

# Planned Disconnections for Mobile Databases\*

JoAnne Holliday      Divyakant Agrawal      Amr El Abbadi  
Department of Computer Science  
University of California at Santa Barbara  
Santa Barbara, CA 93106  
{joanne46, agrawal, amr}@cs.ucsb.edu

## Abstract

*As mobility permeates today's computing environment, we envision application infrastructures that will increasingly use mobile technologies. Traditional database applications will need to integrate mobile entities: people and computers. In this paper, we develop a distributed database framework for mobile environments. A key requirement in such an environment is to support frequent connection and disconnection of database sites.*

## 1 Introduction

As mobility permeates into today's computing and communication arena, we envision application infrastructures that will increasingly rely on mobile technologies. Current mobility applications tend to have a large central server and use mobile platforms only as caching devices. We want to elevate the role of mobile computers to first class entities in the sense that they allow the mobile user work/update capabilities independent of a central server. In such an environment, several mobile computers may collectively form the entire distributed system of interest. These mobile computers may communicate together in an ad hoc manner by communicating through networks that are formed on demand. Such communication may occur through wired or wireless networks. At any given time, a subset of the computer collection may connect and would require reliable and dependable access to relevant data of interest.

In this paper, we consider a distributed database that can be made up entirely of mobile components. These component sites are peers and may be replicated both for fault-tolerance, dependability, and to compensate for sites which are currently disconnected. Thus we have a distributed replicated database where several sites must participate in the synchronization of transactions. The capabilities of the

distributed replicated database are extended to allow a site or team member to plan to disconnect with minimal disruption to the remaining sites. A "check-out" mode allows for independent update of a portion of the database by a disconnected member. These updates are automatically synchronized and integrated into the database upon reconnection.

Walborn and Chrysanthis [8] describe the use of mobile computers in the trucking industry. Faiz and Zaslavsky [1] discuss the impact of wireless technologies and mobile hosts on a variety of replication strategies. Distributed replicated file systems such as Ficus and Coda [6] have extensive experience with disconnected operations. In this paper, by using the notion of planned disconnection, we present a framework to allow disconnected update using a check-out idea adapted from version control software.

## 2 Motivation

In our system, the components of the distributed database are laptop computers belonging to members of a small team. The database is partially or fully replicated at the laptop computers which carry the only active copies of the database. Normal operations ensure that all team members are completely synchronized with each other's progress. Team members gather together in various (sometimes remote) locations and form ad hoc networks to perform their work. On these occasions they need update capabilities on their database without having to connect to a fixed location computer.

We consider three scenarios covering ways in which the team members might want to interact with their database: basic sign-off, check-out, and relaxed check-out. In the first scenario, team members occasionally disconnect for long periods of time, for instance when one or more of them are traveling. When this happens, the remaining team members need to be informed so that the database can continue to process updates knowing that the missing processor has not failed but rather has voluntarily disconnected. The database copy of the traveling member can continue to process read-

---

\*This research was partially supported by the NSF under grant numbers EIA98-18320, CCR97-12108, IIS98-17432 and IIS99-70700.

only transactions. We call this mode *Basic Sign-off*.

Our first scenario involves Bob, the CEO of a small company. The three laptop computers of the senior executives contain replicas of the corporate database, which is generally updated during their frequent meetings when they connect their computers with an ad-hoc network. The database is backed-up occasionally onto a non-mobile computer. When Bob needs to travel on business, he wants the others to be able to continue their work with the database in his absence. He also wants to take his laptop on his trip so he can peruse the database while he is waiting at the airport or in his hotel room. The database system provides a sign-off capability so that the system will be able to synchronize all of the database components and can allow updates in spite of the missing member. When Bob returns and reconnects, his copy of the database will be automatically brought up to date as part of the sign-on procedure.

In the next scenario, the traveling team member wishes to “check-out” a portion of the database so that updates can be made to that portion while disconnected from the rest of the members. Thus, this mode is called *Check-out* mode. This mode could be needed when the team member is traveling to a location where he will be able to gather information relevant to that portion of the database or perhaps that portion of the database is concerned with the work or output of that particular team member. When the traveling team member has checked-out a portion of the database, the other team members are completely unable to access that data. It is as if the traveling team member had a write lock on the data that was checked-out. The traveling member can access other portions of the database that he did not check-out in read-only mode. When the traveling team member returns and reconnects his laptop to the system, the database changes he has made are automatically synchronized and integrated into the database. This is in contrast to a system without check-out mode that requires manual integration of the changes into the database.

An example of the check-out mode would involve Alice, who is part of a small sales team that currently covers the pacific northwest states. She meets with her team almost every day at which time they bring their laptops together to form a network and share their information. At that time, their common database is updated with production and product availability information from the factory. Before the computers are disconnected, each team member is given a sales assignment and a portion of the database is checked-out to them to update as they make sales and delivery contracts. Alice can update the portion that is checked-out to her as well as query the portions of the database that were not checked-out to anyone. She cannot access the portions of the database that were checked out to other team members, however, that is not necessary for her to do her work. At the next meeting, the laptops are connected and

the updates to the database are smoothly integrated without the need of manual intervention.

The third scenario, *Relaxed Check-out* mode, is like check-out except those portions of the database that have been checked-out to other team members can be queried or “browsed”. Unlike the above two scenarios where all transactions are serializable, the resulting transactions are not serializable. However, there are many situation in which a relaxation of serializability requirements is acceptable.

Jane is a field researcher in a remote part of the world. She and her fellow researchers take their laptops to remote observation platforms and weather stations. They record animal migration patterns and download weather information from sensor stations. While she is recording animal migrations she frequently needs to check recordings of an earlier day or a different location to help her in interpreting her measurements for her field notes. The database on the computers of Jane and her fellow researchers allows relaxed check-out mode. She checks out the portion of the database that pertains to the location she is going to or the animal species she is studying. She and her fellow researchers can also query the portions that are checked-out to the others, although the query results reflect old values from the last time the researchers gathered together and formed a network. She knows she may be viewing stale data, but it is still useful for her to do so. When she checks-in her work at the end of the day, it is automatically integrated into the database.

### 3 Planned Disconnection Types

Unplanned disconnection is a disconnection without informing the distributed system of the intention to disconnect and reconnect in an orderly manner. The result is that the disconnected site is detected as a failure. Planned disconnection [5] involves informing the distributed system of the intention to disconnect and may include the appointing of a proxy so that the remaining connected sites can continue processing with minimal disruption. In this section, we explore the possible kinds of planned disconnection.

We consider a distributed database consisting of  $n$  sites labeled  $S_1, S_2, \dots, S_n$  with partial or full replication. Users interact with the database by invoking transactions at any one of the database sites. The criterion for correctness in databases is the *serializable* execution of transactions [2]. Since two-phase locking is widely used to guarantee serial executions, we assume that each site in our distributed system enforces two-phase locking locally. In a replicated database, multiple physical copies must appear to the user as a single logical copy. Therefore, replicated databases should enforce *one-copy serializability* [2]. We assume a mechanism to enforce one-copy serializability is used.

Our assumptions and requirements for the distributed

system are as follows. The number of team members is fixed, however, some of those team members may be disconnected and members will need to gather and form an ad hoc network in different geographical locations. Database consistency is important and serializability is enforced locally with two-phase locking. We do not specify which method is used to enforce one-copy serializability. The distributed system must be able to process database updates even though some of the team members are not available, however, we don't want one of the members to be the primary site which must always be present when a network is formed. Thus, all members are equal participants.

### 3.1 Basic Sign-off

In simple planned disconnection or *basic sign-off*, the database of the disconnected site becomes read-only. The connected sites continue to read and update the data item or objects. The planned disconnection will be accomplished with the help of a *proxy*. When a site disconnects, it appoints another site to vote on its behalf to ensure that replicas can be updated and in any other actions of the distributed system which require consensus. The power to vote on behalf of another member is called a proxy and the site with that power is also referred to as a proxy.

If site  $S_i$  wants to disconnect from a distributed database, it carries out a disconnect dialog so that the system is aware that it has not failed, but will merely disconnect for a period of time.  $S_i$  contacts another member  $S_j$  to be  $S_i$ 's proxy and synchronizes the database copies. For those data items at site  $S_i$  that are not replicated at another currently connected site,  $S_j$  is given a copy of the data items and the right to process update transactions on them. For those data items at site  $S_i$  that are replicated at other sites,  $S_j$  is given the right to vote for  $S_i$  in matters concerning updates to the data items. During the disconnection,  $S_i$  can only read its local copy of the data. When  $S_j$  sees a message for  $S_i$ , it answers on behalf of  $S_i$  while  $S_i$  is disconnected.  $S_j$  also keeps track of the updates to the database that  $S_i$  has missed because of the disconnection. (We assume that all updates are sent to all copies, and so,  $S_j$  can respond on behalf on  $S_i$ .) When  $S_i$  reconnects, it must go through a sign-on procedure to regain its proxy. To do this,  $S_i$  reconnects and contacts the proxy before processing messages from other sites. If  $S_i$ 's proxy has itself disconnected,  $S_i$  broadcasts messages to determine who the new proxy is, so that it can update its database and retrieve its voting rights.

In the case where all sites are disconnecting and  $S_i$  is the last to sign-off, there is no other site for  $S_i$  to give its proxy to. In this case,  $S_i$  simply disconnects without assigning a proxy as no one in the distributed database system will be performing updates or anything else, for that matter.

The Basic Sign-off protocol produces executions that are

one-copy serializable. Since all of the transactions of the disconnected site are read-only, the values of the data that are read are those of a snapshot taken at the time of disconnection. All of the read-only transactions of the disconnected site can be serialized at the time of disconnection. The Basic Sign-off mode is explained in more detail in [3].

### 3.2 Check-out mode

If the site  $S_i$  wants to disconnect and be able to update a particular data object, it declares its intention to do so before disconnection and "checks-out" or "takes" the object for writing. In order to maintain serializability in *check-out* mode, the remaining connected sites are prevented from accessing the object which  $S_i$  has checked-out (as if  $S_i$  had a write lock). An object can only be checked-out to one site at a time. It makes sense to implement *check-out* mode using the existing locking mechanisms. The site that wishes to disconnect, say,  $S_i$ , acquires a write lock on the item or object it wants to update while disconnected. This write lock is like an ordinary write lock except that the "transaction" which holds it should not be aborted due to deadlock with ordinary transactions. In order to distinguish these "transactions" from ordinary user transactions, we will call them *pseudo-transactions*.

Assume that site  $S_i$  wishes to disconnect and "check-out" a set of items  $X$ . The procedure is as follows:

1. Site  $S_i$  selects a proxy as in basic sign-off mode and follows all the same steps to handle voting rights for replicated and non-replicated items.
2. At the same time,  $S_i$  initiates a pseudo-transaction to obtain write locks on the items in  $X$ .
3. If the pseudo-transaction is successful,  $S_i$  disconnects with update privileges on all the items in  $X$ . If the pseudo-transaction is not successful,  $S_i$  tries again or disconnects with update rights on a subset of the items.

When  $S_i$  reconnects, the reconnect procedure is the same as for basic sign-off except that  $S_i$  must complete the effects of the pseudo-transactions by transmitting the new value for all items in  $X$  and releasing the corresponding locks.

Figure 1 shows the activity at three sites,  $S_i$ ,  $S_j$  and  $S_k$ . Time proceeds from left to right and an asterisk (\*) indicates the disconnection and reconnection points in time.  $X_i$  indicates the version of data item  $X$  written by transaction  $i$ . In Figure 1,  $S_i$  first acquires a write lock on  $X$  with a pseudo-transaction (pt) and disconnects.  $t_1$  and  $t_2$  are examples of transactions that may be executed at the disconnected site  $S_i$ . Site  $S_j$  later disconnects with a write lock on  $Y$  and executes transactions that read  $Y$  and  $Z$  and update  $Y$ , e.g.,  $t_4$ . Site  $S_k$  remains connected and executes transaction  $t_3$ .  $S_i$  can execute transactions during its disconnection



$S_i$	*pt: wl-X & disconnect					*reconnect & w( $X_2$ ) c							
		$t_1$	$r(X_0)$	$r(Z_0)$	$w(X_1)$	c		$t_2$	$r(X_1)$	$w(X_2)$	c		
$S_j$	*pt: wl-Y & *disconnect										*reconnect & w( $Y_4$ ) c		
				$t_4$	$r(Y_0)$	$w(Y_4)$	c						
$S_k$		$t_3$	$r(Z_0)$	$w(Z_3)$	c			$t_5$	$b(X_0)$	$b(Y_0)$	$r(Z_3)$	$w(Z_5)$	c

**Figure 2. Relaxed Check-out Mode Using Browse locks**

to access the “old” value of  $X$  during the disconnect? In the relaxed check-out mode, we allow reads of the old value of  $X$  by allowing transactions at other sites to get a browse lock when they cannot get a read lock because the item is write locked by a pseudo-transaction. In this way, other sites can see the old value of  $X$  and  $S_i$  can read and write  $X$  while disconnected. Relaxed check-out mode allows some non-serializable executions and permits greater concurrency. Figure 2 shows an example execution using browse locks. In this example, transaction  $t_5$  at site  $S_k$  cannot get read locks on  $X$  or  $Y$ , so it gets browse locks on them. The non-serializable executions occur because of the browse lock. In Figure 2, there is a dependency cycle involving  $t_1, t_3$  and  $t_5$ .

In *Relaxed check-out* mode, we allow a site  $S_i$  to disconnect and check-out some items,  $X$ , with write locks and also allow  $S_i$  to read those items that were not locked. Other sites can choose to let their transactions get browse locks on  $X$  realizing that they may be reading old data. The rules that must be followed are:

1. Only those items,  $X$ , locked by its pseudo-transactions at disconnect time, can be modified by  $S_i$  during disconnect.
2. Those items in  $X$  can be browsed but not modified by other sites. Any transaction which browses an item in  $X$  knows it has given up serializability.
3. Items not locked by pseudo-transactions at disconnect time can be accessed for reading in transactions executed by  $S_i$  during disconnect.

This corresponds to degree 2 ANSI isolation levels [2] “cursor stability” if browse locks are only used on write locks held by pseudo-transactions. The browse permits the reading of old data and non-repeatable reads if the item is browsed again after the disconnected site reconnects. However, the read will never be that of an uncommitted and thus truly dirty value. If browse locks are allowed on items that are write locked by any transaction, pseudo or “real”, then the appropriate isolation level is degree 1, “browse”.

## 4 Discussion

Mobile computers have added a lot of flexibility to distributed systems, however, distributed databases have traditionally not been very flexible because of the need to synchronize transactions at all sites. The three proposed modes increase flexibility and allow distributed databases to take advantage of mobility. The implementation in a synchronous system with eager update is fairly straight-forward as the pseudo-transactions simply obtain locks in the usual manner. The implementing in an asynchronous system which uses an “epidemic” algorithm for replica update that allows update-anywhere capability and maintains one-copy serializability is explained in [4].

## References

- [1] M. Faiz and A. Zaslavsky. Database Replica Management Strategies in Multidatabase Systems with Mobile Hosts. In *Proceedings of the 6th International Hong Kong Computer Society Database Workshop*, Mar. 1995.
- [2] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufman, 1993.
- [3] J. Holliday, D. Agrawal, and A. El Abbadi. Exploiting Planned Disconnections in Mobile Environments. In *Proceedings, 10th IEEE Workshop on Research Issues in Data Engineering (RIDE2000)*, pages 25–29, Feb. 2000.
- [4] J. Holliday, D. Agrawal, and A. El Abbadi. Planned Disconnections for Mobile Databases. Technical Report TRCS 00-07, Department of Computer Science, University of California at Santa Barbara, May 2000.
- [5] P. Keleher. Decentralized Replicated-Object Protocols. In *Proceedings of the 18th ACM Symposium on Principles of Distributed Computing*, Apr. 1999.
- [6] P. Reiher, J. Heidemann, D. Ratner, G. Skinner, and G. Popek. Resolving File Conflicts in the Ficus File System. In *Proceedings, Summer USENIX Conf.*, pages 183–195, June 1994.
- [7] R. Unland and G. Schlageter. A Transaction Manager Development Facility for Non Standard Database Systems. In A. K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, pages 400–466. Morgan Kaufmann Publishers, 1992.
- [8] D. Walborn and P. K. Chrysanthis. Pro-motion: Management of mobile transactions. In *Proceedings of the 11th ACM Symposium on Applied Computing*, 1997.