MIT Sloan School of Management

Working Paper 4416-03
CISL 2002-02
March 2003

Asynchronous Backup and Initialization of a Database Server for Replicated Database Systems

Subhash Bhalla and Stuart E. Madnick

Asynchronous Backup and Initialization of a Database Server for Replicated Database Systems

Subhash Bhalla[1] and Stuart E. Madnick[2]

[1] Database Systems Laboratory, The University of Aizu,
Aizu-Wakamatsu City, Fukushima 965-8580, Japan
bhalla@u-aizu.ac.jp
[2]Sloan School of Management, Massachusetts Institute of Technology,
Cambridge, Massachusetts, MA 02142, USA
smadnick@mit.edu

AUTHORS' ADDRESS FOR ALL CORRESPONDENCE :
Subhash Bhalla
Database Systems Laboratory,
The University of Aizu,
Aizu-Wakamatsu, Fukushima 965-8580, Japan
email : bhalla@u-aizu.ac.jp
Fax : +81(242)37-2753

(Final version 29 March 2003)
Database Systems Laboratory,
The University of Aizu,
Aizu-Wakamatsu, Fukushima, PO 965-8580

Asynchronous Backup and Initialization of a Database Server for Replicated Database Systems

Subhash Bhalla and S. E. Madnick

Abstract : A possibility of a temporary disconnection of database service exists in many computing environments. It is a common need to permit a participating site to lag behind and re-initialize to full recovery. It is also necessary that active transactions view a globally consistent system state for ongoing operations. We present an algorithm for on-the-fly backup and site-initialization. The technique is non-blocking in the sense that failure and recovery procedures do not interfere with ordinary transactions. As a result the system can tolerate disconnection of services and reconnection of disconnected services, without incurring high overheads.

Keywords : Concurrency control, distributed algorithms, distributed databases, non-blocking protocols, serializability.

1 Introduction

The need for high availability of database services is on the increase. It is accompanied by the the problem of growing sizes of databases. In such an environment, customers frequently require backup of a large file or archive of a database. The purpose of the database backup may vary from application to application. Usually, consumers of database services require backup for protection against one or both of the following items,

- protection from media failures, or
- protection from application errors.

Traditionally, a database service depends on a stable database (S). To protect against failures involving data in S, the recovery system provides -

1. an additional copy of the database (called a backup B), and
2. a recovery log that is applied to the backup to roll its state forward to the desired committed state.

To achieve a recovery after failure, the media recovery system first restores S to a recoverable state by copying B. The recovery log operations are applied to the restored state to roll forward to a more recent time (desired time, usually the latest time before the crash). High availability requires that the database backup activity be performed on-line. The subsequent recovery also needs to be on-line, to reduce the database outage time. Conventional techniques of doing incremental backup incur high overheads during the backup activity [4, 24, 27]. To the best of our knowledge few studies consider on-line procedure for recovery. Lomet [24] points out that media failure is subsequent to a failure and may be acceptable as an off-line procedure. Most available research studies do not incorporate recovery from application errors. Recovery from application errors is increasing in significance with increasing volume of E-Commerce and other financial service applications.

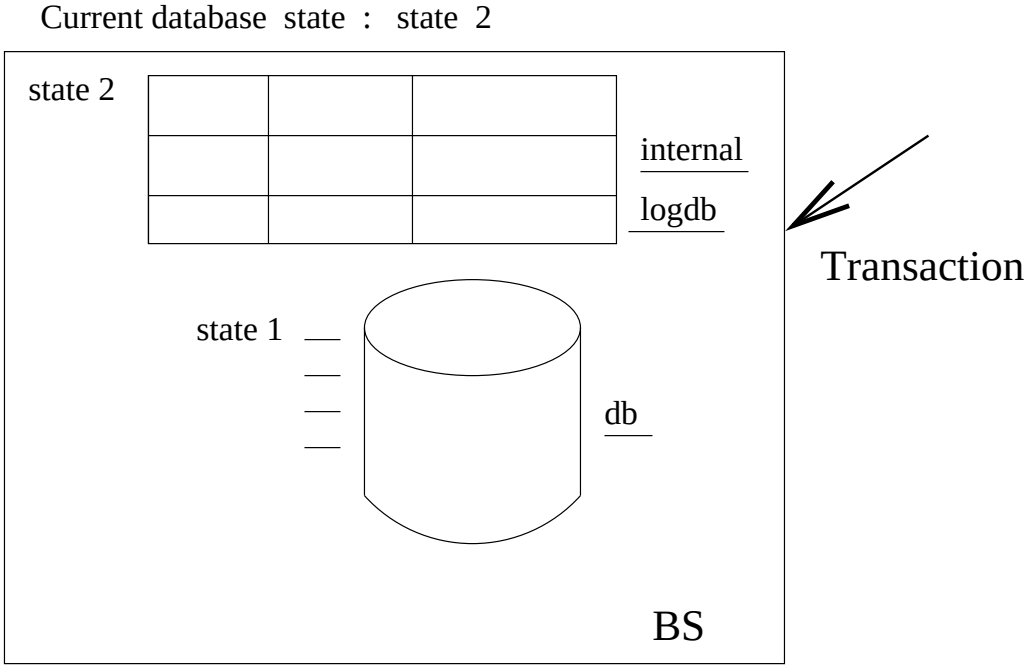

Figure 1: A Backup Server in 'internal disconnect' state

1.1 Backup and Restoration

In order to present a conceptual model, we introduce new paradigms. With successive refinement of the model, we demonstrate the available techniques.

We consider a database service through a replicated database system (a duplicate database is named as backup server (BS) in case of a disconnection until recovery). In absence of failures, there is no BS. Each site functions as a normal site in a replicated database system. The first paradigm considered by us is a state of 'internal disconnect'. A site can disconnect itself and be classified as a BS (for repair or reorganization) with virtually no visible effects for other external sites. It stops to provide the functions of a transaction manager (TM). It redirects arriving new update transactions to another site. It can receive updates from on-going global and local transactions and gather these as pending updates, in the form of a log file, which are to be subsequently appended to the database. In this situation, the state of the database system has two levels. The first is the level of the database at the time of the disconnect (lower level). The second level is the current level of the database which is represented by the level of the external database sites (correct level). To provide a read-only transaction execution service (servicing read requests) during the 'internal disconnect' state the site can first read the data from the database (db). Next, the transaction needs to validate and upgrade the data contents against the new database items in its log file called (internal logdb) (Figure 1). In a large replicated database system, only the BS site maintains a (internal logdb).

In order to achieve an implementation possibility for the paradigm in practice, and for sake of a seamless integration of concurrency control checks on completion of internal disconnect, we consider that the database system uses validation based (optimistic concurrency control (OCC)) (This assumption can be relaxed later on (Appendix A)). The OCC is convenient

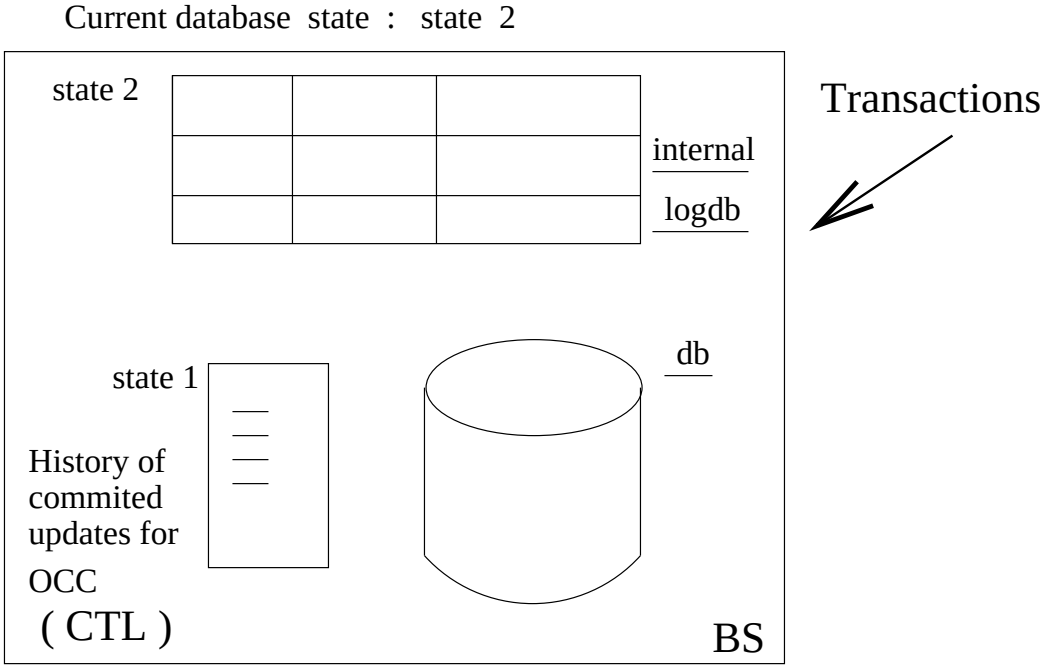

Figure 2: Synchronization and recovery information at the BS

because it also uses a similar structure of a log (a history of committed transactions) for performing concurrency control functions (Figure 2). The structure of the above two data structures can be made to be compatible for functions of recovery and concurrency. In the event of recovery at a later point of time, to be able to reconnect the BS system to a real-world database system, a mechanism for integration is obtained in a straight forward manner by combining the history in committed transactions list (CTL) and internal logdb contents (Figure 3). Earlier studies point to the need for such an integration [19, 32, 33, 34].

Consider a second paradigm to safe guard against communication failures. An 'external disconnect' state in which the external (other) database site maintains a external logdb to protect the update data for the disconnected member. This site continues to advance its database states. The disconnected site maintains its existing state. The external accesses to the database at the disconnected database server pass through an additional stage of accesses and concurrency checks (with reference to the external logdb) at the external end to achieve a consistent state. At the beginning, it is assumed that there is only one database site that is in a disconnected state All active database sites maintain an external logdb and aim at improving the link and accesses at the BS.

The external log (external logdb) is a log of the differences between the states of two database servers (Figure 4.). Participating external sites perform additional concurrency control. Such a system supports integration of procedures to facilitate the lag of state between the two types of servers. Based on the two paradigms above, a database backup and initialization model has been proposed in the subsequent sections. The adoption of the backup and recovery model can reduce the database outage time in many cases. For example, in some cases the 'external' or 'internal' logdb may be small in size, or it may have entries that may confine to

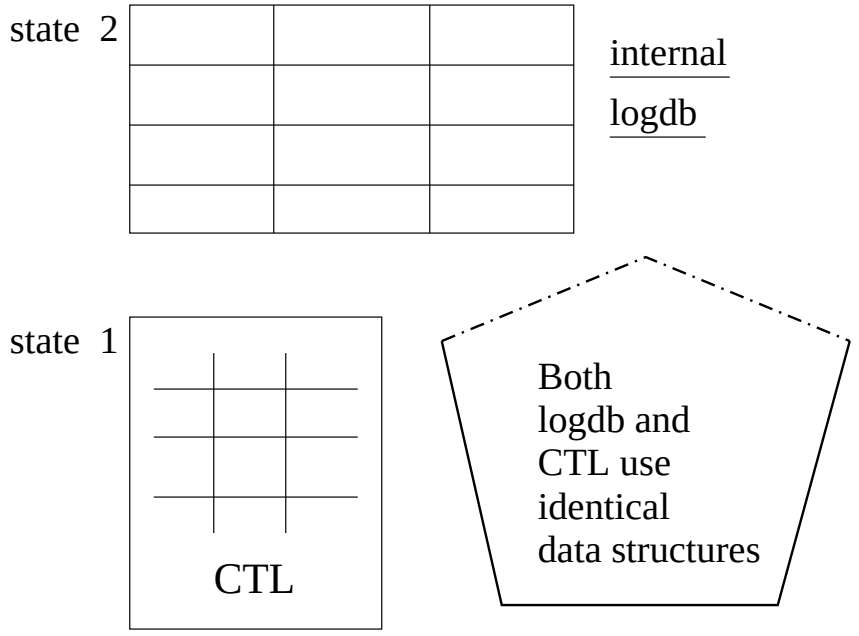

Figure 3: Integration of synchronization and recovery information

a localized part of the database. Transactions from other parts may pass through the checks with a quick validation.

2 Existing High Speed Database Backup Techniques

There have been a number of earlier studies that aim at transaction level backup and recovery [21, 19, 18]. There have been many recent research studies that concern the global-reading of a consistent copy of a database [22, 23, 24]. Earlier efforts in the area of database backup considered obtaining a copy of the entire database as a special case of *Long-lived transactions*.

Lomet [24] considers buffer-management level recovery overheads and generation of backups at a high-speed. The paper provides an overview of earlier research efforts on this topic. Mohan and Narang [27] have considered high speed backup for archival purpose. These studies consider incremental backup of an entire database. This procedure is based on reading small portions of the database without holding read-locks on the (entire) contents (that have been read by the 'database copy algorithm'). An overview of related research activities is described and compared in [4]. The authors point out that although the database copying accesses available database segments, the process needs to block some of the on-going updates that attempt to occur in parallel. The high overheads due to strict conflict graph serializability criteria as in [4] have been partially avoided in [24] by considering page level operations during backup procedure. The management of conflicts based on instance graph serialization is a buffer management activity based on logging operations. It does not consider recovery based on after-image logging. The study can not meet the possibilities of on-demand backup generation for recovery from application errors [7, 8]. It also does not consider generation of a snapshot of the database at the end of a given period, or termination point.

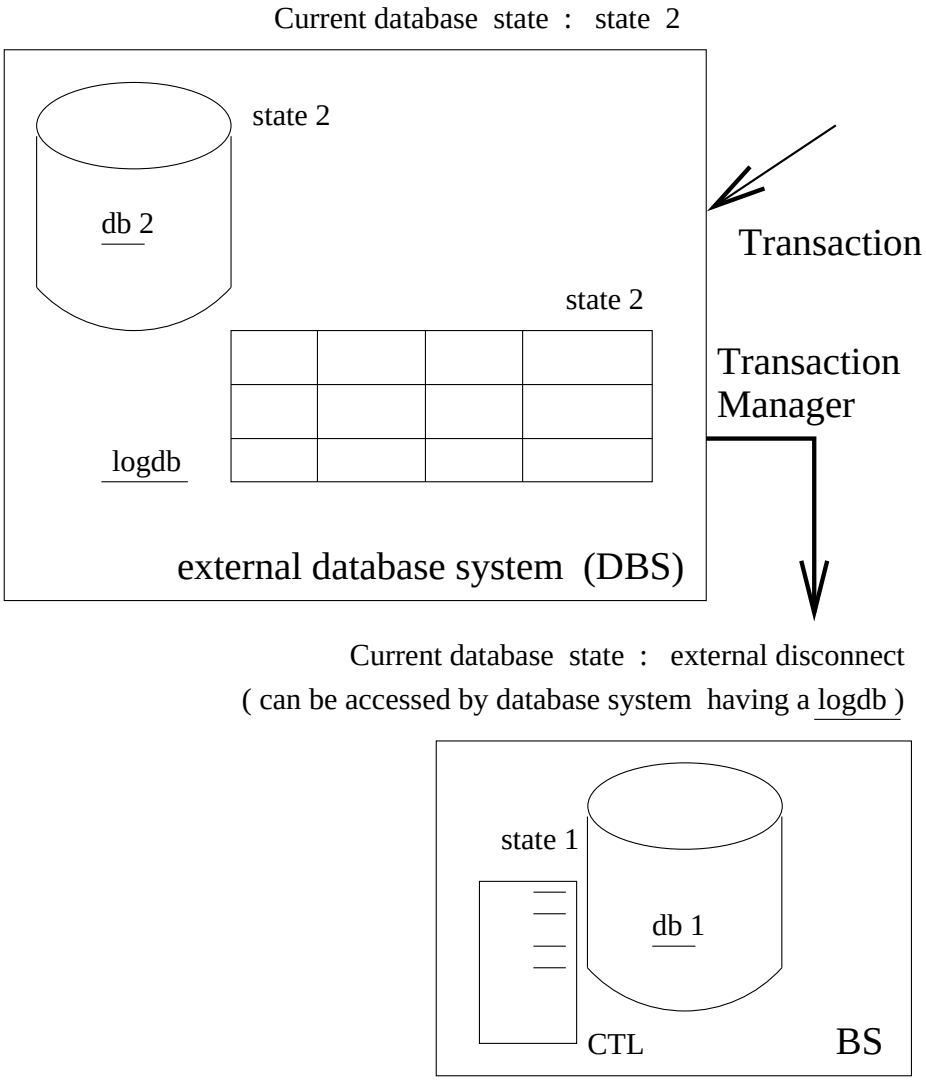

Figure 4: Backup Server in 'external disconnect' state

Supporting long-lived read transactions under the rules of concurrency control based on 2-phase locking, had been studied earlier by [11, 1, 2, 31]. Adoption of 2-phase locking, renders a large part of the database inaccessible, and is not efficient for long running transactions. Using multi-versions for supporting long running transactions has been studied by [5, 26].

The idea to consider asynchronous processing is being studied by many recent studies. Studies point to synchronized transaction processing activity as the cause which causes blocking [15, 20, 9, 10, 30]. A number of recent research studies mention that delays exist in case of replicated databases with respect to synchronization activity and point out the need for improvements in techniques [3, 12, 13]. Currently, there are few proposals in this area of research [15, 20].

3 Replicated Database System Model

The replicated database system (RDBS) consists of a set of data items (the smallest accessible unit of data). The idea of data within a RDBS, is same as in conventional systems [6]. Each data item is stored at each database site (full replication). However, this assumption does not restrict the algorithm in any way. It will be relaxed to consider more general cases later (section on Implementation Issues). The RDBS consists of two types of sites. The sites that perform transaction processing are called normal sites. The other sites that can not perform the normal functions related to processing of transactions due to failures or recovery are called 'disconnected sites' or 'recovering sites'. These sites within the replicated database management system (RDBMS), are called 'backup servers' (BS).

Transactions are identified as $T_i, T_j, \ldots$; and sites are represented by $S_k, S_l \ \ldots$; where, $i, j, k, l \ \ldots$ are integer values. The database sites are connected by a computer network. Each site supports a transaction manager (TM) and a data manager (DM). The TM supervises the execution of the transactions. The DMs manage individual databases. The network is assumed to detect failures, as and when these occur. When a site fails, it simply stops running and other sites detect this fact. The communication medium is assumed to provide the facility of message transfer between sites. A site always hands over a message to the communication medium, which delivers it to the destination site in finite time. For any pair of sites S_i and S_j the communication medium always delivers the messages to S_j in the same order in which they were handed to the medium by S_i.

4 Transformation of Database State

Each transaction that updates objects in a database transforms the database to a new state. The system state includes assertions about values of records and about the allowed transformations of the values. Transactions obey the assertions (laws of consistency constraints) by transforming consistent states into new consistent states. In the proposed scheme, the committing transactions are allotted a Commit Sequence Number (CSN) at the time of commit. The CSN value indicates the identity of a new state of the database.

4.1 Transaction Processing at the Database Server

The transaction processing activity at the DBS may access data items at the BS, during the 'external disconnect' state. On receiving the read-set of such a transaction, the DBS examines the associated site-status value (state1). All the available entries (later than the state1) in the external logdb are checked for a conflict. In case no conflict exists, the transaction proceeds in the normal way as the read items are current (at the level of state2). On completion of the transaction, the next CSN value is assigned to the transaction and its updates are added to the external logdb.

Consider T as the set of committed transactions within external logdb. A transaction t arrives with a site-status value "ss" (ss $\neq$ state 2), as the status of the BS site. The validation check and the subsequent tasks performed by the DBS can be expressed as indicated below.

For all validated transactions, the update values for the items in the write-set are processed as a normal activity. These are added to the external logdb. For the purpose of performing

```
valid := true;
    for   CSN   from  (ss+1)   to  state2
    do
       if   (write-set of transaction T intersects with
             read-set of t)
       then  for the conflicting items t (read-set) = T (write-set);
    end ;
      compute t until termination
      next CSN = latest CSN+1
      commit t, append  t in  external logdb   with next CSN,
                              ---------------
      state2 = next CSN;
 end ;
```

Figure 5: Algorithm for processing transactions at the DBS with data accesses at the BS.

concurrency control, the DBS is assumed to depend on any of the conventional techniques [6].

On recovery of a failed link or at the end of the external disconnect state, the external logdb entries are also transfered to BS in the CSN order, for the recovery of the state at the BS. As soon as the BS server has incorporated the entries (state2 = state1), the external disconnect status for the BS is aborted.

4.2 Transaction Processing at the Backup Server

The transaction processing activity at the DBS (in state 2) may access data items at the BS (in state 1), during its 'internal disconnect' state. On receiving the read request of a transaction, the BS examines the associated site-status value (state 2). It accesses its database copy (db1). For the purpose of incorporating the later updates, all the available entries (later than the state1) in the internal logdb are checked for a conflict. In case no conflict exists, the transaction proceeds in the normal way as the read items are current (at the level of state 2). The validation check for updation with respect to the contents in internal logdb is done to upgrade the transaction read-set contents.

On completion of the transaction the updates are received by BS. The next CSN value is assigned to the transaction and the update is added to the internal logdb.

At the end of the internal disconnect state at BS, the internal logdb entries are incorporated into the database at BS in the CSN order, for the recovery of the state at the BS. As soon as the BS server has incorporated the entries (state2 = state1), the internal disconnect status for the BS is aborted.

Allotted Commit Sequence Number (CSN)	Transaction-id	Read-set	Write-set
--	--	--	--

Figure 6: Committed Transactions List (CTL).

4.3 Information Management

A Transaction-id is commonly assigned to each transaction on its arrival. It contains information related to site-of-origin, site-status, transaction type (read-only class/ duration class/ priority class; (if known)), and local sequence number. The site-status value indicates the Commit Sequence Number (CSN) of the most recent transaction update received and implemented in the local database copy. Both kind of participating sites maintains this value. It is also noted along with read accesses made during transaction processing. It is used at the time of transaction validation to verify that data items read for transaction computation have the most recent value in the case of update transactions.

The sites maintain a table of entries of all recently certified update transactions in a Committed Transaction List (CTL) for the purpose of concurrency control or as external logdb, or as internal logdb as shown in Figure 6. This list contains entries for:

1. Transaction-id;
2. Read-set and associated details such as site at which access was made and the status of the site at the time of the Read-access;
3. Write set; and
4. Allotted Commit Sequence Number (CSN). These numbers are assigned in ascending order, i.e., Next CSN value = Previous CSN value + 1. The CSN also indicates the identity of the coordinator-site of the logdb.

In addition to the above CTL, the sites also maintain a current Network Status Table, which contains information about the status of each participating site (Figure 7). The entries in this table are updated on the basis of regular messages received from the participating sites. Each entry in this table contains :

1. The identifier of the site, Site-id;
2. The site-status value;
3. Up/down status; and
4. Disconnection status and site (acting as external logdb coordinator site : yes or no).

Site Identification	Site-status in terms of most recent CSN known	Connect status (normal/ lagging)	Coordinator External Disconnect (yes/no)
−−	−−	−−	−−

Figure 7: Network Status Table.

5 Backup Generation

A copy of the database is generated by reading the database through an incremental backup procedure. In order to copy the database from a site, a candidate site is declared as a BS (called Incremental Backup Site (IBS)). It notes node-status of the DBS (DBS(initial)). It copies the database disregarding any on-going change to the state of the database (blind copying). After completing a non-stop copying procedure, the status of the DBS is noted as DBS(final). The IBS further incorporates the changes to the state of the database from DBS(initial) to DBS(final) through the list of committed transactions appended as external logdb entries, during the copying. The list entries are used to overwrite the data in the IBS copy. These entries can be down loaded and obtained off-line from common database logging disk in case of a large size.

As the logging is an on-going activity, subsequently a backup copy may continue to be updated and incorporate improvements in its state status.

5.1 Exclusive Incremental Copy Generation

A scheme for processing a database-read algorithm is shown in Figure 8. Algorithm for generation of a consistent database copy after a global-read is presented in Figure 9.

In phase 1, the algorithm carries out an on-the-fly reading of the contents of the database (no consistency). A log of committing transactions (Figure 6) committing in parallel is created as external logdb. In phase 2, the algorithm generates a consistent copy of the database by overwriting the updates from this log.

5.2 Database Snapshot

It is possible to generate a copy of the database at the end of a financial period, (say), for example, at the end of a day on 31 December, at 24:00 hours (cut-off time). A consistent copy of the database can be generated before the cut-off time. The generation of the logdb entries is continued until it is past the cut-off time. A copy at the cut-off time can be easily generated by a roll-forward of the state by using the logdb entries.

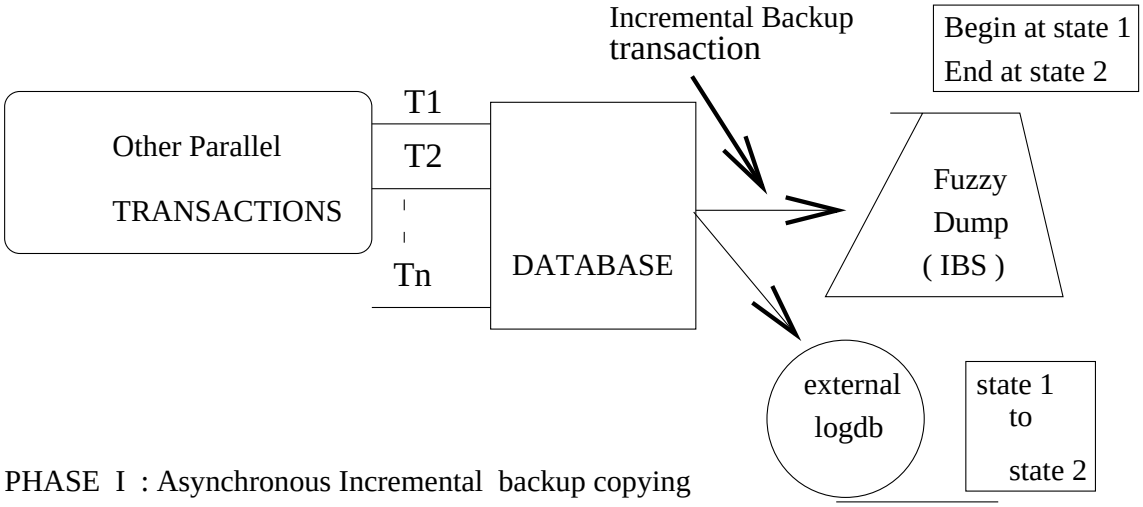

Figure 8: Asynchronous transaction processing for reading entire database copy.

Many situations, require a snapshot of an earlier state as may the case for recovery from errors or reconciliation from conflicts between conflicting transactions (committed due to network partitions in the absence of communications) [14]. For this purpose, a three phase computation is required. A roll-back (reverse) computation phase is further needed to change the level of incorporated updates to make a consistent database copy at the cut-off level (say defined by a transaction at T hours). An algorithm for generation of database snapshot at the cut-off level is shown in Figure 10. In order to implement phase 3, it is assumed that DBS(initial) > cut-off > DBS(final). Also, for each parallel transaction committed in phase 1, it is assumed that its Write-set (T) is fully contained within its Read-set (T).

6 Site Failure and Recovery

6.1 Site Recovery

A site may have a communication link failure and may recover in a short time. Such a disconnection can be taken care by other sites, by the initiation of 'external disconnect' state for the failed site. On recovery the site may incorporate the missed updates from external logdb available from the nearest site. In case of a long duration of the link or the site failure, the the RDBMS sites may tend to offload the logs on to permanent media such as disks or tapes.

```
/* Phase 1 -- External 'logdb' generation with creation
              of Fuzzy dump (at Incremental Backup Site (IBS))  */
  BEGIN
  csn = DBS(initial);
  first-csn = csn;
      WHILE   incremental-read at IBS
         FOR  each transaction T  (commit by other parallel Transactions)
              csn = csn + 1
              WRITE  in external 'logdb'
              ( csn, Transaction-id, Read-set (T), Write-set (T) )
      END WHILE
      last-csn  =  csn
      DBS(final) = last-csn
   END

/* Phase 2 -- Add transformations of database state
              ( roll-forward from 'state 1' to 'state 2' using logdb */
  BEGIN
       FOR   first-csn, last-csn
             IF  write-set (T)  <>  empty-set
                 WRITE  in IBS database copy
                        ( Write-set (T) )
  END
```

Figure 9: Two phase algorithm for Generation of a consistent database.

The subsequent recovery process needs to begin recovery of the missed updates by using the offloaded log data.

6.2 Site Initialization

A site that holds a current copy of the database (at the most up to date state (state 2)), is an ideal candidate for site recovery. The site can be connected and declared to be an operational site within the RDBMS. A site that has recovered a consistent version (an earlier version (state 1)) of the database can also become a candidate.

After copying and generating a consistent version of the database, the site informs about its status to a DBS as 'state1', and requests to be in put under an external disconnect state. The DBS, initiates an external logdb and informs the BS site about the its site status (state2). The BS site compares its site status (usually, state1 $>$ state2). If required, the BS further upgrades its status up to state2 with the help of a log file available off-line (Figure 11). Next, the BS site joins the system in an external disconnect state (until full recovery and integration with the operational RDBMS).

```
/* Phase 1 -- External 'logdb' generation with creation
             of Fuzzy dump at at Incremental Backup Site (IBS) */
  BEGIN
  csn = DBS(initial);
  first-csn = csn;
      WHILE   incremental-read at IBS
         FOR  each transaction T  (commit by other parallel Transactions)
              csn = csn + 1
              WRITE  in external 'logdb'
              ( csn, Transaction-id, Read-set (T), Write-set(T) )
      END WHILE
      last-csn  =  csn
      DBS(final) = last-csn
   END

/* Phase 2 -- Add transformations of database state
             ( roll-forward from 'state 1' to 'state 2' using logdb */
  BEGIN
       FOR   first-csn, last-csn
             IF write-set  (T)  <>  empty-set
                      WRITE  in IBS database copy
                      ( Write-set  (T) )
  END

/* Phase 3 -- roll-back from 'state 2' for an earlier cut-off point
             using logdb  */
  BEGIN
       FOR    csn   FROM   last-csn   TO   cut-off   STEP   -1
              FOR an item in Write-set (T)
                           WRITE in IBS database copy
                           ( Read-set (T) );     /* write old values */
  END
```

Figure 10: Algorithm for database snapshot at a given reference point/time.

7 Proof of Correctness

While the earlier proposals avoid inconsistency by not allowing certain transactions to commit, this proposal permits a normal execution activity during the execution of the incremental-read transaction. Consistency is achieved by updating the inconsistent copy of the database by adopting a 'missed update transaction' approach. A log of the 'updates in progress' is used for creating a consistent version of the data, by over-witting on the inconsistent copy. All such updates that could have been missed partially or fully, are rewritten on the database copy, during phase 2.

The proposed algorithm considers only one BS site for the incremental backup at a given time, that is attached to a DBS. Its extension to multiple global-reads and for considering a distributed system is a simple extension of the mechanism. Each site may independently support its logging and multiple BS sites [28]. Also, for supporting multiple reads, separate incremental logs as per the 'Exclusive Incremental Copy Algorithm' in figure 9, it can be adopted. All the external logdb files can be merged into a single file with its lower state being equal to the lowest among the connected BS sites.

7.1 Proof of Consistency

Informally, a similar approach (use of update history) is also used by database recovery algorithms. These use incremental logs with deferred updates, in a similar manner. A brief outline of the proof of consistency is presented for the proposed approach.

An informal proof is being presented on similar lines as [4, 16].

A transaction is a sequence of n steps :

T = ($(T, a_1, e_1), \ldots, (T, a_i, e_i), \ldots, (T, a_n, e_n)$), where $1 =< i =< n$.

where T is the transaction, and a_i is the action at step i, and e_i is the entity acted upon at the step. A schedule (S) of transaction dependencies, can be written as :

Considering a mix of an update transaction and a read transaction, that access (read) a data entity ' e ', Dependency(S) exists as (T_1, e, T_2), such that e is the output of T_1, and input of T_2, such that, either T_1, or T_2 updates data entity 'e'. Two schedules, Dependency(S1) and Dependency(S2) are equivalent, if Dependency(S1) = Dependency(S2). A schedule S is consistent, if there exists an equivalent serial schedule. All schedules formed by well formed transactions following 2-phase locking are consistent [16].

Assertion 1 : Given that, T_1 is a read-only transaction. A schedule (S) of transaction dependencies (T_1, e, T_2), can be transformed to schedule (S'), as (T_2, e, T_1), if all writes of T_2, are later overwritten on data read by T_1.

Proof : A conflict between the incremental backup (fuzzy data dump) at the IBS and the first conflicting update transaction (T1) is eliminated by overwriting of the updates done by T1 on the copied version (by assertion 1). By induction, each consecutive application of the subsequent updates (occurring until completion of the fuzzy dump), eliminates the conflicts

between the copy version and its parallel update activity transactions.

8 Performance Considerations

The proposed algorithm does not introduce any blocking except requesting shared locks for a small portion of the database. On completion of a incremental-read, an off-line computation is capable of generating a consistent copy of the database. The main overheads in this proposal are - the delay in generation of database copy during the phase 2 processing. The size of the conflict log is not significant because a separate medium can normally be used to maintain serial logs. Also, the overhead can be avoided by using the normal system generated transaction recovery logs (in the place of conflict logs). The generation of transaction logs are a matter of routine processing for database systems [6, 24].

The performance improvement alternatives high lighted by the earlier proposals [4] based on use of color logs to improve performance can be adopted to reduce the size of the logs. Such a procedure may generate large computational steps. These can be avoided considering the off-line nature of the subsequent log processing. The proposed approach differs from all the earlier proposals by offering to turn a non-stop read (blind read with no concurrency control), into a consistent database copy. The second phase of the algorithm can be carried out (optionally) in an *on-line* manner. The earlier procedure for increasing the data availability incur a high overhead [29]. The proposed approach is based on the use of historical data [19, 25, 32]. We assume an independent model of logging for high performance distributed architecture [28] (Figure 13.).

9 Implementation Issues

On page 2, the assumption has been made that the system uses OCC. This can be relaxed. As shown in appendix A, systems that execute transactions as per 2-phase locking or optimistic concurrency control can cooperate among each other at the level of transaction execution steps. Thus, OCC is not an essential part of the mechanisms being proposed. So, far OCC has been a nice theoretical construction that has not been often used in practice in the absence of a mechanism as proposed above.

Also, the algorithm stores the read sets of transactions. The read set is the set of data items that has been read until the moment of time under consideration. The read-set data concerns update transactions that require exclusive access to data resources. For many applications, typically those that require application recovery in banking and e-commerce systems, it may be possible to find the read sets with small size.

During the "check for conflicts" between the write set of the log (in external logdb or internal logdb) and read set of the incoming transaction, the read set is upgraded to read from values in the log, if needed. These result in additions to the log for the new write set of the new transaction. This implies that one cannot interleave operations of different transactions. Transactions must be processed one after the other in a sequence, after the prior transaction

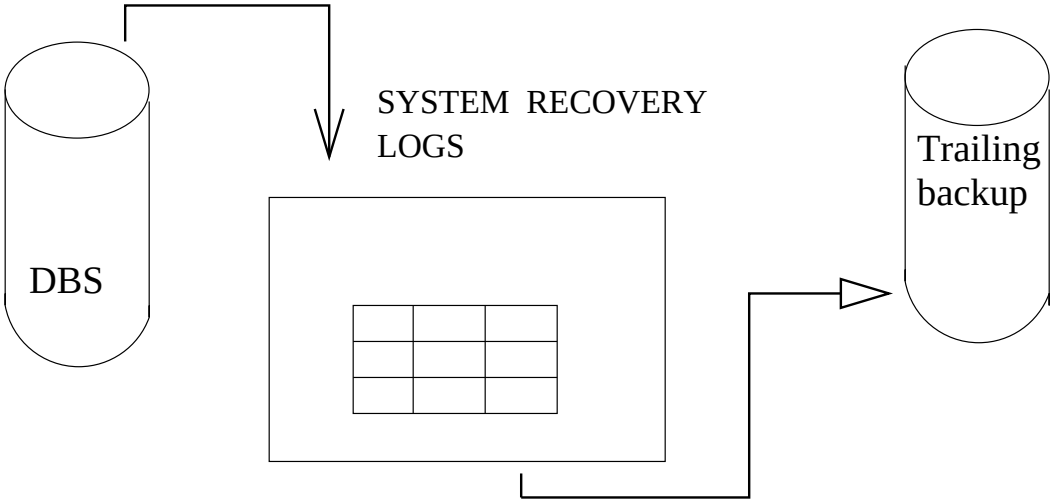

Figure 11: Trailing backup system improving its database state status

finishes (and is appended to the log) before the second one can start. This results in a slow process. As a remedy, the upgraded read-set of the transaction is added as a dummy write set at the end of the log. This provides exclusive access to requested data items for the transaction that seeks these items. On receiving the actual updates from this transaction, the dummy write set information can be removed from the log.

If the number of disconnected sites, becomes more than one, the same external logdb can be utilized by the RDBMS sites. The lowest level entry will depend on the site with an earliest failure for the external logdb. In the event of prolonged outage of a failed site, the size of a external logdb may grow beyond a possible limit. In such a case the contents of the log may be transferred to a stable storage medium. The recovering site will need to process the contents off-line, before joining the system for a site-initialization.

9.1 Partially Replicated Database System

The system assumption about full replication within a RDBMS can be relaxed. The Figure 12 shows a collection of three databases that are part of a collection of cooperating databases. These databases can independently be replaced by three groups of RDBMSs with multiple sites and be an RDBMS with its own disconnected or recovering sites.

As a further extension to the proposal, the network may also support sites that are member of more than one group of RDBMSs. For example, site D, E and F (supporting db1, db2, and db3 respectively) could be merged onto the same physical site (Figure 12). Such a merger can be supported by the exiting methods of time-stamp ordering for generating transaction identity. The disconnection and recovery algorithm needs to be observed within each group of participation unit. Also, the individual RDBMSs are assumed to maintain independent log. Thus, each RDBMS maintains its own log (Figure 13). The associated 'logdb' files may exist depending upon multiplicity of the intermediate site states, that may exist at the participant sites for a given RDBMS.

To consider an extreme case of a non-replicated database system, the assumptions can be

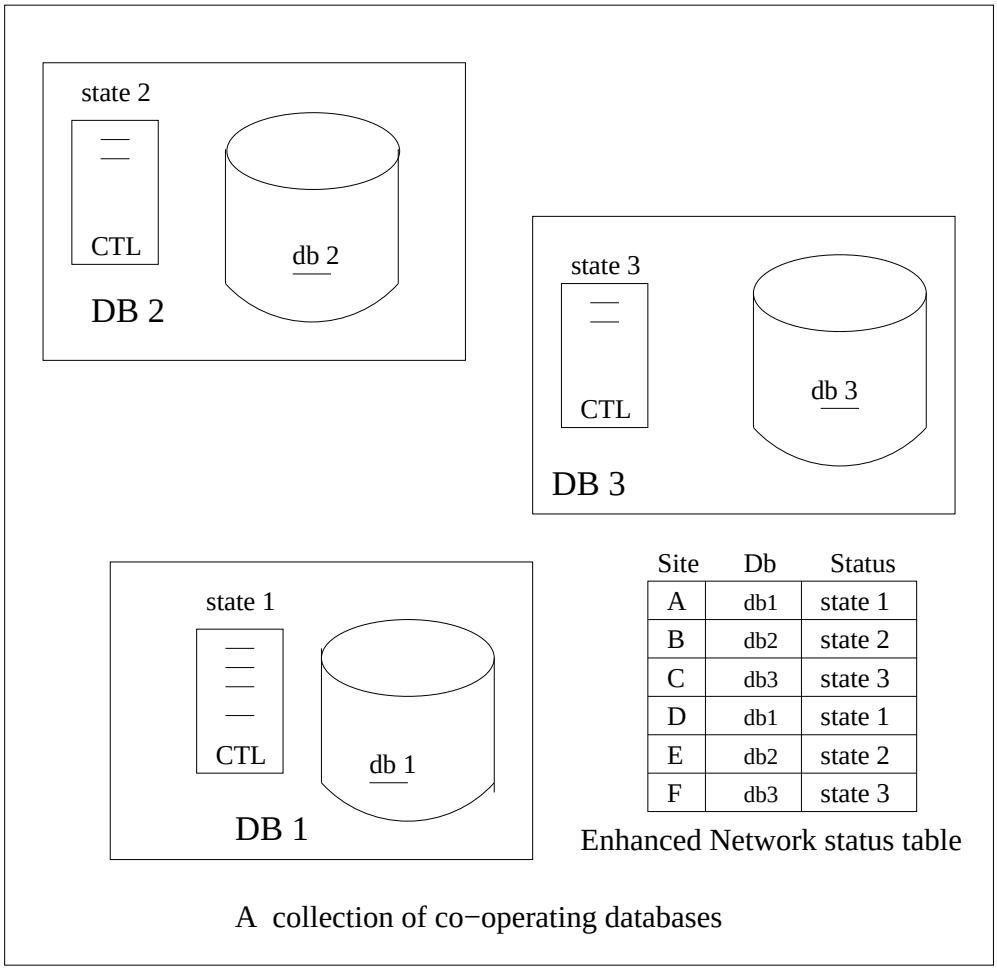

Figure 12: Databases having trailing backup system with lagging state status.

further relaxed. Each database can independently be an non-replicated DBMS with sites and independently maintain its own disconnected or recovering backup sites (Figure 14). Thus, The system may consist of a collection of three databases that are part of a collection of co-operating databases.

10 Discussion

10.1 Incremental Backup

During the internal or external disconnect state, transaction processing has to check both the database and the logdb for correct answer. The proposed mechanism provides for a low cost option for the computing environments that suffer from frequent disconnections and connections. Also, the system state based computing mechanism helps to localize the exact contents of a log that may be long, if the recovery takes a long time. In its absence the logdb could be big and the transaction performance is at a lower level as is the case of the traditional systems. The usual algorithm would apply the log records to the backup and only then allow

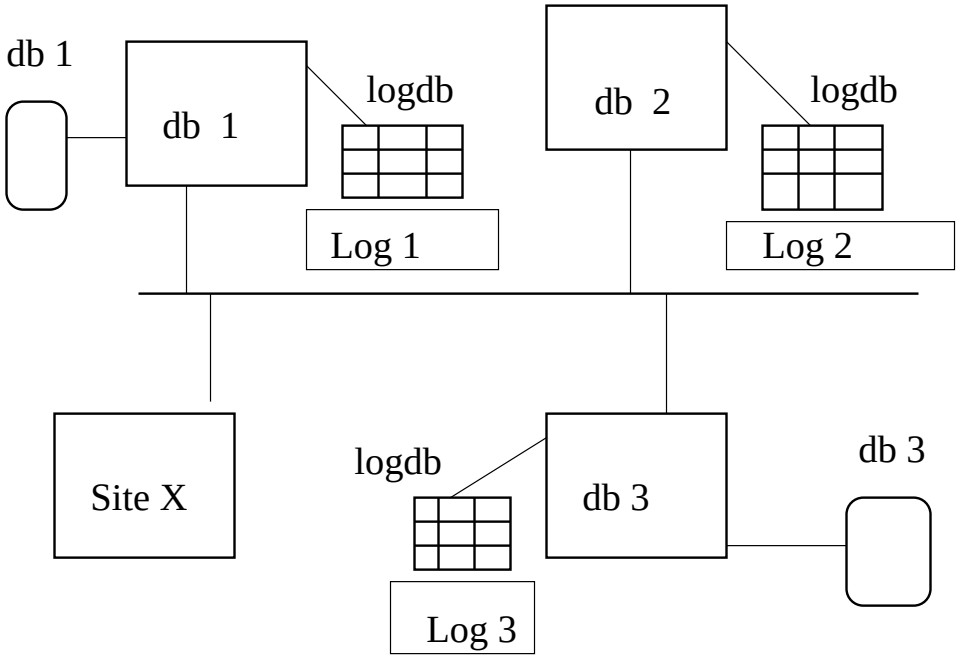

Figure 13: Independent system architecture for a recovery system.

new transactions. If the logdb is small, this won't take much time. If the logdb is large, having to look up records in it (without the proposed site status based mechanism) will take a lot of time.

While describing an incremental backup Lomet mentions [24] - "By identifying the portion of the database state S that has changed since the last backup, one needs to only backup that changed portion". Many database systems support a form of incremental backup [24, 27]. These incur operation level (transaction level) blocking [4]. These techniques do not attempt to convert a blind-read (fuzzy dump) into a consistent copy of the database. The earlier studies have focused on making a **trailing backup** system catch up with current state of the database. In our proposal, the trailing BS can improve its database state in an off-line manner (Figure 11).

In contrast with earlier studies which focus on disk archiving utilities, or buffer management during backup operations, the presented work supports after image backup for of any small or large database, or system files that reside on the system. Thus, the proposal works in combination with the earlier research efforts at the transaction level for recovery from media and application failures. Our proposal is capable of generating a backup of a database or system file under operational conditions. Subsequent to backup generation the database updates can be copied by the trailing BS system (Figure 11). It adopts the conventional recovery logs to be its recovery logs, as the available external logdb from a DBS to improve its database state status.

Similarly, the presented work does not contradict the existing procedure of maintaining a remote backup site [29]. The present study depends on 2 phase commit protocol for transaction commit processing. However, by introduction of asynchronous steps by virtue of an on-line logdb (either 'internal logdb' or 'external logdb') it becomes possible to use asynchronous oper-

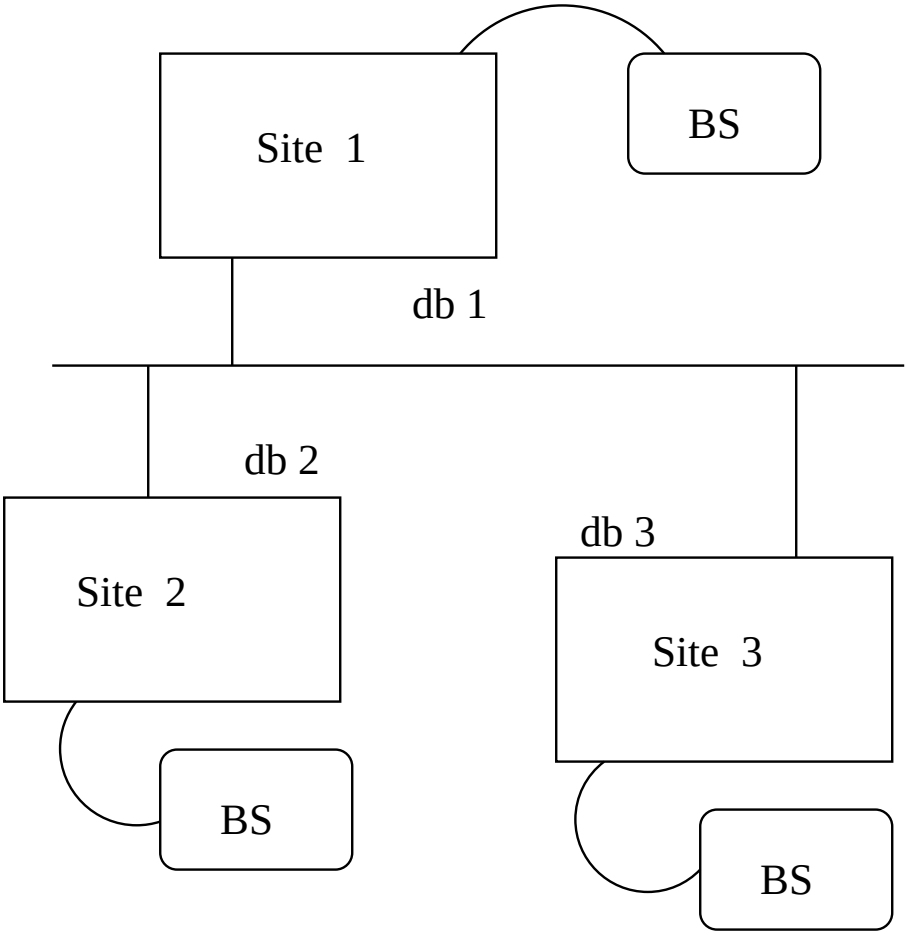

Figure 14: Databases having independent replicated database for backup.

ations for improving parallelism among activities. The model is a useful tool for incorporating alternatives into available mechanisms and algorithms. By extension, it is possible to support a collection of sites within a distributed system that use extended CTL for system wide operations (Figure 12). The opportunity permits the sites to lag behind and recover on-the-fly.

Previously, Gray [18] has described "fuzzy dump" and also how to optimize media recovery by processing the media recovery log. Haerder and Reuter [21] and Bernstein [6] do not provide the details on how the on-line "fuzzy dump" is actually created or used [24].

The main drawback of the conventional systems is that these halt transactions at occurrence of a disconnection. And also halt transactions before and during a reinitialize (to add updates). These drawbacks can add a significant overhead in an environment that has short and frequent disconnections during service.

10.2 Application Errors

On the instance of occurrence of an error, a snapshot of the concerned database or file can be initiated for the purpose of examination for rectifications and recovery process. The option of

an instant backup is a desirable feature within systems supporting financial applications. The after image backups generated by the algorithm can be used to recover the current system state after a failure, by applying a correction after a roll-back of the system.

10.3 Media Failures

Some of the media failures have an effect on a small portion of the data with in a database. For example, a disk sector (a block of data) may become unreadable. In the event of an occurrence of a media failure, the concerned system must be switched to operate in 'internal disconnect' mode. It generates its own internal logdb. In this state, it recovers the damaged data from an alternate site (through support of replication or recovery logs). Thus, the failure is localized to be a small event related to a database segment.

10.4 System Failures

A system failure results in creating an 'external disconnect' state at the other related sites (Figure 12). Each site generates its own external logdb. These operate by gathering the required data from alternate sites (through support of replication or recovery logs). The failed site also begins by creating an 'internal disconnect' state.

The common drawbacks of the conventional systems include that these require a system shut down, in the even of system recovery. The lapsed updates can not be appended on-line. Also, Transaction that need to add updates can not operate with out a system shutdown and recovery.

11 Summary and Conclusions

The paper presents algorithms for incorporating tolerance towards disconnection of database service site and subsequent recovery of a disconnected member site. Among the components for a system of backups and recovery, it provides an algorithm for generation of a consistent copy of the database. It is an essential requirement. The algorithm generates a consistent copy of the database without affecting the ongoing transaction processing activity. It highlights a mechanism for processing site-initialization after recovery. During the recovery phase also, the other update transactions that execute in parallel face no blocking of update activity. To the best of our knowledge, it is an unsolved problem. Earlier algorithms achieve global-reading by restricting on-going transaction processing activity. The proposed algorithm is simple to implement. It preserves the consistency of the backup version, as per the notion of serializability.

The paper considers enhancement of availability in many ways. Firstly, by turning a fuzzy database dump (read copy) into a consistent copy, it provides an easy access to a backup copy. It enhances availability. Secondly, the paper proposes a asynchronous model of computation for backup and site-initialization. Thirdly, the proposal permits treatment of a small portion of disk data (important system files or a selected database) as a unit of backup and initialization. Fourthly, by independently maintaining external or internal 'logdb' files, the failures are localized to the failed parts and the RDBMS can continue user service in the presence of disconnections and re-connections. We also present a procedure based on the proposal for generation of a database snapshot at the end of a time duration. System snapshots are needed

for financial systems (end of the year budget), data mining and data warehousing activities. Most of the existing approaches require a halting of transaction processing for the production of database snapshots. In the event of a crash, when the logdb sizes tend to be large, the proposed system status based processing for recovery, reduces the site-initialization time.

- Appendix A -

Cooperation Among Transaction Processing Systems

Most transaction processing systems use locking based concurrency control. Many research proposals that seek to integrate synchronization and recovery techniques depend on alternative ways to perform concurrency control [32]. There have been many research studies that have considered other techniques such as, optimistic concurrency control (OCC) and time-stamp based concurrency control [6, 17]. We demonstrate a procedure by which it is possible to carry out transaction processing within two cooperating systems that use different concurrency control techniques.

Consider database sites A and B, that perform transaction processing based on two phase locking and optimistic concurrency control, respectively. In order to execute a sub-transaction at site B, the system at A may need to lock the required data items at site B. It send the data access requests for exclusive access to site B. In order to provide the locking privilege for a the selected data items, site B can incorporate the data items (received during the phase one of 2-phase locking) in its committed transactions list as a dummy entry. This step within the OCC prevents the executing transactions that access these data items at site B from updating the database until update values are available after release by the site A at the end of phase two of 2-phase locking. Similarly, site B may need to execute a transaction using data items that exist at site A. On completion of a transaction in OCC, a transaction needs to perform validation. The validation check in the case of a transaction accessing data at site A needs to consider the lock table of site A, in addition to its own list of committed transactions.

References

[1] D. Agrawal and A. El Abbadi, *Locks with Constrained Sharing*, Proceedings of 9th Symposium Principles of Database Systems, pp. 85-93, April 1990.

[2] D. Agrawal, A. El Abbadi and A.E. Lang, *Performance Characteristics of Protocols with Ordered Shared Locks*, Proc. of 7th International Conference on Data Engineering, pp. 592-601, Kobe, Japan, April 1991.

[3] D. Agrawal, A. El Abbadi, and R.C. Steinke, *Epidemic Algorithms in Replicated Databases* Proceedings of the 16th Symposium on Database Systems (PODS), 1997, pp. 161-172.

[4] P. Amann, Sushil Jajodia, and Padmaja Mavuluri, *On-The-Fly Reading of Entire Databases*, IEEE Transactions of Knowledge and Data Engineering, Vol. 7, No. 5, October 1995, pp. 834-838.

[5] R. Bayer, H. Holler and A. Reiser, *Parallelism and recovery in Database Systems*, Volume 5, June 1980, pp 139-156.

[6] P.A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addision-Wesley, 1987.

[7] S. Bhalla, *Improving Parallelism in Asynchronous Reading of an Entire Database* , 7th International Conference on High Performance Computing 2000, (HiPC 2000), December 2000, Proceedings published by Springer-Verlag in Lecture Notes in Computer Science (LNCS) series, vol. 1970, pp. 377-384.

[8] S. Bhalla and S.E. Madnick, *Parallel On-the-Fly Reading of an Entire Database Copy*, International Journal of Computer Research, Vol. 10, No. 4, November 2001.

[9] S. Bhalla, *Asynchronous Transaction Processing for Updates With no Wait-for State*, Proceedings of 9th High Performance Computing (HiPC 2002) conference, Bangalore, December 2002, published by LNCS vol. 2552.

[10] S. Bhalla and S.E. Madnick, *Asynchronous Reading of Consistent Copy of a Large Database*, Applied Parralel Computing, in Advances in Computation Theory and Practices (ACTP) series, Nova Publishers USA, to appear in 2003.

[11] K. Brahmadathan, and K.V.S. Ramarao, *On the Management of Long-living Transactions*, *Journal of Systems and Software*, Vol. 11, No. 1, pp. 45-52, Jan. 1990.

[12] Y. Breitbart and H. F. Korth, *Replication and Consistency: Being Lazy Helps Sometimes*, Proceedings of the 16th Symposium on Database Systems (PODS),pp. 173-184, 1999.

[13] Y. Breitbart, R. Komondoor, R. Rastogi, S. Seshadri, and A. Silberschatz, *Update Propagation protocols For Replicated Databases*, Proceedings of the SIGMOD International Conference on Management of Data, *SIGMOD record*, vol. 28, No. 2, June 1999.

[14] Davidson, S.B., Hector Garcia Molina, and Dale Skeen, *Consistency in Partitioned Networks*, Computing Surveys, Vol. 17, No. 3, Sept. 1985.

[15] L. Do, P. Ram, and P. Drew, *The Need for Distributed Asynchronous Transactions*, SIGMOD Record, vol. 28, No. 2, June 1999.

[16] K.P. Eswaran, J.N. Gray, R.A. Lorie and I.L. Traiger, *The Notion of consistency and predicate locks in a Database System*, Communications of ACM, Vol. 19, Nov. 1976.

[17] H. Garcia-Molina, J.D. Ullman, and J. Widom, *Database System - The Complete Book*, Prentice-Hall publishers, 2002.

[18] J. Gray, *Notes on Database Operating System*, IBM Technical report RJ2188, also LNCS Vol. 60, published by Springer-Verlag, Feb 1978.

[19] J. Gray and A. Reuter, *Transaction Processing : Concepts and Techniques*, Morgan Kaufmann publishers, California, USA, (1993).

[20] J. Gray, P. Helland, P. O'Neil and D. Shasha, *The Dangers of Replication and a Solution*, Proceedings of 1996 Annual SIGMOD conference, *SIGMOD Record*, June 1996, pp. 173-182.

[21] T. Haerder, and A. Reuter, *Principles of Transaction-oriented Database Recovery*, ACM Computing Surveys, 15,4 (Dec. 1983), pp.287-317.

[22] R.P. King, N. Halim, H. Garcia-Molina, and C.A. Polyzois, *Management of Remote backup copy for disaster recovery*, ACM Transactions on Database Systems, Vol. 16, No. 2, pp. 338-368,June 1991.

[23] V. Kumar, and M. Hsu (eds.), *Recovery Mechanisms in Database Systems*, Prentice-Hall, NJ 1998.

[24] D.B. Lomet, *High Speed On-line Backup When Using Logical Log Operations*, International Annual Conference SIGMOD 2000, published in SIGMOD record, June 2000.

[25] D.B. Lomet and B. Salzberg, *Exploiting a History Database for Backup*, Proceedings of Very Large Data Bases (VLDB) Conference, Dublin, pp. 380-390, Sept. 1993.

[26] C. Mohan, H. Pirahesh, and R. Lorie, *Efficient and Flexible Methods for Transient Versioning of Records to Avoid Locking by Read-only Transactions*, Proceedings of SIGMOD International Conference on Management of data, pp. 124-133, June 1992.

[27] C. Mohan, and Inderpal Narang, *An Efficient and Flexible Method for Archiving a Data Base*, Proceedings of SIGMOD International Conference on Management of data, pp. 139-146, 1993.

[28] E. Panagos, A. Biliris, H.V. Jagdish, R. Rastogi, *Client-Based Logging for High Performance Distributed Architectures* , Proceedings of International Conference on Data Engineering (ICDE), pp. 344-351, 1996.

[29] C.A. Polyzois, and Hector Garcia-Molina, *Evaluation of Remote Backup Algorithms for Transaction Processing Systems* , ACM Transactions on Database Systems, Vol. 19, No. 3, September 1994, pp. 423-449.

[30] Reddy P. Krishna, and S. Bhalla, *Asynchronous Operations in Distributed Concurrency Control*, IEEE Transactions on Knowledge and Data Engineering, Vol. 15, No. 3, May 2003.

[31] K. Salem, H. Garcia-Molina and J. Shands, *Altruistic Locking*, ACM Transactions on Database Systems, Vol. 19, No. 1, pp. 117-165, March 1994.

[32] S.K. Sarin, C.W. Kaufman, and J.E. Sommers, *Using History Information to Process Delayed Database Updates*, 12th Intl. Conf. on VLDBs, Kyoto, Aug. 86.

[33] Y.H. Viemont, and G.J. Gadrin, *A Distributed Concurrency Control Algorithm Based on Transaction Commit Ordering*, Proceedings of conference on Fault Tolerant Computer Systems, 1982.

[34] W.E. Weihl, *Data-dependent Concurrency Control and Recovery* , Proceedings of 2nd ACM Symposium on Principles of Distributed Computing, 1983.