A Probabilistic Analysis of Snapshot Isolation with Partial Replication [*]

Josep M. Bernabé-Gisbert
Francesc D. Muñoz-Escoí
Instituto Tecnológico de Informática,
46022 Valencia, Spain
{jbgisber, fmunyoz}@iti.upv.es

Vaidė Zuikevičiūtė
Fernando Pedone
University of Lugano,
CH-6904 Lugano, Switzerland
{vaide.zuikeviciute, fernando.pedone}@lu.unisi.ch

Abstract

Snapshot isolation has received a considerable amount of attention in the context of full database replication. Such popularity is mainly because read-only transactions executing under snapshot isolation are never blocked or aborted. In partial replication, where each replica holds only a part of the database, transactions may require access to remote databases. Each remote read operation of the transaction must execute in a consistent global database snapshot as the local operations; if such a snapshot is not available, the transaction must be aborted.

In this paper we are interested in the effects of distributed transactions on the abort rate of partially replicated snapshot isolation systems. We present a simple probabilistic analysis of transaction abort rates for two different concurrency control mechanisms: lock- and version-based. The former models the behavior of a replication protocol providing one-copy-serializability; the latter models snapshot isolation. Our analysis reveals that in the version-based system the execution abort rate decreases exponentially as the number of data versions available increases. As a consequence, in all cases considered, two versions of each data item were sufficient to eliminate aborts due to distributed transactions.

1. Introduction

Most work on database replication using group communication concentrates on full replication strategies. However, scalability of such protocols is limited under update-intensive workloads: Each replica added to the system allows to submit more transactions; if such transactions modify the database, they will add load to every individual database. To improve the scalability of the system, databases can be replicated partially only. Unfortunately, it is not obvious how to extend many of the protocols developed for full replication to systems with partial replication. If each replica keeps only part of the database, a transaction may require access to data stored on remote replicas and thus, a distributed execution involving more than one replica becomes necessary.

The problems introduced by distributed transactions in partially-replicated systems differ depending on the concurrency control mechanism used. In lock-based systems transactions executing over multiple replicas will acquire locks on remote data items and thus, may increase the likelihood of distributed deadlocks [11], a problem that group communication protocols can mitigate [1]. In version-based systems both local and remote read operations of each transaction must execute in a consistent global database snapshot. However, obtaining the requested snapshot for remote reads may be a challenge in some contexts (e.g., middleware approaches based on standard off-the-shelf databases).

The majority of partial replication solutions in the literature guarantee one-copy serializability (e.g., [5, 6, 7, 10, 12, 13, 15, 17]) and assume that database replicas adopt lock-based concurrency control. Several of these protocols (e.g., [6, 7, 15, 17]) build on the strong assumption that transactions can always execute locally at one database site. Such an assumption requires prior knowledge of the workload and a very precise data distribution over the replicas (e.g., [8]) or at least a single replica that holds the whole database (e.g., [6]). The protocols in [2, 16] allow distributed transactions by ensuring that all remote accesses are able to find the required data version. The work in [2] orders the beginnings of distributed transactions at all the replicas involved and, thus, prevents local executions. The authors of [16] propose to use special "dummy" transactions, created at all database sites every time an update transaction commits. All remote operations must execute within the dummy transaction associated with the required database snapshot. Maintaining a high number of dummy transactions may affect the performance of the system.

[*]This work was partially supported by FEDER/Spanish MEC TIN2006-14738-C02-01, FEDER/IMPIVA IMIDIC/2007/68 and EU ESF/IMPIVA IMAETB/2007/30 grants.

In this paper we are interested in the effects of distributed transactions in partially replicated database systems. Both distributed deadlocks and failed remote read operations result in aborted transactions. Hence, we introduce a probabilistic model for abort rates of partially replicated systems when lock- and version-based concurrency control mechanisms are used. Our study revealed that the abort rate of transactions in the execution phase in a version-based system decreases exponentially with the number of data versions available. As a consequence, in all cases considered, two versions of each data item were sufficient to eliminate the aborts due to distributed transactions. Furthermore, in the version-based system even if the workload over the partially replicated system is dominated by read-only transactions, but the few update transactions perform a lot of write operations, read-only distributed transactions can still suffer from a noticeable number of aborts. This is in contrast to typical fully replicated version-based systems, in which the number of versions available in each replica is unbounded, and thus, read-only transactions never abort.

Summing up, this paper makes the following contributions: (a) we introduce a probabilistic analysis of the abort rates of partially replicated systems when lock- and version-based concurrency control mechanisms are used; (b) we show how the number of data versions available affect the abort rate of the version-based system; (c) we identify the settings under which snapshot isolation can be safely used with partial replication.

The remainder of this paper is organized as follows. In Section 2 we introduce the system and replication models considered. Section 3 describes our probabilistic analysis of abort rate of lock- and version-based systems. We discuss the results of the analytical evaluation in Section 4 and present final conclusions in Section 5.

2. System model

2.1. Sites and communication

We consider an asynchronous distributed system composed of a set of database sites. Sites communicate through message passing and do not have access to a shared memory or a global clock. Sites may fail by crashing, but do not behave maliciously. We also assume the existence of a total-order multicast oracle — the implementation of such an oracle requires additional assumptions about our system, but this is out of the scope of this paper. Total-order multicast allows messages to be sent to a subset of database sites in the system and guarantees that (a) if a database site delivers a message m then every site in the subset delivers m; (b) no two database sites deliver any two messages in different orders; and (c) if a site broadcasts message m and does not fail, then every concerned site eventually delivers m.

2.2. Database and transactions

A database is a set of data items. Database sites have a partial copy of the database. No database site is expected to store the whole set of items, although that is not forbidden. A transaction T_i is a sequence of read and write operations on data items followed by a commit or an abort operation. A transaction is called read-only if it does not contain any write operation; otherwise it is called an update transaction. The transaction's readset and writeset identify the data items read and written by the transaction, denoted as rs and ws, respectively.

2.3. Consistency criteria

We are interested in two consistency criteria for replicated databases: *serializability* (SR) [4] and *snapshot isolation* (SI) [3].

Serializability. A typical correctness criterion for replicated databases is *one-copy serializability* (1SR)[4]. Informally, 1SR requires the execution of concurrent transactions on different replicas to appear as if transactions were executed in some sequential order on a single replica. To guarantee serializability most DBMSs implement *two-phase locking* (2PL) or *strict* 2PL concurrency control [4], where locks on data items are handled by a transaction in two consecutive phases during its execution. In a replicated setting, 2PL may result in high abort rates as aborts grow with the third power of the number of replicas [11]. It has been shown that total-order multicast can be used to reduce the aborts due to replication in the presence of locking [1].

Snapshot Isolation. In snapshot-isolated databases transactions read data from a committed snapshot of the database taken at the time the transaction starts. All transactions execute without interfering with each other, however, transaction T_i can only successfully commit if there exists no other transaction T_j that committed after T_i started and updated the same data items (*first-committer-wins* rule). If no such a transaction exists, then T_i can commit and its updates will be visible to all the transactions that start after T_i's commit.

In [9] the authors extend snapshot isolation to replicated databases and define *generalized snapshot isolation* (GSI). GSI is based on the observation that a transaction need not necessarily observe the latest snapshot. In the rest of this paper we use the terms snapshot isolation and generalized snapshot isolation interchangeably.

In a partially replicated system, transactions execute using a global database snapshot that may be composed of several individual partial snapshots, taken by different replicas. Obviously, the global snapshot should be consistent across replicas. A global snapshot composed of several partial snapshots is consistent if it could be taken by a single replica containing all the database.

2.4. Replication model

We assume a partial replication model where the original database is partitioned and replicated over the database sites. We call $local$ the replica to which the transaction is submitted, and $remote$ the replica which contains data items accessed by the transaction and not stored at the local site. Similarly, an operation is called $local$ if it is executed on a local replica, and $remote$ otherwise. Transactions that access data at more than one database site during execution are called $distributed$.

We distinguish two phases through which transactions pass during processing:

1. *Execution phase.* Transactions are initially submitted to one database site. However, in partial replication, where each replica only holds a subset of the database, transactions may require access to data items not available locally. In such a case, transaction's operations are forwarded to one of the database sites holding the required items and executed remotely. If the database engine adopts lock-based concurrency control, such distributed transactions inevitably introduce the possibility of distributed deadlocks; if the replicas implement snapshot isolation, the key problem is to obtain a consistent global snapshot of the database composed of individual snapshots taken at each replica involved in the execution of the distributed transaction. We assume that every remote request includes the required snapshot version. Upon processing such a request the remote site demands the correct snapshot from the database. If such a version of the data is not available, the transaction is aborted.

2. *Termination phase.* Read-only transactions commit immediately upon request. Update transactions are forwarded to all (or a subset of) database sites using the total-order multicast primitive. We assume that all database sites involved in the execution of the transaction eventually reach a consistent decision on the transaction's fate: commit or abort. Depending on the replication protocol, this may require a voting phase as part of the transaction's termination (e.g., [17]). For different consistency criteria, different certification tests are used. Two concurrent update transactions T_i and T_j are allowed to commit only if:

   - (to ensure 1SR) T_i and T_j executed at distinct replicas, T_i has been delivered first and $ws(T_i) \cap rs(T_j) = \emptyset$ [14]. If T_i and T_j execute at the same replica, the local database scheduler guarantees a serializable execution.
   - (to ensure SI) The writesets of T_i and T_j do not intersect, that is, $ws(T_i) \cap ws(T_j) = \emptyset$. We assume that SI is implemented using strict *first-committer-wins* rule, i.e., transactions are never aborted during the execution phase because of a write-write conflict.

   If the transaction passes the certification test, its updates are applied on all copies of modified data items.

3. Analytical Model

In partial replication settings where each replica holds only a subset of the database, support for execution of distributed transactions is inevitable, unless "perfect data partitioning" is assumed.[1] In lock-based systems distributed transactions may get involved in distributed deadlocks, while in version-based systems remote read operations may be unable to obtain the requested database snapshot at remote replicas. Both distributed deadlocks and failed remote read operations result in aborted transactions. Hence, the goal of our probabilistic analysis is twofold: (a) to quantify the abort rate of transactions due to distributed execution; and (b) to estimate the abort rate of transactions at the termination phase.

The replicated system is modeled as a number of database servers, $sites$, and a fixed-size database composed of DB_SIZE items. Every database item has a number of $copies$ uniformly distributed over the replicas. Thus, the entire system consists of $DB\_SIZE \cdot copies$ resources. All transactions submitted to the database have the same number of operations op and all operations take the same op_time to execute. An operation is defined as a read or a write on one data item; as a consequence, a single SQL statement may consist of many operations. Each data item has the same probability of being accessed (there are no hotspots). We model neither communication between database sites — both local and remote accesses to data items have the same cost — nor failures of the replicas.

Every database site receives TPS transactions per second, so the total load over the replicated system is $TotalTPS = TPS \cdot sites$ and there are always $txn = TotalTPS \cdot op \cdot op\_time$ concurrent transactions executing. Every transaction is read-only with the probability of L. Each operation within the update transaction has the probability k to be a read operation. The number of concurrent

[1]If the database is partitioned so that every transaction can execute at a single site, support for distributed transactions is not needed. However, such an approach requires prior knowledge of the workload and a very particular data distribution over the replicas or at least a single site that holds the whole database.

read-only transactions in the system is $r\_txn = L{\cdot}txn$, each with *op* read operations. The number of update transactions is given by $w\_txn = (1 - L) \cdot txn$ with $r\_op = k \cdot op$ read and $w\_op = (1 - k) \cdot op$ write operations. We also require that the average number of data items accessed by concurrent transactions do not exceed the database size. The main parameters of the model are listed in Table 1.

DB_SIZE	database size
TPS	number of transactions per second submitted to the database site
L	fraction of read-only transactions
op	number of operations in a transaction
k	fraction of read operations in update transactions
op_time	time to execute an operation in seconds
sites	number of replicas in the system
copies	number of copies of each data item

Table 1. Model parameters

In the following two sections we introduce our probabilistic analysis for evaluating the abort rate of partial replication when lock- and version-based concurrency control mechanisms are used. We assume that the lock-based model ensures 1SR, while the version-based model provides GSI.

3.1. Lock-based system

Our model has been strongly influenced by the analytical model introduced by Gray et al. in [11], where the authors analyze the deadlock rate of fully replicated database systems based on locking only. Besides the assumptions considered throughout our probabilistic modelling, the work in [11] does not account for read operations — all transactions are composed of updates only. In this paper we model read operations within update transactions as well as read-only transactions. To calculate the abort rate at the termination phase, we have followed the ideas introduced in [14].

Execution phase. As in [11], we suppose that, in average, each transaction is about half way complete, thus the number of resources locked by executing transactions is at most

$$res\_locked = ro\_read\_locks + u\_locks, \tag{3.1}$$

where

$$ro\_read\_locks = \frac{r\_txn \cdot op}{2}, \tag{3.2}$$

$$\begin{aligned} u\_locks &= u\_write\_locks + u\_read\_locks \\ &= \frac{w\_txn \cdot w\_op}{2} + \frac{w\_txn \cdot r\_op}{2} = \frac{w\_txn \cdot op}{2}. \end{aligned} \tag{3.3}$$

From Eq. 3.3, the probability that a read operation waits because of update transactions is

$$\begin{aligned} p\_r\_op\_waits\_u &= \frac{u\_write\_locks}{DB\_SIZE \cdot copies} \\ &= \frac{w\_txn \cdot w\_op}{2 \cdot DB\_SIZE \cdot copies}. \end{aligned} \tag{3.4}$$

Similarly, $p\_w\_op\_waits\_u$ and $p\_w\_op\_waits\_r$ are the probabilities that a write operation waits for resources locked by update and read-only transactions:

$$\begin{aligned} p\_w\_op\_waits\_u &= \frac{u\_locks}{DB\_SIZE \cdot copies} \\ &= \frac{w\_txn \cdot op}{2 \cdot DB\_SIZE \cdot copies}, \end{aligned} \tag{3.5}$$

$$\begin{aligned} p\_w\_op\_waits\_r &= \frac{ro\_read\_locks}{DB\_SIZE \cdot copies} \\ &= \frac{r\_txn \cdot op}{2 \cdot DB\_SIZE \cdot copies}. \end{aligned} \tag{3.6}$$

Now we can calculate the probability that a read-only transaction waits for resources held by update transactions,

$$p\_r\_tran\_waits\_u = 1 - (1 - p\_r\_op\_waits\_u)^{op}, \tag{3.7}$$

and the probability that an update transaction waits because of other update transactions,

$$\begin{aligned} p\_u\_tran\_waits\_u = 1 - (1 - p\_r\_op\_waits\_u)^{r\_op} \\ \times (1 - p\_w\_op\_waits\_u)^{w\_op}, \end{aligned} \tag{3.8}$$

and because of read_only transactions,

$$p\_u\_tran\_waits\_r = 1-(1-p\_w\_op\_waits\_r)^{w\_op}. \tag{3.9}$$

A deadlock is created if transactions form a cycle waiting for each other. We do not consider deadlocks that involve more than two transactions: deadlocks composed of cycles of three or more transactions are very unlikely to occur [11]. So the probability for a read-only transaction to deadlock is

$$p\_r\_deadlock \approx \frac{p\_r\_tran\_waits\_u \cdot p\_u\_tran\_waits\_r}{r\_txn}, \tag{3.10}$$

and the probability that an update transaction deadlocks is

$$\begin{aligned} p\_w\_deadlock \approx \frac{p\_u\_tran\_waits\_u^2}{w\_txn} \\ + \frac{p\_u\_tran\_waits\_r \cdot p\_r\_tran\_waits\_u}{w\_txn}. \end{aligned} \tag{3.11}$$

From Eq. 3.10 and 3.11, read-only and update transactions deadlock rates are:

$$r\_deadlock\_rate = \frac{p\_r\_deadlock}{op \cdot op\_time}, \tag{3.12}$$

$$w\_deadlock\_rate = \frac{p\_w\_deadlock}{op \cdot op\_time}. \quad (3.13)$$

Finally, we can estimate the total number of deadlocks of the system (in transactions per second) as

$$\begin{aligned} aborts\_deadlock = {} & r\_deadlock\_rate \cdot r\_txn \\ & + w\_deadlock\_rate \cdot w\_txn. \end{aligned} \quad (3.14)$$

Termination phase. If there is only one copy of each data item (i.e., there is no replication), *strict* 2PL ensures serializability and thus transactions are not aborted during the termination phase. For more than one copy, two conflicting transactions executing concurrently at distinct database sites may violate 1SR. As mentioned in Section 2.4, to ensure 1SR, each committing transaction has to pass the certification test which checks that there is no transaction that executed concurrently and updated data items read by the committing transaction. Notice that conflicts appear only if transactions access different copies of the same item.

We consider only those transactions that were not aborted during execution. Thus, $TotalTPS$, the number of read-only and update transactions are:

$$TotalTPS' = TotalTPS - aborts\_deadlock, \quad (3.15)$$

$$r\_txn' = r\_txn \cdot (1 - p\_r\_abort), \quad (3.16)$$

$$w\_txn' = w\_txn \cdot (1 - p\_w\_abort), \quad (3.17)$$

$$txn' = TotalTPS' \cdot op \cdot op\_time. \quad (3.18)$$

If there are only two concurrent transactions in the system, the probability that an update transaction passes the certification test is $(1 - w\_op/DB\_SIZE)^{r\_op}$. Then the probability that the i-th transaction passes the certification test after the commit of $(i-1)$ transactions is

$$p\_i\_txn\_pass = \left(1 - \frac{(i-1) \cdot w\_op}{DB\_SIZE}\right)^{r\_op}. \quad (3.19)$$

On average, the probability that a transaction does not pass the certification test is

$$p\_txn\_no\_pass = 1 - \frac{1}{N} \cdot \sum_{i=1}^{N} p\_i\_txn\_pass, \quad (3.20)$$

where N is the number of concurrent update transactions, excluding those that execute at the same replica and do not cause certification aborts:

$$N = w\_txn' \cdot \frac{sites - 1}{sites}. \quad (3.21)$$

Consequently, the abort rate of update transactions that do not pass the certification test is defined as follows:

$$u\_abort\_rate = \frac{p\_txn\_no\_pass}{op \cdot op\_time}. \quad (3.22)$$

And at last, the total number of aborts due to the certification test is

$$aborts\_sr\_cert = u\_abort\_rate \cdot w\_txn'. \quad (3.23)$$

3.2 Version-based system

During the execution, transactions are aborted if the requested versions of the data items are not available. We assume that all database sites are able to maintain up to V versions per data item, e.g. with $V = 1$, transactions can only obtain the current version of the data item. Notice that we assume a strict *first-committer-wins* rule, i.e., transactions are never aborted during the execution phase due to write-write conflict; such conflicts are resolved at termination.

Execution phase. In the same way as Eq. 3.1, during its execution, a transaction updates at most w_op resources. Therefore, at any time there are $res\_upated\_exec = (w\_txn \cdot w\_op)/2$ resources updated because of the transactions in the execution phase. Some of these transactions will be successfully certified and their updates will be propagated to all the copies of the data items accessed. These remote updates will influence the total number of resources updated.[2] Therefore, during termination there are

$$\begin{aligned} res\_updated\_term = {} & \frac{(copies - 1) \cdot w\_txn' \cdot w\_op}{2} \\ & \times p\_commit \end{aligned} \quad (3.24)$$

resources updated, where w_txn' is defined in Eq. 3.34. p_commit is the probability for an update transaction to pass the certification test and is equal to $1 - p\_w\_abort\_term$ (see Eq. 3.38). Hence, the total number of resources updated is $res\_updated = res\_updated\_exec + res\_updated\_term$ and, consequently, the probability for an item to be updated V times by concurrent transactions is:

$$p\_item\_v\_updated = \left(\frac{res\_updated}{DB\_SIZE \cdot copies}\right)^{V}. \quad (3.25)$$

The probability for a read operation to abort is the same as the probability of waiting for V locks, i.e., the probability of V concurrent transactions to update the same item:

$$p\_r\_op\_abort = p\_item\_v\_updated. \quad (3.26)$$

[2] We do not account for remote updates in the lock-based model since in general the deadlock rate is very small and some remote updates will not affect significantly the final deadlock rate.

Since each read-only transaction has op operations, the probability for a read-only transaction to abort is

$$p\_r\_abort = 1 - (1 - p\_r\_op\_abort)^{op}, \quad (3.27)$$

and the probability of abort of an update transaction is

$$p\_w\_abort = 1 - (1 - p\_r\_op\_abort)^{r\_op}. \quad (3.28)$$

From Eq. 3.27 and 3.28, the abort rates for read-only and update transactions are as follows:

$$r\_abort\_rate = \frac{p\_r\_abort}{op \cdot op\_time}, \quad (3.29)$$

and

$$w\_abort\_rate = \frac{p\_w\_abort}{op \cdot op\_time}. \quad (3.30)$$

Therefore, the total number of aborts during the execution phase of transactions is

$$aborts\_exec = r\_abort\_rate \cdot r\_txn + w\_abort\_rate \cdot w\_txn. \quad (3.31)$$

Termination phase. Similarly to Eqs.3.15–3.18, we have to recalculate the number of concurrent transactions that reach the termination phase:

$$TotalTPS' = TotalTPS - aborts\_exec, \quad (3.32)$$

$$r\_txn' = r\_txn \cdot (1 - p\_r\_abort), \quad (3.33)$$

$$w\_txn' = w\_txn \cdot (1 - p\_w\_abort), \quad (3.34)$$

$$txn' = TotalTPS' \cdot op \cdot op\_time. \quad (3.35)$$

Furthermore, transactions aborted during the execution phase also affect the fraction of read-only and update transactions present at the termination phase:

$$L' = \frac{r\_txn'}{txn'}. \quad (3.36)$$

Thus, the probability that a write operation conflicts with another write operation is

$$p\_w\_op\_con = \frac{w\_txn' \cdot w\_op}{2 \cdot DB\_SIZE}. \quad (3.37)$$

The probability that an update transaction aborts is

$$p\_w\_abort\_term = 1 - (1 - p\_w\_op\_con)^{w\_op}. \quad (3.38)$$

Update transactions abort rate due to write-write conflicts is determined as

$$w\_abort\_rate\_term = \frac{p\_w\_abort\_term}{op \cdot op\_time}. \quad (3.39)$$

Finally, the total number of aborts at the termination phase is

$$aborts\_si\_cert = w\_abort\_rate\_term \cdot w\_txn'. \quad (3.40)$$

4. Analytical Evaluation

4.1. Objectives

We have analytically estimated the transaction abort rate of a partially replicated system to answer the following questions:

- What is the impact of distributed transactions on the abort rate of 1SR and SI systems?
- How do data versions affect the abort rate of SI systems in the context of partial replication?
- Under which environments are SI systems comparable to 1SR lock-based systems ?

In the following we present the parameters used throughout the evaluation.

4.2. Parameter values

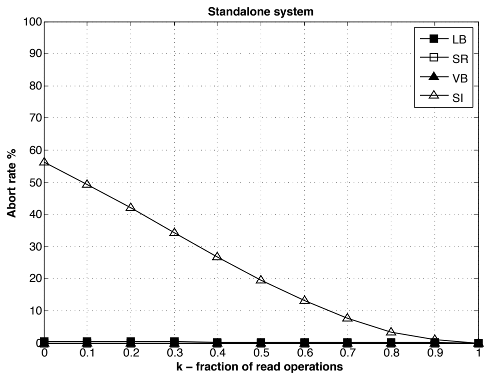
As a base scenario we consider a system composed of 8 database sites and 2 copies of 2.500.000 items database. Every database site executes 100 transactions per second. Each transaction takes 0.170 seconds to execute and is composed of 200 operations. 90% of the transactions in the workload are update transactions.[3] All the parameters used are summarized in Table 2; parameters of the base scenario are highlighted in bold.

Parameter	Values considered
DB_SIZE	**2.500.000**
TotalTPS	400, **800**
op	**200**
op_time	**0.170**/*op*
L	0, **0.1**...1
V	1, 2, 3, 4
sites	1, **8**
copies	1, **2**, 8

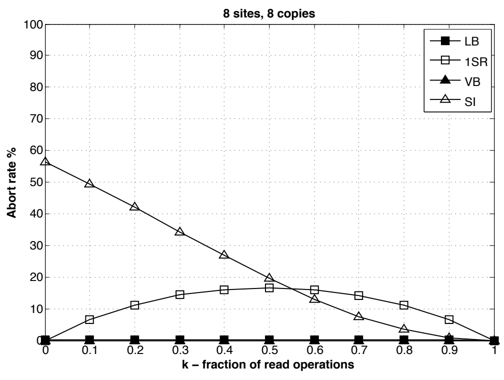
Table 2. Model parameter values

In all figures we report the percentage (%) of transactions aborted during execution and termination or just the total system abort rate. In the execution phase the lock-based system is denoted as LB; VB V represents the version-based system, where V is the number of versions available per data item (e.g. VB 1 indicates a scenario where only the current data version is obtainable). In the termination phase we denote the different systems as 1SR and SI V.

[3]We used the TPC-C benchmark [18] as a reference for our base case parameters. Our implementation of the benchmark for 5 warehouses results in a database of 2.595.055 items and an average transaction response time of 0.170 seconds. In TPC-C update transactions account for 92% of the workload.

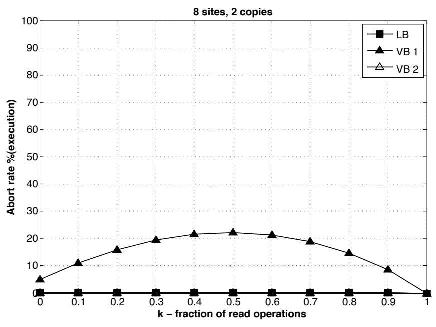


(a) standalone system

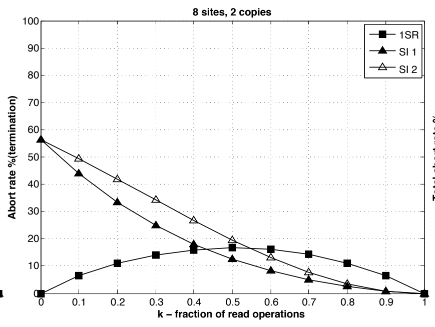


(b) fully replicated system

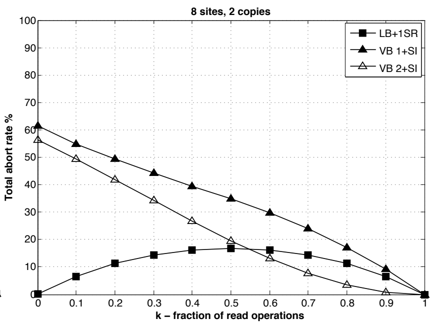
Figure 1. Standalone vs. fully replicated system, $TotalTPS = 800$



(a) during execution



(b) during termination



(c) total abort rate

Figure 2. The effects of distributed transactions; base scenario

4.3. Standalone vs. fully replicated system

Figure 1 compares the execution and termination abort rates for standalone and fully replicated systems. The standalone lock-based system has very low deadlock rate and there are no aborts due to the certification test (see Figure 1(a)): if two conflicting transactions execute concurrently at the same replica, the local database scheduler will serialize them, and thus both transactions can commit. Adding replicas increases the number of transactions executing at distinct sites, thus the aborts due to lack of synchronization grow accordingly (see Figure 1(b)). Differently from the model introduced by Gray et al. [11], where the deadlock rate rises as the third power of the number of replicas, in our replication model all commits of update transactions are ordered and thus replication does not increase the deadlock rate of the 1SR system — the aborts in Figure 1(b)) are due to the certification test. We further use the lock-based 1SR system as a baseline for analyzing the aborts of partially replicated SI systems. In a version-based system, even if two conflicting transactions execute at the same database site, only one is allowed to commit — notice that in SI two concurrent transactions conflict if they update the same data item. Therefore, replication does not affect the system abort rate. Moreover, there are no aborts due to unavailable consistent snapshots in the standalone and the fully replicated systems; in both cases the number of data versions available is unbounded.

4.4. Two data versions are sufficient to eliminate execution aborts

Figure 2 presents the transaction abort rate during (a) execution and (b) termination, and (c) the total system abort rate for the base scenario configuration. If only a single data version is available during the execution of transactions under version-based concurrency control, up to $\approx 22\%$ of transactions may abort due to failed reads (see Figure 2(a), VB 1 curve). The abort rate depends on the number of write operations in update transactions. With 100% of write operations in update transactions, only read-only transactions can abort due to not obtaining the requested database snapshot. On the other hand, with 100% of read operations there are no updates and, hence, no failed reads.

The availability of at least one additional data version is sufficient to almost completely eliminate the aborts during execution (Figure 2(a), VB 2 curve)! This is because the number of data versions available reduces the abort rate exponentially (see Eq. 3.25). We investigate this phenomenon further in Figure 3, which depicts the system abort rate at the execution phase when increasing the number of available versions. Two versions of each data item reduce the execution aborts so that they become insignificant (0.06% in the worst case). Therefore, increasing further the number of versions available will not affect remarkably the abort rate.

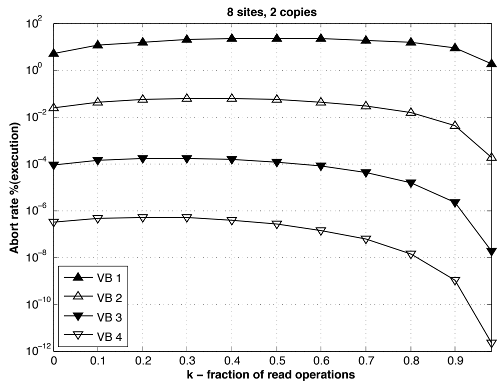


Figure 3. The effects of versions available; base scenario, y-axis in logarithmic scale

If update transactions contain a high number of read operations, VB 1+SI can be as good as LB+1SR, while VB 2+SI may even outperform LB+1SR (see Figure 2(c), when 60% or more of read operations in update transactions). However, if the workload is dominated by update operations, version-based systems abort more transactions, regardless of the number of versions per data item available. This is due to the differences in the certification test. As presented in Section 2, to ensure SI the certification test of a version-based system checks write-write conflicts between concurrent transactions, while to ensure 1SR the certification test of a lock-based system checks read-write conflicts. With a lot of update operations the probability of write-write conflicts increases and thus, the version-based system exhibits higher abort rate. During the termination phase, the abort rate of SI 1 is lower than SI 2 (see Figure 2(b)). This is because in the termination phase our model accounts for aborts that happen during execution. Since in VB 1 a lot of transactions are aborted during execution, fewer transactions reach the termination phase, and consequently, fewer transactions are aborted.

4.5. The impact of read-only transactions

To evaluate the impact of read-only transactions in the workload, we have varied the L parameter. Figure 4 illustrates the total system abort rate when $L = 0.9$. The abort rate of both LB+1SR and VB 2+SI is very low, since with very few updates the termination abort rate is small and it is unlikely that transactions deadlock during the execution. However, for VB 1+SI, if the fraction of write operations in the transactions is high, the execution aborts dominate the total system abort rate. If all update transactions perform a lot of write operations, read-only transactions still have a significant probability of aborting due to not obtaining the requested version of the data item. For example, for VB 1+SI, $L = 0.9$ and $k = 0$, the probability for a read operation to abort due to not obtaining the correct version is $0.541 \cdot 10^{-3}$, but the probability for a read-only transaction to abort during execution is 0.102 (see Eq. 3.26 and 3.27, respectively). Thus, even if the workload over the partially replicated system is dominated by read-only transactions, but the few update transactions perform a lot of updates, read-only distributed transactions can still cause a noticeable number of aborts. This is in contrast to typical fully replicated SI systems, in which the number of versions available in each replica is unbounded, and thus, read-only transactions never abort.

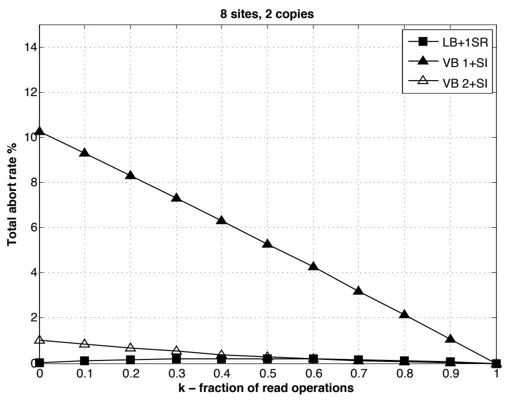
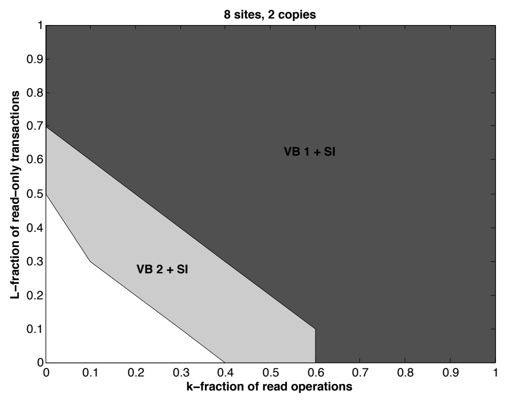


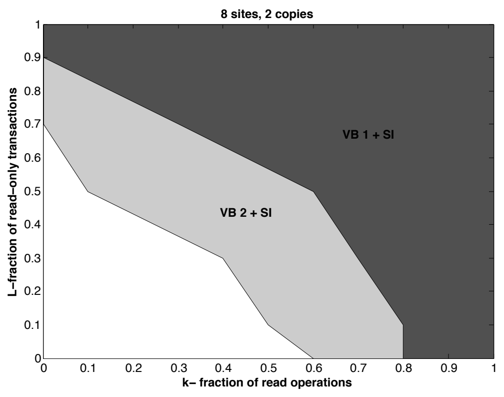
Figure 4. The effects of increasing read-only transactions in the workload; $L = 0.9$

4.6. The environments beneficial to SI version-based systems

Figure 5 depicts the environments under which SI can be safely used with partial replication. We have varied the L and k parameters and report the results where the total system abort rate is equal to or below 20% with $TotalTPS = 400$ (Figure 5(a)) and $TotalTPS = 800$ (Figure 5(b)). The dark and light gray areas represent configurations of version-based SI systems with one (i.e., VB 1+SI) and two data versions available (i.e., VB 2+SI). Workloads composed of a lot of read-only transactions and



(a) $TotalTPS = 400$



(b) $TotalTPS = 800$

Figure 5. Configurations where abort rate of SI systems is $\leq 20\%$; base scenario

workloads where update transactions contain many read operations represent environments beneficial to SI systems.

5. Final Remarks

This paper presents a simple probabilistic analysis of abort rates in partially replicated systems. Two concurrency control mechanisms are considered: lock- and version-based. The lock-based system models the behavior of a replication protocol providing one-copy serializability, while the version-based system ensures snapshot isolation.

In our probabilistic model adding replicas to the system has a different impact on lock- and version-based systems. The difference comes from the distinct certification tests. The throughput over a database site has a linear impact on the total system abort rate, while the number of operations has an exponential effect on the aborts for both lock- and version-based systems. Augmenting the number of operations increases the number of concurrent transactions in the system ($TPS * op * op\_time$), and thus has a quadratic impact on the number of resources updated or locked (e.g., see Eq. 3.1). However, once we calculate various probabilities (e.g. probability that transaction waits), the number of operations appears in the exponent of the formula (e.g. see Eq. 3.7). The impact of the database size is smaller: being just at the denominator of the Eq. 3.1 formula, reducing database size, increases the total system aborts.

The presented analytical evaluation revealed that in the version-based system the number of data versions available decreases the execution abort rate exponentially. As a consequence, in all cases considered, two versions of each data item were sufficient to eliminate the aborts due to distributed transactions. Furthermore, in the version-based system even if the workload over the partially replicated system is dominated by read-only transactions, but the few update transactions perform a lot of updates, read-only distributed transactions can still cause a noticeable number of aborts, as opposed to typical full replication protocols, in which the number of versions available is unbounded, and thus, read-only transactions executing under snapshot isolation are never aborted.

References

[1] D. Agrawal, G. Alonso, A. E. Abbadi, and I. Stanoi. Exploiting atomic broadcast in replicated databases. In *Proceedings of the 3rd International Euro-Par Conference on Parallel Processing*, pages 496–503, 1997.

[2] J. E. Armendáriz-Íñigo, A. Mauch-Goya, J. R. G. de Mendívil, and F. D. Muñoz-Escoí. SIPRe: A partial database replication protocol with SI replicas. In *Proceedings of the 23rd ACM Symposium on Applied computing*, pages 2181–2185, 2008.

[3] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil. A critique of ANSI SQL isolation levels. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 1–10, 1995.

[4] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[5] L. J. Camargos, F. Pedone, and M. Wieloch. Sprint: a middleware for high-performance transaction processing. In *Proceeding of EuroSys*, pages 385–398, 2007.

[6] E. Cecchet, J. Marguerite, and W. Zwaenepoel. C-JDBC: Flexible database clustering middleware. In *Proceedings of USENIX Annual Technical Conference, Freenix track*, 2004.

[7] C. Coulon, E. Pacitti, and P.Valduriez. Consistency management for partial replication in a high performance database cluster. In *Proceeding of the 11th International Conference on Parallel and Distributed Systems*, pages 809–815, 2005.

[8] S. Elnikety, S. Dropsho, and W. Zwaenepoel. Tashkent+: Memory-aware load balancing and update filtering in replicated databases. In *Proceeding of EuroSys*, pages 399–412, 2007.

[9] S. Elnikety, W. Zwaenepoel, and F. Pedone. Database replication using generalized snapshot isolation. In *Proceedings of the 24th IEEE Symposium on Reliable Distributed Systems*, pages 73–84, 2005.

[10] U. J. Fritzke and P. Ingels. Transactions on partially replicated data based on reliable and atomic multicasts. In *Proceeding of the 21st International Conference on Distributed Computing Systems*, pages 284–291, 2001.

[11] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of Data*, pages 173–182, 1996.

[12] J. Holliday, D. Agrawal, and A. E. Abbadi. Partial database replication using epidemic communication. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 485– 493, 2002.

[13] B. Kemme. *Database Replication for Clusters of Workstations*. PhD thesis, Swiss Federal Institute of Technology Zürich, Switzerland, 2000.

[14] F. Pedone. *The Database State Machine and Group Communication Issues*. PhD thesis, École Polytechnique Fédérale de Lausanne, Switzerland, 1999.

[15] N. Schiper, R. Schmidt, and F. Pedone. Optimistic algorithms for partial database replication. In *Proceedings of the 10th International Conference on Principles of Distributed Systems*, pages 81–93, 2006.

[16] D. Serrano, M. Patiño-Martínez, R. Jiménez-Peris, and B. Kemme. Boosting database replication scalability through partial replication and 1-copy-snapshot-isolation. In *Proceedings of the 13th Pacific Rim International Symposium on Dependable Computing*, pages 290–297, 2007.

[17] A. Sousa, F. Pedone, F. Moura, and R. Oliveira. Partial replication in the database state machine. In *Proceedings of IEEE International Symposium on Network Computing and Applications*, pages 298–309, 2001.

[18] Transaction Proccesing Performance Council (TPC). TPC benchmark C. Standard Specification, 2005.