1994

Maintaining Consistency of data in Mobile Distributed Environments

Evaggelia Pitoura

Bharat Bhargava
Purdue University, bb@cs.purdue.edu

Report Number:
94-025

MAINTAINING CONSISTENCY OF DATA IN MOBILE DISTRIBUTED ENVIRONMENTS

Evaggelia Pitoura
Bharat Bhargava

CSD TR-94-025
April 1994

Maintaining Consistency of Data in Mobile Distributed Environments

Evaggelia Pitoura and Bharat Bhargava
Department of Computer Science
Purdue University
West Lafayette, IN 47907-1398
{pitoura, bb}@cs.purdue.edu

Report CSD-TR-94-025

Abstract

It is expected that in the near future millions of users will have access to on-line distributed databases through mobile computers. This possibility raises challenging questions regarding the consistency and availability of data. To deal with the frequent, predictable and varying in degree disconnections that occur in a mobile environment, we introduce weaker notions of consistency and special transaction operations for handling inconsistencies. Specifically, a database is partitioned into a set of clusters. While all data inside a cluster are mutually consistent, degrees of inconsistency are allowed among data at different clusters. The cluster configuration is dynamic and when clusters are merged, strict consistency is restored. We allow transactions to exhibit certain degrees of tolerance for inconsistencies by introducing strict and loose operations. Loose operations are operations that can be executed under weaker consistency requirements. We define correctness criteria for schedules that involve loose operations and compare them with traditional serializability criteria. Finally, we argue that our model is appropriate for a variety of other applications where the distributed sites are connected through limited bandwidth, costly communication links or tend to be highly autonomous.

1 Introduction

In the recent past, technical advances in the development of portable computers and the rapidly expanding cordless technology have provided the

basis for accessing on-line distributed databases through wireless connections. Today, when users move, unplug their computer from the local area network, transport it, and plug it back to the local area network at their destination. In a mobile environment, users will have the ability to retain their network connection even while moving. This possibility raises new challenging problems regarding the availability and consistency of data. Though the problem is old and well-studied, the new environment adds new parameters to it.

1.1 Characteristics of the Mobile Environment

Mobile environments consist of two distinct sets of entities: mobile hosts and fixed hosts. Some of the fixed hosts, which are called *base stations* or *Mobile Support Stations (MSS)*, are augmented with a wireless interface to communicate with mobile hosts [IDGQM91]. Each mobile host can directly communicate with one support station, the one covering the geographical area in which the mobile host moves. A distributed database may be stored at mobile as well as static hosts, and be queried and updated over the wired and the wireless network. For example, insurance agents may interact through their mobile station with a database storing consumer records, while traveling salesperson may access inventory databases [IB93b].

In this environment, *bandwidth limitations* impose severe restrictions on the volume of data that can be transferred. While the growth in physical network bandwidth has been tremendous (in current technology Ethernet provides 10Mbps, FDDI 100 Mbps and ATM 155 Mbps), products for wireless communication achieve only 2Mbps for radio communication, and 9-14 kbps for cellular telephony [FZ93]. Besides, bandwidth consumption should be a major concern of mobile computing designs, because data transmission over the air is also *monetarily expensive* [Hay92].

Mobile computing environments are characterized by much greater variation in network bandwidth than traditional designs leading to various *degrees of disconnections* depending on the available bandwidth and noise of the communication channel [FZ93, IB93a]. Thus, while in a non-mobile distributed system a host operates in one of two modes regarding its connection to the rest of the network (either connected to it or totally disconnected from it), in a mobile environment there may be various possible modes of operation, depending on the bandwidth availability. The *frequency* of these disconnection is very high in comparison with non-mobile environments. Added to the above sources of disconnections is the limited lifetime

of batteries, usually two or three hours under "normal" use. Finally, certain disconnections are considered *foreseeable*, since they can be detected by changes in the signal strength, by predicting the battery's lifetime, or by utilizing knowledge of the bandwidth distribution [AK93, IB93a].

1.2 Maintaining Consistency

Bandwidth limitations and the frequent, predictable, and varying in degrees disconnections add new decisive parameters to the problem of efficient data distribution and make it a very important issue for increasing both performance and availability. Current research in mobile databases has addressed the issue of data distribution by proposing dynamic replication schemas [WJ92, HSW94, AZ93] and cache techniques [BI93]. In this paper, we tackle a different problem in data distribution, that of the maintenance of consistency among the distributed data. To increase performance and availability, we propose a model that allows inconsistencies between copies in a mobile database by taking advantage of the particularities of the environment.

Maintaining full data consistency over all distributed sites imposes unbearable overheads in mobile environments [AK93, IB93a, PB93]. In this paper, we propose a more flexible model. Semantically related or closely located data are grouped together to form a cluster. While full consistency is required for all data inside a cluster, degrees of consistency are defined for data located at different clusters. The degree may depend on the availability of bandwidth in that when bandwidth is in high demand, users may have to tolerate higher degrees of inconsistency. The cluster configuration is dynamic as new clusters may be defined or clusters may merge during operation. Degrees of consistency have a different meaning for different types of constraints between data. In this paper, we focus on replication constraints. Users can explicitly specify through loose and strict operations whether loose consistency is appropriate for their applications.

The organization of the remainder of this paper is as follows. Section 2 introduces the concept of clusters, and section 3 introduces loose and strict operations and the transactions that employ them. In Section 4, we specify correctness criteria and graph-based methods for maintaining intra-cluster consistency. Section 5 presents criteria and graph-based tests for restoring inter-cluster consistency during cluster merging. In Section 6 we compare our work with related research and finally, in Section 7, we offer conclusions and future work.

2 Consistency Clusters

The items of a database are partitioned into *clusters*. Clusters may be associated with the degree of connection. For instance, data stored in hosts that are partially or totally disconnected from the fixed network may be considered as forming a cluster. Alternatively, clusters may be defined based on the type of data. A characteristic example of such data is the data representing the location of a mobile host. This kind of data are fast-changing and the maintenance of their full consistency could cause unbearable overheads.

The cluster configuration is dynamic rather than static. By taking advantage of the predictable nature of disconnections, clusters of data may be explicitly created or merged upon the disconnection or connection of the associated mobile host. The definition of clusters may be also explicitly provided by the users based on the semantics of their data or applications. Finally, information stored into a user's profile may be utilized to determine clusters. For example, data that are most often accessed by a user or data that are in a great extent private can be considered as belonging to the same cluster independent of their location. Clusters are the units of consistency in that all data inside a cluster are required to be fully consistent.

Formally, a *mobile database* MD is a finite set of data items. An MD is partitioned into a set of clusters Cl_i, $i \in N$, where Cl_i is a set of data items (see Figure 1 for an example of clustering based on location). We use the notation x_i to indicate that the data item $x_i \in Cl_i$. We say that an item $x \in$ MD iff $x \in Cl_i$ for some $i \in N$. A database (or a cluster) *state* is defined as a mapping of every data item to a value of its domain. Data items are related by a number of restrictions called *integrity constraints* that express relationships of data items that a database state must satisfy. Integrity constraints among data items inside the same cluster are called *intra-cluster* constraints and constraints among data items at different clusters are called *inter-cluster* constraints.

Definition 1 (m-consistency) *A cluster state is consistent iff all intra-cluster integrity constraints hold. A mobile database state is* m-consistent *iff all cluster states are consistent and all inter-cluster integrity constraints are* m-degree consistent.

The definition of m-degree consistency depends on the type of inter-cluster constraints. In this paper, we consider only replication constraints. A data item $x_i \in Cl_i$ is a *copy (replica)* of a data item $x_j \in Cl_j$ if $i = j \Rightarrow x_i =$

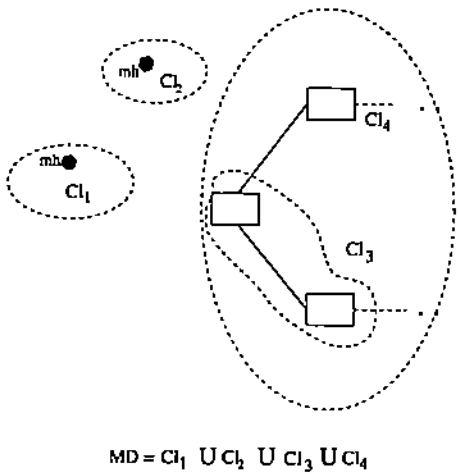

Figure 1: Example Cluster Configuration

x_j, and i $\neq$ j $\Rightarrow x_i$ = m-degree(x_j). In other words, copies are data items that have the same data value while in the same cluster, and while in different clusters their values are associated by an appropriately defined m-degree relation. For cached (replicated) data, the m-degree relation may express the divergence from the value of the primary copy as in quasi copies [ABGM90]. In this case, the allowable degree may be bounded by limiting the number of versions, by setting a maximum value on the allowable deviation, or by limiting the number of transactions that can operate on inconsistent values. The degree may vary based on the availability of network bandwidth by allowing little deviation in cases of higher bandwidth availability and higher deviation in cases of low bandwidth availability.

There are many other alternative ways of defining degrees [SR90]. For instance, one such way is by limiting the number of data items or data copies that can diverge. The criteria developed in this paper are independent of how the degree is defined. In the following, we will use the term data item to refer to any replica of a data item, and the term copy to refer to a specific physical copy of a data item.

3 Transactions with Loose and Strict Semantics

We have defined degrees of consistency and related them with the degrees of disconnection. In this section, we discuss how user transactions can deal with degrees of inconsistency.

3.1 Loose and Strict Operations

In an m-consistent MD there are data items whose values may not satisfy inter-cluster constraints. However, during total or partial disconnections this data may be the only data that is available to a user. To maximize local processing and limit network accesses, we propose allowing the user to interact with locally (in a cluster) available m-consistent data by introducing two new kinds of operations, loose reads and loose writes. These operations allow users to operate on m-consistent data when the lack of strict consistency does not affect the semantics of their transactions. We call the standard read and write operations strict read and strict write operations to differentiate them from the loose operations.

Formally, a transaction is a partial order of read, write, commit and abort operations. The exact semantics of loose and strict operations depend on the type of data on which they operate. In the case of data copies, a *loose read* operation ($LR[x]$) reads the locally available copy of x, that is the value written by the last loose or strict write operation. A *loose write* operation ($LW[x]$) writes the local copy and is not permanent unless it is committed in the merged network. A *strict read* operation ($SR[x]$) reads the value of x written by the last strict write operation. Finally, the value written by a *strict write* operation ($SW[x]$) becomes permanent after commitment. $\mathrm{LR}_j[x]$, $\mathrm{SR}_j[x]$, $\mathrm{LW}_j[x]$, $\mathrm{SW}_j[x]$ represent database operations issued by the transaction T_j for data item x. A_j and C_j are the abort and commit operations of transaction T_j.

3.2 Transaction Types

We distinguish two types of transactions: (a) transactions that consist only of loose read and loose write operations and are called *loose transactions*; and (b) transactions that consist only of strict read and strict write operations and are called *strict transactions*. Loose transactions access data copies that belong to the same cluster. Finally, *read-only* or *query* transactions are a special type of loose or strict transactions that consist only of read

operations. We discuss read-only transactions further in Section 6.1.

	$LR(x_i)$	$SR(x_i)$	$LW(x_i)$	$SW(x_i)$
$LR(x_j)$	NO	NO	YES, if $i=j$	YES $i=j$
$SR(x_j)$	NO	NO	NO	YES $i=j$
$LW(x_j)$	YES, if $i=j$	NO	YES, if $i=j$	YES $i=j$
$SW(x_j)$	YES $i=j$	YES $i=j$	YES $i=j$	YES $i=j$

$x_i = copy(x_j)$

Table 1: Conflict Relation for Operations on Data Copies

Loose transactions have two commit points, a *local commit* in the associated cluster and an implicit *global commit* after merging. The local commit point is expressed by an explicit commit operation. Updates made by locally committed transactions are only revealed to other loose transactions in the same cluster. These changes are revealed to strict transactions only after merging, that is when local transactions become globally committed. The updates of a loose transaction are considered permanent only after global commitment. Before global commitment a loose transaction may be undone even after being locally committed. Conceptually, we can think of local transaction managers as maintaining two copies of a data item, one that is updated by strict transactions and one that is updated by both strict and loose transactions. The first is read by strict read operations and the second by loose read operations. We use the notation $C_j[i]$ to indicate that transaction T_j is locally committed in cluster Cl_i.

Definition 2 (loose transaction) *A loose transaction is a partial order (<) of loose read, loose write, abort and local commit operations, which must specify the order of conflicting data operations and contains exactly one abort or commit operation which is the last in the order. Two loose*

data operations conflict *if they access the same copy of a data item and at least one of them is a loose write operation. Note, that loose transactions have two commit points, a local commit point specified by the local commit operation and a global commit point after merging.*

Definition 3 (strict transaction) *A strict transaction is a partial order (<) of strict read, strict write, abort and commit operations, which must specify the order of conflicting data operations and contains exactly one abort or commit operation which is the last in the order. Two strict data operations* conflict *if they access the same copy of a data item and at least one of them is a strict write operation. Strict transactions have only one global commit point.*

A loose transaction is a transaction that may read m-consistent data and whose writes may be undone any time before merging. Strict transactions have the usual semantics. Loose transactions may be useful for many areas that do not require exact values of data such as gathering information for statistical purpose. Allowing updates in loose transactions adds functionality and it is appropriate for handling mostly private data for which conflicts are rare.

3.3 Schedules

To process operations of a transaction, a DBMS translates operations on data items into operations on the replicated copies of those data items. We formalize this translation by a function h. An LR operation is translated into a loose read of the locally available copy and an LW operation to a loose write of the locally available copy. This update cannot be seen by other strict transactions until cluster merging, when loose transactions are globally committed. Abort and commit operations of a loose transaction are mapped to local abort and commit operations in the associated cluster. For strict operations, h maps each $SR[x]$ into $SR[x_i]$, where x_i is a copy of x, and each $SW[x]$ into $SW[x_{j_1}]$, ..., $SW[x_{j_k}]$, for some copies x_{j_1}, ..., x_{j_k}, of x. Abort and commit operations are mapped to (global) abort and commit operations.

Intuitively, a (complete) intra-cluster schedule, IAS, is an observation of an interleaved execution of transactions before merging, that includes locally committed loose transactions and (globally) committed strict transactions. Formally,

Definition 4 (intra-cluster schedule) *A (complete) intra-cluster schedule, IAS, over* $T = \{T_0, T_1, ..., T_n\}$ *is a pair (OP, <) where < is a partial ordering relation where*

1. $OP = h(\bigcup_{i=0}^{n} Ti)$ *for some translation function h*
2. *for each* T_i *and all operations* op_i, op_j *in* T_i, *if* $op_i < op_j$, *then every operation in* $h(op_i)$ *is related by <* *to every operation in* $h(op_j)$.
3. *for all read operations* $R_j[x_i]$ ($R_j[x_i] = SR_j[x_i]$ *or* $R_j[x_i] = LR_j[x_i]$) *there is at least one* $SW_k[x_i]$ *operation,* $SW_k[x_i] < R_j[x_i]$
4. *all pairs of conflicting operations are related by <* *(where conflicts are defined in Table 1)*
5. *if* $SW_j[x] < SR_j[x]$ *and* $h(SR_j[x]) = SR_j[x_i]$ *then* $SW_j[x_i] \in h(SW_j[x])$
6. *if* $SW_j[x_i] \in h(SW_j[x])$ *for some strict transaction* T_j *then* $SW_j[y_i] \in h(SW_j[y])$ *for all y written by* T_j *for which there is a* $y_i \in Cl_i$.

Condition (6) above indicates that for a strict transaction, if a write is translated to a write on a data copy at a cluster Cl_i then all other writes of this transaction must also write the corresponding copies at cluster Cl_i. This condition is necessary for ensuring that loose transactions do not see partial results of a strict transaction.

Thus, in an intra-cluster schedule a loose read operation *reads-x_i-from* the transaction that has last loose or strict written x_i. A strict read operation *reads-x_i-from* the transaction that has last strict written x_i. A (loose or strict) transaction *reads-x-from* a transaction if for some copy x_i it reads-x_i-from that transaction. Note, that if we want the loose operations at a cluster to read only values written by loose transactions, we can define h such that no strict transaction writes at that cluster.

A (complete) inter-cluster schedule, IES, models execution after merging, where all final local write operations are taken into consideration.

Definition 5 (inter-cluster schedule) *An inter-cluster schedule IES based on* $IAS = (OP, <)$ *is a pair* $(OP', <')$ *where*

1. $OP' = OP$
2. *for any* op_i *and* $op_j \in OP'$, *if* $op_i < op_j$ *in IAS then* $op_i <' op_j$ *in IES and*
3. *in addition,* $LW_j[x_i]$ *conflicts with* $SR_j[x_i]$.

4 Intra-Cluster Serializability

Before merging transactions must maintain m-consistency among clusters and strict consistency inside each cluster. In this section we propose a criterion and develop a graph-based test for characterizing correct intra-cluster schedules.

4.1 Correctness Criterion

Depending on the definition of degrees of consistency we can maintain m-degree consistency of the inter-cluster constraints by limiting the number of local updates, and by controlling the h function. If for example a loose transaction at a cluster C_i must read data only k-version old, we may define h to modify the data at this cluster every k updates.

If each transaction maintains m-consistency when executed alone, and if the projection of all transactions on all copies at a cluster is serializable, then serializability of each of the projections suffices to ensure that each loose transaction gets a consistent view, that is the integrity constraints between the data that it reads hold. Note that this is true because if a transaction maintains m-consistency, then its projection on each cluster also maintains m-consistency, as a consequence of condition (6) of the definition of an IAS schedule.

In addition, for hiding replication from strict transactions, we must require that strict transactions operate as if there is only one copy of each data item in the system. Thus strict transactions must be one-copy (1C) serializable [BHG87]. This is not true however for loose transactions which are allowed to read out-of-date copies.

Definition 6 (IAS Weak Correctness) *An intra-cluster schedule S is correct if its projection on strict transactions is 1SR (equivalent to an 1C schedule S_{1C}) and each of its projections on a cluster is conflict-equivalent to a serial schedule.*

Weak correctness ensures m-consistency only when the only inter-cluster constraints are replication constraints. Furthermore, although the above correctness criterion suffices to ensure that each loose transaction gets a consistent view, it does not suffice to ensure that loose transactions at different clusters get the same view, even in the absence of inter-cluster constraints. The following example demonstrates that even when all projections are serializable there may not be a compatible serial order for the whole schedule.

Example 1 *Assume we have two clusters* $C_1 = \{x_1, y_1\}$ *and* $C_2 = \{w_2, z_2, l_2\}$ *and the following strict transactions (after applying h):* $ST_1 = SW_1(x_1)$ $SW_1(w_2)C_{ST_1}$ *and* $ST_2 = SW_2(y_1)SW_2(z_2)SR_2(x_1)C_{ST_2}$. *In addition, at cluster* Cl_1 *we have the loose transaction* $LT_3 = LR_3(x_1)LR_3(y_1)C_{LT_3}[1]$, *and at cluster* Cl_2 *the loose transactions:* $LT_4 = LR_4(z_2)$ $LW_4(l_2)C_{LT_4}[2]$, *and* $LT_5 = LR_5(w_2)$ $LR_5(l_2)C_{LT_5}[2]$. *(For simplicity, we do not show the transaction that initially writes all items.)*

The following is a possible IAS schedule:

$$LR_5(w_2)SW_1(x_1)LR_3(x_1)SW_1(w_2)C_{ST_1}SW_2(y_1)SW_2(z_2)SR_2(x_1)$$
$$C_{ST_2}LR_3(y_1)C_{LT_3}[1]LR_4(z_2)LW_4(l_2)C_{LT_4}[2]LR_5(l_2)C_{LT_5}[2]$$

The projection on C_1*:*

$$SW_1(x_1)LR_3(x_1)C_{ST_1}SW_2(y_1)SR_2(x_1)LR_3(y_1)C_{LT_3}[1]$$

is serializable as $ST_1 \rightarrow ST_2 \rightarrow LT_3$

The projection on C_2*:*

$$LR_5(w_2)SW_1(w_2)C_{ST_1}SW_2(z_2)C_{ST_2}LR_4(z_2)$$
$$LW_4(l_2)C_{LT_4}[2]LR_5(l_2)C_{LT_5}[2]$$

is serializable as $ST_2 \rightarrow LT_4 \rightarrow LT_5 \rightarrow ST_1$ □

Weak consistency may be appropriate when there are no integrity constraints between clusters other than replication. In general, the following is a stronger correctness criterion that ensures both that loose transactions get the same consistent view and that intra-cluster constraints hold. Obviously, strong correctness implies weak correctness.

Definition 7 (IAS Strong Correctness) *An intra-cluster schedule S is correct if its projection on strict transactions is ISR (equivalent to an IC schedule* S_{1C}*) and it is conflict-equivalent to a serial schedule* S_S *such that the order of transaction in* S_S *is consistent with the order of transactions in* S_{1C}.

Note, that since loose transactions do not directly conflict with loose transactions at other clusters, the following is an equivalent statement of the above definition,

Definition 8 (IAS Strong Correctness (alternative definition)) *An intra-cluster schedule S is correct if its projection on strict transactions is 1SR (equivalent to an 1C schedule S_{1C}) and each of its projection on a cluster Cl_i is conflict-equivalent to a serial schedule S_{S_i} such that the order of transactions in S_{S_i} is consistent with the order of transactions in S_{1C}.*

4.2 Graph Characterization

To determine whether an IAS schedule is correct we will use a modified serialization graph, that we call the *intra-cluster serialization graph* (IASG) of the IAS schedule. In the following we make the assumption that the readset of a transaction contains its writeset. The reason for this assumption is to avoid NP-complete problems in checking serializability.

To represent conflicts between strict transactions, we use a replicated data Serialization Graph (SG). An SG [BHG87] is a serialization graph augmented with additional edges to take into account the fact that operations on different copies of the same data item may also cause conflicts. Acyclicity of the SG implies 1C serializability of the corresponding schedule.

To represent conflicts between loose transactions in the same cluster and conflicts between loose and strict transactions we use a simple serialization graph. In general, there are three types of edges in this graph [DGMS85]:

a. *dependency edges* that represent the fact that a transaction reads a value produced by another transaction;

b. *precedence edges* that represent the fact that a transaction reads a value that was later changed by another transaction;

c. *interference edges* that indicate that a transaction reads an item written by a transaction in another cluster.

In an IASG, we have all types of edges between strict transactions, and dependency and precedence edges between loose transactions in the same cluster. The following lemma shows the types of edges between loose and strict transactions. Note that there exist no edge between loose transactions at different clusters.

Lemma 1 *Let LT_i represent a loose transaction at cluster Cl_i and ST a strict transaction, then the IASG graph induced by an IAS can include* **only** *the following edges between them:*

- *a dependency edge from ST to LT_i*
- *a precedence edge from LT_i to ST*

Proof: Straightforward from the conflict relation, since the only conflicts between loose and strict transactions are due to strict writes and loose reads of the same copy of a data item. □

Theorem 1 *If the IASG is acyclic then the IAS is strongly correct.*

Proof (brief): When a graph is acyclic then each of its subgraphs is acyclic thus SG is acyclic. Acyclicity of the SG implies 1C serializability of strict transactions since strict transactions read only values written by strict transactions. Let T_1, T_2, ... , T_n be all transactions in IAS. Thus T_1, T_2, ... , T_n are the nodes of the IASG. Since IASG is acyclic it can be topologically sorted. Let $T_{i_1}, T_{i_2}, \ldots, T_{i_n}$ be a topological sort of the edges in IASG, then by a straightforward application of the serializability theorem [BHG87] the IAS is conflict equivalent to the serial schedule $S_S = T_{i_1}, T_{i_2}, \ldots, T_{i_n}$. This order is consistent with the partial order induced by a topological sorting of the SG graph, let S_{1C} be the corresponding serial schedule. Thus the order of transactions in S_S is consistent with the order of transactions in S_{1C}. □

Note that if we employ weak *IAS* correctness as our correctness criterion, then the transaction managers at each cluster must only synchronize projections on that cluster. Global control is needed only for synchronizing strict transactions.

5 Cluster Merging

During merging we must enforce full consistency, that is reconcile values of different copies of the same data item at different clusters. There are different approaches to the problem of reconciliation varying from purely syntactic to purely semantic [DGMS85]. In this paper, we adopt a purely syntactic application-independent approach. We attempt to accept as many loose writes as possible without violating the 1C serializability of strict transactions.

5.1 Correctness Criterion

Intuitively a IES schedule is correct if it is equivalent to an 1C serializable schedule ignoring the fact that loose transactions may not read the value produced by the latest write operation but a value written by an older write.

Definition 9 (IES Correctness) *An inter-cluster schedule is correct if it is based on a correct IAS schedule S_{IAS} and all strict transactions have the same read from relation as in the S_{IAS}.*

To resolve conflicts in inter-cluster schedules we roll back the transactions with loose write operations.

Lemma 2 *Undoing a loose transaction results in undoing only loose transactions in the same cluster.*

Proof (brief): At each step we have to undo all transactions that read the value written by a loose transaction, only loose transactions can read this value. □

5.2 Graph Characterization

First, we will see how the serialization graph IASG for an intra-cluster schedule is modified if we take into consideration conflicts between LW and SR.

Lemma 3 *Let LT_i be a loose transaction at cluster Cl_i and ST a strict transaction, then the serialization graph IESG induced by an IES may include in addition to the edges of the IAG the following edges:*

- *a dependency edge from LT_i to ST and*
- *a precedence edge from ST to LT_i*

Proof: If $LW[x_i] > SR[x_i]$ then we add a dependency edges from LT_i to ST. If $SR[x_i] > LW[x_i]$ then we add a precedence edge from ST and LT_i. Note, that interference edges between transactions at different clusters are not reported. □

In the original graph transactions that access the same item but different copies of the same item do not conflict. Thus serializability of the above graph does not suffice. To force such conflicts we further expand the IESG graph by:

1. first inducing a write order as follows, if T_i and T_k (loose or strict) write any copy of an item x then either $T_i \rightarrow T_k$ or $T_k \rightarrow T_i$; and
2. then inducing a strict read order as follows, if a strict transaction ST_j reads-x from ST_i at S_{IAS} and a loose transaction LT follows ST_i then we add an edge $ST_j \rightarrow LT$.

Theorem 2 *If the IESG graph is acyclic then the IES is correct.*

Proof: Clearly if the graph is acyclic, the corresponding graph for the IAS is acyclic (since to get the new graph we only add edges). We will show that if the graph is acyclic then the read-from relation for strict transactions in the inter-cluster schedule S_{IES} is the same as in the underlying intra-cluster schedule S_{IAS}. Assume that ST_j reads-x-from ST_i in S_{IAS}. Then $ST_i \rightarrow ST_j$. Assume for the purposes of contradiction, that ST_j reads-x-from LT. Then LT writes x in S_{IES} and since ST_i also writes x either (a) $ST_i \rightarrow LT$ or (b) $LT \rightarrow ST_i$. In case (a) $ST_j \rightarrow LT$, contradiction since ST_j reads-x-from LT. In case (b) $LT \rightarrow ST_i$, that is LT precedes ST_i which precedes ST_j, which again contradicts the assumption that ST_j reads-x-from LT. □

6 Relation to other Criteria

Strict consistency requires that all replicas of a data item are synchronized. One-copy serializability [BHG87] hides from the user the fact that there can be multiple replicas of a data item and ensures strict consistency. Whereas 1C-serializability may be an acceptable criterion for strict transactions, it is too restrictive for applications that could tolerate m-consistent locally available copies.

The partitioning of a database into clusters resembles the *network partition problem* [DGMS85], where site or link failures fragment the network of database sites into isolated subnetworks called partitions. Clustering is conceptually different than partitioning in that it is electively done to increase performance. Furthermore, whereas all partitions are isolated, clusters may be partly connected. Strategies for network partition face similar competing goals of availability and correctness. These strategies range from *optimistic*, where any transaction is allowed to be executed in any partition, to *pessimistic*, where transactions in a partition are restricted by making worst-case assumptions about what transactions at other partitions are doing. Our model offers a hybrid approach. Strict transactions may be performed only if 1C-serializability is ensured (in a pessimistic manner). Loose transactions may be performed locally (in an optimistic manner). To merge updates performed by loose transactions we use a strictly syntactic approach.

6.1 Read-only transactions

In this section we compare our model with weaker notions of consistency associated with read-only or query transactions. Read-only transactions do not modify the database state, thus their execution cannot lead to inconsistent database states. In our framework read-only transactions with weaker consistency requirements are considered as a special case of a loose transaction.

In [GMW82] two requirements for query transactions were introduced: consistency and currency requirements. *Consistency* requirements specify the degree of consistency needed by a read-only transaction. In this framework, a read-only transaction may have: (a) *no* consistency requirements; (b) *weak* consistency requirements if it requires a consistent view (that is, if all consistency constraints that can be fully evaluated with the data read by it must be true); or (c) *strong* consistency requirements if the schedule of all update transactions together with all other strong consistency queries must be consistent. While in our model strict read-only transaction always have strong consistency requirements, loose read-only transactions can be tailored to have any of the above degrees based on the criterion used for IAS correctness. Loose read-only transactions may have no consistency requirement if they are ignored from the IAS schedule, weak consistency if they are part of a weakly correct IAS schedule, and strong consistency if they are part of a strongly correct schedule. The *currency* requirements specifies what update transactions should be reflected by the data read. In terms of currency requirements, strict read-only transactions read the most-up-to-date data item available (i.e. committed). Loose read-only transactions may read older versions of data, depending on the definition of the m-degree.

Epsilon-serializability (ESR) [PL91] allows temporary and bounded inconsistencies in replicas to be seen by queries during the period among the asynchronous updates of the various copies of a data item. Read-only transactions in this framework are similar to loose read-only transactions with no consistency requirements. ESR bounds inconsistency directly by bounding the number of updates. In [WA92] a generalization of ESR was proposed for high-level type specific operations on abstract data types. In contrast, our approach deals with low-level read and write operations.

6.2 File Systems for Mobile Environments

Coda [KS92] treats disconnections as network partitions and follows an optimistic strategy. An elaborate reconciliation algorithm is used for merging file updates after the sites are connected to the fixed network. No degrees of consistency are defined and no transaction support is provided.

The idea of using different kinds of operations to access data is also adopted in [TD91] where a loose read operation was added to a file service interface. The semantics of operations are different in that no loose write is provided and since there is no transaction support, the correctness criterion is not based on 1C serializability.

7 Conclusions and Future Work

In this paper, we have studied consistency issues for distributed databases in mobile environments. Compared to traditional distributed environments, in mobile environments, communication is more expensive, bandwidth is scare and shows much greater variation, and disconnections are frequent and predictable. As a consequence, accessing locally available data, can lead to a significant improvement in availability, bandwidth utilization and performance. However, there are significant trade-offs between availability and performance versus consistency maintenance. To deal with this problem, we propose weaker notions of consistency appropriate for mobile environments.

The main contributions of this paper are as follows. First, we have formalized the notion of locality by introducing the idea of data clustering. Clusters are not the inevitable result of a network partition but can be explicitly defined to express physical locality, semantic proximity or similarity of consistency requirements. To take advantage of the predictability of disconnections in mobile environments, the cluster configuration is dynamic. Degrees of consistency among data that reside at different clusters may be used to model the degree of connection among those clusters. Second, we have defined two kinds of operations (loose and strict) to allow different applications to specify whether loose or strict consistency is appropriate for their correct execution. We have shown how loose transactions can be part of a concurrency controller and we have developed criteria and graph-based tests for the correctness of the schedules that include them. By offering to the applications the ability to specify explicitly when strict consistency is necessary for their execution, we gain in both resource utilization and availability.

Clustering may be also appropriate for large databases. As distributed databases grow in size and cover large geographical areas, new challenging problems regarding the availability and consistency of data are raised. Communication delays and packet losses will be a major concern in environments where communication is achieved through wide area networks [ZB93]. Clustering data which reside in sites located in the same geographical area seems to be a reasonable approach. Then communication inside a cluster will be relatively inexpensive and reliable. Clustering seems to be appropriate as well for databases that scale in the number of sites (versus scale in geographical distribution). In that case maintaining consistency of data residing in numerous sites is unrealistic. Clustering semantically related data seems an appropriate way to overcome this problem. Finally, the idea of degrees of consistency and loose operation may prove useful in multidatabase systems, which are confederations of autonomous pre-existing database systems. Local sites may be viewed as clusters and loose transaction as loose operations. However, in such environments, a more elaborate semantic-based method will be needed for merging local copies.

In this paper, we have focussed on a special type of dependency between clusters, namely data replication. In future studies, we plan to investigate how the semantics of loose operations can be generalized to operate on different types of dependencies, such as vertical and horizontal partitions or constraint dependencies [SR90]. We are currently developing a simulation system to evaluate the performance of our method.

References

[ABGM90] Rafael Alonso, Daniel Barbara, and Hector Garcia-Molina. Data Caching Issues in an Information Retrieval System. *ACM Transactions on Database Systems*, 15(3):359–384, September 1990.

[AK93] Rafael Alonso and Henry F. Korth. Database System Issues in Nomadic Computing. In *Proceedings of the 1993 SIGMOD Conference*, Washington, D.C., May 1993.

[AZ93] Swarup Acharya and Stanley Zdonik. An Efficient Scheme for Dynamic Data Replication. Technical Report CS-93-43, Dept. of Computer Science, Brown University, September 1993.

[BHG87] Philip A. Bernstein, Vassos Hadjilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems.* Addisson-Wesley, 1987.

[BI93] Danieal Barbara and Tomasz Imielinksi. Sleepers and Workaholics: Caching Strategies in Mobile Environments. Technical Report MITL-TR-58-93, MITL, 1993. to appear in SIGMOD 94.

[DGMS85] Susan B. Davidson, Hector Garcia-Molina, and Dale Skeen. Consistency in Partitioned Networks. *ACM Computing Surveys*, 17(3):341–370, September 1985.

[FZ93] George H. Forman and John Zahorjan. The Challenges of Mobile Computing. Technical Report UW-CSE-TR #93-11-03, University of Washington, Dept. of Comp. Science& Engineering, December 1993.

[GMW82] Hector Garcia-Molina and Gio Wiederhold. Read-Only Transactions in a Distributed Database. *ACM Transactions on Database Systems*, 7(2):209–234, June 1982.

[Hay92] D. Hayden. The New Age of Wireless. *Mobile Office*, May 1992.

[HSW94] Yixiu Huang, Prasad Sistla, and Ouri Wolfson. Data Replication for Mobile Computers. to appear in SIGMOD 94, 1994.

[IB93a] Tomasz Imielinksi and B. R. Badrinath. Data Management for Mobile Computing. *SIGMOD Record*, 22(1):34–39, March 1993.

[IB93b] Tomasz Imielinksi and B. R. Badrinath. Wireless Mobile Computing: Challenges in Data Management. Technical Report DCS-TR-296 and WINLAB TR-49, Rutgers, Dept. of Comp. Sciences, 1993. to appear in Communications of the ACM, 1994.

[IDGQM91] John Ioannidis, Dan Duchamp, and Jr Gerald Q. Maguire. IP-based Protocols for Mobile Internetworking. In *Proceedings of the ACM SIGCOMM Symposium on Communications, Architectures and Protocols*, pages 235–245, September 1991.

[KS92] James J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems*, 10(1):3–25, February 1992.

[PB93] Evaggelia Pitoura and Bharat Bhargava. Dealing with Mobility: Issues and Research Challenges. Technical Report TR-93-070, Purdue University, Dept. of Comp. Sciences, 1993.

[PL91] Calton Pu and A. Leff. Replica Control in Distributed Systems: An Asynchronous Approach. In *Proceedings of the ACM SIGMOD*, pages 377–386, 1991.

[SR90] Amit Sheth and Marek Rusinkiewicz. Management of Interdependent Data: Specifyin Dependency and Conssitency Requirements. In *Proceedings of the Workshop on the Management of Replicated Data*, pages 133–136, Houston, Texas, November 1990.

[TD91] Carl D. Tait and Dan Duchamp. Service Interface and Replica Management Algorithm for Mobile File System Clients. In *Proceedings of the First International Conference on Parallel and Distributed Information Systems*, pages 190–197, 1991.

[WA92] M.H. Wong and D. Agrawal. Tolerating Bounded Inconsistency for Increasing Concurrency in Database Systems. In *Proceedings of the 11th ACM PODS*, pages 236–245, 1992.

[WJ92] Ouri Wolfson and Sushil Jagodia. Distributed Algorithms for Dynamic Replicated Data. In *Proceedings of the 11th ACM PODS*, pages 149–163, 1992.

[ZB93] Yongguang Zhang and Bharat Bhargava. Wance: A Wide Area Network Communication Emulation System. In *Proceedings of IEEE Workshop on Advances in Parallel and Distributed Systems*, pages 40–45, Princeton, NJ, October 1993.