Data Consistency in Intermittently Connected Distributed Systems

Evaggelia Pitoura, *Member, IEEE Computer Society*, and Bharat Bhargava, *Fellow, IEEE*

Abstract—Mobile computing introduces a new form of distributed computation in which communication is most often intermittent, low-bandwidth, or expensive, thus providing only weak connectivity. In this paper, we present a replication scheme tailored for such environments. Bounded inconsistency is defined by allowing controlled deviation among copies located at weakly connected sites. A dual database interface is proposed that in addition to read and write operations with the usual semantics supports weak read and write operations. In contrast to the usual read and write operations that read consistent values and perform permanent updates, weak operations access only local and potentially inconsistent copies and perform updates that are only conditionally committed. Exploiting weak operations supports disconnected operation since mobile clients can employ them to continue to operate even while disconnected. The extended database interface coupled with bounded inconsistency offers a flexible mechanism for adapting replica consistency to the networking conditions by appropriately balancing the use of weak and normal operations. Adjusting the degree of divergence among copies provides additional support for adaptivity. We present transaction-oriented correctness criteria for the proposed schemes, introduce corresponding serializability-based methods, and outline protocols for their implementation. Then, some practical examples of their applicability are provided. The performance of the scheme is evaluated for a range of networking conditions and varying percentages of weak transactions by using an analytical model developed for this purpose.

Index Terms—Mobile computing, concurrency control, replication, consistency, disconnected operation, transaction management, adaptability.

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 networking environment that provides varying, intermittent, and weak connectivity.

In particular, mobile clients encounter *wide variations* in connectivity ranging from high-bandwidth, low latency communications through wired networks to total lack of connectivity [7], [13], [27]. Between these two extremes, connectivity is frequently provided by wireless networks characterized by low bandwidth, excessive latency, or high cost. To overcome availability and latency barriers and reduce cost and power consumption, mobile clients most often deliberately avoid use of the network and thus operate switching between connected and disconnected modes of operation. To support such behavior, *disconnected operation*, that is the ability to operate disconnected, is essential for mobile clients [13], [14], [30], [25]. In addition to disconnected operation, operation that exploits *weak connectivity*, that is connectivity provided by intermittent, low-bandwidth, or expensive networks, is also desirable [20], [12]. Besides mobile computing, weak and intermittent connectivity also applies to computing using portable laptops. In this paradigm, clients operate disconnected most of the time and occasionally connect through a wired telephone line or upon returning to their working environment.

Private or corporate databases will be stored at mobile as well as static hosts and mobile users will query and update these databases over wired and wireless networks. These databases, for reasons of reliability, performance, and cost will be distributed and replicated over many sites. In this paper, we propose a replication scheme that supports weak connectivity and disconnected operation by balancing network availability against consistency guarantees.

In the proposed scheme, 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 are tolerated for copies at different clusters. The interface offered by the database management system is enhanced with operations providing weaker consistency guarantees. Such weak operations allow access to locally, i.e., in a cluster, available data. Weak reads access bounded inconsistent copies and weak writes make conditional updates. The usual operations, called strict in this paper, are also supported. They offer access to consistent data and perform permanent updates.

The scheme 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 provides for better bandwidth utilization, latency and cost. In cases of strong connectivity, using only strict operations makes the scheme reduce to the usual one-copy semantics. Additional support

• E. Pitoura is with the Department of Computer Science, University of Ioannina, GR 45110 Ioannina, Greece. E-mail: pitoura@cs.uoi.gr.
• B. Bhargava is with the Department of Computer Sciences, Purdue University, West Lafayette, IN 47907. E-mail: bb@cs.purdue.edu.

*Manuscript received 2 July 1996; revised 19 May 1997, accepted 22 Jan. 1998.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 104369.*

for adaptability is possible by tuning the degree of inconsistency among copies based on the networking conditions.

In a sense, weak operations offer a form of *application-aware adaptation* [21]. Application-aware adaptation characterizes the design space between two extreme ways of providing adaptability. 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. Since, in general, the system is not aware of the application semantics, it 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. They are tools offered by the database system to applications. The application can at its discretion use weak or strict transactions based on its semantics. The implementation, consistency control, and the underlying transactional support is the job of the database management system.

The remainder of this paper is organized as follows. In Section 2, we introduce the replication model along with an outline of a possible implementation that is based on distinguishing data copies into core and quasi. In Sections 3 and 4, we define correctness criteria, prove corresponding serializability-based theorems, and present protocols for maintaining weak consistency under the concurrent execution of weak and strict transactions and for reconciling divergent copies,, respectively. Examples of how the scheme can be used are outlined in Section 5. In Section 6, we develop an analytical model to evaluate the performance of the scheme and the interplay among its various parameters. The model is used to demonstrate how the percentage of weak transactions can be effectively tuned to attain the desired 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. In Section 7, we provide an estimation of the reconciliation cost. This estimation can be used for instance to determine an appropriate frequency for the reconciliation events. In Section 8, we compare our work with related research, and we conclude the paper in Section 9 by summarizing.

2 The Consistency Model

The sites of a distributed system are grouped together in sets called *physical clusters* (or p-clusters) so that sites that are strongly connected with each other belong to the same p-cluster, while sites that are weakly connected with each other belong to different p-clusters. Strong connectivity refers to connectivity achieved through high-bandwidth and low-latency communications. Weak connectivity includes connectivity that is intermittent or low bandwidth. The goal is to support autonomous operation at each physical cluster, thus eliminating the need for communication among p-clusters since such intercluster communication may be expensive, prohibitively slow, and occasionally unavailable. To this end, weak transactions are defined as access copies at a single p-cluster. At the same time, the usual atomic, consistent, durable, and isolated distributed transactions, called strict, are also supported.

2.1 The Extended Database Operation Interface

To increase availability and reduce intercluster communication, direct access to locally, e.g., in a p-cluster, available copies is achieved through *weak read* (WR) and *weak write* (WW) operations. Weak operations are local at a p-cluster, i.e., they access copies that reside at a single p-cluster. We say that a copy or item is locally available at a p-cluster if there exists a copy of that item at the p-cluster. We call the standard read and write operations *strict read* (SR) and *strict write* (SW) operations. To implement this dual database interface, we distinguish the copies of each data item as core and quasi. *Core* copies are copies that have permanent values, while *quasi* copies are copies that have only conditionally committed values. When connectivity is restored, the values of core and quasi copies of each data item are *reconciled* to attain a system-wide consistent value.

To process the operations of a transaction, the database management system translates operations on data items into operations on copies of these data items. This procedure is formalized by a *translation function* h. Function h maps each read operation on a data item x into a number of read operations on copies of x and returns one value (e.g., the most up-to-date value) as the value read by the operation. That is, we assume that h when applied to a read operation returns one value rather than a set of values. In particular, h maps each $SR[x]$ operation into a number of read operations on core copies of x and returns one from these values as the value read by the operation. Depending on how each weak read operation is translated, 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 operation into a number of read operations only on locally available quasi copies and returns the most up-to-date such value.

Based on the time of propagation of updates of core copies to quasi copies, we define two types of translation functions: an *eventual* translation function that maps an $SW[x]$ into writes of core copies only and an *immediate* translation function that in addition updates the quasi copies that reside at the same p-cluster with the core copies written. For an immediate h, conservative and best-effort have the same result. Each $WW[x]$ operation is translated by h into a number of write operations of local quasi copies of x. Table 1 summarizes the semantics of operations.

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 or ROWA [3].

Users interact with the database system through transactions, that is, through the execution of programs that include database operations:

TABLE 1
Variations of the Translation Function

Weak Read (WR)	Reads local copies and returns as the value read the most recent one		
	Variations	**Best-effort h:**	Reads both core and quasi local copies
		Conservative h:	Reads only local quasi copies
Strict Read (SR)	Reads core copies and returns as the value read the most recent one		
Weak Write (WW)	Writes local quasi copies		
Strict Write (SW)	Variations	**Eventual h:**	Writes only core copies
		Immediate h:	Writes both core and quasi copies at the corresponding p-clusters

Definition 1. *A transaction (T) is a partial order (OP, $<$), where OP is the set of operations executed by the transaction, and $<$ represents their execution order. The operations include the following data operations: weak (WR) or strict reads (SR) and weak (WW) or strict writes (SW), as well as abort (A) and commit (C) operations. 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 or strict data operations conflict if they access the same copy of a data item and at least one of them is a weak or strict write operation.*

Two types of transactions are supported, weak and strict. A *strict* transaction (*ST*) is a transaction where *OP* does not include any weak operations. Strict transactions are atomic, consistent, isolated and durable. A *weak* transaction (*WT*) is a transaction where *OP* does not include any strict operations. Weak transactions access data copies that reside at the same physical cluster and thus are executed locally at this p-cluster. Weak transactions are *locally* committed at the p-cluster at which they are executed. After local commitment, their updates are visible only to weak transactions in the same p-cluster; other transactions are not aware of these updates. Local commitment of weak transactions is conditional in the sense that their updates might become permanent only after reconciliation when local transactions may become globally committed. Thus, there are two commit events associated with each weak transaction, a local conditional commit in its associated cluster and an implicit *global commit* at reconciliation.

Other types of transactions that combine weak and strict operations can be envisioned; their semantics, however, become hard to define. Instead, weak and strict transactions can be seen as transaction units of some advanced transaction model. In this regard, upon its submission, each user transaction is decomposed into a number of weak and strict subtransaction units according to semantics and the degree of consistency required by the application.

2.2 Data Correctness

As usual, a database is a set of data items and a database *state* is defined as a mapping of every data item to a value of its domain. Data items are related by a number of restrictions called *integrity constraints* that express relationships among their values. A database state is consistent if the integrity constraints are satisfied [22]. In this paper, we consider integrity constraints to be arithmetic expressions that have data items as variables. Consistency maintenance in traditional distributed environments relies on the assumption that normally all sites are connected. This assumption, however, is no longer valid in mobile computing since the distributed sites are only intermittently connected. Similar network connectivity conditions also hold in widely distributed systems as well as in computing with portable laptops.

To take into account intermittent connectivity, instead of requiring maintenance of all integrity constraints of a distributed database, we introduce *logical clusters* as units of consistency. A logical cluster, l-cluster, is the set of all quasi copies that reside at the same p-cluster. In addition, all core copies constitute a single system-wide logical cluster independently of the site at which they reside physically. We relax consistency in the sense that integrity constraints are ensured only for data copies that belong to the same logical cluster. An intracluster integrity constraint is an integrity constraint that can be fully evaluated using data copies of a single logical cluster. All other integrity constraints are called intercluster integrity constraints.

Definition 2. *A mobile database state is consistent if all intracluster integrity constraints are satisfied and all intercluster integrity constraints are bounded-inconsistent.*

In this paper, we focus only on replication intercluster integrity constraints. For such integrity constraints, bounded inconsistency means that all copies in the same logical cluster have the same value while among copies at different logical clusters there is bounded divergence [31],

TABLE 2
Divergence Among Copies

	maximum number of transactions that operate on quasi copies
	range of acceptable values that a data item can take
d is the:	maximum number of copies per data item that diverge from a pre-assigned primary copy of the item, i.e., the core copies
	maximum number of data items that have divergent copies
	maximum number of updates per data item that are not reflected at all copies of the item

[1]. Bounded divergence is quantified by a positive integer d, called degree of divergence; possible definitions of d are listed in Table 2. A replication constraint for x thus bounded is called d-consistent. The degree of divergence among copies can be tuned based on the strength of connection among physical clusters, by keeping the divergence small in instances of high bandwidth availability and allowing for greater deviation in instances of low bandwidth availability.

Immediate Translation and Consistency. To handle integrity constraints besides replication, in the case of an *immediate* translation function h, h should be defined such that the integrity constraints between quasi copies in the same logical cluster are not violated. The following example is illustrative.

Example 1. For simplicity, consider only one physical cluster. Assume two data items x and y, related by the integrity constraint $x > 0 \Rightarrow y > 0$, and a consistent database state $x_c = -1$, $x_q = -1$, $y_c = 2$, and $y_q = -4$, where the subscripts c and q denote core and quasi copies, respectively.

Consider the transaction program:

```
x = 10
if y < 0
   then y = 10
```

If the above program is executed as a strict transaction $SW(x)\ SR(y)\ C$, we get the database state $x_c = 10$, $x_q = 10$, $y_c = 2$, and $y_q = -4$ in which the integrity constraint between the quasi copies of x and y is violated. Note that the quasi copies were updated because we considered an immediate translation function.

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 them. Similar problems occur when refreshing individual copies of a cache [1]. Possible solutions include: 1) Each time a quasi copy is updated at a physical cluster as a result of a strict write, the quasi copies of all data in this cluster 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 x and y should have been reconciled prior to the execution of the transaction, producing for instance the database state $x_c = -1$, $x_q = -1$, $y_c = 2$, and $y_q = 2$. Then, the execution of the transaction would result in the database state $x_c = 10$, $x_q = 10$, $y_c = 2$, and $y_q = 2$, which is consistent. 2) If a strict transaction updates a quasi copy at a physical cluster, its read operations are also mapped into reads of quasi copies at this cluster. In cases of incompatible reads, again, a reconciliation procedure is initiated having a result similar to the one above. 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 $x_c = 10$, $x_q = -1$, $y_c = 2$, and $y_q = -4$, which is consistent. The first two approaches may force an immediate reconciliation among copies, while the third approach defers this reconciliation and is preferable in cases of low connectivity among physical clusters.

3 Weak Connectivity Operation

In this section, we provide serializability-based criteria, graph-based tests and a locking protocol for correct executions that exploit weak connectivity. When o_j is an operation, the subscript j denotes that o belongs to transaction j, while the subscript on a data copy identifies the physical cluster at which the copy is located. Without loss of generality, we assume that there is only one quasi copy per physical cluster. This assumption can be easily lifted but with significant complication in notation. Since all quasi copies in a physical cluster have the same value, this single copy can be regarded as their representative. Read and write operations on copies are denoted by *read* and *write*, respectively.

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

Definition 3 *(Intracluster schedule). Let $T = \{T_0, T_1, \ldots, T_n\}$ 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 $SW_j[x] <_j SR_j[x]$ and $read_j(x_i) \in h(SR_j[x])$, then $write_j(x_i) \in h(SW_j[x])$.*
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 at physical cluster 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 $<_i$ stipulated by each transaction T_i 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 core copy at a physical cluster Cl_i then all other writes of this transaction must also write any corresponding copies at this cluster. 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 (i.e., returns as the value read) a copy of x written by T_i and no other transaction writes this copy in between. We say that a transaction T_i has the *same reads-from* relationship in schedule S_1 as in schedule S_2 if, for any data item x, T_i reads-x-from T_j in S_1, then it reads-x-from T_j in S_2. 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 physical cluster* Cl_k is the schedule obtained from S by deleting all operations that access copies not in Cl_k. Two schedules are *conflict equivalent* if they are defined over the same set of transactions, have the same set of operations and order conflicting operations of committed transactions the same way [3]. A *one-copy* schedule is the single-copy interpretation of an (intracluster) schedule where all operations on data copies are represented as operations on the corresponding data item.

3.1 Correctness Criterion

A correct concurrent execution of weak and strict transactions must maintain bounded-inconsistency. First, we consider a weak form of correctness, in which the only requirement for weak transactions is that they read consistent data. The requirement for strict transactions is stronger, because they must produce a system-wide consistent database state. In particular, the execution of strict transactions must hide the existence of multiple core copies per item, i.e., it must be view-equivalent to a one-copy schedule [3]. A replicated-copy schedule S is *view-equivalent to a one-copy schedule*S_{1C} if 1) S and S_{1C} have the same reads-from relationship for all data items, and 2) for each final write $WR_i(x)$ in S_{1C}, $write_i(x_j)$ is a final write in S for some copy x_j of x. These requirements for the execution of strict and weak transactions are expressed in the following definition:

Definition 4 *(IAS weak correctness). An intracluster schedule S_{IAS} is weakly correct iff all the following three conditions are satisfied:*

1. *All transactions in S_{IAS} have a consistent view, i.e., all integrity constraints that can be evaluated using the data read are valid;*
2. *There is a one copy serial schedule S such that: a) it is defined on the same set of strict transactions as S_{IAS}, b) strict transactions in S have the same reads-from relationship as in S_{IAS}, and c) the set of final writes in S is the same as the set of final writes on core copies in S_{IAS}; and*
3. *Bounded divergence among copies at different logical clusters is maintained.*

Next, we discuss how to enforce the first two conditions. Protocols for bounding the divergence among copies at different logical clusters are outlined at the end of this section. A schedule is *one-copy serializable* if it is equivalent to a serial one-copy schedule. The following theorem defines correctness in terms of equivalence to serial executions.

Theorem 1. *Given that bounded divergence among copies at different logical clusters is maintained, if the projection of an intracluster schedule S on strict transactions is one-copy serializable and each of its projections on a physical cluster is conflict-equivalent to a serial schedule, then S is weakly correct.*

Proof. Condition 1 of Definition 4 is guaranteed for strict transactions from the requirement of one-copy serializability since strict transactions get the same view as in a one-copy serial schedule and read only core copies. For weak transactions at a physical cluster, the first condition is provided from the requirement of serializability of the projection of the schedule on this cluster given that the projection of each (weak or strict) transaction on the cluster satisfies all intracluster integrity constraints when executed alone. Thus, it suffices to prove that such projections maintain the intracluster integrity constraints. This trivially holds for weak transactions, since they are local at a physical cluster. The condition also holds for strict transactions since, if a strict transaction maintains consistency of all database integrity constraints, then its projection on any cluster also maintains the consistency of intracluster integrity constraints as a consequence of Condition 6 of Definition 3. Finally, one copy serializability of the projection of strict transactions suffices to guarantee Conditions 2b and 2c since strict transactions read only core copies and weak transactions do not write core copies, respectively. □

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

Example 2. Assume two physical clusters $Cl_1 = \{x, y\}$ and $Cl_2 = \{w, z, l\}$ that have both quasi and core copies of the corresponding data items, 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 translation function h. For notational simplicity, we do not use any special notation for the core and quasi copies, since which copies are read is inferred by the translation function.

Assume that the execution of the above transactions produces the following schedule which is weakly correct:

$$S = WR_5[w]SW_1[x]WR_3[x]SW_1[w]C_1SW_2[y]SW_2[z]$$
$$SR_2[x]C_2WR_3[y]C_3WR_4[z]WW_4[l]C_4WR_5[l]C_5.$$

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 of S 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 all weak transactions get the same consistent view. Obviously, strong correctness implies weak correctness.

Definition 5 *(IAS strong correctness). An intracluster schedule S is strongly correct iff there is a serial schedule S_S such that*

1. *S_S is conflict-equivalent with S;*
2. *There is a one-copy schedule S_{1C} such that a) strict transactions in S_S have the same reads-from relationship as in S_{1C} and b) the set of final writes on core copies in S_S is the same as in S_{1C}; and*
3. *Bounded divergence among copies at different logical clusters is maintained.*

Lemma 1. *Given that bounded divergence among copies at different logical clusters is maintained if the projection of an intracluster schedule S is conflict-equivalent to a serial schedule S_S and its projection on strict transactions is view equivalent to a 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}, S is strongly correct.*

Proof. We need to prove that in S_{1C} strict transactions have the same reads-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 conflict with weak transactions at other clusters directly, the following is an equivalent statement of the above lemma:

Corollary 1. *Given that bounded divergence among copies at different logical clusters is maintained, if the projection of an intracluster schedule S on strict transactions is view equivalent to a one-copy serial schedule S_{1C}, and each of its projections on a physical 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}, S is strongly correct.*

If weak *IAS* correctness is used as the correctness criterion, then the transaction managers at each physical cluster must only synchronize projections on their cluster. Global control is required only for synchronizing strict transactions. Therefore, no control messages are necessary between transaction managers at different clusters for synchronizing weak transactions. The proposed scheme is flexible, in that any coherency control method that guarantees one-copy serializability (e.g., quorum consensus or primary copy) can be used for synchronizing core copies. The scheme reduces to one-copy serializability when only strict transactions are used.

3.2 The Serialization Graph

To determine whether an *IAS* schedule is correct, we use a modified serialization graph, that we call the *intracluster serialization graph* (IASG) of the *IAS* schedule. To construct the *IASG*, first a replicated data serialization graph (SG) is built that includes all 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 create conflicts. Acyclicity of the SG implies one-copy serializability of the corresponding schedule. Then, the *SG* is augmented to include weak transactions as well as edges that represent conflicts between weak transactions in the same cluster and weak and strict transactions. An edge is called a *dependency edge* if it represents the fact that a transaction reads a value produced by another transaction. An edge is called 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 and*
- *a precedence edge from WT_i to ST.*

Proof. The proof is straightforward 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 2. *Let S_{IAS} be an intracluster schedule. If S_{IAS} has an acyclic $IASG$, then S_{IAS} is strongly correct.*

Proof. When a graph is acyclic then each of its subgraphs is acyclic, thus the underlying SG on which the $IASG$ was built is acyclic. Acyclicity of the SG implies one-copy serializability of strict transactions, since strict transactions only read values produced by strict transactions. Let $T_1, T_2, \ldots, T_n$ be all transactions in S_{IAS}. Thus, $T_1, T_2, \ldots, T_n$ are the nodes of the IASG. Since IASG is acyclic it can be topologically sorted. Let $T_{i_1}, T_{i_2}, \ldots, T_{i_n}$ be a topological sort of the edges in IASG, then by a straightforward application of the serializability theorem [3], S_{IAS} is conflict equivalent to the serial schedule $S_S = T_{i_1}, T_{i_2}, \ldots, T_{i_n}$. This order is consistent with the partial order induced by a topological sorting of the SG, allowing S_{1C} to be the serial schedule corresponding to this sorting. 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 ensures that all copies of a data item have the same value. In the proposed scheme, we must maintain this property globally for core and locally for quasi copies. Concurrency control ensures the maintenance of the other integrity constraints, here the intracluster integrity constraints. For coherency control, we assume a generic quorum-based scheme [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 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. Fig. 1 depicts the compatibility of locks for various types of translation functions and is presented to demonstrate the interference between operations on items. Differences in compatibility stem from the fact the operations access different kinds of copies. The basic overhead on the performance of weak transactions imposed by these protocols is caused by other weak transactions at the same cluster. This overhead is small, since weak transactions do not access the slow network. Strict transactions block a weak transaction only when they access the same quasi copies. This interference is limited and can be controlled, e.g., by letting in cases of disconnections, strict transactions access only core copies and weak transactions access only quasi copies.

	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)

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

Bounding divergence among copies. At each p-cluster, the degree for each data item 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 by either limiting the number of weak writes pending global commitment or by controlling the h function. In Table 3, we outline ways of maintaining d-consistency for different ways of defining d.

4 A Consistency Restoration Schema

After the execution of a number of weak and strict transactions, for each data item, all its core copies have the same value, while its quasi copies may have as many different values as the number of physical clusters. Approaches to reconciling the various values of a data item so that a single value is selected vary from purely syntactic to purely semantic ones [5]. Syntactic approaches use serializability-based criteria, while semantic approaches use either the semantics of transactions or the

TABLE 3
Maintaining Bounded Inconsistency

Definition of divergence (d):	Applicable method:
The maximum number of transactions that operate on quasi copies	Appropriately bound the number of weak transactions at each physical cluster. In the case of a dynamic cluster reconfiguration, the distribution of weak transactions at each cluster must be re-adjusted.
A range of acceptable values a data item can take	Allow only weak writes with values inside the acceptable range.
The maximum number of divergent copies per data item	For each data item, bound the number of physical clusters that can have quasi copies.
The maximum number of data items that have divergent copies	Bound the number of data items that can have quasi copies.
The maximum number of updates per data item not reflected at all copies	Define h so that each strict write modifies the quasi copies at each physical cluster at least after d updates. Note, that this cannot be ensured for disconnected clusters, since there is no way of notifying them for remote updates.

semantics of data items. We adopt a purely syntactic and thus application-independent approach. The exact point when reconciliation is initiated depends on the application requirements and the distributed system characteristics. For instance, reconciliation may be forced to keep inconsistency inside the required limits. Alternatively, it may be initiated periodically or on demand upon the occurrence of specific events, such as the restoration of network connectivity, for instance when a palmtop is plugged-back to the stationary network or a mobile host enters a region with good connectivity.

4.1 Correctness Criterion

Our approach for reconciliation is based on the following rule: A weak transaction becomes globally commited if the inclusion of its write operations in the schedule does not violate the one-copy serializability of strict transactions. That is, we assume that weak transactions at different clusters do not interfere with each other even after reconciliation, that is weak operations of transactions at different clusters never conflict. A (complete) intercluster schedule, IES, models execution after reconciliation, where strict transactions become aware of weak writes, i.e., weak transactions become globally committed. Thus, in addition to the conflicts reported in the intracluster schedule, the intercluster schedule reports all relevant conflicts between weak and strict operations. In particular:

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. *$OP' = OP$, and for any op_i and $op_j \in OP'$, if $op_i <_a op_j$ in S_{IAS}, then $op_i <_e op_j$ in S_{IES}, in addition:*
2. *For each pair of weak write $op_i = WW_i[x]$ and strict read $op_j = SR_j[x]$ operations, either for all pairs of operations, $cop_i \in h(op_i)$ and $cop_j \in h(op_j)$, $cop_i <_e cop_j$ or $cop_j <_e cop_i$;*
3. *For each pair of weak write $op_i = WW_i[x]$ and strict write $op_j = SW_j[x]$ operations, either for all pairs of operations, $cop_i \in h(op_i)$ and $cop_j \in h(op_j)$, $cop_i <_e cop_j$ or $cop_j <_e cop_i$.*

We extend the reads-from relationship for strict transactions so that weak writes are taken into account. A strict read operation on a data item x *reads-x-from* a transaction T_i in an IES schedule, if it reads a copy of x and T_i has written this or any quasi copy of x and no other transaction wrote this or any quasi copy of x in between. A weak write is acceptable as long as the extended reads-from relationship for strict transactions is not affected, that is strict transactions still read values produced by strict transactions. In addition, correctness of the underlying *IAS* schedule implies one-copy serializability of strict transactions and consistency of weak transactions.

```
Until there are no cycles in the IESG
    rollback a weak transaction WT in the cycle
    rollback 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
```

Fig. 2. The reconciliation steps.

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

1. *It is based on a correct IAS schedule S_{IAS}, and*
2. *The reads-from relationship for strict transactions is the same with their reads-from relationship 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 access 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, we add an edge $ST_j \rightarrow WT$.

Theorem 3. *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) and thus the IAS schedule is correct. We shall show that if the graph is acyclic, then the reads-from relationship 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 IESG, 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 T may result in *cascading aborts* of transactions that have read the values written by T; that is, transactions that are related to T through a dependency edge. Since weak transactions write only quasi copies in a single physical cluster and since only weak transactions in the same cluster can read these quasi copies, we get the following lemma:

Lemma 2. *Only weak transactions in the same physical cluster read values written by weak transactions in that cluster.*

The above lemma ensures that when a weak transaction is aborted to resolve conflicts in an intercluster schedule, only weak transactions in the same p-cluster are affected. In practice, fewer transactions ever need to be aborted. In particular, we need to abort only weak transactions whose output depends on the exact values of the data items they read. We call these transactions *exact*. Most weak transactions are not exact 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 IESG can be hard. The difficulties arise from the fact that between transactions that wrote a data item an edge can have any direction, thus resulting in *polygraphs* [22]. Polynomial tests for acyclicity are possible, if we make the assumption that transactions read a data item before writing it. Then, to get the IES graph from the IAS graph, 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 $ST \rightarrow WT$.

Fig. 2 outlines the reconciliation steps.

5 Discussion

In the proposed hybrid scheme, weak and strict transactions coexist. Weak transactions let users process local data thus avoiding the overhead of long network accesses. Strict transactions need access to the network to guarantee permanence 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. *Weak writes* allow users to update local data without confirming these updates immediately. Update validation is delayed until the physical 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.

The cluster configuration is dynamic. Physical clusters may be explicitly created or merged upon a forthcoming disconnection or connection of the associated mobile clients. To accommodate migrating locality, a mobile host may join a different p-cluster upon entering a new support environment. Besides defining clusters based on the *physical location* of data, other definitions are also possible. 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. Some examples follow.

Example 1 (Cooperative environment). 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. Physical 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 physical cluster (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 physical clusters. In each physical 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.

Example 2 (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 when strict consistency is required and weak otherwise. 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.

Example 3 (Caching 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.

6 Quantitative Evaluation of Weak Consistency

To quantify the improvement in performance attained by sacrificing strict consistency in weakly connected environments and understand the interplay among the various parameters, we have developed an analytical model. The analysis follows an iteration-based methodology for coupling standard hardware resource and data contention as in [39]. 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 Fig. 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 varying disconnection intervals.

6.1 Performance Model

We assume a cluster configuration with n physical 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 physical 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 p-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 p-cluster into hot and cold data sets. Let D be the number of data items per p-cluster, D_c of which are cold and D_h hot. To capture *locality*, we assume that a fraction o of 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, a transaction may move to a different physical cluster and thus the data it accesses may no longer belong to the hot data of the new cluster. This can be modeled by letting o diminish. Locality is taken advantage of by the replication scheme by assuming that the probability that a hot data has a core copy at a p-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 p-cluster. Let q_r be the read and q_w the write quorum and N_S be the mean number of operations on data copies per strict transaction. 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 $1, 2, \ldots, n_L$ 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

$$r_{trans} = r_{INPL} + r_E + \sum_{j=1}^{n_w} r_{w_j} + t_{commit}, \tag{A}$$

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 $1, 2, \ldots, n_L$ 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.

6.1.1 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: the processing time, t_q, of a query on a data copy, the time t_u to install an update on a data copy, and the overhead time, t_b, 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 become disconnected and 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 among clusters is:

$$M = 2nc[\lambda_q(p_l(q_r - 1) + (1-p_l)q_r) + \lambda_u(p_l(q_w - 1) + (1-p_l)q_w)].$$

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 $R^w_q = w + t_q$ and for a weak update $R^w_u = w + t_u$, 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

$$R^s_q = p_l[w + t_q + (q_r - 1)t_b + b_r(2t_r + t_q + w)] + (1-p_l)(q_r t_b + 2t_r + t_q + w)$$

and for a strict write

$$R^s_u = p_l[w + t_u + (q_w - 1)t_b + b_w(2t_r + t_u + w)] + (1-p_l)(q_w t_b + 2t_r + t_u + w).$$

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, $t_r = 1/s_r + w_r$. The arrival rate λ_r at each link is Poisson with mean $M/(n(n-1))$. 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 (since $\lambda \leq E[x]$, where λ is the arrival rate of all requests at each cluster and $E[x]$ is the mean service time); b) the available bandwidth (since $\lambda_r \leq t_r$); and c) the disconnection intervals (since $\lambda_r \leq E[v]$, where $E[v]$ is the mean duration of a disconnection).

6.1.2 Data Contention Analysis

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

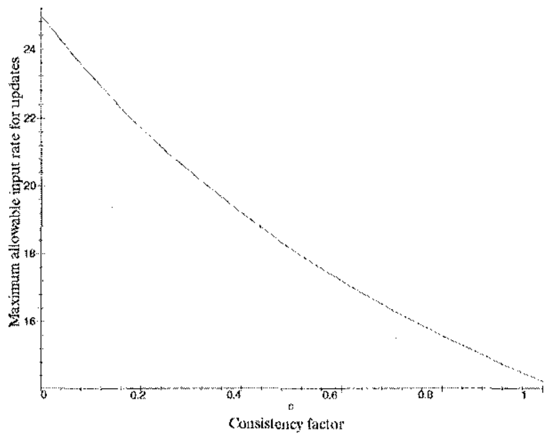
$$R_{strict-transaction} = R_{INPL} + R_{E_{strict}} + N_q P_q R_{SR} + N_u P_u R_{SW} + T_{commit}$$

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

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.

TABLE 4
Input Parameters

Parameter	Description	Value
n	number of physical 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	512 bits
D_c	number of cold data items per p-cluster	800
D_h	number of hot data items per p-cluster	200
N	average number of operations per transaction	10



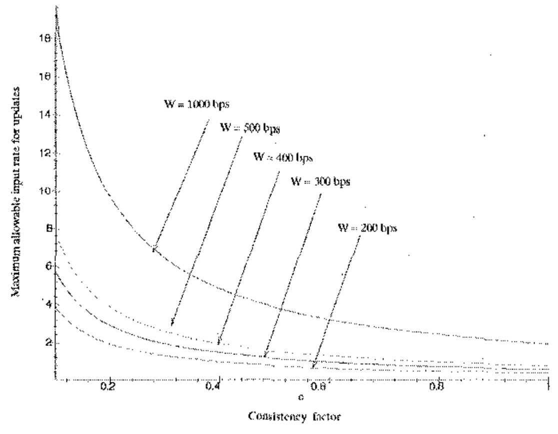


Fig. 3. Maximum allowable input rate of updates for various values of the consistency factor. (a) Limits imposed by the processing rate at each cluster ($\lambda \leq E[x]$). (b) Limits imposed bandwidth restrictions ($\lambda_r \leq t_r$).

6.2 Performance Evaluation

The following performance results show how the percentage of weak and strict transactions can be effectively tuned based on the prevailing networking conditions such as the available bandwidth and the duration of disconnections to attain the desired throughput and latency. Table 4 depicts 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 [7].

6.2.1 *System Throughput*

Fig. 3a, Fig. 3b, Fig. 4a, and Fig. 4b 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 Fig. 3a, 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

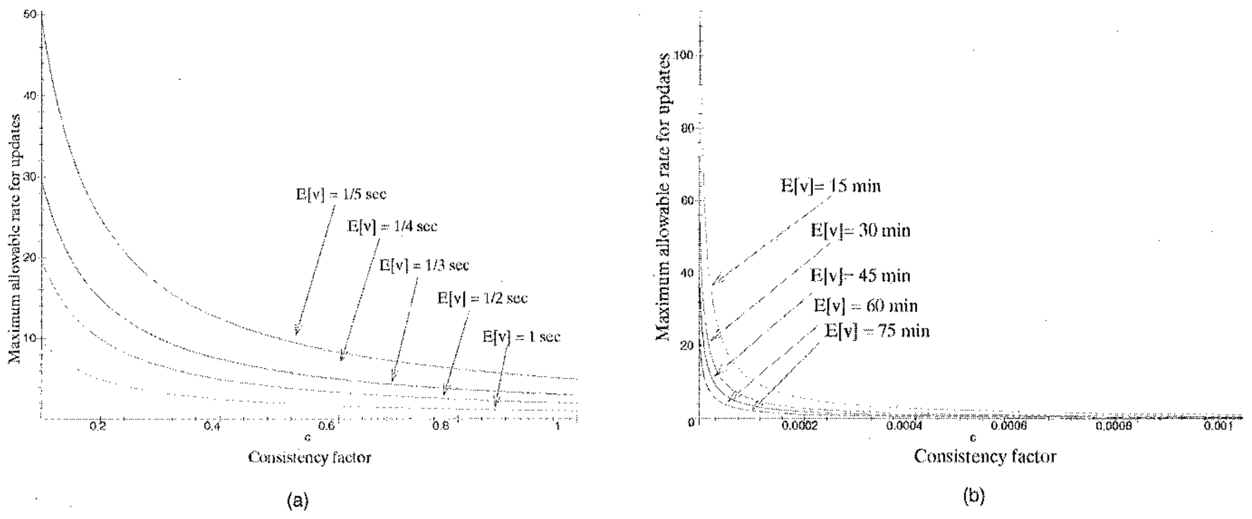


Fig. 4. Maximum allowable input rate for updates for various values of the consistency factor. Limits imposed by disconnections and their duration ($\lambda_r \leq E[v]$). (a) Disconnections lasting from 1/5 to 1 seconds and (b) longer disconnections lasting from 15 to 75 minutes.

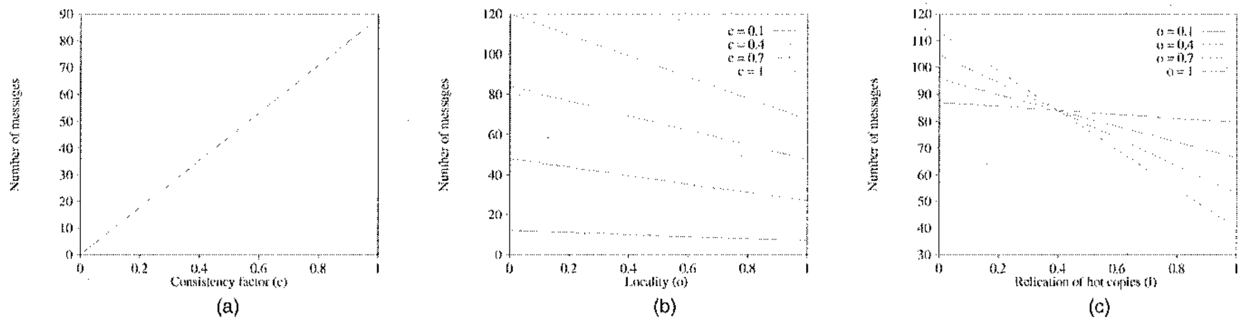


Fig. 5. Number of messages. (a) For various values of c. (b) With locality. (c) For different replication of hot core copies. Unless stated otherwise: $o = 0.6$, $l = 0.9$, $l' = 0.4$, $h = 0.9$, and $c = 0.7$.

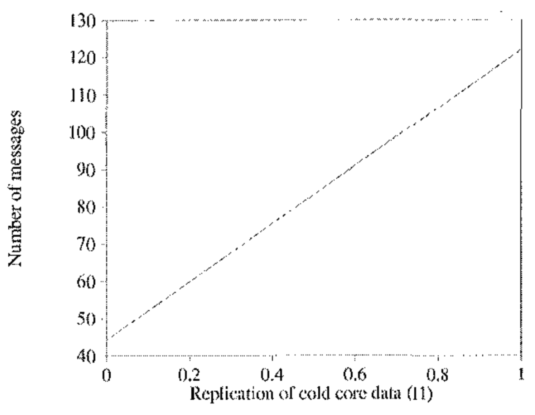
the networking conditions such as the duration of disconnections and the available bandwidth. As indicated in Fig. 3b, to get, for instance, the same throughput with 200 bps as with 1,000 bps and $c = 1$ we must lower the consistency factor below 0.1. The duration of disconnections may vary from seconds when they are caused by hand offs [19] to minutes, for instance when they are voluntary. Fig. 4 depicts the effect of the duration of a disconnection on the system throughput for short (Fig. 4a) and long disconnections (Fig. 4b). For long disconnections, only a very small percentage of strict transactions can be initiated at disconnected sites (Fig. 4a). To keep the throughput comparable to that for shorter disconnections (Fig. 4b), the consistency factor must drop at around three orders of magnitude.

6.2.2 Communication Cost

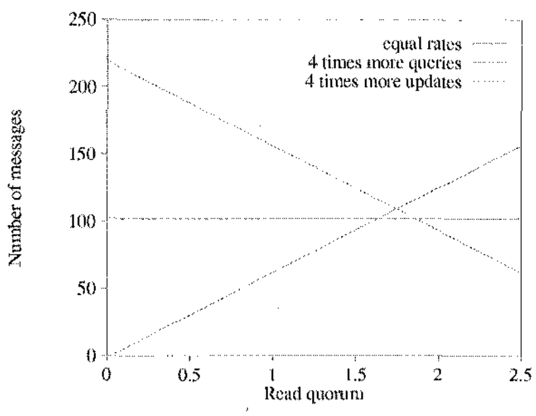
We estimate the communication cost by the number of messages sent. The number of messages depends on the following parameters of the replication scheme: 1) the consistency factor c, 2) the data distribution l for hot and l' for cold data, 3) the locality factor o, and 4) the quorums, q_r and q_w, of the coherency scheme. We assume a ROWA scheme ($q_r = 1$, $q_w = n_d$), if not stated otherwise. As shown in Fig. 5a, 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 (Fig. 5b). The number of messages decreases, when the replication factor of hot core copies increases (Fig. 5c). The decrease is more evident since most operations are queries and the coherency scheme 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 (Fig. 6a). Finally, the relationship between the read quorum and the number of messages depends on the relative number of queries and updates (Fig. 6b).

6.2.3 Transaction Response Time

The response time for weak and strict transactions is depicted in Fig. 7 for various values of c. 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



(a)



(b)

Fig. 6. Number of messages. (a) For different replication of cold core copies. (b) For different values of the read quorum. Unless stated otherwise: $o = 0.6$, $l = 0.9$, $l' = 0.4$, $h = 0.9$, and $c = 0.7$.

as indicated in Table 4. 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 such as those caused by hand-offs [19], and 4) the coherency control scheme 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 increases, since more conflicts occur. The increase is more dramatic for smaller values of bandwidth. Fig. 8a and Fig. 8b show the response time distribution, respectively, for strict and weak transactions and 2Mbps bandwidth. For strict transactions, the most important overhead is due to 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.

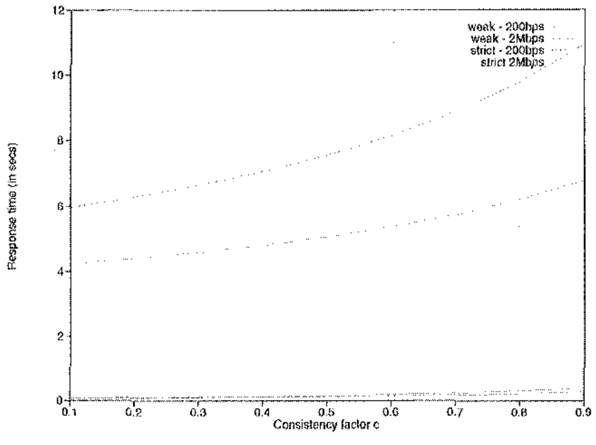


Fig. 7. Comparison of the response times for weak and strict transactions for various values of the consistency factor.

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.

A weak transaction WT is rolled back, if its writes conflict with a read operation of a strict transaction ST 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, $P = P_1P_2$ is the probability that a weak transaction is rolled back. Assume that reconciliation occurs after N_r transactions of which $k = cN_r$ 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 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, $P_1 \simeq 1 - (1 - N_u/D)^{N_q}$.

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, where the type of a transaction is either weak or strict. Let $p'_{KL}(m, m')$ be the probability that in an 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 for p_{KL} and $p'_{KL}(m, m')$ 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 transac-

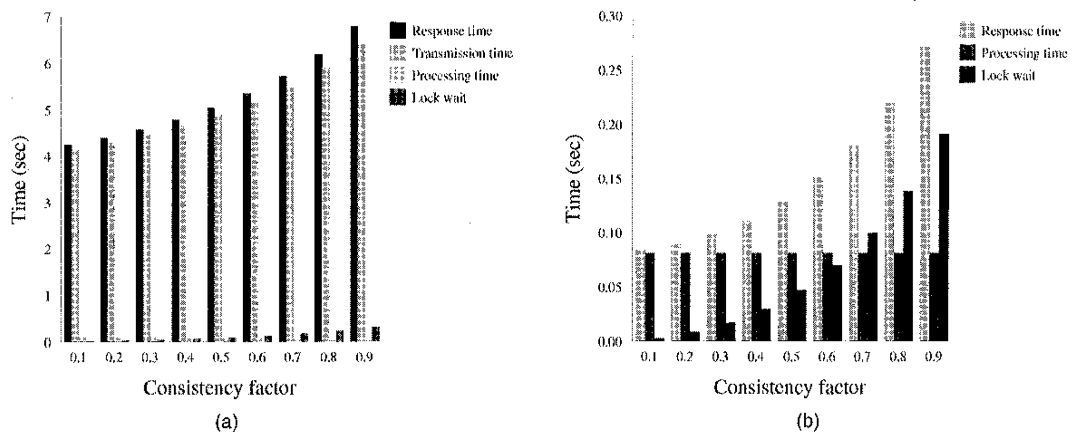


Fig. 8. (a) Response time distribution for strict transactions. (b) Response time distribution for weak transactions.

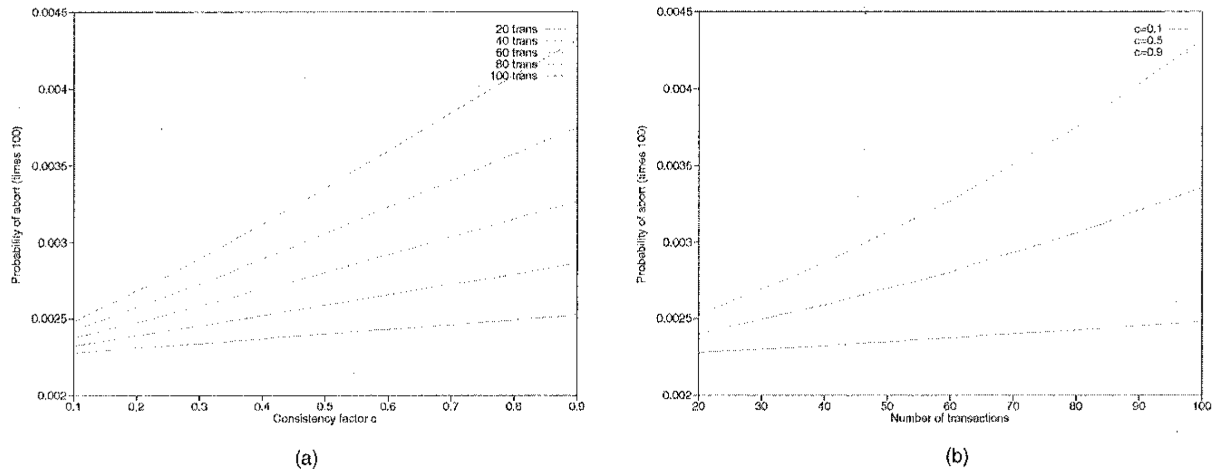


Fig. 9. Probability of abort for 3,000 data items.

tion to a given strict transaction in an 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:

$$\begin{array}{ll} p(m, m', 1) = p_{WS}, & \text{for all m} > 0, \text{m}' > 0 \\ p(m, m', 0) = 0, & \text{for all m} > 0, \text{m}' > 0, \\ p(m, 0, i) = 0, & \text{for all m} \geq 0, \text{i} > 0, \text{ and} \\ p(0, m', i) = 0 & \text{for all m}' \geq 0, \text{i} > 0 \end{array}$$

$$p(k, k', i+1) = 1 - [(1 - p'_{WW}(k, k')p(k, k'-1, i))$$
$$\left(\prod_{j=1}^{i-1} 1 - (p'_{WS}(k, k') \prod_{l=1}^{i-j-1} p'_{SS}(k-l, k'-1)\right.$$
$$\left. p'_{SW}(k-i+j, k'-1)p(k-i+j-1, k'-2, j)\right)$$
$$(1 - p'_{WS}(k, k') \prod_{l=1}^{i-1} p'_{SS}(k-l, k'-1)p_{SS})],$$

where the first term is the probability of a path whose first edge is between weak transactions, the second of a path whose first edge is between a weak and a strict transaction and includes at least one more weak transaction and the last of a path whose first edge is between a weak and a strict transaction and does not include any other weak transactions. Thus, the actual number of weak transactions that need to be undone or compensated for 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 that is aborted. Let e be the percentage of weak transactions that are exact, then $N_{roll} = e[1 - (1 - N_u/D)^{N_q}]k' N_{abort}$.

Fig. 9 depicts the probability that a weak transaction cannot be accepted because of a conflict with a strict transaction for reconciliation events occurring after varying number of transactions and for different values of the consistency factor. Fig. 10 shows the same probability for varying database sizes. More accurate estimations can be achieved for specific applications for which the access patterns of transactions are known. These results can be used to determine an appropriate reconciliation point that balances the cost of initiating reconciliations against the number of weak transactions that need to be aborted. For

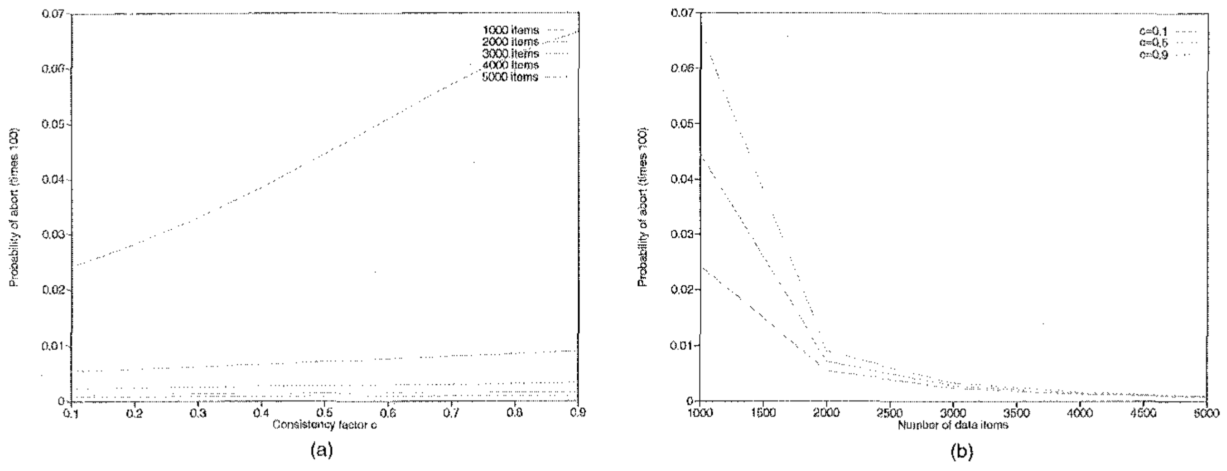


Fig. 10. Probability of abort for 60 transactions.

instance, for a given $c = 0.5$, to keep the probability below a threshold of say 0.00003, reconciliation events must take place as often as every 85 transactions (Fig. 9b).

8 Related Work

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 also causes unbearable overheads in cases of weak connectivity. The weak transaction model described in this paper was first introduced in [26], while preliminary performance results were presented in [24].

8.1 Network Partitioning

The partitioning of a database into clusters resembles the *network partition problem* [5], 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. Thus, clients may operate as physically disconnected even while remaining physically connected. Strategies for network partition face similar competing goals of availability and correctness as clustering. 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.

8.2 Read-Only Transactions

Read-only transactions do not modify the database state, thus their execution cannot lead to inconsistent database states. In our scheme, read-only transactions with weaker consistency requirements are considered a special case of weak transactions that have no write operations.

Two requirements for read-only transactions were introduced in [8]: 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 integrity 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 our 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 ignored from the IAS schedule; weak consistency, if part of a weakly correct IAS schedule; and strong consistency, if part of a strongly correct schedule. *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) [28], [29] 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 [38], 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 [15]. 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 incorporated in the proposed model to define d as the ignorance factor N of weak transactions.

8.3 Mobile Database Systems

The effect of mobility on replication schemes 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 the above requirements. The restrictive nature of one-copy serializability for mobile applications is also pointed out in [16] and a more relaxed criterion is proposed. This criterion although sufficient for 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* [35], [6], [23] 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 scheme 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 nondecreasing 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 [36].

In the *two-tier replication* scheme [9], replicated data have two versions at mobile nodes: master and tentative versions. A master version records the most recent value received while the site was connected. A tentative version records local updates. There are two types of transactions somewhat analogous to our weak and strict transactions: tentative and base transactions. A *tentative transaction* works on local tentative data and produces tentative data. A *base transaction* works only on master data and produces master data. Base transactions involve only connected sites. Upon reconnection, tentative transactions are reprocessed as base transactions. If they fail to meet some application-specific acceptance criteria, they are aborted and a message is returned to the mobile node. Our scheme extends two-tier replication in that weak connectivity is supported by employing a combination of weak and strict transactions. We have provided the foundations for the correctness of the proposed scheme as well as an evaluation of its performance.

8.4 Mobile File Systems

Coda [14] 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. Isolation-only transactions (IOTs) [17], [18] extend Coda with a new transaction service. 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 [10]. 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 [32], [33], where a weak read operation was added to a file service interface. The semantics of file 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 scheme 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. Weak operations query local, potentially inconsistent copies and perform tentative updates. The usual operations, called strict in this framework in contradistinction to weak, are also supported. Strict operations access consistent data and perform permanent updates.

Clients may operate disconnected by employing only weak operations. To accommodate weak connectivity, a mobile client selects 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. The idea of providing weak operations can be applied to other types of inter-cluster integrity constraints besides replication. Such constraints can be vertical and horizontal partitions or arithmetic constraints [31]. Another way of defining the semantics of weak operations is by exploiting the semantics of data. In [37], data are fragmented and later merged based on their object semantics.

APPENDIX

A. 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:

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

Then, the wait time by using the Pollaczek-Khinchin (P-K formula) [4] is:

$$w = \frac{\lambda E[x^2]}{2(1-\lambda E[x])}.$$

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 [34]. 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 rate and v the vacation interval, that is the duration of a disconnection.

B. DATA CONTENTION ANALYSIS

From the resource contention analysis,

$$R_{E_{strict}} = N_q R_q^s + N_u R_u^s$$

and

$$R_{E_{weak}} = N_q R_q^w + N_u R_u^w.$$

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 into 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_{op}^h (d_{op}^c) be the mean number of hot (cold) copies written by an op operation and I_{op}^c the mean number of op operations per copy. For instance, for $op = WR$ and hot copies,

$$d_{WR}^h = oh + \frac{(1-o)D_h}{D},$$

and

$$I_{WR}^h = \frac{(1-c)\lambda_q d_{WR}^h}{D_h}.$$

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,

$$P_{WR/WW} = d_{WR}^c I_{WW}^c T_W + d_{WR}^h I_{WW}^h T_W$$

and

$$P_{WR} = d_{WR}^h(I_{SW}^h T_S + I_{WW}^h T_W) + d_{WR}^c(I_{SW}^c T_S + I_{WW}^c T_W).$$

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

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

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

Let $N_W^{i_a}$ ($N_S^{i_a}$) be the mean number of weak (strict) transactions per cluster in substate i_a and $CP_{op_1/op_2}^{i_a}$ 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,

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

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:

$$s_{W_i} = (N-i)\left[\frac{N_q}{N}(a_{WR} + b_{WR}) + \frac{N_u}{N}(a_{WW} + b_{WW})\right] + T_c.$$

C. Reconciliation

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

$$\begin{aligned}p_{WW} &= \left[1 - (1 - \frac{N_u}{D})^N\right] p_c \\ p'_{WW}(m, m') &= 1 - (1 - p_{WW})^{(m'-1)} \\ p_{WS} &= 1 - \left(1 - \frac{N_q}{n(lD_h + l'D_c)}\right)^{(N_u q_W)} \\ p'_{WS}(m, m') &= 1 - (1 - p_{WS})^{m} \\ p_{SW} &= p_{WS} \\ p'_{SW}(m, m') &= 1 - (1 - p_{SW})^{m'} \\ p_{SS} &= 1 - \left(1 - \frac{N_u}{D}\right)^N \\ p'_{SS}(m, m') &= 1 - (1 - p_{SS})^{(m-1)},\end{aligned}$$

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

Acknowledgments

Bharat Bhargava was supported, in part, by the U.S. National Science Foundation Grant No. 9405931.

References

[1] R. Alonso, D. Barbara, and H. Garcia-Molina, "Data Caching Issues in an Information Retrieval System," *ACM Trans. Database Systems*, vol. 15, no. 3, pp. 359–384, Sept. 1990

[2] D. Barbará and H. Garcia-Molina, "Replicated Data Management in Mobile Environments: Anything New under the Sun?" *Proc. IFIP Conf. Applications in Parallel and Distributed Computing*, Apr. 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] S.B. Davidson, H. Garcia-Molina, and D. Skeen, "Consistency in Partitioned Networks," *ACM Computing Surveys*, vol. 17, no. 3, pp. 341–370, Sept. 1985.

[6] A. Demers, K. Petersen, M. Spreitzer, D. Terry, M. Theimer, and B. Welch, "The Bayou Architecture: Support for Data Sharing among Mobile Users," *Proc. IEEE Workshop Mobile Computing Systems and Applications*, pp. 2–7, Dec. 1994.

[7] G.H. Forman and J. Zahorjan, "The Challenges of Mobile Computing," *Computer*, vol. 27, no, 6, pp. 38–47, June 1994.

[8] H. Garcia-Molina and G. Wiederhold, "Read-Only Transactions in a Distributed Database," *ACM Trans. Database Systems*, vol. 7, no. 2, pp. 209–234, June 1982.

[9] J. Gray, P. Helland, P. O'Neil, and D. Shasha, "The Dangers of Replication and a Solution," *Proc. ACM SIGMOD Conf.*, pp. 173–182, 1996.

[10] P. Honeyman and L.B. Huston, "Communication and Consistency in Mobile File Systems," *IEEE Personal Comm.*, vol. 2, no. 6, Dec. 1995

[11] Y. Huang, P. Sistla, and O. Wolfson, "Data Replication for Mobile Computers," *Proc. 1994 SIGMOD Conf.*, pp. 13–24, May 1994.

[12] L.B. Huston and P. Honeyman, "Partially Connected Operation," *Computing Systems*, vol. 4, no. 8, Fall 1995.

[13] T. Imielinksi and B. R. Badrinath, "Wireless Mobile Computing: Challenges in Data Management," *Comm. ACM*, vol. 37, no. 10, Oct. 1994.

[14] J.J. Kistler and M. Satyanarayanan, "Disconnected Operation in the Coda File System," *ACM Trans. Computer Systems*, vol. 10, no. 1, pp. 213–225, Feb. 1992.

[15] N. Krinshnakumar and A.J. Bernstein, "Bounded Ingnorance: A Technique for Increasing Concurrency in a Replicated System," *ACM Trans. Database Systems*, vol. 19, no. 4, pp. 586–625, Dec. 1994.

[16] N. Krishnakumar and R. Jain, "Protocols for Maintaining Inventory Databases and User Profiles in Mobile Sales Applications," *Proc. Mobidata Workshop*, Oct. 1994.

[17] Q. Lu and M. Satyanarayanan, "Improving Data Consistency in Mobile Computing Using Isolation-Only Transactions," *Proc. Fifth Workshop Hot Topics in Operating Systems*, May 1995.

[18] Q. Lu and M. Satyanarayanan, "Resource Conservation in a Mobile Transaction System," *IEEE Trans. Computers*, vol. 46, no. 3, pp. 299–311, Mar. 1997.

[19] K. Miller, "Cellular Essentials for Wireless Data Transmission," *Data Comm.*, vol. 23, no. 5, pp. 61–67, Mar. 1994.

[20] L.B. Mummert, M.R. Ebling, and M. Satyanarayanan, "Exploiting Weak Connectivity for Mobile File Access," *Proc. 15th ACM Symp. Operating Systems Principles*, Dec. 1995.

[21] B.D. Noble, M. Price, and M. Satyanarayanan, "A Programming Interface for Application-Aware Adaptation in Mobile Computing," *Computing Systems*, vol. 8, no. 4, Winter 1996.

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

[23] K. Petersen, M. Spreitzer, D.B. Terry, M. Theimer, and A.J. Demers, "Flexible Update Propagation for Weakly Consistent Replication," *Proc. 17th ACM Symp. Operating Systems Principles*, pp. 288–301, 1997.

[24] E. Pitoura, "A Replication Schema to Support Weak Connectivity in Mobile Information Systems," *Proc. Seventh Int'l Conf. Database and Expert Systems Applications (DEXA '96)*, pp. 510–520, Sept. 1996.

[25] E. Pitoura and B. Bhargava, "Building Information Systems for Mobile Environments," *Proc. Third Int'l Conf. Information and Knowledge Management*, pp. 371–378, Nov. 1994.

[26] E. Pitoura and B. Bhargava, "Maintaining Consistency of Data in Mobile Distributed Environments," *Proc. 15th IEEE Int'l Conf. Distributed Computing Systems*, pp. 404–413, May 1995.

[27] E. Pitoura and G. Samaras, *Data Management for Mobile Computing*. Kluwer, 1998.

[28] C. Pu and A. Leff, "Replica Control in Distributed Systems: An Asynchronous Approach," *Proc. ACM SIGMOD*, pp. 377–386, 1991.

[29] K. Ramamritham and C. Pu, "A Formal Characterization of Epsilon Serializability," *IEEE Trans. Knowledge and Data Eng.*, vol. 7, no. 6, pp. 997–1,007, 1995.

[30] M. Satyanarayanan, J.J. Kistler, L.B. Mummert, M.R. Ebling, P. Kumar, and Q. Lu, "Experience with Disconnected Operation in a Mobile Comp uting Environment," *Proc. 1993 Usenix Symp. Mobile and Location-Independent Computing*, Aug. 1993.

[31] A. Sheth and M. Rusinkiewicz, Management of Interdependent Data: Specifying Dependency and Consistency Requirements, *Proc. Workshop Management of Replicated Data*, pp. 133–136, Nov. 1990.

[32] C.D. Tait and D. Duchamp, "Service Interface and Replica Management Algorithm for Mobile File System Clients," *Proc. First Int'l Conf. Parallel and Distributed Information Systems*, pp. 190–197, 1991.
[33] C.D. Tait and D. Duchamp, "An Efficient Variable-Consistency Replicated File Service," *Proc. Usenix File Systems Workshop*, pp. 111–126, May 1992.
[34] H. Takagi, *Queueing Analysis, Vol. 1: Vacation and Priority Systems.* North-Holland, 1991.
[35] D. Terry, A. Demers, K. Petersen, M. Spreitzer, M. Theimer, and B. Welch, "Session Guarantees for Weakly Consistent Replicated Data," *Proc. Int'l Conf. Parallel and Distributed Information Systems*, pp. 140–149, Sept. 1994.
[36] 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," *Proc. 15th ACM Symp. Operating Systems Principles*, Dec. 1995.
[37] G. Walborn and P. K. Chrysanthis, "Supporting Semantics-Based Transaction Processing in Mobile Database Applications," *Proc. 14th Symp. Reliable Distributed Systems*, Sept. 1995.
[38] M.H. Wong and D. Agrawal, "Tolerating Bounded Inconsistency for Increasing Concurrency in Database Systems," *Proc. 11th ACM Symp. Principles of Database Systems (PODS)*, pp. 236–245, 1992.
[39] P.S Yu, D.M. Dias, and S.S Lavenberg, "On the Analytical Modeling of Database Concurrency Control," *J. ACM* vol. 40, no. 4, pp. 831–872, Sept. 1993.

Evaggelia Pitoura received her diploma from the Department of Computer Science and Engineering of the University of Patras, Greece, in 1990; and her MSc and PhD degrees in computer science from Purdue University in 1993 and 1995, respectively. Since September 1995, she has been with the Department of Computer Science at the University of Ioannina, Greece. Her research interests include database issues in mobile computing, heterogeneous databases, and distributed systems. Her publications in the above areas include several journal and conference proceedings articles, plus a recently published book on mobile computing. She is a member of the IEEE Computer Society.

Bharat Bhargava is currently a professor in the Department of Computer Science at Purdue University. His research involves both theoretical and experimental studies in distributed systems. His research group has implemented a robust and adaptable distributed database system, called RAID, to conduct experiments in replication control, checkpointing, and communications. He has conducted experiments in large-scale distributed systems, communications, and overheads in implementing object support on top of the relational model. He has recently developed an adaptable video conferencing system using the NV system from Xerox PARC. He is conducting experiments with research issues in large-scale communication networks to support emerging applications such as digital libraries and multimedia databases. He chaired the IEEE Symposium on Reliable and Distributed Systems that was held at Purdue in 1998, and he is on the editorial board of three international journals. At the 1988 IEEE Data Engineering Conference, he and John Riedl received the best paper award for their work on "A Model for Adaptable Systems for Transaction Processing." He is a fellow of the IEEE and Institute of Electronics and Telecommunication Engineers. He has been awarded the Gold Core member distinction by the IEEE Computer Society for his distinguished service. He received the Outstanding Instructor award from the Purdue chapter of the ACM.