A Replication Schema to Support Weak Connectivity in Mobile Information Systems *

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

Abstract

We present a replication schema appropriate for distributed environments where connectivity is partial, weak, and variant such as in mobile information systems. The schema is based on augmenting the database interface with operations with weaker consistency guarantees. A implementation of the schema is presented by distinguishing copies into quasi and core and protocols for enforcing the schema are introduced. Then, some practical examples of its applicability are described. The performance of the weak consistency schema is evaluated for various networking conditions by using an analytical model developed for this purpose.

1 Introduction

Advances in telecommunications and in the development of portable computers have provided for wireless communications that permit users to actively participate in distributed computing even while moving. The resulting distributed environment is subject to restrictions imposed by the nature of the wireless medium [8, 10, 19, 20].

Mobile computing is susceptible to frequent network disconnections. Besides an increased number of disconnections resulting from site or communication failures, certain other disconnections such as those caused by battery limitations or handoffs are considered predictable. Frequently, predictable disconnections are voluntary since users deliberately avoid use of the wireless links to reduce cost and power consumption and to overcome availability and latency barriers. Thus, most mobile clients are only *occasionally* connected, switching between connected and disconnected modes of operation. Similar considerations are applicable to portable computing. Disconnections have been discussed

*University of Ioannina, Computer Science Department, Techical Report No: 96-08. A shorter version will appear in the 7th International Conference on Database and Expert Systems Applications, September 1996

extensively in the context of network partition [6]. The frequency of their occurrence in portable computing, however, forces making them part of normal operation and considering a new mode of operation called *disconnected operation* [10, 11, 25, 23]. Another characteristic of mobile distributed computing is weak connectivity. Wireless networks deliver much lower bandwidth than wired networks and have higher error rates [8]. Finally, mobile systems are characterized by high variation in network bandwidth, that can shift one to four orders of magnitude, depending on whether the host is plugged in or using wireless access and on the type of its current connection [8, 10].

Mobile users will desire access to private or corporate databases that will be stored at mobile as well as static hosts and queried and updated over the wired and the wireless network. For instance, insurance agents may interact through their mobile station with a database storing consumer records, while traveling salespersons may access inventory databases. Requirements for database use through wireless communications stem also from personal computing applications. Accessing databases is necessary for instance for consumers to purchase goods, get traffic information, or make travel plans while using their notebooks. These databases, for reasons of reliability, performance, and cost will be distributed and replicated over many sites.

In this paper, we propose a replication schema appropriate for environments where connectivity is partial, weak and variant such as in mobile computing. Instead of requiring mutual consistency of all copies of a data item we allow bounded inconsistencies. Specifically, all data located at strongly connected hosts are grouped together to form a cluster. While all data inside a cluster are consistent, degrees of inconsistency are defined for replicas at different clusters. The proposed mechanism enchances the interface offered by database systems with operations with weaker consistency guarantees which allow access to data that exhibit bounded inconsistency. It is the responsibility of the individual application to decide between weak and strict operations so that bandwidth is utilized. In addition, users can still operate even when disconnected by accessing local data. Finally, the degree of inconsistency can be tuned based on the networking conditions.

The weak consistency replication schema was first introduced in [21], where its theoretical formulation was presented. In this paper, we focus on practical aspects of the model. First, we refine and clarify the model and present an implementation based on distinguishing copies into core and quasi. Then, protocols are introduced for realizing the proposed schema. We also describe some practical examples of its use. An analytical model is developed to evaluate the performance of the schema. The model is used to study the effect of weakening consistency on the system performance. The performance parameters considered are the system throughput, the number of messages, and the response time. The study is performed for a range of networking conditions, that is for different values of bandwidth and for varying disconnection intervals.

The remainder of this paper is organized as follows. Section 2 introduces the replication schema. In Section 3, we present protocols for implementing the schema along with some practical scenarios for the use of the model. In Section 4, we develop an analytical model, and in Section 5, the model is used to evaluate the performance of the schema. In Section 6, we compare our work with related research. Finally, concluding remarks are offered in Section 7.

2 A Weak Consistency Model

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

2.1 Consistency Units

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

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

The definition of *d-degree consistency* for an integrity constraint depends on its type. In this paper, we focus on replication constraints, where copies x_i of the same data item x are expected to have the same value. The d-degree relation bounds the divergence among the values of the copies of the data item. This divergence may be expressed in terms of the time lag or fuzziness of value of the copies in deviating from mutual consistency [24, 1]. The values of the data copies are occasionally reconciled to obtain a mutual consistent value. Reconciliation can be performed for instance periodically, at times of higher network availability, or when required for keeping the divergence inside the d-bound. The degree may vary based on the availability of network bandwidth by allowing little deviation in instances of higher bandwidth availability and high deviation in instances of low bandwidth availability. Thus, bounded inconsistency makes applications able to adjust to the limitations of the communication environment by providing users with data of variable level of detail or quality. The cluster configuration is dynamic. By taking advantage of the predictable nature of disconnections, clusters of data may be explicitly created or merged upon a forthcoming disconnection or connection of the associated mobile host. To accommodate migrating locality, a mobile host can also move to a different cluster when it enters a new environment.

2.2 Weak and Strict Transactions

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

Definition 2 (transaction) *A transaction (T) is a partial order (OP, $<$), where OP is the set of weak or strict read, weak or strict write, abort and commit operations executed by the transaction, and $<$ represents their execution order. The partial order must specify the order of conflicting data operations and contains exactly one abort or commit operation which is the last in the order. Two weak (strict) data operations* conflict *if they access the same copy of a data item and at least one of them is a weak (strict) write operation.*

Two types of transactions are supported, weak and strict. A *weak* transaction (WT) is a transaction where OP does not include strict operations. A *strict* transaction (ST) is a transaction where OP does not include weak operations. Weak transactions access data copies that belong to the same cluster and thus are local at that cluster. User transactions are decomposed into a number of weak and strict subtransactions units according to the degree of consistency required by the application.

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

Implementation. To implement this schema we distinguish the copies of a data item into two broad categories: core and quasi copies. *Core* copies are copies whose values are up-to-date and permanent, while *quasi* copies are copies whose values may be obsolete and are only conditionally committed. To process the operations of a transaction, the database management system translates operations on data items into operations on the copies of those data items. In general, strict transactions, except of occasional updates of quasi copies, access only core copies and weak transactions operate on local, quasi or core, copies. We formalize this procedure by a *translation function h*. Function h maps each $SR[x]$ operation into a number of read operations on core copies of x and each $WW[x]$ operation into a number of write operations of local quasi copies of x. Depending on the translation of a weak read operation we define two types of translation functions: a *best-effort* translation function maps each $WR(x)$ operation into a number of read operations on the most up-to-date locally available core or quasi copies of x, and a *conservative* translation function maps each weak read into a number of read operations only on locally available quasi copies. In addition, based on the time of propagation of updates of core copies to quasi copies, we define two types of translation function: an *eventual* translation function maps a $SW[x]$ into writes of only core copies, while an *immediate* translation function also updates the quasi copies of the corresponding cluster. For an immediate h, conservative and best-effort have the same result. How many and which core or quasi copies are actually read or written when a database operation is issued on a data item depends on the coherency algorithm used, e.g, quorum consensus, ROWA. Without loss of generality, we assume that there is only one quasi copy per cluster. This assumption can be easily lifted but with significant complication in notation. Since all quasi copies in a cluster have the same value, this single copy can be considered to be their representative.

2.3 Weak Connectivity Operation

A complete intracluster schedule, IAS, is an observation of an interleaved execution of transactions in a given cluster configuration that includes locally committed weak transactions and (globally) committed strict transactions. For a formal definition, see [21]. Given a schedule S, the *projection* *of S on strict transactions* is the schedule obtained from S by deleting all weak operations, and the *projection of S on a cluster Cl_k* is the schedule obtained from S by deleting all operations of S that do not access Cl_k. A correct concurrent execution of weak and strict transactions must maintain d-consistency among clusters and strict consistency inside each cluster. Specifically, an intracluster schedule is weakly correct iff (1) all transactions have a consistent view, i.e., all constraints that can be evaluated using the data read are valid, (2) replication is hidden from strict transactions, and (3) the d-degree relation is maintained. Weak correctness ensures that each transaction gets a consistent view, but it does not ensure that weak transactions at different clusters get the same view. The following theorem provides a serializability-based criteria that also provides for this condition.

Theorem 1 *Given that $d-$degree is maintained, an intracluster schedule S is correct if its projection on strict transactions is equivalent to an one-copy (1C) schedule S_{1C}, and each of its projections on a cluster Cl_i is conflict-equivalent to a serial schedule S_{S_i} such that the order of transactions in S_{S_i} is consistent with the order of transactions in S_{1C}.*

Protocols for maintaining the d-degree relation are given in Section 3. The proposed schema is flexible, in the sense that any coherency control method that guarantees one-copy serializability can be used for synchronizing core copies. The schema reduces to one-copy serializability when only strict transactions are employed.

Graph characterization. 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 represent conflicts between strict transactions, we construct a replicated data Serialization Graph (SG). An SG [3] is a serialization graph augmented with additional edges to take into account the fact that operations on different copies of the same data item may also cause conflicts. Acyclicity of the SG implies one-copy serializability of the corresponding schedule. We augment the SG with additional edges to represent conflicts between weak transactions in the same cluster and conflicts between weak and strict transactions. We add an edge $T_i \rightarrow T_j$ between two transactions T_i and T_j in IAS, if some operation $op_i \in T_i$ conflicts with an operation $op_j \in T_j$. A *dependency edge* is an edge that represents the fact that a transaction reads a value produced by another transaction. A *precedence edge* is an edge that represent the fact that a transaction reads a value that is later changed by another transaction. It can be shown, that if an intracluster schedule has an acyclic IASG then it is strongly correct.

2.4 Reconciliation of Core and Quasi Copies

Approaches to reconciling core and quasi copies vary from purely syntactic to purely semantic ones [6]. We adopt a purely syntactic application-independent approach. Our correctness criterion is based on the following principle: if a core copy is written, and a strict transaction has read it, the value of the core copy must be the value selected. Otherwise, the value of any quasi copy can be selected. A (complete) intercluster schedule, IES, models execution after reconciliation, when global transaction should become aware of local writes, i.e., local transaction become globally committed. To capture this, in the IAS, we add conflicts between weak and strict operations, specifically we ask that in addition to maintaining the same order as in the IAS, each pair of weak write and strict read operations on the same item and each pair of weak write and strict write on the same item are also ordered. In an IES schedule, a strict transaction *reads-x-from* the last transaction in the schedule that wrote any copy (core or quasi) of x. We accept as many weak writes as possible without violating the one-copy serializability of strict transactions. Specifically, an intercluster schedule is correct iff (1) it is based on a correct IAS schedule S_{IAS}, and (2) strict transactions have the same reads-from relation as in the S_{IAS}.

Graph characterization To determine correct IES schedules, we construct an *intercluster serialization graph* (IESG) by augmenting the serialization graph of the underlying intracluster schedule. In the IASG graph, transactions that access different copies of the same item do not conflict. To force such conflicts, we induce

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

It can be shown that if an intercluster schedule based on an IAS has an acyclic IESG, then it is correct.

3 Weak Consistency in Practice

3.1 Protocols

Serializability. We distinguish between protocols for coherency and protocols for concurrency control. Coherency control refers to ensuring that all copies of a data item have the same value, here

	WR	WW	SR	SW
WR	X		X	
WW			X	X
SR	X	X	X	
SW		X		

Table 1: Lock compatibility matrix. A X entry indicates that the lock modes are compatible.

we must maintain this property globally for core and locally for quasi copies. Concurrency control refers to maintaining consistency of the other integrity constraints, here the intracluster constraints. For coherency control, we adopt a quorum-based schema [3]. Each strict transaction reads q_r core copies and writes q_w core copies per strict read and write operation. The values of q_r, and q_w for a data item x are such that $q_r + q_w > n_d$, where n_d is the number of available core copies of x. To maintain correctness we employ *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. The compatibility of locks is defined based on the conflict relation, where two lock modes are compatible if the corresponding operations do not conflict. The lock compatibility for an eventual and best-effort h is depicted in Table 1.

Degree Maintenance. At a 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. Therefore, the degree can be bounded by either limiting the number of weak writes pending commitment or controlling the h function. We outline below some possible ways to define the degree and corresponding methods for enforcing them. (a) If the degree is expressed as the number of transactions that are allowed to read inconsistent data then we bound the number of weak transactions at each cluster. (b) If the degree d is bounded by limiting the number of versions, that is if a weak transaction at a cluster must read only d-version old data, we define h such that a strict write modifies quasi copies at the cluster at least every d updates. This definition of degree cannot be ensured for involuntarily disconnected clusters, since there is no way of notifying them for remote updates of core copies. (c) If the value of a data item is arithmetic and the degree is expressed as a range d of acceptable values that each copy of a data item can take, then only weak writes with values in the allowable bounds are accepted. (d) If the degree d is defined as the number of data items that are allowed to diverge, we bound the number of quasi copies that are allowed to diverge from the core copies to d. This is accomplished by allowing weak reads and writes only on d data items. (e) Finally, if the degree d is defined as the number of data copies that are allowed to diverge, we bound the number of quasi copies that are allowed to diverge at each cluster so that the total number of quasi copies that differ from core copies in all clusters is d. This is achieved by bounding appropriately the number of weak writes at each cluster.

Reconciliation. To get a correct schedule we need to break potential cycles in the IES graph. Since to construct the IES we start from an acyclic graph and add edges between a weak and a strict transaction, there is always at least one weak transaction in each cycle. We rollback such weak transactions. Undoing a transaction normally results in *cascading aborts*, that is, in aborting transactions which have read the values written by that transaction; that is, transactions that are related with a dependency edge to the transaction undone. Since weak transactions write only quasi copies in a cluster, and since only weak transactions in the same cluster can read these quasi copies we get the following lemma:

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

The above lemma ensures that only weak transactions in the same cluster may need to be aborted when a weak transaction is aborted to resolve conflicts in an intercluster schedule. In practice, fewer transactions ever need to be aborted. In particular, we need to abort only transactions whose output depends on the exact values of the data items they read. We call these transactions *exact*. Many weak transactions do not naturally fall in this category, since by definition, weak transactions are transactions that read local d-consistent data. Thus even if the value they read was produced by a transaction that was later aborted, this value was inside an acceptable range of inconsistency and this may be sufficient to guarantee their correctness.

Detecting cycles in the IEG can be hard. The difficulties raise from the fact that between transactions that wrote a data item an edge can have any direction, thus resulting in *polygraphs* [17]. Polynomial tests for acyclicity are possible, if we made the assumption that transactions read a data item before writing it. Then, to get the IES graph from the IAS we need only:

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

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

Table 2: The algorithm for reconciliation.

Table 2 outlines the reconciliation steps.

3.2 Examples

In the following, we give some examples illustrative of the applicability of the weak/strict operation interface.

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

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

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

4 Performance Model

To evaluate the proposed schema and quantify the improvement in performance attained by sacrificing strict consistency in weakly connected environments, we have developed an analytical model. The analysis follows an iteration-based methodology for coupling standard hardware resource and data contention as in [30]. 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 changing locality, (b) two different transaction types, weak and strict, and (c) various types of data access, as indicated by the compatibility matrix of Table 1. An innovative feature of the analysis is the employment of a vacation system to model disconnections of the wireless medium.

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

Let c be the *consistency factor* of the application under consideration, that is c is the fraction of the arrived operations that are strict. To model hotspots, we divide data at each cluster into hot and cold data sets. To capture *locality* we assume that a fraction o of the transactions exhibit locality, that is they access data from the hot set with probability h and data from the cold set with probability $1 - h$. The remaining transactions access hot and cold data uniformly. Due to mobility o may diminish with time. Locality is taken into consideration by the replication schema, by assuming that the probability that a hot data has a core copy at a cluster is l, and that a cold data has a core copy is l', where normally, $l' < l$.

For simplicity, we assume that there is one quasi copy of each data item at each cluster. Let q_r be the read and q_w the write quorum, then the mean number of operations on data copies per strict transaction, N_S, is equal to $N_u q_w + N_q q_r$. In the following let x be the probability that an operation at a cluster access data for which a core copy exists at that cluster. The transaction model consists of

$n_L + 2$ states, where n_L is the random variable of items accessed by the transaction and N_L its mean. Without loss of generality, we assume that N_L is equal to the number of operations. The transaction has an initial setup phase, state 0. Then, it progress to states 1,2, ..., n_L in that order. If successful, at the end of state n_L the transaction enters into the commit phase at state n_{L+1}. The transaction response time r_{trans} can be expressed as

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

where n_w is the number of lock waits during the run of the transaction, r_{w_j} is the waiting time for the jth lock contention, r_E is the sum of the execution times in states 1,2, ..., n_L excluding lock waiting times, r_{INLP} is the execution time in state 0, and t_{commit} is the commit time to reflect the updates in the database. We use lower case letters to represent random variables and upper case letters to represent their corresponding means.

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

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

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

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

Execution time. For simplicity, we ignore the overhead of communications inside a cluster by assuming either that each cluster consists of a single node or that the communication among the nodes inside a cluster is fast. Without taking into account data contention, the average response time for a weak read on a data item is $R_q^w = w + t_q$ and for a weak update $R_u^w = w + t_u$, where w is the average wait time at each cluster. Similar formulas hold for the response time R_q^s and R_u^s of the strict transactions. The computation of w is given in the appendix.

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

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

4.2 Data contention analysis

We assume that for maintaining consistency we use a strict 2PL schema where a transaction release its locks only after commitment. In the following, *op* stands for one of WR, WW, SR, SW. Using the formula from the previous section the response time for strict and weak transaction is:

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

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

where P_{op} is the probability that a transaction contents for an *op* operation on a data copy, and R_{op} is the average time spent waiting to get an *op* lock given that lock contention occurs. P_q and P_u are respectively the probability that at least one operation on a data copy per strict read or write conflicts. Specifically, $P_q = 1 - (1 - P_{SR})^{q_r}$ and $P_u = 1 - (1 - P_{SW})^{q_w}$. An outline of the estimation of P_{op} and R_{op} is given in the appendix, for a detailed description of the model see [18].

5 Performance Evaluation

Table 3 shows some realistic values for the various parameters. For the bandwidth the values depend 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 [8]. We assume an eventual and best effort h.

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

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

Table 3: Model parameters.

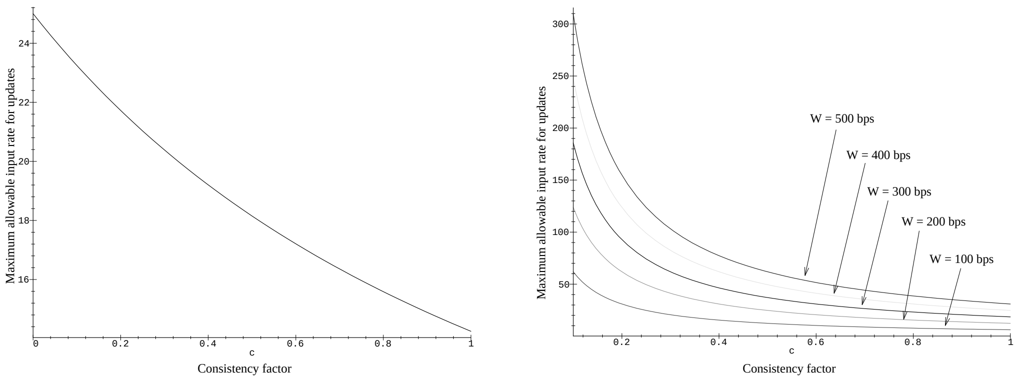

Figure 1: (left) Maximum allowable input rate for queries for various values of the consistency factor. Limits imposed by the processing rate at each cluster ($\lambda \leq E[X]$). (right) Maximum allowable input rate for updates for various values of the consistency factor. Limits imposed by bandwidth restrictions ($\lambda_r \leq t_r$).

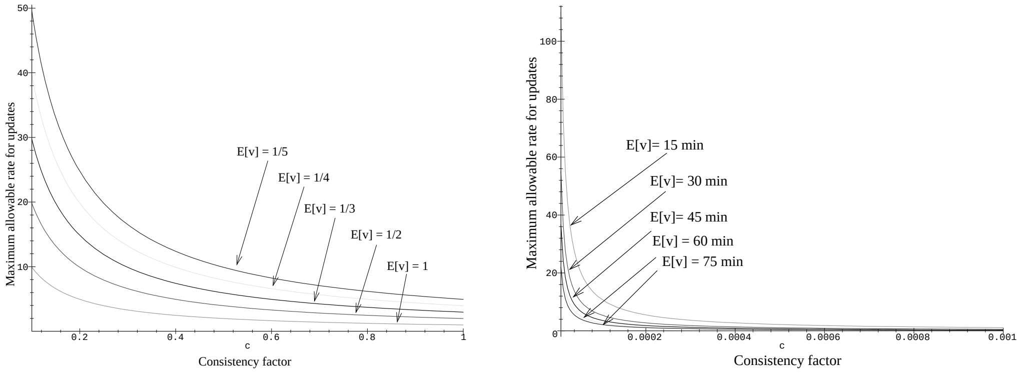

Figure 2: Maximum allowable input rate for updates for various values of the consistency factor. Limits imposed by disconnections and their duration ($\lambda_r \leq E[V]$).

System throughput. Figures 1(left), 1(right), and 2 show how the maximum transaction input, or system throughput, is bounded by the processing time, the available bandwidth, and the disconnection intervals respectively. We assume that queries are four times more common than updates $\lambda_q = 4\,\lambda_u$. As shown in Figure 1(left), the allowable input rate when all transactions are weak ($c = 0$) is almost double the rate when all transactions are strict (strict consistency, $c = 1$). This is the result of the increase in the workload with c caused by the fact that strict operations on data items may be translated into more than one operation on data copies. The percentage of weak transactions can be effectively tuned to attain the desired throughput based on the networking conditions such as the duration of disconnections and the available bandwidth. As indicated in Figure 1(right), to get for instance, the same throughput with 100bps as with 500bps and $c = 1$ we must lower the consistency factor below 0.1. The duration of disconnections may vary from seconds when they are caused by handoffs ([15]) to minutes for instance when they are voluntary. Figure 2 depicts the effect of the duration of a disconnection on the system throughput for both short durations (Figure 2(left)) and longer ones (Figure 2(right)). For long disconnections (Figure 2(right)), only a very small percentage of strict transactions can be processed. To keep the throughput comparable to that for shorter disconnections (Figure 2(left)) the consistency factor must drop at around three orders of magnitude.

Communication cost. We estimate the communication cost by the number of messages sent. The number of messages depends on the following parameters of the replication schema: (1) the consistency factor c, (2) the data distribution l for hot and l' for cold data, (3) the locality factor o and (4) the quorums, q_r and q_w, of the coherency schema. We assume a ROWA schema ($q_r = 1$, $q_w = n_d$) if not otherwise stated. As shown in Figure 3(left) the number of messages increases linearly with the consistency factor. As expected the number of messages decreases with the percentage of transactions

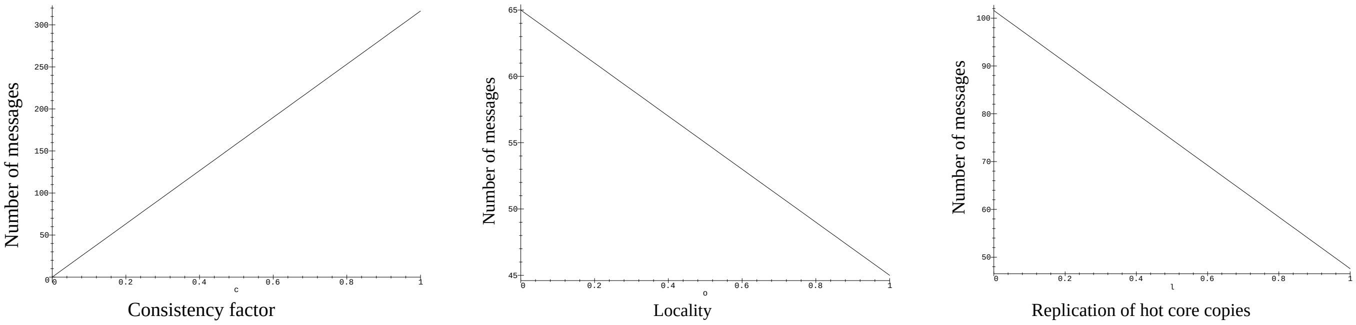

Figure 3: (left) Number of messages for various values of c. (middle) Number of messages with locality (right) Number of messages for different replication of cold core copies.

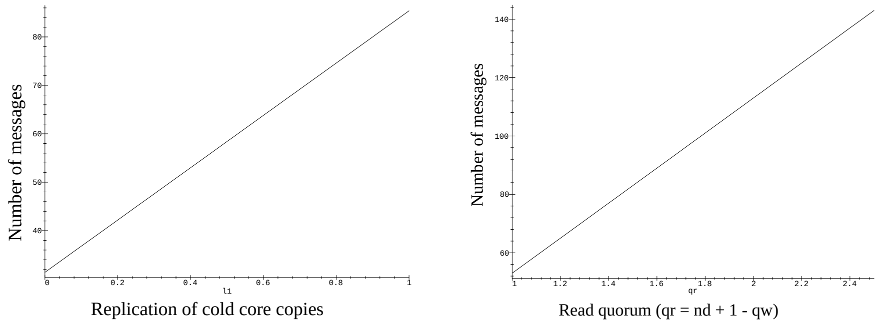

Figure 4: (left) Number of messages for different replication of hot core copies. (right) Number of messages for different values of the read quorum.

that access hot data, since then local copies are more frequently available (Figure 3(middle)). Thus the communication overhead in mobile computing is increased due to the frequently changing locality. The number of messages decreases when the replication factor of hot core copies increases (Figure 3(right)). The decrease is more dramatic since most operations are queries and the coherency schema is ROWA, thus for most operations no messages are sent. On the contrary, the number of messages increases with the replication factor of cold core copies because of additional writes caused by coherency control (Figure 4(left)). Finally, since most operations are queries keeping the read quorum smaller than the write quorum decreases the number of messages (Figure 4(right)).

Transaction response time. The response time for weak and strict transactions for various values of c is depicted in Figure 5. The larger values of response times are for 200bps bandwidth, while faster response times are the result of higher network availability set at 2Mbps. The values for the other input parameters are as indicated in Table 3. The additional parameters are set as follows. The locality parameters are $o = 0.9$ and $h = 0.9$. The data replication parameters are $l' = 0.2$ and $l = 0.8$. The disconnection parameters are $p = 0.1$ and the vacation intervals are exponentially

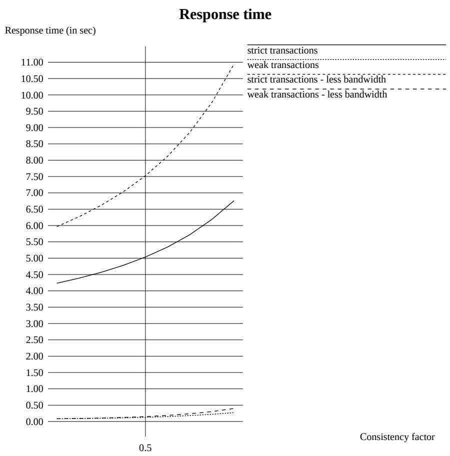

Figure 5: Comparison of the response times for weak and strict transactions for various values of the consistency factor.

distributed with $E[V] = 1/5$ sec, to model disconnection intervals that correspond to short involuntary disconnections such as those caused by handoffs [15]. The coherency control schema is ROWA. The latency of weak transactions is about 50 times greater than that of strict transactions. However, there is a trade-off involved in using weak transactions, since weak updates may be aborted later. The time to propagate updates during reconciliation is not counted. As c increases the response time for both weak and strict transactions increase since more conflicts occur. Figure 6(left) and Figure 6(right) show the response time distribution for strict and weak transactions respectively for 2Mbps bandwidth. For strict transactions, the most important overhead is network transmission. All times increase as c increases. For weak transactions, the increase in the response time is the result of longer waits for acquiring locks, since weak transactions that want to read up-to-date data conflict with strict transactions that write them.

6 Related Work

General weak consistency schemas. The partitioning of a database into clusters resembles the *network partition problem* [6], where site or link failures fragment a network of database sites into isolated subnetworks called partitions. Clustering is conceptually different than partitioning in that it is electively done to increase performance. Furthermore, whereas all partitions are isolated, clusters may be partly connected. Strategies for network partition 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 do-

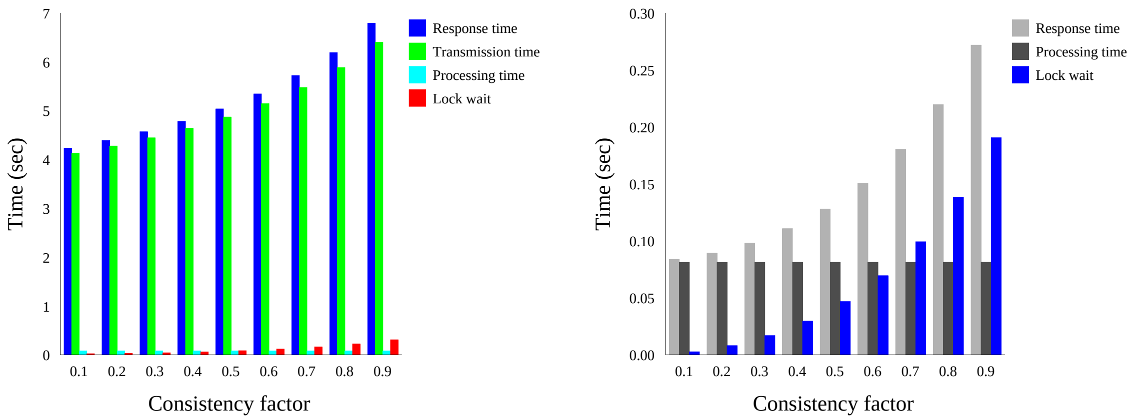

Figure 6: (left) Response time distribution for strict transactions. (right) Response time distribution for weak transactions.

ing. 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.

Read-only transactions [9] do not modify the database state, thus their execution cannot lead to inconsistent database states. In the proposed schema read-only transactions with weaker consistency requirements are considered a special case of weak transactions. *Epsilon-serializability* (ESR) [22] 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 [29] a generalization of ESR was proposed for high-level type specific operations on abstract data types. In contrast, our approach deals with low-level read and write operations.

Mobile information Systems. The effect of mobility on replication schemas is discussed in [2]. The need for the management of cached copies to be tuned according to the available bandwidth and the currency requirements of the applications is stressed. In this respect, d-degree consistency and weak transactions realize both of the above requirements. The restrictive nature of one-copy serializability for mobile applications is also pointed out in [12] and a more relaxed criterion is proposed. This criterion although sufficient for a specific kind of data typical of sales applications is not appropriate for general application and distinguishable data. Furthermore, the criterion does not support any form of adaptability to the current network conditions.

The *Bayou system* [7, 28] is a platform of replicated highly available, variable-consistency, mobile databases on which to build collaborative applications. A read-any/write-any weakly-consistent replication schema is employed. Each Bayou database has one distinguished server, the primary, which is responsible for committing writes. The other secondary servers tentatively accept writes and propagate them towards the primary. Each server maintains two views of the database: a copy that only reflects committed data and another full copy that also reflects tentative writes currently known to the server. Applications may choose between committed and tentative data. Tentative data are similar to our quasi data, and committed data similar to core data. Correctness is defined in terms of session, rather than on serializability as in our model. A *session* is an abstraction for the sequence of read and writes of an application.

The *Coda* file system [11] 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. [13, 14] extend Coda with a new transaction service called *isolation-only transactions* (IOT). IOTs are sequences of file accesses that unlike traditional transactions have only the isolation property. IOTs do not guarantee failure atomicity and only conditionally guarantee permanence. IOTs are similar to weak transactions.

The idea of using different kinds of operations to access data is also adopted in [5, 26], where a weak read operation was added to a *file service interface*. The semantics of operations are different in that no weak write is provided and since there is no transaction support, the correctness criterion is not based on one-copy serializability.

7 Conclusions

To overcome bandwidth, cost, and latency barriers, clients of mobile information systems switch between connected and disconnected modes of operation. In this paper, we propose a replication schema appropriate for such operation. The schema is based on extending the database interface with weak operations. Mobile clients can operate even when disconnected using weak operations. Bandwidth can by utilized by deliberately using weak transactions. In addition, the degree of consistency can be appropriately tuned to achieve the desired system performance. An implementation of the schema was presented based on distinguishing copies into core and quasi and then protocols were presented for enforcing it. The performance of the schema was evaluated for various connectivity conditions using an analytical model.

Weak operations offer what is termed *application-aware adaptation* [16]. Application-aware adaptation characterizes the design space between two extremes. At one extreme, adaptivity is entirely the responsibility of the application, that is there is no system support or any standard way of providing adaptivity. At the other extreme, adaptivity is subsumed by the system, here the database management system. In general, the system is not aware of the application semantics and cannot provide an 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.

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[20] E. Pitoura and B. Bhargava. Revising Transaction Concepts for Mobile Environments. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, pages 164–168, Santa Cruz, California, December 1994.

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

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

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

[24] A. Sheth and M. Rusinkiewicz. Management of Interdependent Data: Specifying Dependency and Consistency Requirements. In *Proceedings of the Workshop on the Management of Replicated Data*, pages 133–136, Houston, Texas, November 1990.

[25] P. D. Skopp and G. E. Kaiser. Disconnected Operation in a Multi-User Software Development Environment. In *Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems*, Princeton, New Jersey, October 1993.

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

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

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

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

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

Appendix

Computation of 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 $\theta_4 = t_q$. The formulas the corresponding arrival rates and service times for updates queries are anologous. 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:

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

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

Transmission waiting time. We consider a nonexhaustive vacation system where after the end of each service the server takes a vacation with probability $1-p$ or continues service with probability p. This is called a queue system with Bernoulli scheduling [27]. 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.

Coupling resource and data contention.

From the resource contention analysis,

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

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

Let d^h_{op} (d^c_{op}) be the mean number of hot (cold) copies written by an *op* operation and I^c_{SW} the mean number of *op* operations per copy. Given a mean lock holding time of T_W (T_S) for weak (strict) transactions and assuming that the lock request times are a poisson process, the probability of contention on a lock request for a copy equals the lock utilization. Let P_{op_1/op_2} stand for the probability that an op_1-lock request conflicts with an op_2-lock request, then for example

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

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

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

Then $T_W = \frac{G_W}{N}$. Similar formulas hold for G_S and T_S.

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

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

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