Boosting Database Replication Scalability through Partial Replication and 1-Copy-Snapshot-Isolation *

D. Serrano, M. Patiño-Martinez, R. Jimenez-Peris
Universidad Politenica de Madrid (UPM), Spain
{dserrano,mpatino,rjimenez}@fi.upm.es

B. Kemme
McGill University, Canada
kemme@cs.mcgill.ca

Abstract

Databases have become a crucial component in modern information systems. At the same time, they have become the main bottleneck in most systems. Database replication protocols have been proposed to solve the scalability problem by scaling out in a cluster of sites. Current techniques have attained some degree of scalability, however there are two main limitations to existing approaches. Firstly, most solutions adopt a full replication model where all sites store a full copy of the database. The coordination overhead imposed by keeping all replicas consistent allows such approaches to achieve only medium scalabilitiy. Secondly, most replication protocols rely on the traditional consistency criterion, 1-copy-serializability, which limits concurrency, and thus scalability of the system. In this paper, we first analyze analytically the performance gains that can be achieved by various partial replication configurations, i.e., configurations where not all sites store all data. From there, we derive a partial replication protocol that provides 1-copy-snapshot isolation as correctness criterion. We have evaluated the protocol with TPC-W and the results show better scalability than full replication.

Keywords: Large-Scale Database Replication, Partial Replication, Database Replication Middleware, Snpashot Isolation, One-Copy Snapshot Isolation

1. Introduction

The software industry is evolving to modern service oriented architectures that are targeted to provide service to increasingly large numbers of users. In order to provide appropriate scale-out any kind of bottleneck in the service infrastructure needs to be avoided. Currently, one of the most common bottlenecks in these information systems is the backend database that stores the persistent data. This is one of the reasons why database replication has received a lot of attention in the last few years.

Early approaches to eager (also called synchronous) database replication based on distributed locking lacked any scalability [10]. This paper triggered a new wave of research trying to overcome the scalability limitations. A lot of research has focused on scalable replication protocols that provide 1-copy-serializability as correctness criterion [4], and assume full database replication [15, 20, 22, 2, 21, 12]. The problem of protocols providing serializability is that all types of read/write and write/write conflicts must be considered. In particular read/write conflicts result in an inherent scalability ceiling since they are very frequent and therefore, limit the amount of potential concurrency in the system. In optimistic replication protocols [15, 22, 12] read-write conflicts result in transaction aborts, whilst pessimistic protocols [20, 2, 21] result in low concurrency. Snapshot isolation [3] is a multi-version concurrency control in which transactions see a snapshot of the database as it was when the transaction started. Moreover, readers and writers do not conflict. Compared to serializability snapshot isolation provides a very similar level of consistency (it passes the tests for serializability of standard benchmarks such as the ones from TPC). Today many databases provide snapshot isolation as their highest isolation level (e.g., Oracle, PostgreSQL, etc.). 1-copy-snapshot-isolation is a correctness criterion for replicated databases that run under snapshot isolation [16].

Full database replication means that all sites store copies of all data items. An analytical study in [13] has shown the scalability limits of full eager replication. The problem is that updates have to be executed at all replicas. In some protocols update transactions are executed at all sites (*symmetric processing*) to preserve all the databases identical (consistent). Therefore, the replicated database does not scale under update workloads (all sites do the same work). Other protocols use *asymmetric processing*. That is, a transaction is first executed at one site, its changes are collected in

*This work has been partially funded by Microsoft Research Cambridge under a European PhD Award, the Spanish Research Council, MEC, under project TIN2004-07474-C02-01, the Madrid Regional Science Foundation (CAM), under project S-0505/TIC/000285, Consejería de Educación de la CAM, F.E.D.E.R y F.S.E.

form of a writeset which is applied at the other sites to keep the copies consistent. Asymmetric processing provides better scalability with update workloads, because applying the writeset usually only requires a fraction of the processing cost of executing the full transaction / update operations. When there is only one site (no replication), all the capacity of the site is used to process transactions. If the system has two replicas (it executes transactions at two sites), then a fraction of the capacity of each replica is devoted to install the writesets produced by the other replica. When the number of replicas increases, there is a point at which adding a new replica does not increase the system capacity anymore. The reason is that a large fraction of the capacity of the new replica is consumed to process updates propagated by the other replicas.

In this paper we elaborate on solutions that aim at increasing the scalability of replication following an update everywhere model (i.e., there is not a distinguished site for running update transactions). First, we provide an analytical model that allows to quantify the scalability of a partially replicated database. Then, we propose an eager replication protocol for partially replicated databases based on snapshot isolation. The protocol is able to handle distributed transactions where different operations are executed at different sites. The challenge in our setting is to provide a globally consistent snapshot for distributed transactions and detect conflicts although no site has global knowledge in the system. We have evaluated our protocol on the industrial benchmark TPC-W showing a much improved scalability over full replication.

The paper is structured as follows. Section 2 surveys the state of the art on database replication. Then, we motivate partial replication through an analytical model of the scalability in Section 3 and the resulting analytical evaluation of different replication alternatives. Then, we present the partial replication protocol in Section 4. Section 5 shows the empirical evaluation of the replication protocol using TPC-W benchmark. We present our conclusions in Section 6.

2. Related work

[17] proposes a two dimensional replication model based on the number of data items and the number of copies per item. The model assumes a primary copy approach, that is, all update transactions are executed at a single site, and changes (writesets) are propagated after commit to the secondary sites which only execute read-only transactions. The model assumes that the cost of applying the changes at the secondaries is the same as the one of executing the full transaction (symmetric update processing). The main differences with our analytical model are as follows. First we consider asymmetric update processing. Second, we consider heterogeneous workloads in which different replicas might process different mixes of read-only and update transactions. Third, we consider that each replica might contain a different fraction of the database.

Full database replication has been heavily studied [2, 5, 21, 6, 19, 1, 9]. The traditional correctness criterion for data replication is one-copy-serializability (1CS) [4]. More recently, some protocols have explored extensions of snapshot isolation for replicated data [7, 23, 27, 16, 8].

All the protocols on partial replication implement 1CS and most of them assume that all data accessed by a transaction is stored on the site where the transaction is submitted. One of the earliest protocols on partial replication is the one proposed in [14], integrated into the Postgres-R prototype. The protocol follows a primary backup approach. The primary sends the writesets (changes) to all of the sites where are decomposed, analyzed and applied if needed.

[25] proposes a protocol in which readsets and writesets are multicast to all sites holding a copy of any data item involved in the transaction. Then, each site checks whether there is any conflict with other committed transaction at that site (certification). The result of the certification is used later when the site that executed the transaction, initiates a distributed commit protocol to determine the outcome of the transaction. More recently, the authors have compared this approach with a protocol in which readsets and writesets are multicast to all replicas, and all replicas run a local certification process to decide on the transaction outcome [24]. There are several differences of this work with ours. The correctness criterion we use is one-copy-snapshot isolation (1CSI), whilst they use one-copy-serializability. The use of 1CSI has the great benefit of not dealing with readsets, in addition to never have read-write conflicts, and therefore, reduces the number of aborted transactions. Second, our protocol only multicasts the writeset. Their protocol needs both writesets and readsets that can be very bulky. Third, they assume that for every transaction there is at least one site that stores all data needed by the transaction.

[11] proposes an epidemic protocol for partially replicated databases in a WAN environment. Each data item has one or more permanent sites that always have a copy. Other sites may have a temporary cached copy. Readsets and writesets are propagated to maintain consistency. If a data item is not stored at the site where the transaction executes, a request is sent to one of the permanent sites and propagated with the associated lock table information. Our work deals with 1CSI and focuses on a cluster environment.

The protocol in [18] supports full and partial replication. It avoids distributed transactions by requiring a priori knowledge of the data accessed by transactions. Partial replication has also been studied in [5]. The model of replication is that some sites do not store the whole database. Transactions are executed at a single site. This may lead to full replication when there are complex requests that access

several tables.

3. Analytical Model for Partial Replication

3.1. System Model

The system consists of a set of n sites, $N = \{1..n\}$. The database consists of a set of o objects, $O = \{O_1, .., O_o\}$. Objects are accessed in a transaction. A transaction is read only if it only reads objects. An update transaction at least writes (insert/update/delete) one object. The model is based on asymmetric processing of transactions (one site executes the update transaction and all other sites only apply the resulting writeset). The load is defined as a pair (A, U). A is the proportion of accesses to each object per time unit, $A = \{A_1, .., A_o\}$, i. e. A_i means the proportion of the accesses (reads and writes) to the object O_i. Given A, U is the proportion of writes per time unit, $U = \{U_1, .., U_o\}$, i.e. U_i means the proportion of accesses to the object O_i that modify it. Both A and U are further decomposed in $A_i = \{a_{1i}, a_{2i}, ..., a_{ni}\}$ and $U_i = \{u_{1i}, u_{2i}, ..., u_{ni}\}$, which define the percentage of accesses (writes) to object i at each site, respectively. There is also a function $r : N \times O \rightarrow \{0, 1\}$, which defines the replication schema i. e. in which sites objects are stored. $r(i, k) = 1$ if O_k is stored at site i, and $r(i, k) = 0$ otherwise. r models a fully replicated database (every object is stored at every site), if $\forall i \in \{1..n\}, \forall k \in \{1..o\} \mid r(i, k) = 1$. We call a hybrid replicated database to a database where at least one site stores a full copy of the database (full replica), and at least one site does not store the full database (partial replica). r models a hybrid replicated database, if $\exists i \in \{1..n\} \forall k \in \{1..o\} \mid r(i, k) = 1 \wedge \exists j \in \{1..n\} \exists k \in \{1..o\} \mid r(j, k) = 0$. Otherwise, r models a pure partially replicated database. That is, a database where no site stores a full copy of the database and data is replicated. Formally: $\forall i \in \{1..n\} \exists k \in \{1..o\} \mid r(i, k) = 0$

3.2. Analytical Model

The goal of the analytical model is to understand the potential scalability gains of partial replication with respect to full replication. The model quantifies the scale out, which determines how many times the replicated system increases the performance of a non replicated system.

We assume that a non replicated database has a processing capacity C, that means that a non-replicated database can execute C transactions per time unit. We assume all sites have the same capacity. In a non-replicated database, the entire processing capacity C is used for executing local transactions, but sites in the replicated database need to use some of its processing capacity C for coordination with other sites. We term the coordination work as remote work. So, each site i in a replicated database uses a fraction of its processing capacity for local work (L_i) and the remaining capacity for remote work (R_i), i. e. $C = L_i + R_i$. Then, the local work performed in a site i is:

$$L_i = C - R_i \tag{1}$$

The scale out is the sum of the amount of local work executed at each site, divided by the processing capacity of a non-replicated database ($scaleout = \frac{\sum_{i=1}^{n} L_i}{C}$). That is, how many times the capacity of a non-replicated system is increased when it is replicated. The more local work (L_i) each site executes, the better the scalability of the system. The total amount of local work L_i at site i is the sum of accesses to objects O_k stored at i (Eq. 2). The objects stored at site i are defined by the function $r(i, k)$.

$$L_i = \sum_{k=1}^{o} C \cdot r(i, k) \cdot a_{ik}, \forall i = 1..n \tag{2}$$

Since our model uses asymmetric processing, writes on an object O_k at site i impose some (remote) work in the other sites that also store a copy of O_k. We call this fraction of remote work the writing overhead, $wo, 0 \leq wo \leq 1$. Therefore, the amount of remote work, R_i, at site i is a fraction (wo) of the writes on every object O_k stored at site i that are executed at the rest of the sites that store a copy of that object (Eq. 3).

$$R_i = wo \cdot \sum_{j=1, j \neq i}^{n} \sum_{k=1}^{o} r(i, k) \cdot r(j, k) \cdot a_{jk} \cdot u_{jk} \tag{3}$$

Therefore, replacing L_i and R_i in Eq. 1 with the expressions in Eq. 2 and Eq. 3 we obtain:

$$\sum_{k=1}^{o} C \cdot r(i, k) \cdot a_{ik} = C - wo \cdot \sum_{j=1, j \neq i}^{n} \sum_{k=1}^{o} r(i, k) \cdot r(j, k) \cdot a_{jk} \cdot u_{jk}, \forall i = 1..n \tag{4}$$

However, maximizing the amount of local work (L_i) each site processes may lead to saturation of other sites. For instance, let us assume that there are three sites $\{s_1, s_2, s_3\}$, two objects $\{O_1, O_2\}$ and a writing overhead $wo = 0.75$. s_1 stores $\{O_1\}$, $s_2 = \{O_2\}$, and $s_3 = \{O_1, O_2\}$. That is, s_1 and s_2 are partial replicas and s_3 is a full replica. Furthermore, s_1 and s_2 use their entire processing capacity for local work ($R_i = 0$, $L_i = C$ at both sites). The amount of remote work at s_3 is $R_3 = (C + C) * 0.75 = 1.5C$. That is, the amount of remote work at s_3 surpasses its processing capacity, and therefore, that site is saturated. In order to avoid saturation of some sites, some of the remaining sites cannot use their entire capacity, e. g. in the previous example, if s_1 and s_2 use only $\frac{2}{3}$ of their capacity, $R_3 = (\frac{2}{3}C + \frac{2}{3}C)0.75 = C$, s_3 could process all remote work. We model this fact by reducing the capacity of sites. C_i ($0 \leq C_i \leq 1.$) is the percentage of the capacity site i uses to prevent saturation of other sites. So, equation 4 is

now defined as follows:

$$\sum_{k=1}^{o} C \cdot r(i,k) \cdot a_{ik} =$$

$$C \cdot C_i - wo \cdot \sum_{j=1, j \neq i}^{n} \sum_{k=1}^{o} r(i,k) \cdot r(j,k) \cdot a_{jk} \cdot u_{jk}, \forall i = 1..n \quad (5)$$

Moreover, the model must take into account the workload (A, U), and guarantee that the amount of accesses to each object O_k corresponds with the proportion A_k. So, each A_k should be equal to the fraction of accesses to O_k with respect the global number of accesses to all objects:

$$A_k = \frac{\sum_{i=1}^{n} C \cdot r(i,k) \cdot a_{ik}}{\sum_{i=1}^{n} \sum_{k=1}^{o} C \cdot r(i,k) \cdot a_{ik}}, \forall k \in \{1..o\} \quad (6)$$

A similar expression is defined for writes U_k, which are a proportion of writes of the total number of accesses for an object A_k.

$$A_k \cdot U_k = \frac{\sum_{i=1}^{n} C \cdot r(i,k) \cdot a_{ik} \cdot u_{ik}}{\sum_{i=1}^{n} \sum_{k=1}^{o} C \cdot r(i,k) \cdot a_{ik} \cdot u_{ik}}, \forall k \in \{1..o\} \quad (7)$$

Finally, the scale out is the sum of the amount of local work executed at each site, divided by the processing capacity of the non-replicated database (Eq. 8).

$$scale\_out = \frac{1}{C} \sum_{i=1}^{n} \sum_{k=1}^{o} C \cdot r(i,k) \cdot a_{ik} \quad (8)$$

So, given a replicated database of n sites with a replication schema r and a load (A, U), we look for the values C_i, a_{ik}, u_{ik} that maximize the scale out solving the optimization problem (non linear program):

$$max \frac{1}{C} \sum_{i=1}^{n} \sum_{i=1}^{n} C \cdot r(i,k) \cdot a_{ik} \quad (9)$$

subject to Eq. 5, 6, 7 and the domain of variables: $0 \leq C_i \leq 1; 0 \leq a_{ik} \leq 1; 0 \leq u_{ik} \leq 1; \forall i \in \{1..n\}; \forall k \in \{1..o\}$

3.3. Analytical Evaluation

We have analytically evaluated the different replication strategies for a system with 100 sites. $A_o = 0.01$ (even load) and $U_k = 0.2$ (20% updates) for all objects. To simplify the non linear problem we assume that the amount of updates on objects are evenly distributed among nodes, i. e. $u_{ik} = U_k$. This assumption simplifies the non linear problem into a linear one. We compare the scale out of full replication, hybrid replication (one site with the full database and five replicas per object distributed among the remaining sites) and pure partial replication (five replicas per object and the database is distributed among 100 sites). The number of copies per object and the number of partial replicas (sites storing a fraction of the database) determine the portion of the database each site stores. For instance, 5 copies and 90 partial replicas means that each partial replica stores $\frac{5}{90}$ of the database. Figure 1 shows the scale out. Full replication scales to 20 with 40 sites. Then, if more sites are added to the system, the scale out almost does not increase (24 scale out for 100 sites). Hybrid replication scales better reaching a scale out of 35. Pure partial replication shows the best scalability in that setting with a close to linear scale out, outperforming the rest of the replication strategies.

The evolution of scale out of pure partial replication for different numbers of copies per object is shown in Fig.2. As can be seen, the more copies per object, the less scale out. With $\frac{N}{2}$ copies and 100 sites, each site stores half of the database ($\frac{50}{100} = \frac{1}{2}$). So, the amount of remote work at each site is considerable even for a medium percentage of updates (20%) and therefore, the maximum scale out is 40 for a system with 100 sites. With 2-10 copies per object the scale out is considerably higher. The reason of the higher scale out for a smaller number of copies is due to the smaller overhead that is introduced for remote updates.

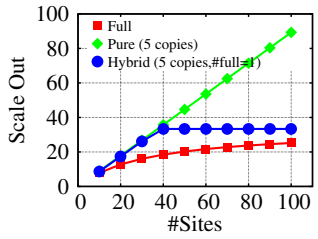


Figure 1. Analytical Scale Out

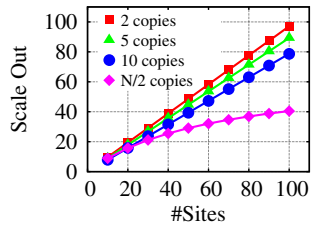


Figure 2. Scale Out. Pure Partial Replication. 20 % Updates

4. One-Copy-Snapshot-Isolation Partial Replication Protocol

Optimistic asymmetric replication protocols for full replication execute transactions at a single site and before committing, a validation process is executed for update transactions to determine if the transaction conflicts with any other concurrent update transaction executed at any

other site. In this case the transaction aborts, otherwise, it commits. This validation process is done only for update transactions. Since only write-write conflicts are checked with snapshot isolation, read only transactions can safely commit when they finish at the site where they were executed without any further action.

In order to validate an update transaction the site that executes the transaction multicast (total order) the transaction updates to all sites. Since all sites know all update transactions, they can validate the transaction without any further communication. Thanks to the use of total order multicast (all sites will receive messages in the same order) and the deterministic validation, all sites will produce the same outcome for all transactions. One may think that with partial replication validation should only be executed by the sites that store data that has been modified. However, this may produce inconsistencies.

For instance, let us assume that there are three sites $\{S_1, S_2, S_3\}$, the database consists of three objects $\{a, b, c\}$. Site S_1 stores $S_1 = \{b, c\}$, $S_2 = \{a, b\}$, and $S_3 = \{a, c\}$. There are two concurrent transactions executed at S_1 and S_2, respectively. t_1 updates $\{b\}$ and t_2 updates $\{a, b\}$. The total (multicast) and validation order is t_1, t_2. If S_1 only receives writesets related to b and c, it would commit t_1 and abort t_2 (they are concurrent, conflict and t_1 validates before t_2). The same would happen at S_2. However, S_3 would validate (and commit) t_2 because there is no other concurrent conflicting transaction at S_3 (S_3 does not validate t_1 because S_3 does not store b). This means that t_2 changes to a would be applied at S_3, but not at S_1 and S_2 yielding to inconsistencies. If S_3 had received t_1, t_2 validation would also have failed at S_3. Therefore, all sites must receive and validate update transactions.

Another important issue is the selection of the site where transactions are executed. Most of the partial replication protocols do not address this issue. If transactions only have a single SQL statement, it is easy to determine the site where the transaction may execute. The system may provide a directory that stores for each site the data that site stores. However, if transactions may have several SQL statements and they are not known a priori, it may happen that a transaction is submitted to a site that stores the data used in the first SQL statement but, that site does not store the data the transaction needs for the following SQL statements. Even if there is a priori knowledge about the data used by transactions, it may happen that transactions access a lot of data and therefore, at least one site may have a full copy of the database. As we have shown with the analytical model, hybrid replication does not scale. So, a general solution for partial replication should be able to deal with distributed transactions.

4.1. Partial Replication Protocol

Snapshot isolation (SI) is a multi-version concurrency control used in databases [3]. One of the most important properties of SI is that readers and writers do not conflict. When a transaction T begins, it gets a *start timestamp* ($T.ST$). This timestamp denotes the snapshot of the transaction which consists of the latest committed values (version) of the database (*snapshot*). Every read operation of T is performed on the snapshot associated to T. That is, the snapshot a transaction reads from reflects the number of committed update transactions at the time that transaction started. When T updates data, it creates a new version that becomes valid at the time the transaction commits. Subsequent read operations of T will read its own writes. When T finishes it gets a *commit timestamp* ($T.CT$). The two timestamps of a transaction are used to validate the transaction, that is, to determine if the transaction will commit or abort. T will commit if there is no other concurrent committed transaction T_j that has written some common object. That is, T commit timestamp is in the interval of the start and commit timestamp of T_j.

With partial replication the number of committed transactions may be different among sites (not all sites store the full database). However, all sites validate all update transactions and therefore increment the snapshot value in the same way.

The protocol proceeds as follows (Fig. 3). A client connects to a site that at least stores the data accessed in the first operation of the transaction. That site will coordinate the transaction and associate the transaction with its start timestamp (*T.ST*). It will also execute the transaction operations if the site stores all the needed data. Otherwise, it will forward the operation to another site (*redirect*). If the operation is redirected to another site, that site will execute the operation and send the results and changes back to the coordinator. When a transaction executes at several sites it must read from the same snapshot at all the sites. The transaction start timestamp travels with the transaction. When a transaction operation is forwarded to a site, then that site finds the right snapshot (*getDummyTransaction(*$T.ST$*)*) and executes the forwarded operation in the corresponding snapshot.

When a transaction is redirected to another site, that site may have to apply changes of that transaction before executing the forwarded operation. For instance, if there are three sites, S_1, S_2, S_3 and a database with three objects A, B, C. S_1 stores AB, S_2 BC and S_3 AC. Now a client starts a transaction at S_1 and updates A. Then, it updates C using information introduced in A. Since S_1 does not store C, the operation is forwarded to S_3. However, S_3 does not know the updates executed on behalf of the transaction at S_1. Therefore, the transaction will not be session consistent. It

modified some data and when it reads them later, it does not see its own changes. For this reason, the changes produced by a transaction are applied at the forwarded site before executing the forwarded operation (*execute(changes(T))*). On the other hand, the operation may produce changes that are also needed at the coordinator and therefore, are propagated with the result of the operation (*send($T.coordinator$, result(op_j), newChanges(T_k)...)*).

When the client submits the commit operation, if it is a query, the transaction can commit at all participating sites (*multicast(commit,T)*). If it is an update transaction the coordinator multicasts in total order the transaction writeset. Now all sites perform validation. It will be the same at all sites since all sites have received the same set of transactions for validation and in the same (total) order. During validation (*validate*), the transaction is compared with concurrent update transactions that already validated (committed). If validation succeeds, the writeset is applied and T can commit. At sites that have already performed some operations on behalf of T, applying the writeset will only apply the missing updates.

When a transaction operation is forwarded from the transaction coordinator site to another site, the transaction must be executed using the same snapshot that at the coordinator. For this to be correct, each site starts a number of dummy transactions after each transaction commit (*startTransaction(), storeDummyTransaction*(T), only one dummy transaction is started in the algorithm). These transactions are associated with the current database snapshot. When a redirected operation arrives, an unused dummy transaction with the correct snapshot will be used for the execution of this operation (*getDummyTransaction($T.ST$)*). There is also a garbage collection algorithm (not shown) that aborts unused dummy transactions corresponding to old snapshots that will not be needed anymore. Since starting transactions has a negligible cost, as well as aborting transactions that did not access any data, the strategy attains its goal with an almost null overhead.

5. Evaluation

5.1. Evaluation Setup

We have evaluated our protocol using the TPC-W benchmark [26]. TPC-W simulates an online bookstore and defines a transactional web benchmark for evaluating e-commerce systems. TPC-W establishes three different workloads: browsing (10% updates), shopping (20%), and ordering (50%). The benchmark establishes a database with 10 tables but provides freedom to change the schema of the databases as far as all the information of the tables is kept. We split the table item into two tables to separate the updated part, from the read only part. There is also a new

Initialization
snapshot= 0 ;
commitedTX= $\emptyset$;
transactionTable= $\emptyset$;

Upon receiving operation op_j **of transaction** T **from** $client$
if $first(op_j, T)$ **then**
T=startTransaction();
$T.coordinator = S_k$;
$T.ST = snapshot$;
transactionTable.store(T, T);
end
if $dataIsLocal(op_j, S_k)$ **then**
$execute(op_j, T.ST)$;
return $result(op_j)$ to the client;
else
$redirect(op_j, getChanges(T), T)$;
end

Upon receiving $redirect(op_j, changes(T), T)$
if *firstOperationAtSite(T)* **then**
$T_k = getDummyTransaction(T.ST)$;
transactionTable.store(T, T_k);
else
T_k =transactionTable.getTX(T);
end
$execute(changes(T) \cap localData(S_k), T_k)$;
$execute(op_j, T_k)$;
$send(T.coordinator, result(op_j), newChanges(T_k), T)$;

Upon receiving $(T.coordinator, result(op_j), newChanges, T)$
return $result(op_j)$ to the client;
$execute(newChanges \cap localData(S_k), T)$;

Upon receiving commit/abort T **from the client**
if $op_j = abort$ **then**
multicast(abort,T);
else
$T.writeset = getWriteset(T)$;
if $T.writeset = \emptyset$ **then**
multicast(commit,T));
else
multicastTotal(commit, T);
end
end

Upon receiving (commit, T**) / (abort,** T**)**
if *executedAtSiteK(T,S_k)* **then**
T_k= transactionTable.getTX(T);
commit(T_k) / abort(T_k);
end

Upon receiving (commit,T**) in total order**
if *not executedAtSiteK(T,S_k)* **then**
T_j = getDummyTransaction($T.ST$);
else
T_j = transactionTable.getTX(T);
end
$T_j.CT$ = snapshot + 1;
if *validate(T_j)* **then**
snapshot = snapshot + 1;
if $T.writeset \cap localData(S_k) \neq \emptyset$ **then**
if *not executedAtSiteK(T_j,S_k)* **then**
execute(T_j, $T.writeset \cap localData(S_k)$);
DeleteDummyTransaction(T_j);
else
execute(T_j, $T.writeset \cap localData(S_k)$ not previously executed);
transactionTable.deleteTX(T_j);
end
commit(T_j) ;
committedTx = committedTx $\cup$ $\{T_j\}$;
end
T_j=startTransaction();
$T_j.ST = snapshot$;
storeDummyTransaction(T_j);
else
abort(T_j)
end

validate(T)
return $\nexists T_j \in$ commitedTX $\wedge$ $T_j.ST \leq T.CT \leq T_jCT \wedge$ $(T.writeset \cap T_j.writeset \neq \emptyset)$;

Figure 3. Replication protocol at site S_k

table, stock, with the item id and stock. The item table does not have the stock attribute. The database was fragmented in an even way across replicas. Furthermore, read only tables (item, author, country), order and order_line are replicated at every replica. The database population parameters were 100 emulated browsers and 10,000 items which generated a database of nearly 650MB. Due to the lack of space we present the results for the ordering mix that is the most challenging mix due to the higher update ratio (50% updates), and can show better the benefits of partial replication.

The experiments were run in a cluster of up to 14 homogeneous sites running Fedora Core 3 interconnected through a 100-MBit Ethernet switch. Each site is equipped with two processors AMD Athlon 2GHz, 1 GB of RAM running Postgres 7.2 and Yakarta Tomcat 5.5.9.

5.2. Scale Out

This experiment aims at quantifying the scale out for an increasing number of sites and at the same time compares the empirical results with the ones provided by the analytical model. The scale out measures the number of times the maximum throughput of a non-replicated system is multiplied by replicating the system. We ran the ordering mix of TPC-W for an increasing number of sites and different degrees of replication: full replication, minimum degree of partial replication (2 copies per object), and a high degree of partial replication (number of copies = N/2). For each configuration TPC-W was configured to inject an increasing load until the system was saturated (maximum throughput). The scale out is obtained dividing the maximum throughput of each configuration by the maximum throughput of one site.

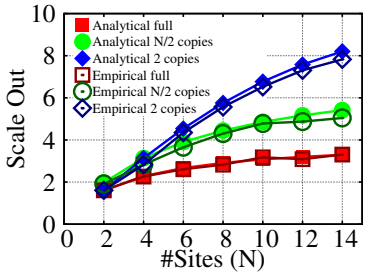


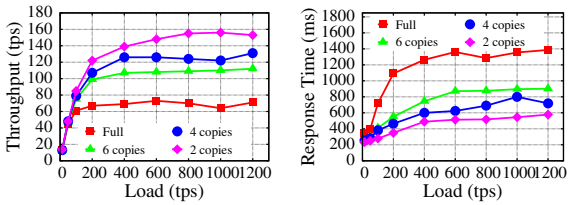
Figure 4. Analytical vs Empirical Scale Out

The empirical scale out for full replication was very poor (Fig. 4). It barely triplicated the maximum throughput of a non-replicated system with a configuration of 14 sites. With a high degree of replication ($\frac{N}{2}$), the scale out improves significantly, above 5 with 14 sites. What is more, from 10 sites to 14 sites, full replication increases the scale out by 0.25, whilst with $\frac{N}{2}$ copies the increase is close to 1. With the minimum degree of replication (2 copies) the scale out is substantially boosted, scaling till a scaleout factor of 8. This contrasts sharply with the low scale out of full replication.

The analytical curves shown in Fig. 4 correspond to the same setting as the empirical ones. As shown in the figure the analytical curves almost match the empirical ones and therefore, the model predicts very accurately the scale out.

5.3. Performance Evolution under Increasing Loads

In this experiment we study the behavior under increasing loads of full and pure partial replication. The performance results show both the throughput and response time (Fig. 5) for 12 replicas. Each graph includes curves for full replication, and 2, 4 and 6 copies of each object.



(a) Throughput (b) Response time

Figure 5. TPC-W Results. 12 sites

The maximum throughput with full replication is 70 tps (Fig. 5(a)). However, this maximum throughput is achieved when the system is already in saturation. The throughput increases up to a load of 100 tps (transactions per second), but beyond that load, the growth is very mild due to the system saturation. The response time for full replication is worse than expected looking at the throughput results (Fig. 5(b)). It increases very sharply till reaching 1200 ms. Then, it increases with a milder slope. This behavior has to do with the fact that TPC-W clients are synchronous. That is, they do not send a request until they receive the response to the previous one. Therefore, when the server saturates, the planned load (x axis) is not really achieved. That is why the response time does not increase exponentially when saturation is reached, as it would happen with asynchronous clients. The response time stabilizes at about 1.4 seconds.

The partial replication results (2, 4, and 6 replicas per object) show that the lower the number of replicas, the higher the throughput. With a configuration of 6 copies per object and 12 sites, each site holds a fraction $\frac{6}{12} = \frac{1}{2}$ of the database. The throughput is better than the one of full replication, reaching a maximum of 110 tps (Fig. 5(a)). This means a significant increase of 60% over full replication. The response time stabilizes at 900 ms (a 25% decrease over full replication). With 4 copies per object, the throughput

increases to a peak of almost 130 tps (20% increase compared to 6 copies). The response time is also smaller, between 100 and 200 ms for medium and high loads. Finally, the throughput with 2 copies is the best one, reaching a peak of 150 tps (Fig. 5(a)). This represents an increase of 210% over full replication. The response time reaches a maximum of 600 ms that is a 50% decrease over full replication. In summary, partial replication does not only scale significantly more than full replication, but also behaves much better under overloads as the ones shown in the graphs.

6. Conclusions

We have presented a new protocol for data replication that overcomes the two scalability limitations of current approaches: lack of concurrency due to 1CS and the overhead of full replication. The protocol combines 1CSI and partial replication. We have motivated analytically the need for partial replication and discarded hybrid replication as an alternative because of its lack of scalability. The protocol has been evaluated with TPC-W and the results have shown excellent scalability as predicted by the analytical model.

References

[1] F. Akal, C. Türker, H.-J. Schek, Y. Breitbart, T. Grabs, and L. Veen. Fine-grained replication and scheduling with freshness and correctness guarantees. In *Proc. of the Int. Conf. on Very Large Data Bases (VLDB)*, pages 565–576, 2005.

[2] C. Amza, A. L. Cox, and W. Zwaenepoel. Distributed versioning: Consistent replication for scaling back-end databases of dynamic content web sites. In *Int. Middleware Conf.*, pages 282–304, 2003.

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

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

[5] E. Cecchet, J. Marguerite, and W. Zwaenepoel. Partial replication: Achieving scalability in redundant arrays of inexpensive databases. In *(OPODIS)*, pages 58–70, 2003.

[6] C. Coulon, E. Pacitti, and P. Valduriez. Consistency management for partial replication in a high performance database cluster. In *Int. Conf. on Parallel and Distributed Systems (ICPADS)*, pages 809–815, 2005.

[7] K. Daudjee and K. Salem. Lazy database replication with snapshot isolation. In *Int. Conf. on Very Large Data Bases (VLDB)*, pages 715–726, 2006.

[8] S. Elnikety, W. Zwaenepoel, and F. Pedone. Database replication using generalized snapshot isolation. In *Symp. on Reliable Distributed Systems (SRDS)*, 2005.

[9] S. Gancarski, H. Naacke, E. Pacitti, and P. Valduriez. The leganet system : Transaction processing in a database cluster. *Information Systems Journal*, 2007.

[10] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. In *ACM SIGMOD Conf.*, 1996.

[11] J. Holliday, D. Agrawal, and A. E. Abbadi. Partial database replication using epidemic communication. In *Int. Conf. on Distributed Computing Systems (ICDCS)*, 2002.

[12] L. Irún-Briz, H. Decker, R. de Juan-Marín, F. Castro-Company, J. E. Armendáriz-Iñigo, and F. D. Muñoz-Escoí. Madis: A slim middleware for database replication. In *Int. Euro-Par Conf.*, pages 349–359, 2005.

[13] R. Jiménez-Peris, M. Patiño-Martínez, G. Alonso, and B. Kemme. Are quorums an alternative for data replication? *ACM Trans. on Database Systems*, 28(3):257–294, 2003.

[14] B. Kemme. *Database Replication for Clusters of Workstations*. PhD thesis, ETHZ, 2000.

[15] B. Kemme and G. Alonso. Don't be lazy, be consistent: Postgres-r, a new way to implement database replication. In *Int. Conf. on Very Large Data Bases (VLDB)*, 2000.

[16] Y. Lin, B. Kemme, M. Patiño-Martínez, and R. Jiménez-Peris. Middleware based data replication providing snapshot isolation. In *ACM SIGMOD Conf.*, pages 419–430, 2005.

[17] M. Nicola and M. Jarke. Performance modeling of distributed and replicated databases. *IEEE Trans. Knowl. Data Eng.*, 12(4):645–672, 2000.

[18] E. Pacitti, C. Coulon, P. Valduriez, and M. T. Özsu. Preventive replication in a database cluster. *Distributed and Parallel Databases*, 18(3):223–251, 2005.

[19] C. L. Pape, S. Gançarski, and P. Valduriez. Trading freshness for performance in a cluster of replicated databases. In *CoopIS*, pages 14–15, 2003.

[20] M. Patiño-Martínez, R. Jiménez-Peris, B. Kemme, and G. Alonso. Scalable Replication in Database Clusters. In *Int. Conf. on Distributed Computing (DISC)*, 2000.

[21] M. Patiño-Martínez, R. Jiménez-Peris, B. Kemme, and G. Alonso. MIDDLE-R: Consistent database replication at the middleware level. *ACM Trans. Comput. Syst.*, 23(4):375–423, 2005.

[22] F. Pedone, R. Guerraoui, and A. Schiper. The database state machine approach. *Distributed and Parallel Databases*, 14(1), 2003.

[23] C. Plattner and G. Alonso. Ganymed: Scalable replication for transactional web applications. In *Int. Middleware Conf.*, pages 155–174, 2004.

[24] A. Sousa, A. C. Jr., F. Moura, J. Pereira, and R. Oliveira. Evaluating certification protocols in the partial database state machine. In *Int. Conf. on Availability, Reliability and Security (ARES)*, pages 855–863, 2006.

[25] A. Sousa, R. Oliveira, F. Moura, and F. Pedone. Partial replication in the database state machine. In *Int. Symp. on Network Computing and Applications (NCA)*, page 298, 2001.

[26] Transaction Processing Performance Council. *TPCW Benchmark*. http://www.tpc.org/tpcw.

[27] S. Wu and B. Kemme. Postgres-R(SI): Combining replica control with concurrency control based on snapshot isolation. In *Int. Conf. on Data Engineering (ICDE)*, 2005.