Maintaining Consistency of Data in Mobile Distributed Environments*

Evaggelia Pitoura and Bharat Bhargava
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

Abstract

To deal with the frequent, foreseeable and variable disconnections that occur in a mobile environment, we introduce a flexible, two-level consistency model. Semantically related or closely located data are grouped together to form a cluster. While all data inside a cluster are mutually consistent, degrees of inconsistency are allowed among data at different clusters. To take advantage of the predictability of disconnections, and to accommodate mobility, the cluster configuration is dynamic. We allow transactions to exhibit certain degrees of tolerance for inconsistencies by introducing strict and weak operations. Weak operations are operations that can be executed under weaker consistency requirements. We define correctness criteria for schedules that involve weak operations and compare them with traditional serializability criteria. Finally, we argue that our model is appropriate for a variety of other environments including very large databases and multidatabases.

1 Introduction

In the recent past, technical advances in the development of portable computers and the rapidly expanding cordless technology have provided portable computers with wireless connections that permit users to actively participate in distributed computing even while moving. The resulting distributed environment is subject to restrictions imposed by the nature of the wireless medium, and the resulting mobility of users.

A mobile distributed system consists of two types of hosts: mobile hosts and fixed hosts [12]. Some of the fixed hosts, called *base stations*, are augmented with a wireless interface to communicate with mobile hosts. The geographical area covered by a base station is called a *cell*. Each mobile host can directly communicate with one base station, the one covering its current geographical area.

Mobile users will desire access to private or corporate databases that will be stored at mobile as well as static hosts, and 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 salespersons may access inventory databases [12]. These databases, for reasons of reliability, performance, and cost will be distributed and replicated over many sites. The use of replicated databases entails both a schema for placing and locating copies, and concurrency and replica control protocols for ensuring correctness. Previous research on replication and mobility has mainly addressed the first issue [3, 10]. In this paper, we propose schemas for maintaining the consistency of replicated data, in a way that takes into consideration the restrictions imposed by the nature of mobile computing.

Motivation. Wireless communications and mobility introduce a new paradigm of distributed computing [8, 12, 17]. Today's computer systems often depend heavily for their operation on the rest of the network. Mobile computers, however, are very susceptible to network disconnections. The majority of these disconnections are voluntary. Frequently, users will deliberately avoid use of the network for cost or power consumption or because no networking capability is available at their current location. Thus, many users will be only occasionally connected to the rest of the network. Handling disconnections has been discussed extensively in the context of network partition [7]. Their frequency, however, forces making them part of normal operation and considering a new mode of operation called *disconnected operation* [12, 15]. Furthermore, in network partitions, disconnections are involuntary and mostly unpredictable since they result from network or host failures.

Wireless networks deliver much lower bandwidth than wired networks, and have higher error rates [8]. Mobile systems are also characterized by high variation in network bandwidth, that can shift one to four orders of magnitude, depending on whether the host is plugged in or using wireless access and on the type of connection at its current cell [8, 12]. Finally, while in static designs the location of users is fixed, in mobile environments it varies, thus the notion of locality changes as users move into new cells.

As a consequence, concurrency control schemas for mobile distributed databases should meet novel objec-

*in Proceedings of the 15th International Conference on Distributing Computing Systems, Vancouver, British Columbia, Canada, May 30 - June 2, 1995

This and related reports are available through www, URL address: http://www.cs.purdue.edu/people/pitoura tives. In particular, these schemas should:

1. support the autonomous operation of mobile hosts during disconnections,
2. reflect a greater concern for bandwidth consumption and constraints,
3. be able to adapt to varying connectivity conditions, and
4. take into account the changing locality.

Although replica control schemas that address some of the above concerns have already been studied in the context of mobile file systems (for instance, see [15, 22]), the database community is only starting to address these issues [4].

Our Approach. Maintaining data consistency over all distributed sites imposes unbearable overheads on mobile computing [2, 8, 11, 18]. 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 replicated data at different clusters. The degree of consistency may vary depending on the availability of network bandwidth among clusters. The cluster configuration is dynamic. We augment the database interface by adding weak operations. Users access locally (i.e., in a cluster) consistent data by issuing weak transactions and globally consistent data by issuing strict transactions. Weak operations support disconnected operation since a mobile host can operate disconnected as long as applications are satisfied with local copies. Furthermore, by allowing applications to specify their consistency requirements, better bandwidth utilization is achieved. Finally, the proposed schema models operations on imprecise location data.

The organization of the remainder of this paper is as follows. Section 2 introduces the concepts of clusters and degrees of consistency, and Section 3 introduces weak 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 and bounded inconsistency among clusters. 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. In Section 7, we show how the concepts of clustering and weak transactions can prove useful in two other contexts, namely in very large databases and in multidatabases. Finally, in Section 8, we offer conclusions and future work.

2 Consistency Clusters

A database is distributed at fixed and mobile hosts. Data are stored or cached at a mobile host to support its autonomous operation during disconnections and to avoid costly use of the network. We assume a fully distributed environment where users submit transactions from mobile or static hosts. Transactions may involve both remote data and data stored locally at the user's host. We consider the items of a database to be partitioned into *clusters.* Clusters are the units of consistency in that all data items inside a cluster are required to be fully consistent, while data items residing at different clusters may exhibit bounded inconsistencies.

Clustering may be based on the physical location of data. Under this definition, data located at the same, neighbor, or strongly connected hosts are considered to belong to the same cluster, while data residing at disconnected or remote hosts are regarded as belonging to separate clusters. The cluster configuration is dynamic. By taking advantage of the predictable nature of disconnections, clusters of data may be explicitly created or merged upon a forecoming disconnection or connection of the associated mobile host. Furthermore, to accommodate migrating locality, a mobile host can move to a different cluster when it enters a new cell.

Other definitions of clusters are also feasible. First, clusters may be defined based on the *semantics of data* with the most characteristic such case being location data. Location data, which represent the address of a mobile host, are fast changing data replicated over many sites. These data are often imprecise, since updating all their copies imposes unbearable overheads [13]. Second, the definition of clusters may be explicitly *provided by users* based on the requirements 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 some user or data that are in a great extent private to a user can be considered to belong to the same cluster independent of their location or semantics.

Formally, a *mobile database* MD is a finite set of data items. An MD is partitioned into a finite set of clusters Cl_i, i ϵ N, where Cl_i is a set of data items. We say that an item x ϵ MD iff x ϵ Cl_i for some i ϵ 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* for an integrity constraint depends on the type of the inter-cluster constraint. In this paper, we focus on the special case of replication constraints.

Degrees of Consistency for Replicated Data. A replication constraint R over a set of data items $\{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\}$ is defined as: $R(x_{i_1}, x_{i_2}, \ldots, x_{i_k})$ = (*for any* j, $l \in [1\ldots k]$ and ($x_{i_j} \in Cl_m$, $x_{i_l} \in$

Cl_n)) ($m = n \Rightarrow x_{i_j} = x_{i_l}$) *and* ($m \neq n \Rightarrow$ *m-degree*(x_{i_j}, x_{i_l})). That is, 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. In the following, we call the set $\{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\}$ a data item and denote it by x. We call the elements $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$ of the set, which are the physical entities stored in the database, (data) copies of x. For clarity, in the following, we collectively denote by x_i a representative of all copies of x located at cluster Cl_i.

For cached data, the m-degree relation may express the divergence of a cached (secondary) copy from the value of the primary copy (as in quasi copies [1]). In this case, the allowable degree of deviation may be bounded by (a) setting a maximum value on the allowable deviation, (b) setting a limit on the number of transactions that can operate on inconsistent copies, or (c) limiting the number of allowable versions. There are many alternative ways of defining degrees [20]. For instance, the m-degree relation may be defined by limiting the number of (d) data items or (e) data copies that can diverge. In Section 4.2.1, we show how to maintain degrees for each of the above cases.

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. This is an important property of bounded inconsistency since it provides applications with the capability to adapt to the currently available bandwidth, providing the user with data of variable level of detail or quality. For example, in the instance of a cooperative editing environment where multiple users are coediting a book, the application can display only one chapter or old versions of chapters under weak network connections and up-to-date copies of all chapters under strong network connections.

3 Weak and Strict Transactions

In an m-consistent MD, some data items do not satisfy inter-cluster constraints in the strict sense. However, during disconnections or when the connection is weak or costly, this may be the only data that a user can afford to access. To maximize local processing and reduce network access, we allow the user to interact with locally (in a cluster) available m-consistent data by introducing two new kinds of operations, *weak reads* and *weak writes*. These operations allow users to operate on m-consistent data when the lack of strict consistency can be tolerated by the semantics of their applications. We call the standard read and write operations strict read and strict write operations to differentiate them from weak operations. We distinguish two basic types of transaction units regarding consistency: (a) transactions that consist only of weak read and weak write operations and are called *weak transactions*; and (b) transactions that consist only of strict read and strict write operations and are called *strict transactions*. Weak transactions access data copies that belong to the same cluster, and can be considered local at that cluster. Each submitted transaction is decomposed to a number of weak and strict subtransactions according to the degree of consistency required by the application.

Of particular interest are a special kind of weak or strict transactions called *query* or *read-only* transactions, which are transactions that consist only of read operations and therefore do not interfere with any integrity constraints. Weak and strict read-only transactions are discussed further in Section 6.

3.1 Notation and formalism

We support two types of data operations, weak and strict. A *weak read* operation on a data item x ($W\_Read[x]$) reads a locally available copy of x, that is the value written by the last weak or strict write operation at that cluster. A *weak write* operation ($W\_Write[x]$) writes a local copy and is not permanent unless it is committed in the merged network. A *strict read* operation ($S\_Read[x]$) reads the value of x written by the last strict write operation. Finally, a *strict write* operation ($S\_Write[x]$) writes one or more copies of x. We use the subscript j to denote that a database operation is issued by a transaction T_j. A_j and C_j are the abort and commit operations of transaction T_j.

Weak transactions have two commit points, a *local commit* in the associated cluster and an implicit *global commit* after cluster merging. The local commit point is expressed by an explicit commit operation. We use the notation $\mathrm{C}_j[i]$ to indicate that transaction T_j is locally committed in cluster Cl_i. Updates made by locally committed weak transactions are only revealed to other weak transactions in the same cluster. These changes are revealed to strict transactions only after merging, that is when local transactions become globally committed. Thus, the updates of a weak transaction are considered permanent only after global commitment, since before global commitment a weak transaction may be undone even after being locally committed. Formally,

Definition 2 (weak and strict transactions) *A transaction (T) is a partial order (OP, <), where OP is the set of weak or strict read, weak or strict write, abort and local commit operations executed by the transaction, and < represents their execution order. The partial order must specify the order of conflicting data operations and contains exactly one abort or commit operation which is the last in the order. Two weak (strict) data operations* conflict *if they access the same copy of a data item and at least one of them is a weak (strict) write operation. Two types of transactions are supported, weak and strict. A* **weak** *transaction (WT) is a transaction where OP does not include any strict operations. A* **strict** *transaction (ST) is a transaction where OP does not include any weak operations.*

To implement the above mentioned semantics it suffices for each local transaction manager in cluster Cl_k to maintain two versions, x_k^s and x_k^w, of each data copy x_k. The version x_k^s is updated by strict transactions and is called *strict version*, whereas x_k^w is updated by both strict and weak transactions and is called *weak*

	$W\_Read_j(a_k)$	$S\_Read_j(a_k)$	$W\_Write_j(a_k)$	$S\_Write_j(a_k)$
$W\_Read_i(a_k)$			x	x
$S\_Read_i(a_k)$				x
$W\_Write_i(a_k)$	x		x	x
$S\_Write_i(a_k)$	x	x	x	x

Table 1: Conflict relation, a "x" entry indicates that the operations for the given row and column conflict. Row entries correspond to operations of transaction T_i and column entries to operations of transaction T_j

version. Strict transactions read x_k^s while weak transactions read x_k^w. Although in most cases, one version is sufficient to implement the above schema, for generality and understandability, we will use both versions in the following discussion.

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 *translation function h*. An $W\_Read(x)$ operation in a cluster Cl_k is translated into a read of the locally available copy $Read(x_k^w)$ and an W_Write operation to a write of the locally available copy $Write(x_k^w)$. Note, that this update cannot be seen by other strict transactions until cluster merging, when weak transactions are globally committed. Abort and commit operations of a weak transaction are mapped to local abort and commit operations in the associated cluster. For strict operations in a cluster Cl_k, h maps each $S\_Read[x]$ into $Read[x_i^s]$, where x_i^s is a copy of x, and each $S\_Write[x]$ into $Write[x_{j_1}^s]$, ..., $Write[x_{j_k}^s]$, and $Write[x_{j_1}^w]$, ..., $Write[x_{j_k}^w]$ for some copies x_{j_1}, ..., x_{j_k}, of x. Abort and commit operations are mapped to (global) abort and commit operations.

Which data copies are actually read or written when a database operation is issued on a data item depends on the coherency algorithm used. For example, h may be defined so that each strict write operation updates only one copy of each data item. In this case, the copy updated corresponds to a *primary copy* for that data item and one version of each data copy is sufficient for implementing the schema.

In the following, *Write* and *Read* operations are prefixed with W or S to indicate whether they have originated from weak or strict transactions respectively. To simplify the notation we omit the superscripts in data copies. Thus, a read operation on a data copy of x in cluster Cl_k that originated from a weak transaction is denoted $W\_Read(x_k)$. Table 1 summarizes conflicts between weak and strict data operations of two transactions. Two operations conflict if they access the same version of a data copy and one of them is a write operation.

Integrity Constraints and the Translation Function. In the case of integrity constraints other than replication constraints between data items, the translation function h can produce transactions that are inconsistent, in particular transactions that violate integrity constraints between weak versions in the same cluster. The following example is illustrative.

Example 1 *For simplicity consider only one cluster. Assume two data items b and c, related by the integrity constraint* $b > 0 \Rightarrow c > 0$*, and a consistent database state* $b^s = -1$, $b^w = -1$, $c^s = 2$ *and* $c^w = -4$. *Consider the transaction program:*

```
b = 10
 if c < 0
  then c = 10
```

If the above program is executed as a strict transaction $S\_Write(b)$ $S\_Read(c)$*, we get the database state* $b^s = 10$, $b^w = 10$, $c^s = 2$ *and* $c^w = -4$*, where the integrity constraint between the weak versions of b and c is violated.* □

The problem is that weak versions are updated to the current value of the strict version without taking into consideration integrity constraints among weak versions. Similar problems are faced when refreshing individual copies of a cache [1]. Some possible solutions include: (1) Each time a weak version is updated as a result of a strict write, the weak versions of all data related to it by some integrity constraint are also updated either after or prior to the execution of the transaction. This update is done following a reconciliation procedure for merging weak and strict versions (as in Section 5). In the above example, the versions of b and c should have been reconciled prior to the execution of the transaction, producing for instance the database state $b^s = -1$, $b^w = -1$ $c^s = 2$ and $c^w = 2$. Then, the execution of the transaction would result in the database state $b^s = 10$, $b^w = 10$, $c^s = 2$ and $c^w = 2$, which is consistent. (2) Each transaction program that writes a data item at a cluster is also translated to a weak transaction. The above example would result in transactions $S\_Write(b)$ $S\_Read(c)$ and $W\_Write(b)$ $W\_Read(c)$ $W\_Write(c)$ and a database state $b^s = 10$, $b^w = 10$, $c^s = 2$ and $c^w = 10$ which is consistent. (3) Updating weak versions is postponed by deferring any updates of weak copies that result from writes of the corresponding strict copies. A log of weak writes resulting from strict writes is kept. In this scenario, the execution of the transaction results in the database state $b^s = 10$, $b^w = -1$, $c^s = 2$ and $c^w = -4$, which is consistent.

3.2 Discussion

The above hybrid schema allows two types of transactions, weak and strict, to coexist. Weak transactions allow users to process local data without the overhead of long network accesses. Strict transactions need access to the network to guarantee consistency of their updates.

Weak reads give users the choice of reading an approximately accurate value of a data even under disconnections or weak connections. This value is sufficient for a variety of applications which do not require exact values. Such applications include gathering information for statistical purposes or making high-level decisions and reasoning in expert systems which can tolerate bounded uncertainty in input data. For instance, a customer may check the available credit before buying an item, and a traveling salesperson can check the available stock of an item before a sale. Finally, in another context, getting the approximate location of a mobile user may be sufficient for determining what type of location-based services, e.g., traffic information, is applicable to her/him.

Weak writes allow users to update local data without confirming those updates immediately. The validation of the updates is delayed till clusters are connected. Delayed updates can be performed during periods of low network activity to reduce demand on the peaks. Furthermore, transmitting many weak writes together as a block rather than one at a time can improve bandwidth usage. For example, a salesperson can locally update many data items, till these updates are finally confirmed, when the machine is plugged back to the network at the end of the day. However, since weak writes may not be finally accepted they must be used only when compensating transactions are available, or when the likelihood of conflicts is very low. For example, users can use weak transactions to update mostly private data and strict transactions to update highly used common data.

4 Maintaining Consistency in a Cluster Configuration

In this section we define correctness criteria and graph-based characterizations for the correct concurrent execution of weak and strict transactions in a given cluster configuration.

4.1 Schedules

Intuitively, a (complete) intra-cluster schedule, IAS, is an observation of an interleaved execution of transactions in a given cluster configuration, that includes locally committed weak transactions and (globally) committed strict transactions. Formally,

Definition 3 (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. *$OP = h(\bigcup_{i=0}^{n} T_i)$ for some translation function h.*
2. *For each T_i and all operations op_k, op_l in T_i, if $op_k < op_l$, then every operation in $h(op_k)$ is related by $<$ to every operation in $h(op_l)$.*
3. *All pairs of conflicting operations are related by $<$ (where conflicts are defined in Table 1).*
4. *For all read operations $S\_Read_j[x_i]$ there is at least one $S\_Write_k[x_i]$ operation, such that $S\_Write_k[x_i] < S\_Read_j[x_i]$ and for each $W\_Read_j[x_i]$ there is at least one write operation $Write_k[x_i]$, either weak or strict, such that $Write_k[x_i] < W\_Read_j[x_i]$.*
5. *If $S\_Write_j[x] < S\_Read_j[x]$ and $h(S\_Read_j[x]) = S\_Read_j[x_i]$, then $S\_Write_j[x_i] \epsilon\ h(S\_Write_j[x])$.*
6. *If $S\_Write_j[x_i] \epsilon\ h(S\_Write_j[x])$ for some strict transaction T_j then $S\_Write_j[y_i] \epsilon h(S\_Write_j[y])$ for all y written by T_j for which there is a $y_i \ \epsilon\ Cl_i$.*

Condition (1) states that the transaction managers translate each operation submitted by a transaction into appropriate operations on data copies. Condition (2) states that the intra-cluster schedule preserves the ordering stipulated by each transaction. Condition (3) states that the schedule records the execution order of conflicting operations. Condition (4) states that a transaction cannot read a copy unless it has been previously initialized. Condition (5) states that if a transaction writes a data item x before it reads x, then it must write to the same copy of x that it subsequently reads. Finally, condition (6) 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 weak transactions do not see partial results of a strict transaction.

Thus, in an intra-cluster schedule a weak read operation *reads-x_i-from* the transaction that has last weak or strict written x_i. A strict read operation *reads-x_i-from* the transaction that has last strict written x_i. A (weak 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 weak operations at a cluster to read only values written by weak transactions, we can define h such that no strict transaction writes at that cluster.

Given a schedule S, the *projection of S on strict transactions* is the schedule obtained from S by deleting all weak operations, and the *projection of S on a cluster Cl_k* is the schedule obtained from S by deleting all operations of S that do not access Cl_k.

4.2 Correctness criterion

A correct concurrent execution of weak and strict transactions must maintain m-consistency among clusters and strict consistency inside each cluster.

4.2.1 Maintaining bounded inconsistency

In our schema the degree for each data item at a cluster expresses the divergence of each local (weak) version from the value of the strict version. This difference may result either from globally uncommitted weak writes or from updates of strict versions that have not yet been reported at the cluster. As a consequence, the degree may be bounded either by limiting the number of weak writes pending commitment or by controlling the h function.

Specifically: (a) If the value of a data item is arithmetic and the degree is expressed as a range m of acceptable values that each copy of a data item can take, then only weak writes that are in the allowable bounds are accepted. (b) If the degree is expressed as the number of transactions that are allowed to read inconsistent data then we bound the number of weak transactions that are allowed at each cluster. (c) If the degree m is bounded by limiting the number of versions, that is if a weak transaction at a cluster Cl_i must read only m-version old data, we define h such that a strict write modifies the data at a cluster at least every m updates. (d) If the degree m is defined as the number of data items that are allowed to diverge, we bound the number of weak versions that are allowed to diverge from the strict versions at all clusters to m. This is accomplished by allowing weak reads and writes only on m data items. (e) Finally, if the degree m is defined as the number of data copies that are allowed to diverge, we bound the number of weak versions that are allowed to diverge at each cluster so that the total number of weak versions that differ from strict versions in all clusters is m. This is achieved by bounding appropriately the number of weak writes at each cluster.

4.2.2 Maintaining intra-cluster consistency

If each transaction maintains m-consistency when executed alone, and if the projections of the schedule at each cluster are conflict-serializable [5], then serializability of each of the projections suffices to ensure that each weak transaction gets a consistent view, that is the integrity constraints among the data it reads hold. This is true because if a strict transaction maintains m-consistency, then its projection on any 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 serializable (1SR), that is (view) equivalent to an 1C (one-copy) schedule [5].

Definition 4 (IAS Weak Correctness) *An intra-cluster schedule S is correct if its projection on strict transactions is 1SR and each of its projections on a cluster is conflict-equivalent to a serial schedule.*

Weak correctness guarantees that both weak and strict transactions get a consistent view of the database, i.e., one that is equivalent to the result of a serial execution of a number of consistent transactions. For strict transactions this is guaranteed from the one-copy serializability of the schedule of strict transactions, since strict transactions do not read values written by weak transactions. For weak transactions in each cluster this is guaranteed from the serializability of the projection of the schedule on this cluster.

Note, that inter-cluster constraints other than replication constraints among weak versions of data items at different sites may be violated. Weak transactions however are unaffected of such violations, since they read only local data. Although, the above correctness criterion suffices to ensure that each weak transaction gets a consistent view, however it does not suffice to ensure that weak transactions at different clusters get the same view, even in the absence of inter-cluster constraints. The following example is illustrative.

Example 2 *Assume two clusters* $Cl_1 = \{x_1, y_1\}$ *and* $Cl_2 = \{w_2, z_2, l_2\}$ *and the following two strict transactions (after the application of h)* $ST_1 = S\_Write_1(x_1)\ S\_Write_1(w_2) C_1$ *and* $ST_2 = S\_Write_2(y_1) S\_Write_2(z_2) S\_Read_2(x_1) C_2$. *In addition, at cluster* Cl_1 *we have the weak transaction* $WT_3 = W\_Read_3(x_1)\ W\_Read_3(y_1)\ C_3[1]$, *and at cluster* Cl_2 *the weak transactions* $WT_4 = W\_Read_4(z_2)\ W\_Write_4(l_2)\ C_4[2]$, *and* $WT_5 = W\_Read_5(w_2)\ W\_Read_5(l_2)\ C_5[2]$. *(For simplicity, we do not show the transaction that initially writes all data copies.)*

Their execution results in the weakly correct schedule
$S = W\_Read_5(w_2) S\_Write_1(x_1) W\_Read_3(x_1)$
$S\_Write_1(w_2) C_1 S\_Write_2(y_1) S\_Write_2(z_2)$
$S\_Read_2(x_1) C_2 W\_Read_3(y_1) C_3[1] W\_Read_4(z_2)$
$W\_Write_4(l_2) C_4[2] W\_Read_5(l_2) C_5[2]$

The projection of S on strict transactions is:
$S\_Write_1(x_1) S\_Write_1(w_2) C_1 S\_Write_2(y_1)$
$S\_Write_2(z_2) C_2$ *which is equivalent to the 1SR schedule:* $S\_Write_1(x) S\_Write_1(w)\ C_1\ S\_Write_2(y)$
$S\_Write_2(z)\ C_2$

The projection of S on Cl_1: $S\_Write_1(x_1)$
$W\_Read_3(x_1) C_1 S\_Write_2(y_1) S\_Read_2(x_1)$
$W\_Read_3(y_1) C_3[1]$ *is serializable as* $ST_1 \rightarrow ST_2 \rightarrow WT_3$

The projection on Cl_2: $W\_Read_5(w_2) S\_Write_1(w_2)$
$C_1 S\_Write_2(z_2) C_2 W\_Read_4(z_2) W\_Write_4(l_2) C_4[2]$
$W\_Read_5(l_2) C_5[2]$ *is serializable as* $ST_2 \rightarrow WT_4 \rightarrow WT_5 \rightarrow ST_1$ □

Thus, weak correctness does not guarantee that there is a serial schedule equivalent to the intra-cluster schedule as a whole, that is including both weak and strict transactions. The following is a stronger correctness criterion that ensures that weak transactions get the same consistent view. Obviously, strong correctness implies weak correctness.

Definition 5 (IAS Strong Correctness) *An intra-cluster schedule S is correct if its projection on strict transactions is equivalent to an 1C schedule* S_{1C} *and S is conflict-equivalent to a serial schedule* S_S *such that the order of transactions in* S_S *is consistent with the order of transactions in* S_{1C}.

Since weak transactions do not directly conflict with weak transactions at other clusters, the following is an equivalent statement of the above definition,

Definition 6 IAS Strong Correctness (alternative definition) *An intra-cluster schedule S is correct if its projection on strict transactions is equivalent to an 1C schedule* S_{1C}, *and each of its projections 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}.

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. Therefore, no control messages are needed between transaction managers at different clusters for synchronizing weak transactions.

The proposed schema is very flexible. Any coherency control method that guarantees one-copy serializability (e.g., quorum consensus, primary copy) can be used for synchronizing strict versions. The schema reduces to one-copy serializability when only strict transactions are used.

4.3 Graph characterization

To determine whether an *IAS* schedule is correct we use a modified serialization graph, that we call the *intra-cluster serialization graph* (IASG) of the *IAS* schedule. To represent conflicts between strict transactions, we construct a replicated data Serialization Graph (SG). An SG [5] 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 one-copy serializability of the corresponding schedule. Then, we augment *SG* with additional edges to represent conflicts between weak transactions in the same cluster and conflicts between weak and strict transactions. In general, there are three types of edges in the resulting *IASG* [7]:

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.

Property 1 below characterizes the type of edges that exist between weak transactions, and Property 2 the type of edges between weak and strict transactions.

Property 1 *Between weak transactions in the same cluster, there may exist dependency and precedence edges. Between weak transactions at different clusters, no edge exist.*

Proof: Straightforward since weak transactions at different clusters read different versions of a data item. □

Property 2 *Let WT_i represent a weak transaction at cluster Cl_i and ST a strict transaction, then the IASG graph induced by an IAS may include* **only** *the following edges between them:*

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

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

Now we prove,

Theorem 1 *Let S_{IAS} be an intra-cluster schedule. If S_{IAS} has an acyclic IASG then S 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 one-copy serializability of the strict transactions since strict transactions read only values written by strict transactions. Let $T_1, T_2, \ldots, T_n$ be all transactions in S_{IAS}. Thus $T_1, T_2, \ldots, T_n$ 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 [5] S_{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, 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}. □

5 Restoring Consistency upon Cluster Merging

When clusters are merged we must enforce full consistency, that is reconcile values of different copies of the same data item located at different clusters.

5.1 Schedules

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

Definition 7 (inter-cluster schedule) *An inter-cluster schedule (IES) S_{IES} based on an intra-cluster schedule $S_{IAS} = (OP, <)$ is a pair $(OP', <')$ where*

1. *$OP' = OP$,*
2. *for any op_i and $op_j \in OP'$, if $op_i < op_j$ in S_{IAS} then $op_i <' op_j$ in S_{IES}, and*
3. *in addition, for each $W\_Write_i[x_k]$ and $S\_Read_j[x_k]$ either $W\_Write_i[x_k] <' S\_Read_j[x_k]$ or $S\_Read_j[x_k] <' W\_Write_i[x_k]$.*

5.2 Correctness criterion

There are different approaches to the problem of reconciliation varying from purely syntactic to purely semantic [7]. In this paper, we adopt a purely syntactic application-independent approach. We accept as many weak writes as possible without violating the one-copy serializability of strict transactions.

Definition 8 (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 transactions whose weak writes conflict with strict transactions. Undoing a transaction normally results in *cascading aborts*, that is, in aborting transactions which have read the values written by that transaction. In our case, since weak transactions write only weak versions in a cluster, and since only weak transactions in the same cluster can read these weak versions we get the following lemma:

Lemma 1 *Only weak transactions in the same cluster read values written by weak transactions in that cluster.*

The above lemma ensures that only weak transactions in the same cluster may need to be aborted when a weak transaction is aborted to resolve conflicts in an inter-cluster schedule. However, for most applications, aborting a weak transaction does not change the semantics of other committed weak transactions that have read the values produced by it. This is because weak transactions are only interested on approximate values of data items, thus even if the value they read was produced by a transaction that was later aborted, this value was in an acceptable range and this fact suffices to guarantee their correctness.

5.3 Graph characterization

To determine correct *IES* schedules we define a modified serialization graph that we call the *inter-cluster serialization graph* (IESG). To construct the IESG, we first augment the serialization graph IASG of the underlying intra-cluster schedule to take into consideration conflicts between W_Write and S_Read. Specifically:

Property 3 *Let WT_i be a weak 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 IASG the following edges:*

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

Proof: If $W\_Write[x_i] > S\_Read[x_i]$ then we add a dependency edges from WT_i to ST. If $S\_Read[x_i] > W\_Write[x_i]$ then we add a precedence edge from ST and WT_i. Note, that interference edges between transactions at different clusters are not reported. □

In the original IASG graph, transactions that access different versions 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 inducing:

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

Theorem 2 *Let S_{IES} be an IES schedule based on an IAS schedule S_{IAS}. If S_{IES} has an acyclic IESG then S_{IES} is correct.*

Proof: Clearly, if the IESG graph is acyclic, the corresponding graph for the IAS is acyclic (since to get the IESG we only add edges to the IASG). 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 a weak transaction WT. Then WT writes x in S_{IES} and since ST_i also writes x either (a) $ST_i \rightarrow WT$ or (b) $WT \rightarrow ST_i$. In case (a), from the definition of the *IESG*, we get $ST_j \rightarrow WT$, which is a contradiction since ST_j reads-x-from WT. In case (b) $WT \rightarrow ST_i$, that is WT precedes ST_i which precedes ST_j, which again contradicts the assumption that ST_j reads-x-from WT. □

6 Relation to other Criteria

Strict consistency requires that all copies of a data item are synchronized. One-copy serializability [5] hides from the user the fact that there can be multiple copies of a data item and ensures strict consistency. Whereas one-copy serializability may be an acceptable criterion for strict transactions, it is too restrictive for applications that could tolerate m-consistent locally available copies.

Network Partitioning. The partitioning of a database into clusters resembles the *network partition problem* [7], where site or link failures fragment a 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 1SR is ensured (in a pessimistic manner). Weak transactions may be performed locally (in an optimistic manner). To merge updates performed by weak transactions we adopt a purely syntactic approach.

Read-only 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 a special case of weak transactions.

In [9] two requirements for read-only 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 the transaction 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 transactions always have strong consistency requirements, weak read-only transactions can be tailored to have any of the above degrees based on the criterion used for IAS correctness. Weak 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 specify 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). Weak read-only transactions may read older versions of data, depending on the definition of the m-degree.

Epsilon-serializability (ESR) [19] allows temporary and bounded inconsistencies in copies 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 weak read-only transactions with no consistency requirements. ESR bounds inconsistency directly by bounding the number of updates. In [23] 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.

Mobile Database Systems. The effect of mobility on replication schemas is discussed in [4]. The need for the management of cached copies to be tuned according to the available bandwidth and the currency requirements of the applications is stressed. In this respect, m-degree consistency and weak transactions realize both of the above requirements. The restrictive nature of one-copy serializability for mobile applications is also pointed out in [16] and a more relaxed criterion is proposed. This criterion although sufficient for a specific kind of data typical of sales applications is not appropriate for general application and distinguishable data. Furthermore, the criterion does not support any form of adaptability to the current network conditions.

Mobile File Systems. Coda [15] 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 [22], where a weak read operation was added to a file service interface. The semantics of operations are different in that no weak write is provided and since there is no transaction support, the correctness criterion is not based on one-copy serializability.

7 Other Applications of Clustering

Clustering is also appropriate for *very 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 are a major concern in environments where communication is achieved through wide area networks [24]. 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 is as well appropriate for databases that scale in the number of sites (as oppose to 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.

The idea of degrees of consistency and weak operation may also prove useful in *multidatabase systems*, which are confederations of autonomous pre-existing database systems. In this environment, transaction management is performed at two levels: at a local level by the pre-existing transaction managers of the local databases (LTMs), and at a global level by the global transaction manager (GTM) [6]. Local transaction managers are responsible for the correct execution of transactions executed at their local sites. The global transaction manager retains no control over global transactions after their submission to the LTMs and can make no assumptions about their execution. Local sites may be viewed as clusters, local transactions as weak transactions, and global transactions as strict transactions. Replication constraints express the fact that data items representing the same real-word entity may exist in more than one local database. We can consider two versions of data, weak versions that correspond to pre-existing local data and may be independently updated and strict versions that correspond to global data that are created during integration. Weak correctness of intra-cluster schedules respects the autonomy of local sites, since serializability of the projections at each cluster is guaranteed by the LTMs and one-copy serializability of strict transactions can be ensured by the GTM. Global transactions can read both weak and strict versions and thus can ensure bounded inconsistencies between them. To reconcile weak and strict versions, *polytransactions* [21] can be used. This schema can be augmented to support local transactions with strict semantics using protocols along the lines of [14].

8 Conclusions and Future Work

In this paper, we have studied consistency issues for distributed databases in mobile environments. Our main contributions 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 and to accommodate the changing locality, the cluster configuration is dynamic. While all data inside a cluster are mutually consistent, degrees of inconsistency are allowed among data at different clusters. Degrees may depend on the availability of bandwidth. Second, we have defined two kinds of operations (weak and strict) to allow applications to specify the type of consistency that is appropriate for their correct execution. We have shown how weak transactions can be part of a concurrency controller and developed criteria and graph-based tests for the correctness of schedules that employ them. By allowing applications to specify their consistency requirements, better bandwidth utilization is achieved. Finally, the proposed schema models operations on imprecise location data.

In this paper, we have focussed on a special type of dependencies between clusters, namely data replication. In future studies, we plan to investigate how the semantics of weak operations can be generalized to operate on different types of dependencies, such as vertical and horizontal partitions or constraint dependencies [20].

References

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

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

[3] B. R. Badrinath and T. Imielinski. Replication and Mobility. In *Proceedings of the 2nd IEEE Workshop on Management of Replicated Data*, 1992.

[4] D. Barbará and H. GarciaMolina. Replicated Data Management in Mobile Environments: Anything New Under the Sun? In *Proceedings of the IFIP Conference on Applications in Parallel and Distributed Computing*, April 1994.

[5] P. A. Bernstein, V. Hadjilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addisson-Wesley, 1987.

[6] Y. Breitbart, H. Garcia-Molina, and A. Silberschatz. Overview of Multidatabase Transaction Management. *VLDB Journal*, 1(2):181–239, 1992.

[7] S. B. Davidson, H. Garcia-Molina, and D. Skeen. Consistency in Partitioned Networks. *ACM Computing Surveys*, 17(3):341–370, September 1985.

[8] G. H. Forman and J. Zahorjan. The Challenges of Mobile Computing. *IEEE Computer*, 27(6), April 1994.

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

[10] Y. Huang, P. Sistla, and O. Wolfson. Data Replication for Mobile Computers. In *Proceedings of the 1994 SIGMOD Conference*, pages 13–24, May 1994.

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

[12] T. Imielinksi and B. R. Badrinath. Wireless Mobile Computing: Challenges in Data Management. *Communications of the ACM*, 37(10), October 1994.

[13] T. Imielinski and B. R. Badrinath. Querying in Highly Mobile Distributed Environments. In *Proceedings of 18th VLDB*, pages 41–52, August 1992.

[14] J. Jing, W. Du, A. Elmagarmid, and O. Bukhres. Maintaining Consistency of Replicated Data in Multidatabase Systems. In *Proceedings of the 14th International Conference on Distributed Computing Systems*, Poznan, Polland, June 1994.

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

[16] N. Krishnakumar and R. Jain. Protocols for Maintaining Inventory Databases and User Profiles in Mobile Sales Applications. In *Proceedings of the Mobidata Workshop*, October 1994.

[17] E. Pitoura and B. Bhargava. Revising Transaction Concepts for Mobile Environments. In *Proceedings of the 1st IEEE Workshop on Mobile Computing Systems and Applications*, pages 164-168, 1994.

[18] E. Pitoura and B. Bhargava. Building Information Systems for Mobile Environments. In *Proceedings of the 3rd International Conference on Information and Knowledge Management*, pages 371–378, 1994.

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

[20] A. Sheth and M. Rusinkiewicz. Management of Interdependent Data: Specifying Dependency and Consistency Requirements. In *Proceedings of the Workshop on the Management of Replicated Data*, 1990.

[21] A. P. Sheth, M. Rusinkiewics, and G. Karabatis. Using Polytransactions to Manage Interdependent Data. In Ahmed K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, pages 555–576. Morgan Kaufmann, 1992.

[22] C. D. Tait and D. Duchamp. Service Interface and Replica Management Algorithm for Mobile File System Clients. In *Proceedings of the 1st International Conference on Parallel and Distributed Information Systems*, pages 190–197, 1991.

[23] 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.

[24] Y. Zhang and B. Bhargava. Wance: A Wide Area Network Communication Emulation System. In *Proceedings of IEEE Workshop on Advances in Parallel and Distributed Systems*, pages 40–45, 1993.