active at the time of checkpointing initiation and the transactions on which the active transactions depend, and hence the checkpoints can be safely taken including only the effects of those transactions. For this, the transactions executing during the checkpointing period are provided with a deviated database area, so that each update of newly initiated transactions, after the initiation of checkpointing, are saved separately and only the updates of necessary transactions can be appropriately included in the checkpoint. The overhead involved in providing this service is some space requirement to retain each value of a data item updated by the active transactions, during the first two phases of the checkpointing scheme. As shown in simulation results, the space required in our scheme during the first phase of the checkpointing coordination is almost equal to the scheme proposed in [11]. Moreover, the maximum difference in space required by our scheme over [11] during the second phase and third phase is not great.

In summary, our scheme does not impose any restriction in initiating subtransactions through a two-phase initiation mechanism, which is required for the scheme in [11]. The global transaction in our scheme is not forced to send the subtransactions to the sites at the start of the transactions, but the scheme in [11] requires that the global transactions send the subtransactions at the beginning of its execution.

References

[1] Verhofstad, J.S.M., "Recovery Techniques for Database Systems," *ACM Computing Surveys* Vol.10, No.2, pp. 167–195, June 1978.

[2] Haerder, T. and Reuter, A., "Principles of Transaction-Oriented Database Recovery," *ACM Computing Surveys*, Vol.15 No.4, pp. 287–317, December 1983.

[3] Dadam, P. and Schlageter, G., "Reconstruction of Consistent Global States in Distributed Databases," *Information Processing, North-Holland Publishing Company, Amsterdam*, pp. 191–201, 1980.

[4] Dadam, P. and Schlageter, G., "Recovery In Distributed Databases Basaed On Non-synchronized Local Checkpoints," *Information Processing, North-Holland Publishing Company, Amsterdam*, pp. 457–462, 1980.

[5] McDermid, J., "Checkpointing and Error Recovery in Distributed Systems," *in Proceedings of the 2nd International Conference on Distributed Computing Systems*, pp. 271–282, April 1981.

[6] Kuss, H., "On Totally Ordering Checkpoints in Distributed Databases," *in Proceedings of the ACM SIGMOD*, pp. 293–302, 1982.

[7] Jouve, M., "Reliability Aspects in a Distributed Database Management System," *in Proceedings of AICA*, pp. 199–209, 1977.

[8] Moss, J., "Checkpoint and Restart in Distributed Transaction Systems," *in Proceedings of the 3rd Symposium on Reliability in Distributed Software and Database Systems*, pp. 85–89, 1983.

[9] Attar, R., Bernstein, P., and Goodman, N., "Site Initialization, Recovery, and Backup in a Distributed Database System," *IEEE Transactions on Software Engineering*, Vol.SE-10, No.6, pp. 625–650, November. 1984.

[10] Fischer, M., Griffeth, N., and Lynch, N., "Global States of a Distributed System," *IEEE Transactions on Software Engineering*, Vol. SE-8, No.3, pp. 198–202, May. 1982.

[11] Son, S. and Agrawala, A., "Distributed Checkpointing for Globally Consistent States of Databases," *IEEE Transactions on Software Engineering*, Vol.15, No.10, pp. 1157–1167, October. 1989.

[12] Lim, J. and Moon, S., "A Checkpointing Scheme for Heterogeneous Distributed Database Systems," *in proceedings of the 11th International Conference on Distributed Computing Systems*, pp. 608–615, May 1991.

[13] Son, S. and Agrawala, A., "A Non-Intrusive Checkpointing Scheme in Distributed Database Systems," *in Proceedings of the 15th International Symposium on Fault Tolerant Computing*, pp. 99–104, 1985.

[14] Pilarski, S. and Kameda, T., "A Novel Checkpointing Scheme for Distributed Database Systems," *in ACM Proceedings of the 9th Principles of Database Systems*, pp. 368–378, 1990.

[15] Gray, J., "Notes on data base operating systems. in: Operating systems," *Lecture Notes in Computer Science 60*, pp. 393–481, 1978.

[16] Bernstein, P., Goodman, N., and Hadzilacos, V., "Recovery Algorithms For Database Systems," *Information Processing, North-Holland Publishing Company, Amsterdam*, 1983.

[17] Eswaran, K., "The Notion of Consistency and Predicate Locks in a Database System," *Communications of the ACM*, pp. 624–633, November 1976.

[18] Lamport, L., "Time, Clocks and the Ordering of Events in a Distributed System," *Communications of the ACM 21(7)*, pp. 558–565, July 1978.