Replica Consistency in Intermittently Connected Distributed Systems

Evaggelia Pitoura
Department of Computer Science
University of Ioannina
GR 45110 Ioannina, Greece
email: pitoura@cs.uoi.gr

Bharat Bhargava
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
email: bb@cs.purdue.edu

Ouri Wolfson
EECS Department
University of Illinois
Chicago, IL 60607
email: wolfson@eecs.uic.edu

Abstract

Mobile computing introduces a new form of distributed computation in which communication is achieved by a wide range of networks that most often are intermittent, low-bandwidth, or expensive thus providing only weak connectivity. In this paper, we present a replication schema tailored for such environments, that seeks to adapt consistency guarantees to networking conditions. Bounded inconsistency is defined by allowing controlled deviation among copies located at weakly connected sites. The database interface is extended with weak operations that allow access to local, potentially inconsistent copies and make conditional updates. The usual operations, called strict in this framework, are also supported and offer access to consistent data and permanent updates. The proposed model provides for disconnected operation, since mobile clients can operate even when disconnected by using weak operations. A balanced use of weak and strict transactions can efficiently reduce network usage and latency. Adjusting the degree of divergence among copies provides additional support for adaptability. We present correctness criteria for the schema, prove corresponding serializability-based theorems, and outline protocols for its implementation. Then, some practical examples of its applicability are provided. The performance of the schema is evaluated for various networking conditions and varying percentages of weak transactions by using an analytical model developed for this purpose.

1 Introduction

Advances in telecommunications and in the development of portable computers have provided for wireless communications that permit users to actively participate in distributed computing even while relocating from one support environment to another. The resulting distributed environment is subject to restrictions imposed by the nature of the connectivity conditions [9, 13, 25].

Mobile clients encounter wide variations in connectivity ranging from high-bandwidth, low latency communications through wired networks to total lack of connectivity. Most often, between these two extremes, connectivity is provided by wireless networks characterized by low bandwidth, high latency or high cost. To overcome availability and latency barriers, and reduce cost and power consumption mobile clients deliberately avoid use of the network and operate switching between connected and disconnected modes of operation. Thus, *disconnected operation*, that is the ability to operate disconnected, is essential for mobile clients [13, 15, 28]. Disconnected operation can also be used to extend battery life, and, for short durations, to handle hand offs. Besides disconnected operation, operation exploiting *weak connectivity*, that is connectivity provided by intermittent, low-bandwidth, or expensive networks, is also desirable [21, 11].

Mobile users will access private or corporate databases stored at mobile as well as static hosts and queried and updated over wired and wireless networks. For instance, insurance agents may interact through their mobile station with a database storing consumer records, while traveling salespersons may access inventory databases. Requirements for database use through wireless communications stem also from personal computing applications. Accessing databases is necessary for instance for consumers to purchase goods, get traffic information, or make travel plans while using their notebooks. These databases, for reasons of reliability, performance, and cost will be distributed and replicated over many sites. In this paper, we propose a replication schema that supports weak connectivity and disconnected operation. Schemas that balance network availability against consistency guarantees have been proposed in the context of file systems [11, 15, 6]. In this paper, we concentrate on providing transactional semantics.

In the proposed schema, data located at strongly connected sites are grouped together to form clusters. Mutual consistency is required for copies located at the same cluster while degrees of inconsistency can be tolerated for copies at different cluster. The proposed mechanism enchances the interface offered by database systems with operations with weaker consistency guarantees. Such weak operations allow access to local, i,e., in a cluster, bounded inconsistent copies and make conditional updates. The usual operations, called strict, here, are also supported and offer access to consistent data and permanent updates. The schema supports disconnected operation since users can operate even when disconnected by using only weak operations. In cases of weak connectivity, a balanced use of both weak and strict operations can provide for better bandwidth utilization, latency and cost. In cases of strong connectivity, using only strict operations makes the schema reduce to the usual one-copy semantics. Additional support for adaptability is offered be tuning the degree of inconsistency among copies based on the networking conditions.

We provide correctness criteria, prove corresponding serializability-based theorems and outline protocols for the implementation of the schema as well as for the reconciliation of copies. The presented implementation is based on distinguishing copies into core and quasi. Example use of the schema are outlined. Then, an analytical model is developed to evaluate the performance of the schema. The model is used to study the effect of weakening consistency on the system performance. The performance parameters considered are the system throughput, the number of messages, and the response time. The study is performed for a range of networking conditions, that is for different values of bandwidth and for varying disconnection intervals. We provide also an estimation of the reconciliation cost.

The remainder of this paper is organized as follows. In Section 2, the model is introduced along with an outline of its implementation. In Section 3 and 4, we define correctness criteria, prove serializability-based theorems, and present corresponding protocols for respectively maintaining weak consistency and for reconciliation. In Section 5, we present an analytical model for the schema and use it to estimate its performance. In Section 6, we evaluate the cost of reconciliation. Various issues regarding the schema along with examples of its use are presented in Section 7. In Section 8, we compare our work with related research and conclude in Section 9 by summarizing.

2 The Consistency Model

We consider a distributed database with data located both at mobile and stationary hosts to allow autonomous operation during disconnections and for performance. Transactions are distributed and can be initiated both at mobile and stationary hosts.

2.1 Data Correctness

As usually, a database *state* is defined as a mapping of every data to a value of its domain. Data are related by a number of restrictions called *integrity constraints* that express relationships of values of data that a database state must satisfy. A state is consistent if the integrity constraints are satisfied [3]. In contrast to traditional distributed databases where sites are normally connected, in mobile environments sites are only intermittently connected. Thus, instead of requiring maintenance of all integrity constraints we define units of consistency, called *clusters*, by partitioning the items of a database into clusters Cl_i based on their location, so that data in strongly connected sites belong to the same cluster. Then, we provide a weaker form of consistency by requiring maintenance of intracluster constraints and tolerating bounded inconsistency for intercluster constraints, as follows.

Definition 1 *A cluster state is consistent iff all intracluster integrity constraints hold. A database state is d-*consistent *iff all cluster states are consistent and all intercluster integrity constraints are d-degree consistent.*

The definition of *d-degree consistency* for an integrity constraint depends on its type. In this paper, we focus on replication constraints, where copies x_i of the same data item x are expected to have the same value. For replicated data, d-consistency means mutual consistency of all copies in the same cluster and bounded divergence among copies at different clusters. This divergence may be expressed in terms of the time lag or fuzziness of values of the copies in deviating from mutual consistency [29, 1]. The values of the data copies are occasionally reconciled to obtain a mutual consistent value.

The degree may vary based on the availability of network bandwidth by allowing little deviation in instances of higher bandwidth availability and high deviation in instances of low bandwidth availability. Thus, bounded inconsistency makes applications able to adjust to the limitations of the communication environment by providing users 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 in cases of weak network connectivity and up-to-date copies of all chapters in cases of strong network connectivity. The cluster configuration is dynamic. Clusters of data may be explicitly created or merged upon a forthcoming disconnection or connection of the associated mobile host. To accommodate migrating locality, a mobile host may move to a different cluster when it enters a new environment.

2.2 The Extended Database Operation Interface

To maximize local processing and reduce network access, we allow direct access to locally, e.g., in a cluster, available d-consistent data by introducing *weak reads* and *weak writes*. These operations allow clients to operate on d-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. In particular, a *weak read* operation on a data item x ($WR[x]$) reads a locally available value of x. A *weak write* operation ($WW[x]$) writes locally available copies and becomes permanent after reconciliation. A *strict read* operation ($SR[x]$) reads the value written by the last strict write operation. Finally, a *strict write* operation ($SW[x]$) writes one or more copies of x and is permanent upon the end of the issuing transaction.

Definition 2 (transaction) *A transaction (T) is a partial order (OP, $<$), where OP is the set of weak or strict read, weak or strict write, abort and 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 strict operations. A *strict* transaction (ST) is a transaction where OP does not include weak operations. Weak transactions access data copies that belong to the same cluster and thus are local at that cluster. User transactions can be decomposed into a number of weak and strict subtransactions units according to the degree of consistency required by the application.

There are two commit events associated with each weak transaction, a *local commit* in its associated cluster and an implicit *global commit* at reconciliation. The local commit point is expressed by an explicit commit operation, C. Updates made by locally committed weak transactions are visible only by weak transactions in the same cluster. These updates become permanent and visible by strict transactions only after reconciliation when local transactions become globally committed.

2.3 Implementation Issues

We divide copies into core and quasi. *Core* copies are copies that have up-to-date and permanent values, while *quasi* copies are copies that have potentially obsolete values that are only conditionally committed. All quasi copies at a cluster are mutually consistent and show bounded inconsistency with respect to core copies. Core copies are mutually consistent. Efficient distribution of core and quasi copies may be accomplished using appropriate algoriths for replica placement such as those proposed in [12].

To process the operations of a transaction, the database management system translates operations on data items into operations on the copies of these data items. In general, strict transactions, except of occasional updates of quasi copies, access only core copies and weak transactions operate on local, quasi or core, copies. We formalize this procedure by a *translation function h*.

Function h maps each $SR[x]$ operation into a number of read operations on core copies of x and returns one value (e.g., the most up to date) as the value actually read by the operation. Each $WW[x]$ operation is translated by h into a number of write operations of local quasi copies of x. Depending on the translation of a weak read operation we define two types of translation functions: a *best-effort* translation function that maps each $WR[x]$ operation into a number of read operations on locally available core or quasi copies of x and returns the most up-to-date such value, and a *conservative* translation function that maps each weak read into a number of read operations only on locally available quasi copies and returns the most-up-to-date such value. In addition, based on the time of propagation of updates of core copies to quasi, we define two types of translation functions: an *eventual* translation function that maps a $SW[x]$ into writes of only core copies and an *immediate* translation function that updates as well the quasi copies at the corresponding cluster. For an immediate h, conservative and best-effort have the same result. How many and which core or quasi copies are actually read or written when a database operation is issued on a data item depends on the coherency algorithm used, e.g, quorum consensus, ROWA. Without loss of generality, we assume that there is only one quasi copy per cluster. This assumption can be easily lifted but with significant complication in notation. Since all quasi copies in a cluster have the same value, this single copy can be considered to be their representative.

Immediate writes and consistency

In the case of integrity constraints other than replication constraints between data items, and for immediate translation functions, h should be defined such that integrity constraints between quasi copies in the same cluster are not violated. 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 $b^c = -1$, $b^q = -1$, $c^c = 2$ and $c^q = -4$, where superscripts c, and q stands for core and quasi copies respectively.*

Consider the transaction program:

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

If the above program is executed as a strict transaction $SW(b)\ SR(c)\ C$, we get the database state $b^c = 10$, $b^q = 10$, $c^c = 2$ and $c^q = -4$, where the integrity constraint between the quasi copies of b and c is violated. □

The problem arises from the fact that quasi copies are updated to the current value of the core copy without taking into consideration integrity constraints among quasi copies. Similar problems occur when refreshing individual copies of a cache [1]. Possible solutions include: (1) Each time a quasi copy is updated as a result of a strict write, the quasi copies 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 core and quasi copies (as in Section 4). In the above example, the core and quasi copies of b and c should have been reconciled prior to the execution of the transaction, producing for instance the database state $b^c = -1$, $b^q = -1$ $c^c = 2$ and $c^q = 2$. Then, the execution of the transaction would result in the database state $b^c = 10$, $b^q = 10$, $c^c = 2$ and $c^q = 2$, which is consistent. (2) Each transaction program that writes a data copy at a cluster is also translated to a weak transaction. The above example would result in transactions $SW(b)\ SR(c)C$ and $WW(b)WR(c)\ WW(c)C$ and a database state $b^c = 10$, $b^q = 10$, $c^c = 2$ and $c^q = 10$ which is consistent. (3) Updating quasi copies is postponed by deferring any updates of quasi copies that result from writes of the corresponding core 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 $b^c = 10$, $b^q = -1$, $c^c = 2$ and $c^q = -4$, which is consistent.

3 Weak Connectivity Operation

In this section, we provide serializability-based criteria, graph-based tests and a locking protocol for correct executions that exploits weak connectivity. We use the terms read and write to refer to the operations on data copies. The subscript j of an operation denotes that it belongs to transaction j and the subscript on a data copy identifies the cluster. A complete intracluster 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 (intracluster schedule) *Let $T = \{T_0, T_1, ..., T_n\}$ be a set of transactions. A (complete) intracluster schedule, IAS, over T is a pair (OP, $<_a$) in which $<_a$ is a partial ordering relation such that*

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 $op_k <_i op_l$, then every operation in $h(op_k)$ is related by $<_a$ to every operation in $h(op_l)$.*
3. *All pairs of conflicting operations are related by $<_a$, where two operations conflict if they access the same copy and one of them is a write operation.*
4. *For all read operations, $read_j[x_i]$ there is at least one $write_k[x_i]$ such that $write_k[x_i] <_a read_j[x_i]$.*
5. *If $SW_j[x] <_a SR_j[x]$ and $read_j(x_i) \in h(SR_j[x])$, then $write_j(x_i) \in SW_j[x]$.*
6. *If $write_j[x_i] \in h(SW_j[x])$ for some strict transaction T_j then $write_j[y_i] \in h(SW_j[y])$ for all y written by T_j for which there is a $y_i \in Cl_i$, where x_i is a quasi copy when h is conservative and any, quasi or core, copy when h is best effort.*

Condition 1 states that the transaction managers translate each operation on a data item into appropriate operations on data copies. Condition 2 states that the intracluster schedule preserves the ordering stipulated by each transaction and Condition 3 that it also 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 that may be possibly read by a weak 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.

A read operation on a data item x reads-x-from a transaction T_i if it reads a copy of x written by T_i and no other transaction writes this copy in between. 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. A schedule is one-copy serializable if it is view-equivalent to a serial one-copy schedule [3].

3.1 Correctness Criterion

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

Definition 4 (IAS Weak Correctness) *An intracluster schedule is weakly correct iff*

1. *all transactions have a consistent view, i.e., all constraints that can be evaluated using the data read are valid,*
2. *(a) strict transactions have the same reads-from relationship, and (b) the set of final writes on core copies is the same as in an one copy history,*
3. *it maintains the d-degree relationship among replicas.*

Next, we discuss how to enforce the first two conditions. Protocols for bounding the divergence among copies are outlined at the end of this section. The following theorem, defines correctness in terms of equivalence to serial executions.

Theorem 1 *Given that d-consistency is maintained, an intracluster schedule S is correct if its projection on strict transactions is one-copy serializable and each of its projections on a cluster is conflict-equivalent to a serial schedule.*

Proof: The first condition of the definition of correctness is guaranteed for strict transactions from the requirement of one-copy serializability, since strict transaction get the same view as in an one-copy serial schedule and read only core copies. For weak transactions at a cluster, the condition is provided from the requirement of serializability of the projection of the schedule on this cluster given that the projection of each transaction at the cluster maintains consistency when executed alone. Thus it suffices to prove that such projections maintain consistency. This trivially holds for weak transactions since they are local at each cluster. The condition also holds for strict transactions, since if a strict transaction maintains d-consistency, then its projection on any cluster also maintains d-consistency, as a consequence of condition (6) of the definition of an IAS schedule. Finally, one copy serializability of the projection of strict transactions suffices to guarantee 2(a) and 2(b) since strict transactions read only core copies and weak transactions do not write core copies respectively. □

Note, that intercluster constraints other than replication constraints among quasi copies 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 intercluster constraints. The following example is illustrative.

Example 2 *Assume two clusters $Cl_1 = \{x, y\}$ and $Cl_2 = \{w, z, l\}$ and the following two strict transactions $ST_1 = SW_1[x]\ SW_1[w]C_1$ and $ST_2 = SW_2\ [y]SW_2[z]SR_2[x]C_2$. In addition, at cluster Cl_1 we have the weak transaction $WT_3 = WR_3[x]\ WR_3[y]\ C_3$, and at cluster Cl_2 the weak transactions*

$WT_4 = WR_4[z]\ WW_4[l]\ C_4$, and $WT_5 = WR_5[w]\ WR_5[l]\ C_5$. For simplicity, we do not show the transaction that initializes all data copies. We consider an immediate and best effort h.

The execution of the above transactions produces the weakly correct schedule
$S = WR_5[w]SW_1[x]WR_3[x]SW_1[w]\ C_1SW_2[y]SW_2[z]SR_2[x]C_2WR_3[y]C_3WR_4[z]WW_4[l]C_4$
$WR_5[l]C_5$

The projection of S on strict transactions is: $SW_1[x]\ SW_1[w]\ C_1\ SW_2[y]\ SW_2[z]\ C_2$ which is equivalent to the 1SR schedule: $SW_1[x]\ SW_1[w]\ C_1\ SW_2[y]\ SW_2[z]\ C_2$

The projection of S on Cl_1: $SW_1[x]\ WR_3[x]\ C_1\ SW_2[y]\ SR_2[x]\ WR_3[y]\ C_3$ is serializable as $ST_1 \rightarrow ST_2 \rightarrow WT_3$

The projection on Cl_2: $WR_5[w]\ SW_1[w]\ C_1\ SW_2[z]\ C_2\ WR_4[z]\ WW_4[l]\ C_4\ WR_5[l]\ C_5$ 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 intracluster schedule as a whole, that is including all 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 intracluster schedule S over T is strongly correct iff there is a serial schedule S_S over T such that*

1. *S_S is conflict-equivalent with S, and*
2. *In S_S, (a) strict transactions have the same reads-from relationship, and (b) the set of final writes on core copies is the same as in an one copy history.*

Theorem 2 *An intracluster schedule S over T is correct if it is conflict-equivalent to a serial schedule S_S and its projection on strict transactions is equivalent to an one-copy serial schedule S_{1C} such that the order of transactions in S_S is consistent with the order of transactions in S_{1C}.*

Proof: We need to prove that in S_{1C} strict transactions have the same read-from and final writes as in S_S which is straightforward since strict transaction only read data produced by strict transactions and core copies are written only by strict transactions. □

Since weak transactions do not directly conflict with weak transactions at other clusters, the following is an equivalent statement of Theorem 2,

Theorem 3 *An intracluster schedule S is correct if its projection on strict transactions is equivalent to an one-copy serial 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 weak IAS correctness is used as the correctness criterion, then the transaction managers at each cluster must only synchronize projections on that cluster. Global control is required 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 core copies. The schema reduces to one-copy serializability when only strict transactions are used.

3.2 The Serialization Graph

To determine whether an IAS schedule is correct, a modified serialization graph is used, that we call the *intracluster serialization graph* (IASG) of the IAS schedule. To construct the $IASG$, a replicated data serialization graph (SG) is built to represent conflicts between strict transactions. An SG [3] 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, the SG is augmented with additional edges to represent conflicts between weak transactions in the same cluster and conflicts between weak and strict transactions. We add an edge $T_i \rightarrow T_j$ between two transactions T_i and T_j in IAS, if $op_i <_a op_j$. An edge is called a *dependency edge* if it represents the fact that a transaction reads a value produced by another transaction, and a *precedence edge* if it represents the fact that a transaction reads a value that was later changed by another transaction.

It is easy to see that in the IASG there are no edges between weak transactions at different clusters, since weak transactions at different clusters read different copies of a data item. In addition:

Property 1 *Let WT_i be a weak transaction at cluster Cl_i and ST a strict transaction. 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. □

Theorem 4 *Let S_{IAS} be an intracluster schedule. If S_{IAS} has an acyclic $IASG$ then S is strongly correct.*

Proof: 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, ... , T_n be all transactions in S_{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}, ... , T_{i_n} be a topological sort of the edges in IASG, then by a straightforward application of the serializability theorem [3] S_{IAS} is conflict equivalent to the serial schedule $S_S = T_{i_1}, T_{i_2}, \dots , T_{i_n}$. 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}. □

3.3 Protocols

Serializability. We distinguish between coherency and concurrency control protocols. Coherency control refers to ensuring that all copies of a data item have the same value, here we must maintain this property globally for core and locally for quasi copies. Concurrency control refers to maintaining consistency of the other integrity constraints, here the intracluster constraints. For coherency control, we assume a generic quorum-based schema [3]. Each strict transaction reads q_r core copies and writes q_w core copies per strict read and write operation. The values of q_r, and q_w for a data item x are such that $q_r + q_w > n_d$, where n_d is the number of available core copies of x. For concurrency

	WR	WW	SR	SW
WR	x		x	x
WW			x	x
SR	x	x	x	
SW	x	x		

(a)

	WR	WW	SR	SW
WR	x		x	
WW			x	x
SR	x	x	x	
SW		x		

(b)

	WR	WW	SR	SW
WR	x		x	x
WW			x	
SR	x	x	x	
SW	x			

(c)

	WR	WW	SR	SW
WR	x		x	
WW			x	
SR	x	x	x	
SW				

(d)

Figure 1: Lock compatibility matrices. A X entry indicates that the lock modes are compatible. (a) Eventual and conservative h. (b) Eventual and best effort. (c) Immediate and conservative. (d) Immediate and best effort h.

.

control we use *strict two phase locking* where each transaction releases its locks upon commitment [3]. Weak transactions release their locks upon local commitment and strict transactions upon global commitment. There are four lock modes (WR, WW, SR, SW) corresponding to the four data operations. Before the execution of each operation, the corresponding lock is requested. A lock is granted only if the data copy is not locked in an incompatible lock mode. Figure 1 depicts the compatibility of locks for various types of translation functions and is presented to demonstrate the interference between operations on items. The differences in compatibility stem from the fact the operations access different kinds of copies.

Bounded inconsistency among copies. The degree for each data item at a cluster expresses the divergence of the local quasi copy from the value of the core copy. This difference may result either from globally uncommitted weak writes or from updates of core copies 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. In Table 1, we outline ways of defining the bound on divergence along with methods of implementing them.

4 A Consistency Restoration Schema

For each data item, after the execution of a number of weak and strict transactions all its core copies have the same value while its quasi copies may have as many different values as the number of clusters. In this section, we provide first criteria for characterizing the correctness of protocols for reconciling the different values of copies and then describe such a protocol.

The exact point of reconciliation depends on the application requirements and the distributed system characteristics. Reconciliation may be forced to maintain the inconsistency bound. Alternatively, it may be initiated periodically or on demand upon the occurrence of specific events. Values may be reconciled when the network connection is reestablished, for instance when a palmtop is plugged-back to the stationary network or a mobile host enters a cell that provides good connectivity.

4.1 Correctness Criterion

Approaches to reconciling copies vary from purely syntactic to purely semantic ones [7]. We adopt a purely syntactic application-independent approach. Our correctness criterion is based on the following principle: if a core copy is written, and a strict transaction has read it, the value of the core copy must be the value selected. Otherwise, the value of any quasi copy may be chosen. Some weak transactions that wrote a value that was not selected may need to be undone/compensated or redone. This may lead to roll-back of other weak transactions that have read values written by this transaction. However, transaction roll-back is limited and never crosses the boundaries of a cluster.

A (complete) intercluster schedule, IES, models execution after reconciliation, where global transaction should become aware of local writes, i.e., local transaction become globally committed. In the schedule, we must add additional conflicts between weak and strict operations.

Definition 6 (intercluster schedule) *An intercluster schedule (IES) S_{IES} based on an intracluster schedule $S_{IAS} = (OP, <_a)$ is a pair $(OP', <_e)$ where*

1. *$OP' = OP$,*
2. *for any op_i and $op_j \epsilon OP'$, if $op_i <_a op_j$ in S_{IAS} then $op_i <_e op_j$ in S_{IES}, and in addition:*
3. *for each pair of weak write $WW_i[x]$ and strict read $SR_j[x]$ operations either $WW_i[x] <_e SR_j[x]$ or $SR_j[x] <_e WW_i[x]$*
4. *for each pair of weak write $WW_i[x]$ and strict write $SW_j[x]$ operations either $WW_i[x] <_e SW_j[x]$ or $SW_j[x] <_e WW_i[x]$.*

We redefine the reads-from relationship for strict transactions as follows. A strict read operation on a data item x reads-x-from a transaction T_i if it reads a copy of x and T_i has written this copy or a quasi copy of x and no other transaction wrote this or the quasi copy in between.

We accept as many weak writes as possible without violating the one-copy serializability of strict transactions. Specifically, a weak write is accepted only when it does not violate the read-from relationship for strict transactions.

Definition 7 (IES Correctness) *An intercluster schedule is correct iff*

1. *it is based on a correct IAS schedule S_{IAS}, and*
2. *strict transactions have the same reads-from relation as in the S_{IAS}.*

4.2 The Serialization Graph

To determine correct IES schedules we define a modified serialization graph that we call the *intercluster serialization graph* (IESG). To construct the IESG, we augment the serialization graph IASG of the underlying intracluster schedule. To force conflicts among weak and strict transactions that read different copies of the same data item, we induce

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

Theorem 5 *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 intercluster schedule S_{IES} is the same as in the underlying intracluster 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. □

4.3 Protocol

To get a correct schedule we need to break potential cycles in the IES graph. Since to construct the IES we start from an acyclic graph and add edges between a weak and a strict transaction, there is always at least one weak transaction in each cycle. We rollback such weak transactions. Undoing a transaction may result in *cascading aborts*, that is, in aborting transactions that have read the values written by the transaction; that is, transactions that are related with a dependency edge to the transaction undone. Since weak transactions write only quasi copies in a cluster, and since only weak transactions in the same cluster can read these quasi copies 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 intercluster schedule. In practice, fewer transactions ever need to be aborted. In particular, we need to abort only transactions whose output depends on the exact values of the data items they read. We call these transactions *exact*. Many weak transactions do not fall in this category, since by definition, weak transactions are transactions that read local *d*-consistent data. Thus even if the value they read was produced by a transaction that was later aborted, this value was inside an acceptable range of inconsistency and this is probably sufficient to guarantee their correctness.

Detecting cycles in the IEG can be hard. The difficulties raise from the fact that between transactions that wrote a data item an edge can have any direction, thus resulting in *polygraphs* [23]. Polynomial tests for acyclicity are possible, if we made the assumption that transactions read a data item before writing it. Then, to get the IES graph from the IAS we need only:

- induce a read order as follows: if a strict transaction ST reads an item that was written by a weak transaction WT we add a precedence edge $SR \rightarrow WT$

Table 2 outlines the reconciliation steps.

5 Discussion

5.1 Issues

Weak Operations. The proposed hybrid schema allows for the coexistence of two types of transactions, weak and strict. Weak transactions let users process local data thus avoiding the overhead of long

Definition of the degree of divergence (d)	Method
Up to d transactions can operate on inconsistent data.	The number of weak transactions at each cluster is bounded appropriately.
d is the maximum of divergent versions per item, that is a transaction reads at most up to d-version old data.	The function h is defined such as a strict write modifies quasi copies at a cluster at least after every d updates. This definition cannot be ensured for involuntary disconnected clusters since there is no way of notifying them for remote updates of core copies.
(for data items with arithmetic values) Each copy can take a value only inside a range d of acceptable values	Only weak writes whose values are in the allowable bounds are accepted.
Up to d data items are allowed to diverge.	The number of quasi copies that are allowed to diverge is bounded to d by allowing weak writes only on a set of d data items.
Up to d data copies per item are allowed to diverge.	The number of quasi copies that are allowed to diverge at each cluster is bounded so that the total number of quasi copies that differ from core copies in all clusters is d. This is achieved by bounding appropriately the number of weak writes at each cluster.

Table 1: Possible definitions for bounded inconsistency.

```
Until there are no cycles in the IESG
   rollback a weak transaction WT in the cycle
   unroll all exact transactions related with a dependency edge to WT
Per data item
   If the final write is on a core copy
      propagate this value to all quasi copies
   else
     choose a value of a quasi copy
    propagate this value to all core and quasi copies
```

Table 2: The reconciliation steps.

network accesses. Strict transactions need access to the network to guarantee consistency of their updates.

Weak reads provide users with the choice of reading an approximately accurate value of a datum in particular in cases of total or partial disconnections. This value is appropriate for a variety of applications that do not require exact values. Such applications include gathering information for statistical purposes or making high-level decisions and reasoning in expert systems that can tolerate bounded uncertainty in input data. Consider for instance the case of a customer who takes a look at the available credit before making a purchase, or that of a traveling salesperson who wants an estimation of the available stock before setting prices. In another context, getting the approximate location of a mobile user may suffice to determine what type of location-based services, e.g., traffic information, is applicable.

Weak writes allow users to update local data without confirming these updates immediately. Update validation is delayed till clusters are connected. Delayed updates can be performed during periods of low network activity to reduce demand on the peaks. Furthermore, grouping together weak updates and transmitting them 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 employ weak transactions to update mostly private data and strict transactions to update frequently used, heavily shared data.

Communication and Consistency. Disconnected operation is supported by using only weak operations. Users can choose to be logically disconnected even when physically connected. To accommodate weak connectivity, a mobile client can select an appropriate combination of weak and strict transactions based on the consistency requirements of its applications and on the prevailing networking conditions. Adjusting the degree of divergence provides an additional support for adaptability.

In a sense, weak operations offer some form of *application-aware adaptation* [22]. Application-aware adaptation characterizes the design space between two extremes. At one extreme, adaptivity is entirely the responsibility of the application, that is there is no system support or any standard way of providing adaptivity. At the other extreme, adaptivity is subsumed by the system, here the database management system. In general, the system is not aware of the application semantics and cannot provide a single adequate form of adaptation. Weak and strict operations lie in an intermediate point between these two extremes, serving as middleware between a database system and an application.

Clusters. Clusters were defined based on the *physical location* of data. Data located at the same, neighbor, or strongly connected sites are considered to belong to the same cluster, while data residing at remote or weakly connected sites are regarded as belonging to separate clusters. Other definitions of clusters are also feasible. Clusters may be defined based on the *semantics* of data or applications. Information about access patterns, for instance in the form of a *user's profile* that includes data describing the user's typical behavior, may be utilized in determining clusters. Under this definition, data that are most often accessed by a 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.

Weak Consistency. The idea of providing weak operations can be applied to other type of constraints besides replication. Such constraints can be vertical and horizontal partitions or arithmetic constraints

[29]. Another way of defining the semantics of weak operations is by exploiting the semantics of data. In [35], data are fragmented and later merged based on their object semantics.

5.2 Examples

Cooperative Environments. Consider the case of users working on a common project using mobile hosts. Groups are formed that consist of users who work on similar topics of the project. Clusters correspond to data used by people in the same group who need to maintain consistency among their interactions. We consider data that are most frequently accessed by a group as data *belonging* to this group. At each group, the copies of data items belonging to the group are core copies, while the copies of data items belonging to other groups are quasi. A data item may belong to more than one group if more than one group frequently accesses it. In this case, core copies of that data item exist in all such clusters. In each cluster, operations on items that do not belong to the group are weak, while operations on data that belong to the group are strict. Weak updates on a data item are accepted only when they do not conflict with updates by the owners of that data item.

Caching. Clustering can be used to model caching in a client/server architecture. In such a setting, a mobile host acts as a client interacting with a server at a fixed host. Data are cached at the client for performance and availability. The cached data are considered quasi copies. The data at the fixed host are core copies. Transactions initiated by the server are always strict. Transactions initiated by the client that invoke updates are always weak while read-only client transactions can be strict if strict consistency is required. At reconciliation, weak writes are accepted only if they do not conflict with strict transactions at the server. The frequency of reconciliation depends on the user consistency requirements and on networking conditions.

Location Data. In mobile computing, data representing the location of a mobile user are fast-changing. Such data are frequently accessed to locate a host. Thus, location data must be replicated at many sites to reduce the overhead of searching. Most of the location copies should be considered quasi. Only a few core copies are always updated to reflect changes in location.

5.3 Applications Beyond Weak Connectivity

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 [38]. 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.

Multidatabase Systems. Multidatabase systems 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) [5]. 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 types of copies per data, quasi copies that correspond to pre-existing local data and may be independently updated and core copies 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 core and quasi copies and thus can ensure bounded inconsistencies between them. To reconcile quasi and core copies, *polytransactions* [30] can be used. This schema can be augmented to support local transactions with strict semantics using protocols along the lines of [14].

6 Quantitative Evaluation of Weak Consistency

We have developed an analytical model to quantify the improvement in performance attained by sacrificing strict consistency in weakly connected environments and the interplay among the various parameters. The analysis follows an iteration-based methodology for coupling standard hardware resource and data contention as in [37]. Data contention is the result of concurrency and coherency control. Resources include the network and the processing units. We generalize previous results to take into account (a) nonuniform access of data, that takes into consideration hotspots and the changing locality, (b) weak and strict transaction types, and (c) various forms of data access, as indicated by the compatibility matrix of Table 1. An innovative feature of the analysis is the employment of a vacation system to model disconnections of the wireless medium.

The performance parameters under consideration are the system throughput, the number of messages sent, and the response time of weak and strict transactions. The study is performed for a range of networking conditions, that is for different values of bandwidth and for varying disconnection intervals.

6.1 Performance Model

We assume a cluster configuration with n clusters and a Poisson arrival rate for both queries and updates. Let λ_q and λ_u respectively be the average arrival rate of queries and updates on data items initiated at each cluster. We assume fixed length transactions with N operations on data items, $N_q = [\lambda_q/(\lambda_q + \lambda_u)]N$ of which are queries and $N_u = [\lambda_u/(\lambda_q + \lambda_u)]N$ are updates. Thus the transaction rate, i.e., the rate of transactions initiated at each cluster, is $\lambda_N = \lambda_u/N_u$.

Let c be the *consistency factor* of the application under consideration, that is c is the fraction of the arrived operations that are strict. To model hotspots, we divide data at each cluster into hot and cold data sets. To capture *locality* we assume that a fraction o of the transactions exhibit locality, that is they access data from the hot set with probability h and data from the cold set with probability $1-h$. The remaining transactions access hot and cold data uniformly. Due to mobility o may diminish with time. Locality is taken into consideration by the replication schema, by assuming that the probability that a hot data has a core copy at a cluster is l, and that a cold data has a core copy is l', where normally, $l' < l$. Let p_l be the probability that an operation at a cluster accesses a data item for which there is a core copy at the cluster, $p_l = o[hl + (1-h)l'] + (1-o)[(l'D_c)/D + (lD_h)/D]$.

For simplicity, we assume that there is one quasi copy of each data item at each cluster. Let q_r be the read and q_w the write quorum, then the mean number of operations on data copies per strict transaction, N_S, is equal to $N_u q_w + N_q q_r$. The transaction model consists of $n_L + 2$ states, where n_L is the random variable of items accessed by the transaction and N_L its mean. Without loss of generality, we assume that N_L is equal to the number of operations. The transaction has an initial setup phase, state 0. Then, it progress to states 1,2, …, n_L in that order. If successful, at the end of state n_L the transaction enters into the commit phase at state n_{L+1}. The transaction response time r_{trans} can be expressed as

$$r_{trans} = r_{INPL} + r_E + \sum_{j=1}^{n_w} r_{w_j} + t_{commit} \qquad \text{(A)}$$

where n_w is the number of lock waits during the run of the transaction, r_{w_j} is the waiting time for the jth lock contention, r_E is the sum of the execution times in states 1,2, …, n_L excluding lock waiting times, r_{INLP} is the execution time in state 0, and t_{commit} is the commit time to reflect the updates in the database.

Resource contention analysis

We model clusters as M/G/1 systems. The average service time for the various types of requests, all exponentially distributed, can be determined from the following parameters: t_q processing time for a query on a data copy, t_u time to install an update on a data copy, t_b overhead time to propagate an update or query to another cluster. In each M/G/1 server, all requests are processed with the same priority on a first-come, first-served basis. Clusters are connected and later reconnected. To capture disconnections, we model each connection among two clusters as an M/M/1 system with vacations. A vacation system is a system in which the server becomes unavailable for occasional intervals of time. If W is the available bandwidth between two clusters and if we assume exponentially distributed packet lengths for messages with average size m then the service rate s_r is equal to W/m. Let t_r be the network transmission time.

Number of Messages. The total number of messages transmitted per second is:

$$M = 2nc[\lambda_q(q_r - p_l) + \lambda_u(q_w - p_l)]$$

The first term corresponds to query traffic; the second to update traffic.

Execution Time. For simplicity, we ignore the communication overhead inside a cluster, assuming either that each cluster consists of a single node or that the communication among the nodes inside a cluster is relatively fast. Without taking into account data contention, the average response time for a weak read on a data item is $R_q^w = w + t_q$ and for a weak update $R_u^w = w + t_u$, where w is the average wait time at each cluster. Let b_r be 0 if $q_r = 1$ and 1 otherwise, and b_w be 0 if $q_w = 1$ and 1 otherwise. Then for a strict read on a data item $R_q^s = p_l[w + t_q + (q_r - 1)t_b + b_r(2t_r + t_q + w)] + (1 - p_l)(q_r t_b + 2t_r + t_q + w)$, and for a strict write $R_u^s = p_l[w + t_u + (q_w - 1)t_b + b_w(2t_r + t_u + w)] + (1 - p_l)(q_w t_b + 2t_r + t_u + w)$. The computation of w is given in the Appendix.

Average Transmission Time. The average transmission time t_r equals the service time plus the wait time w_r at each network link, $t_r = \frac{1}{s_r} + w_r$. The arrival rate λ_r at each link is Poisson with mean $\frac{M}{n(n-1)}$. The computation of w_r is given in the Appendix.

Throughput. The transaction throughput, i.e., input rate, is bounded by: (a) the processing time at each cluster, $\lambda \leq E[x]$, where $E[x]$ is the mean service time (b) the available bandwidth, $\lambda_r \leq t_r$, and (c) the disconnection intervals, $\lambda_r \leq E[v]$, where $E[v]$ is the mean duration of a disconnection.

Data contention analysis

We assume an eventual and best effort h. In the following, *op* stands for one of WR, WW, SR, SW. Using formula (A) the response time for strict and weak transaction is:

$$R_{strict-transaction} = R_{INPL} + R_{E_{strict}} + N_q P_q R_{SR} + N_u P_u R_{SW} + T_{commit}$$

$$R_{weak-transaction} = R_{INPL} + R_{E_{weak}} + N_q P_{WR} R_{WR} + N_u P_{WW} R_{WW} + T_{commit}$$

where P_{op} is the probability that a transaction contents for an *op* operation on a data copy, and R_{op} is the average time spent waiting to get an *op* lock given that lock contention occurs. P_q and P_u are respectively the probability that at least one operation on a data copy per strict read or write conflicts. Specifically, $P_q = 1 - (1 - P_{SR})^{q_r}$ and $P_u = 1 - (1 - P_{SW})^{q_w}$. An outline of the estimation of P_{op} and R_{op} is given in the Appendix. For a detailed description of the model see [24].

6.2 Performance Evaluation

Table 3 shows some realistic values for the input parameters. The bandwidth depends on the type of technology used, for infrared a typical value is 1 Mbps, for packet radio 2 Mbps, and for cellular phone 9-14 Kbps [9].

Parameter	Description	Value
n	number of clusters	5
λ_q	query arrival rate	12 queries/sec
λ_u	update arrival rate	3 updates/sec
c	consistency factor	ranges from 0 to 1
q_r	read quorum	ranges from 1 to n
q_w	write quorum	ranges from 1 to n
o	local transactions accessing hot data	ranges from 0 to 1
h	probability that a local transaction access hot data	ranges from 0 to 1
l	probability a hot data has a core copy at a given cluster	ranges from 0 to 1
l'	probability a cold data has a core copy at a given cluster	ranges from 0 to 1
t_u	processing time for an update	0.02 sec
t_q	processing time for a query	0.005 sec
t_b	propagation overhead	0.00007 sec
V	vacation interval	ranges
W	available bandwidth	ranges
m	average size of a message	16 bits
D_c	number of cold data items per cluster	800
D_h	number of cold data items per cluster	200
N	average number of operations per transaction	10

Table 3: Input parameters.

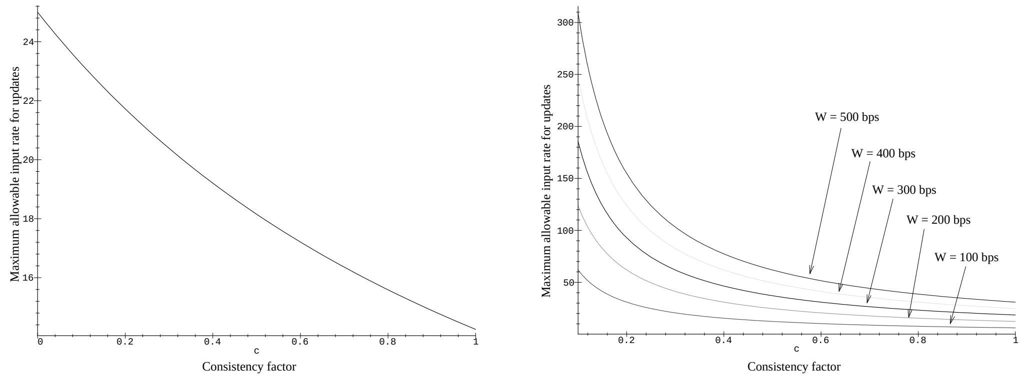

Figure 2: Maximum allowable input rate of updates for various values of the consistency factor. (left) Limits imposed by the processing rate at each cluster ($\lambda \leq E[x]$). (right) Limits imposed by bandwidth restrictions ($\lambda_r \leq t_r$).

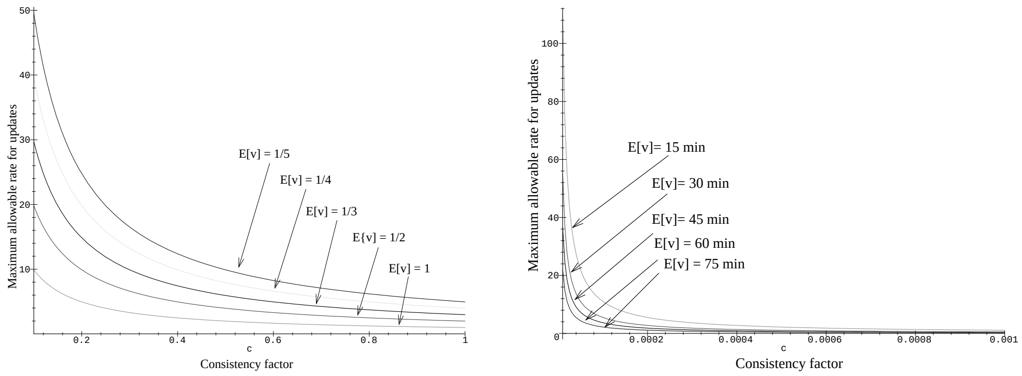

Figure 3: Maximum allowable input rate for updates for various values of the consistency factor. Limits imposed by disconnections and their duration ($\lambda_r \leq E[v]$).

System throughput

Figures 2(left), 2(right), and 3 show how the maximum transaction input, or system throughput, is bounded by the processing time, the available bandwidth, and the disconnection intervals respectively. We assume that queries are four times more common than updates $\lambda_q = 4\,\lambda_u$. As shown in Figure 2(left), the allowable input rate when all transactions are weak ($c = 0$) is almost double the rate when all transactions are strict ($c = 1$). This is the result of the increase in the workload with c caused by the fact that strict operations on data items may be translated into more than one operation on data copies. The percentage of weak transactions can be effectively tuned to attain the desired throughput based on the networking conditions such as the duration of disconnections and the available bandwidth. As indicated in Figure 2(right), to get for instance, the same throughput with 100bps as with 500bps and $c = 1$ we must lower the consistency factor below 0.1. The duration of disconnections may vary from seconds when they are caused by hand offs ([20]) to minutes for instance when they are voluntary. Figure 3 depicts the effect of the duration of a disconnection on the system throughput for both short durations (Figure 3(left)) and longer ones (Figure 3(right)). For long disconnections (Figure 3(right)), only a very small percentage of strict transactions can be processed. To keep the throughput comparable to that for shorter disconnections (Figure 3(left)) the consistency factor must drop at around three orders of magnitude.

Communication cost

We estimate the communication cost by the number of messages sent. The number of messages depends on the following parameters of the replication schema: (1) the consistency factor c, (2) the data distribution l for hot and l' for cold data, (3) the locality factor o and (4) the quorums, q_r and q_w, of the coherency schema. We assume a ROWA schema ($q_r = 1$, $q_w = n_d$) if not otherwise stated. As shown in Figure 4(left) the number of messages increases linearly with the consistency factor. As expected the number of messages decreases with the percentage of transactions that access hot data, since then local copies are more frequently available. To balance the increase in the communication cost caused by diminishing locality there may be a need to appropriately decrease the consistency factor (Figure 4(middle)). The number of messages decreases when the replication factor of hot core copies increases (Figure 4(right)). The decrease is more evident since most operations are queries and the coherency schema is ROWA, thus for most operations no messages are sent. The decrease is more rapid when transactions exhibit locality, that is when they access hot data more frequently. On the contrary, the number of messages increases with the replication factor of cold core copies because of additional writes caused by coherency control (Figure 5(left)). Finally, the relationship between the read quorum and the number of messages depends on the relative number of queries and updates (Figure 5(right)).

Transaction response time

The response time for weak and strict transactions for various values of c is depicted in Figure 6. The larger values of response times are for 200bps bandwidth, while faster response times are the result of higher network availability set at 2Mbps. The values for the other input parameters are as indicated in Table 3. The additional parameters are set as follows: (1) the locality parameters are $o = 0.9$ and $h = 0.9$, (2) the data replication parameters are $l' = 0.2$ and $l = 0.8$, (3) the disconnection parameters are $p = 0.1$ and the vacation intervals are exponentially distributed with $E[v] = 1/5$ sec, to model disconnection intervals that correspond to short involuntary disconnections

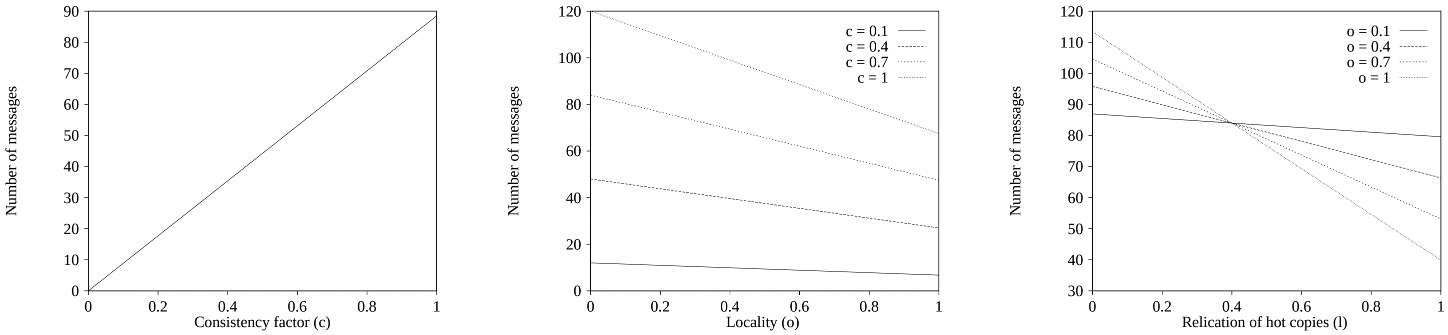

Figure 4: (left) Number of messages for various values of c. (middle) Number of messages with locality (right) Number of messages for different replication of hot core copies. Unless otherwise stated $o = 0.6$, $l = 0.9$, $l' = 0.4$, $h = 0.9$, and $c = 0.7$.

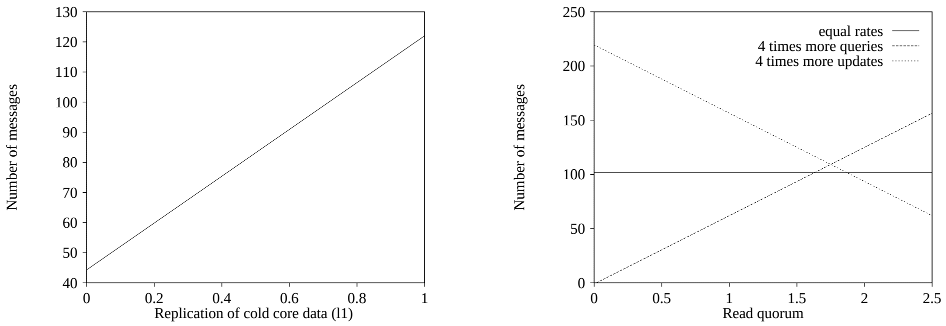

Figure 5: (left) Number of messages for different replication of cold core copies. (right) Number of messages for different values of the read quorum. Unless otherwise stated $o = 0.6$, $l = 0.9$, $l' = 0.4$, $h = 0.9$, and $c = 0.7$.

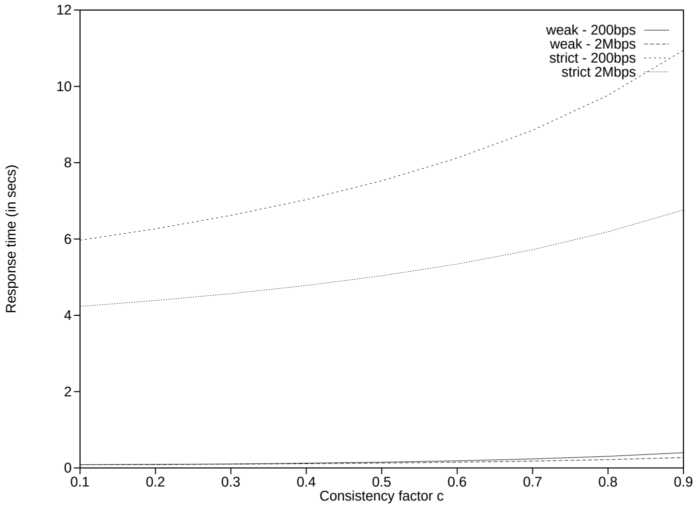

Figure 6: Comparison of the response times for weak and strict transactions for various values of the consistency factor.

such as those caused by hand offs [20], (4) the coherency control schema is ROWA. The latency of weak transactions is about 50 times greater than that of strict transactions. However, there is a trade-off involved in using weak transactions, since weak updates may be aborted later. The time to propagate updates during reconciliation is not counted. As c increases the response time for both weak and strict transactions increase since more conflicts occur. Figure 7(left) and (right) show the response time distribution for strict and weak transactions respectively for 2Mbps bandwidth. For strict transactions, the most important overhead is network transmission. All times increase as c increases. For weak transactions, the increase in the response time is the result of longer waits for acquiring locks, since weak transactions that want to read up-to-date data conflict with strict transactions that write them.

7 Reconciliation cost

We provide an estimation of the cost of restoring consistency in terms of the number of weak transactions that need to be rolled back. We focus on conflicts among strict and weak transactions for which we have outlined a reconciliation protocol and do not consider conflicts among weak transactions at different clusters. A similar analysis is applicable to this case also.

7.1 Estimation

A weak transaction is rolled back if its writes conflict with a read of a strict transaction that follows it in the IASG. Let P_1 be the probability that a weak transaction WT writes a data item read by a strict transaction ST and P_2 be the probability that ST follows WT in the serialization graph. Then the probability $P = P_1 P_2$ is the probability that a weak transaction is rolled back. Assume that reconciliation occurs after N_r transactions $k = c\,N_r$ of which are strict and $k' = (1-c)N_r$ are weak.

For simplicity we assume uniform access distribution. Although it is reasonable to assume that granule access requests from different transactions are independent, independence cannot hold within

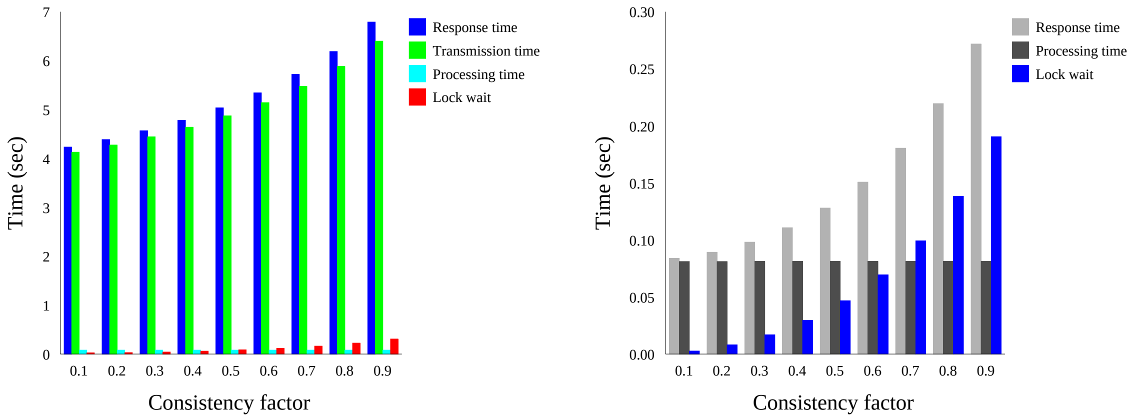

Figure 7: (left) Response time distribution for strict transactions. (right) Response time distribution for weak transactions.

a transaction if a transaction's granule accesses are distinct. However, if the probability of accessing any particular granule is small, e.g., when the number of granules is large and the access distribution is uniform, this approximation should be very accurate. Then $P_1 = 1 - (1 - \frac{N_u}{D})^{N_q}$.

Let p_{KL} be the probability that in the IASG there is an edge from a given transaction of type K to a given transaction of type L. Let $p'_{KL}(m, m')$ be the probability that in the IASG with m strict and m' weak transactions there is an edge from a given transaction of type K to any transaction of type L. The formulas are given in the Appendix.

Let $p(m, m', i)$ be the probability that there is an acyclic path of length i, i.e., a path with $i + 1$ distinct nodes, from a given weak transaction to a given strict transaction in a IASG with m strict and m' weak transactions. Then

$$P_2 = 1 - \prod_{i=1}^{k+k'-1}[1 - p(k, k', i)]$$

The values of $p(k, k', i)$ can be computed from the following recursive relations:

$$p(m, m', 1) = p_{WS}, \quad \text{for all } m > 0,\ m' > 0$$
$$p(m, m', 0) = 0, \text{ for all } m > 0,\ m' > 0,\ p(m, 0, i) = 0, \text{ for all } m \geq 0,\ i > 0, \text{ and } p(0, m', i) = 0 \text{ for all } m' \geq 0,\ i > 0$$
$$p(k, k', i+1) = 1 - [(1 - p'_{WW}(k, k')p(k, k'-1, i)) \\ (\prod_{j=1}^{i-1} 1 - (p'_{WS}(k, k') \prod_{l=1}^{i-j-1} p'_{SS}(k-l, k'-1)p'_{SW}(k-i+j, k'-1)p(k-i+j-1, k'-2, j)) \\ (1 - p'_{WS}(k, k') \prod_{l=1}^{i-1} p'_{SS}(k-l, k'-1)p_{SS})]$$

where the first term is the probability of a path with the first edge between weak transactions, the second of a path with the first edge between a weak and a strict transaction including at least one more weak transaction and the last of a path with the first edge between a weak and a strict transaction and no other weak transactions.

Thus the actual number of weak transaction that need to be undone or compensated because their writes cannot become permanent is $N_{abort} = P k'$. We also need to roll back all exact weak transactions that read a value written by a transaction aborted. Let e be the percentage of weak transactions that are exact, then $N_{roll} = e[1 - (1 - \frac{N_u}{D})^{N_q}]k' N_{abort}$.

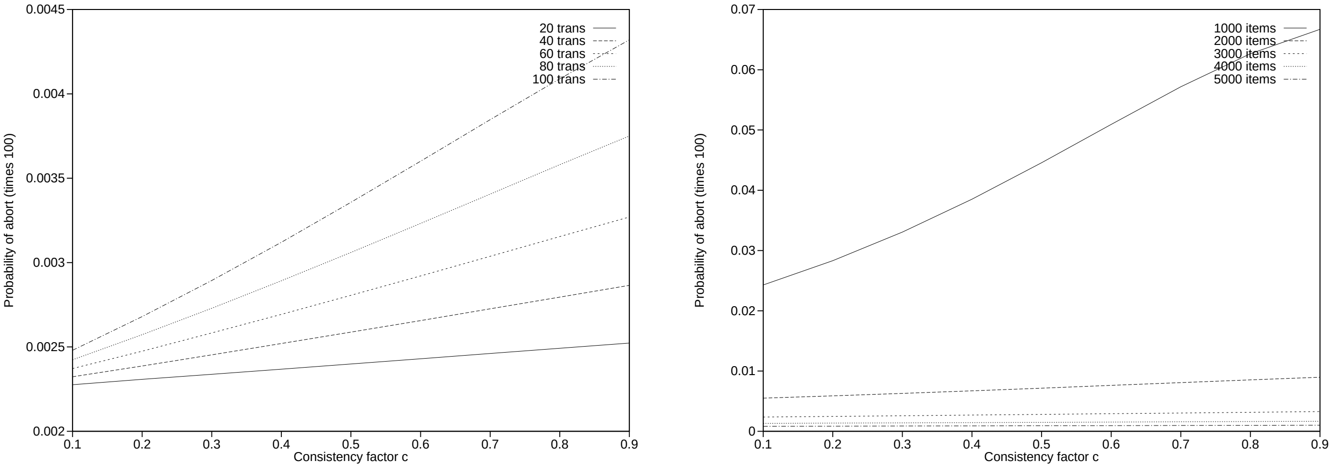

Figure 8: (left) Probability of abort, for 3000 data items. (right) Probability of abort, for 60 transactions.

7.2 Estimated Data

Figure 8 depicts the probability that a weak transaction cannot be accepted because of a conflict with a strict for reconciliation occurring after the execution of different numbers of transactions (left) and for various database sizes (right). More accurate estimations can be achieved for specific applications for which the access patterns of the transactions are known.

8 Related Work

Strict consistency requires that all copies of a data item are synchronized. One-copy serializability [3] 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 tolerate bounded inconsistency and causes unbearable overheads in cases of weak connectivity. An initial high-level description of a weak/strict database interface was presented first in [26].

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. Whereas all partitions are isolated, clusters may be weakly connected. Clients may operate as physically disconnected even while remaining physically 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 one-copy serializability 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 [10] 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 d-degree.

Epsilon-serializability (ESR) [27] 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 [36] 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.

In an *N-ignorant* system, a transaction need not see the results of at most N prior transactions that it would have seen if the execution had been serial [16]. Strict transactions are 0-ignorant and weak transactions are 0-ignorant of other weak transactions at the same cluster. Weak transactions are ignorant of strict and weak transactions at other clusters. The techniques of supporting N-ignorance can be incorporating in the proposed model to define d as the ignorance factor N of weak transactions.

Mobile database systems

The effect of mobility on replication schemas is discussed in [2]. 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, d-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 [17] and a more relaxed criterion is proposed. This criterion although sufficient for aggregate data is not appropriate for general applications and distinguishable data. Furthermore, the criterion does not support any form of adaptability to the current network conditions.

The *Bayou system* [8, 33] is a platform of replicated highly available, variable-consistency, mobile databases on which to build collaborative applications. A read-any/write-any weakly-consistent replication schema is employed. Each Bayou database has one distinguished server, the primary, which is responsible for committing writes. The other secondary servers tentatively accept writes and propagate them towards the primary. Each server maintains two views of the database: a copy that only reflects committed data and another full copy that also reflects tentative writes currently known to the server. Applications may choose between committed and tentative data. Tentative data are similar to our quasi data, and committed data similar to core data. Correctness is defined in terms of session, rather than on serializability as in the proposed model. A *session* is an abstraction for the sequence of read and writes of an application. Four types of guarantees can be requested per session: (a) read your writes, (b) monotonic reads (successive reads reflect a non-decreasing set of writes), (c) writes follow read (writes are propagated after reads on which they depend), and (d) monotonic writes (writes are propagated after writes that logically precede them). To reconcile copies, Bayou adopts an application based approach as opposed to the syntactic based procedure used here. The detection mechanism is based on dependency checks, and the per-write conflict resolution method is based on client-provided merge procedures [34].

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. [18, 19] extend Coda with a new transaction service called *isolation-only transactions* (IOT). IOTs are sequences of file accesses that unlike traditional transactions have only the isolation property. IOTs do not guarantee failure atomicity and only conditionally guarantee permanence. IOTs are similar to weak transactions.

Methods for refining consistency semantics of cached files to allow a mobile client to select a mode appropriate for the current networking conditions are discussed in [11], The proposed techniques are delayed writes, optimistic replication and failing instead of fetching data in cases of cache misses.

The idea of using different kinds of operations to access data is also adopted in [6, 31], 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.

9 Summary

To overcome bandwidth, cost, and latency barriers, clients of mobile information systems switch between connected and disconnected modes of operation. In this paper, we propose a replication schema appropriate for such operation. Data located at strongly connected sites are grouped in clusters. Bounded inconsistency is defined by requiring mutual consistency among copies located at the same cluster and controlled deviation among copies at different clusters. The database interface is extended with weak operations that allow access to local, potentially inconsistent copies and make conditional updates. The usual operations, called strict in this framework in contradistinction to weak, are also supported and offer access to consistent data and permanent updates.

The proposed model provides for disconnected operation, since mobile clients can operate even when disconnected using weak operations. Bandwidth can by utilized by deliberately using weak transactions. In addition, the degree of consistency can be appropriately tuned to achieve the desired system performance. Weak operations offer a form of *application-aware adaptation* [22]. They can be viewed as a tool offered by a database system to an application. The application, can at its discretion use weak or strict transaction based on its semantics. The implementation, consistency control, and the underlying support of the transactions is the job of the database system.

We have defined correctness criteria for the schema, proved corresponding serializability-based theorems and outline protocols for its implementation. The presented implementation is based on distinguishing copies into core and quasi. An analytical model was developed and used to make predictions of the performance of the schema under various networking conditions and for different percentages of weak and strict transactions.

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] D. Barbará and H. Garcia-Molina. 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.

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

[4] D. Bertsekas and R. Gallager. *Data Networks*. Prentice Hall, 1987.

[5] Yuri Breitbart, Hector Garcia-Molina, and Avi Silberschatz. Overview of Multidatabase Transaction Management. *VLDB Journal*, 1(2):181–239, 1992.

[6] D. Tait C and D. 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.

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

[8] A. Demers, K. Petersen, M. Spreitzer, D. Terry, M. Theimer, and B. Welch. The Bayou Architecture: Support for Data Sharing Among Mobile Users. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, pages 2–7, Santa Cruz, California, December 1994.

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

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

[11] P. Honeyman and L.b Huston. Communication and Consistency in Mobile File Systems. *IEEE Personal Communications*, December 1995.

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

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

[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, Poland, 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, February 1992.

[16] N. Krinshnakumar and A. J. Bernstein. Bounded Ingnorance: A Technique for Increasing Concurrency in a Replicated System. *ACM Transactions on Database Systems*, 19(4):586–625, December 1994.

[17] 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.

[18] Q. Lu and M. Satyanarayanan. Isolation-Only Transactions for Mobile Computing. *Operating Systems Review*, pages 81–87, April 1994.

[19] Q. Lu and M. Satyanarayanan. Improving Data Consistency in Mobile Computing Using Isolation-Only Transactions. In *Proceedings of the Fifth Workshop on Hot Topics in Operating Systems*, Orcas Island, Washington, May 1995.

[20] K. Miller. Cellular Essentials for Wireless Data Transmission. *Data Communications*, 23(5):61–67, March 1994.

[21] L. B. Mummert, M. R. Ebling, and M. Satyanarayanan. Exploiting Weak Connectivity for Mobile File Access. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, December 1995.

[22] B. D. Noble, M. Price, and M. Satyanarayanan. A Programming Interface for Application-Aware Adaptation in Mobile Computing. In *Proceedings of the Second USENIX Symposium on Mobile and Location-Independent Computing*, Ann Arbor, Michigan, April 1995.

[23] C. Papadimitriou. *The Theory of Database Concurrency Control*. Computer Science Press, 1986.

[24] E. Pitoura. *Transaction Management in Mobile Heterogeneous Environments*. PhD thesis, Department of Computer Science, Purdue University, 1995.

[25] E. Pitoura and B. Bhargava. Building Information Systems for Mobile Environments. In *Proceedings of the Third International Conference on Information and Knowledge Management*, pages 371–378, Washington, D.C., November 1994.

[26] E. Pitoura and B. Bhargava. Maintaining Consistency of Data in Mobile Distributed Environments. In *Proceedings of the 15th International Conference on Distributed Computing Systems*, pages 404–413, Vancouver, British Columbia, Canada, May 1995.

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

[28] M. Satyanarayanan, J. J. Kistler, L. B. Mummert, M. R. Ebling, P. Kumar, and Q. Lu. Experience with Disconnected Operation in a Mobile Computing Environment. In *Proceedings of the 1993 USENIX Symposium on Mobile and Location-Independent Computing*, Cambridge, Massachusetts, August 1993.

[29] 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*, pages 133–136, Houston, Texas, November 1990.

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

[31] C. D. Tait and D. Duchamp. An Efficient Variable-Consistency Replicated File Service. In *Proceedings of the USENIX File Systems Workshop*, pages 111–126, May 1992.

[32] H. Takagi. *Queueing Analysis, Vol.1: Vacation and Priority Systems.* North-Holland, 1991.

[33] D. Terry, A. Demers, K. Petersen, M. Spreitzer, M. Theimer, and B. Welch. Session Guarantees for Weakly Consistent Replicated Data. In *Proceedings of the International Conference on Parallel and Distributed Information Systems*, pages 140–149, September 1994.

[34] D. B. Terry, M. M Theimer, K. Petersen, A. J. Demers, M. J Spreitzer, and C. H. Hauser. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, December 1995.

[35] G. Walborn and P. K. Chrysanthis. Supporting Semantics-Based Transaction Processing in Mobile Database Applications. In *Proceedings of the 14th Symposium on Reliable Distributed Systems*, September 1995.

[36] 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.

[37] P. S Yu, D. M. Dias, and S. S Lavenberg. On the Analytical Modeling of Database Concurrency Control. *Journal of the ACM*, 40(4):831–872, September 1993.

[38] 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, Princeton, New Jersey, October 1993.

Appendix

Computation of resource waiting times.

Processor waiting time. At each cluster there are the following types of requests. Queries are initiated at a rate of λ_q. From the locally initiated queries, $\lambda_1 = (1-c)\lambda_q$ are weak and are serviced locally with an average service time $\theta_1 = t_q$. Then, from the remaining $c\lambda_q$ strict queries $\lambda_2 = xc\lambda_q$ have service time $\theta_2 = (q_r - 1)t_b + t_q$ and the rest $\lambda_3 = (1-x)c\lambda_q$ have service time $\theta_3 = q_r t_b$. Queries are also propagated from other clusters at a rate $\lambda_4 = [x(q_r - 1) + (1-x)q_r]c\lambda_q$ and have service time $\theta_4 = t_q$. Analogous formulas hold for the arrival rates and service times of updates. The combined flow of request forms a Poisson process with arrival rate, $\lambda = \sum_{i=1}^{8} \lambda_i$. The service time of the combined flow, x, is no longer exponentially distributed but its means and second moments are:

$$E[x] = \sum_{i=1}^{8}(\frac{\lambda_i}{\lambda})\theta_i \qquad E[x^2] = \sum_{i=1}^{8}(\frac{\lambda_i}{\lambda^2})2\theta_i^2$$

Then, the wait time by using the Pollaczek-Khinchin (P-K formula) [4] is: $w = \dfrac{\lambda E[x^2]}{2(1-\lambda E[x])}$
Note, that the above analysis as well as the following analysis on network links are worst cases. In practice, when a locking method is used for concurrency control, a number of transactions is waiting to acquire locks and not competing for system resources. Thus the rate of arrival of operations at the resource queues and the waiting time at each queue may be less than the value assumed in this section.

Transmission waiting time. We consider a nonexhaustive vacation system where after the end of each service the server takes a vacation with probability $1-p$ or continues service with probability p. This is called a queue system with Bernoulli scheduling [32]. In this case:

$$w_r = \frac{E[v^2]}{2E[v]} + \frac{\lambda_r\{s_r^{(2)} + (1-p)(2(1/s_r)E[v] + E[v^2])\}}{2\{1-p-(1-p)\lambda_r E[v]\}}$$

where $s_r^{(2)}$ is the second moment of the service ratei, and v the vacation interval, that is the duration of a disconnection.

Data contention analysis.

From the resource contention analysis,

$$R_{E_{strict}} = N_q R_q^s + N_u R_u^s \qquad \text{and} \qquad R_{E_{weak}} = N_q R_q^w + N_u R_u^w$$

We divide the state i of each weak transaction into two substates, a lock state i_1, and an execution state i_2. In substate i_1 the transaction holds $i-1$ locks and is waiting for the ith lock. In substate i_2 it holds i locks and is executing. Similarly, we divide each state of a strict transaction in three substates i_0, i_1 and i_2. Let $q_x = (N_q/N)q_r + (N_u/N)q_w$. In substate i_0, a transaction is at its initiating cluster, holds $(i-1)q_x$ locks and sends messages to other clusters. In substate i_1 the transaction holds $(i-1)q_x$ locks and is waiting for the ith set of locks. In substate i_2 it holds $(i-1)q_x + q_r$ $((i-1)q_x + q_w)$ locks and is executing. The probability that a transaction enters substate i_1 upon leaving state $i-1$ or i_0 is P_{WR}, P_{WW}, P_q, and P_u respectively, for WR, WW, SR and SW lock requests. The mean time a_{op} spent at substate i_2 is computed from the resource contention analysis, for instance $a_{WR} = w + t_q$. Let c_{op} for a strict *op* be the mean time spent at state i_0 for instance, $c_{SR} = w + (1-p_l)(q_r t_b + t_r) + p_l((q_r - 1)t_b + b_r t_r)$. The time spent at state i_1 is R_{op}, and the unconditional mean time spent in substate i_1 is b_{op}, for instance $b_{WW} = P_{WW} R_{WW}$.

Let d^h_{op} (d^c_{op}) be the mean number of hot (cold) copies written by an *op* operation and I^c_{op} the mean number of *op* operations per copy. For instance, for $op = WR$ and hot copies, $d^h_{WR} = oh + \frac{(1-o)D_h}{D}$, and $I^h_{WR} = \frac{(1-c)\lambda_q d^h_{WR}}{D_h}$. Given a mean lock holding time of T_W (T_S) for weak (strict) transactions and assuming that the lock request times are a poisson process, the probability of contention on a lock request for a copy equals the lock utilization. Let P_{op_1/op_2} stand for the probability that an op_1-lock request conflicts with an op_2-lock request, then for example

$$P_{WR/WW} = d^c_{WR} I^c_{WW} T_W + d^h_{WR} I^h_{WW} T_W \qquad \text{and}$$
$$P_{WR} = d^h_{WR}(I^h_{SW} T_S + I^h_{WW} T_W) + d^c_{WR}(I^c_{SW} T_S + I^c_{WW} T_W)$$

Let G_W (G_S) be the sum of the mean lock holding times over all N copies accessed by a weak (strict) transaction,

$$G_W = \sum_{i=1}^{N} [\frac{N_q}{N}(i\, a_{WR} + (i-1) b_{WR}) + \frac{N_u}{N}(i\, a_{WW} + (i-1) b_{WW})] + NT_c$$

where T_c is the mean time to commit. Then $T_W = \frac{G_W}{N}$. Similar formulas hold for G_S and T_S.

Let $N^{i_a}_W$ ($N^{i_a}_S$) be the mean number of weak (strict) transactions per cluster in substate i_a and $CP^{i_a}_{op_1/op_2}$ be the conditional probability that an op_1-lock request contents with a transaction in substate i_a that holds an incompatible op_2-lock given that lock contention occurs. Now we can approximate R_{op}, for instance

$$R_{WR} = \sum_{i=1}^{N} [CP^{i_1}_{WR/WW}(\frac{R_{WW}}{f_1} + a_{WW} + s_{W_i}) + CP^{i_2}_{WR/WW}(\frac{a_{WW}}{f_4} + s_{W_i}) + CP^{i_0}_{WR/SW}(\frac{c_{SW}}{f_2} +$$
$$R_{SW} + a_{SW} + s_{S_i}) + CP^{i_1}_{WR/SW}(\frac{R_{SW}}{f_1} + a_{SW} + s_{S_i}) + CP^{i_2}_{WR/SW}(\frac{a_{SW}}{f_2} + s_{S_i})] + \frac{NT_c}{G_W}(\frac{c}{f_3})$$

where the factors f_i express the mean remaining times at the corresponding substate and depend on the distribution of times at each substate. Finally, s_{W_i} (s_{S_i}) is the mean time for a weak (strict) transaction from acquiring the ith lock till the end of commit, for instance $s_{W_i} = (N-i)[\frac{N_q}{N}(a_{WR} + b_{WR}) + \frac{N_u}{N}(a_{WW} + b_{WW})] + T_c$.

Reconciliation

The probabilities of edges in the serialization graphs are given below:

$$p_{WW} = [1 - (1 - \frac{N_u}{D})^N] p_c \qquad p'_{WW}(m, m') = 1 - (1 - p_{WW})^{(m'-1)}$$
$$p_{WS} = 1 - (1 - \frac{N_q}{n(lD_h + l'D_c)})^{(N_u q_W)} \qquad p'_{WS}(m, m') = 1 - (1 - p_{WS})^m$$

$$p_{SW} = p_{WS} \qquad p'_{SW}(m, m') = 1 - (1 - p_{SW})^{m'}$$
$$p_{SS} = 1 - (1 - \frac{N_u}{D})^N$$
$$p'_{SS}(m, m') = 1 - (1 - p_{SS})^{(m-1)}$$

where $p_c = \dfrac{1}{n^2}$ is the probability that two given transactions are initiated at the same cluster.