Comparison of Database Replication Techniques Based on Total Order Broadcast

Matthias Wiesmann and André Schiper, *Member*, *IEEE*

Abstract—In this paper, we present a performance comparison of database replication techniques based on total order broadcast. While the performance of total order broadcast-based replication techniques has been studied in previous papers, this paper presents many new contributions. First, it compares with each other techniques that were presented and evaluated separately, usually by comparing them to a classical replication scheme like distributed locking. Second, the evaluation is done using a finer network model than previous studies. Third, the paper compares techniques that offer the same consistency criterion (one-copy serializability) in the same environment using the same settings. The paper shows that, while networking performance has little influence in a LAN setting, the cost of synchronizing replicas is quite high. Because of this, total order broadcast-based techniques are very promising as they minimize synchronization between replicas.

Index Terms—Distributed databases, performance attributes, distributed programming.

1 Introduction

Database replication is traditionally handled in two ways, either with eager replication or with lazy replication [1]. Eager replication implies an atomic commitment protocol and is slow and deadlock prone. Lazy replication is efficient, but does not enforce consistency between the replicas. To address this problem, replicated databases based on group communication have been proposed for some time [2], [3], [4]. Those techniques, which typically replace the two-phase commit protocol [5], [6], [7], [8], promise better performance [9], [10], [11], [12], [13], [14], [15], [16], [17], [18] and less deadlocks [19] than classical eager replication, while maintaining strong consistency.

Techniques based on group communication typically rely on a primitive called *total order broadcast*.[1] This primitive ensures that messages are delivered reliably and in the same order on all replicas. While most techniques described in the literature use total order broadcast, they do not use it in the same way. Some techniques use a single total order broadcast, while others need, additionally, a reliable broadcast. Moreover, some techniques do the broadcast at the start of the transaction and some others do it at the end.

Although the performance evaluation of techniques based on total order broadcast can be found in the literature [13], [16], they are usually used to present new techniques and, thus, only compare new techniques to traditional ones, like distributed locking. For example, in [9], [11], the certification-based database replication technique is presented with a rough performance evaluation. The weak voting technique is presented along with other techniques that do not enforce one-copy serializability with a performance evaluation in [10]. The performance of a prototype based on Postgres is presented in [16]. A set of replication techniques based on group communication is presented with a performance evaluation in [13]. Some of the techniques, explained in [4], are similar to the ones presented in [9] and [10].

The performance of a restricted version of active replication (single operation) is presented in [20] and compared to voting replication based on two-phase commit, but without any indication about the workload. The performance of the certification technique based on a CORBA middleware is presented in [21].

Because these papers typically compare only a new technique to traditional techniques like distributed locking, understanding the relationship between all techniques and comparing their performance is difficult. In this paper, we present a performance comparison of the three main database replication techniques based on total order broadcast. We compare these techniques to each other and to traditional database replication schemes like distributed-locking, lazy replication, and primary copy. All techniques were implemented in the same simulation environment and tested under the same conditions. This allows us to compare the performance of the different techniques and to understand their advantages and their weaknesses. Another shortcoming of existing performance evaluations is the way they model group communications. In previous simulations, the execution of group communication protocols (e.g., of total order broadcast) is simulated as a single atomic operation, that is, the network and the group communication stacks are, *de-facto*, considered as a single shared resource. This hides the complexity of the group communication subsystem and ignores the interactions between the communication system and database system

1. Holliday et al. [18] present an exception: It relies on causal order delivery of messages and epidemic communication.

• M. Wiesmann is with ATLAS-TDAQ, Division PH, European Organisation for Nuclear Research, Ch-1211 Genève 23, Switzerland. E-mail: matthias.wiesmann@a3.epfl.ch.
• A. Schiper is with the Distributed Systems Laboratory, Federal Institute of Technology in Lausanne, Switzerland, Ch-1015 Lausanne, Switzerland. E-mail: andre.schiper@epfl.ch.

Manuscript received 30 Apr. 2004; revised 28 Aug. 2004; accepted 6 Oct. 2004; published online 17 Feb. 2005.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0125-0404.

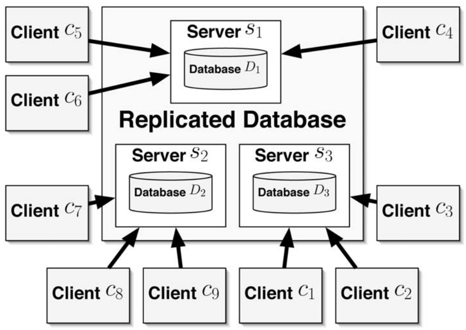


Fig. 1. Database replication model.

(for instance, it ignores that the CPU is used both for processing transactions and sending messages). Our simulations are based on a finer model: low-level resources, e.g., the network, are simulated and group communication protocols are executed inside the simulator. Simulating the system at a finer granularity makes it possible to understand what role low-level resources play in the performance of different replication techniques. This way, we can understand the influence of the network performance on the overall system. The results show that the design of replication techniques has a serious impact on the performance of the system, more so than the number of message "rounds." We also show that while the cost of a total order broadcast is low, communication has an impact on performance because of synchronization issues.

The rest of the paper is structured as follows: Section 2 introduces the communication and database model. Section 3 introduces the different replication techniques and discusses how they relate to techniques presented in the literature. Section 4 presents our simulator and the general simulation settings. Section 6 presents the different experiments and their results. Related work is discussed in Section 7. Section 8 concludes the paper.

2 Model

We assume a set of servers $S = \{s_1, \dots, s_n\}$ and a database D. Each server s_i contains a full copy D_i of the database D (Fig. 1). Server s_i hosts a local transaction manager; this manager can execute local transactions, that is, transactions that update D_i. The local transaction manager ensures the ACID properties of local transactions.

A database replication protocol runs on all servers $s_i \in S$. The correctness criterion is one-copy serializability [22]: All copies of D are kept synchronized at all times. We also consider a set of clients $C = \{c_1 \dots c_m\}$. Clients are the source of transactions. In order to process a transaction t, a client c connects to a server s_d and submits transaction t to s_d. We call the server s_d the *delegate* for transaction t. For some replication schemes (primary copy), only one server (the primary) can act as the delegate server.

The replication techniques we compare in this paper rely on a group communication primitive called *total order broadcast* (the primitive is also called *atomic broadcast*). Roughly speaking, messages that are broadcast using this primitive (which will be denoted by *TO-broadcast*) are delivered (which will be denoted by *TO-deliver*) in the same order on all destination processes. For example, if two processes TO-broadcast messages m and m', all destination processes TO-deliver either m before m', or all TO-deliver m' before m. The precise specification of total order broadcast is not relevant here and is therefore omitted.[2] The implementation of total order broadcast is also outside the scope of the paper. However, it is important to note that the total order broadcast primitives that we consider here order messages even though processes might crash.

We also assume the existence of a two-phase atomic commitment protocol (2PC). This protocol is used in the context of distributed locking.

3 Replication Techniques

Different techniques for database replication based on total order broadcast have been proposed. In this paper, we consider the three most relevant techniques: *active replication*, *certification based replication*, and *weak voting replication*. The performance of these techniques will be compared to the two widely used techniques that do not rely on group communication, namely, *primary copy replication* and *lazy replication*.

The three techniques based on total order broadcast ensure one-copy serializability and have in common the following features:

- They do not rely on an atomic commitment protocol.
- They are update-everywhere replication techniques [1], i.e., a client c can connect to any server to submit its transaction.
- They require $\mathcal{O}(1)$ network interactions [24].

A technique is said to have $\mathcal{O}(1)$ network interactions if the number of interactions between servers in order to process a transaction is constant, i.e., independent of the number of operations in the transaction. Other techniques, i.e., those that require a number of interactions in the order of the number of operations in the transaction, are considered to be very slow, especially if each interaction is a total order broadcast. Experiments confirm that those techniques perform very poorly [25] and, therefore, are not considered here.

3.1 Active Replication

This technique is a direct application of the active replication technique [26] (also called *state machine replication*) to database replication: The whole transaction is put into a message, and the message is broadcast (total order broadcast) to the servers. The only difference here is that the client does not issue the total order broadcast. In a database setting, clients are usually "thin" and do not have access to group communication primitives. In our case, the client sends the transaction to one delegate server that issues the total order broadcast on behalf of the client.

Fig. 2 illustrates the communication scheme of this technique. When the delegate server s_d receives a transaction t from a client c, the server s_d broadcasts t to all servers using a total order broadcast. All servers then deliver t and process t; the processing of t must be deterministic. So each

2. The specification can be found in [23].

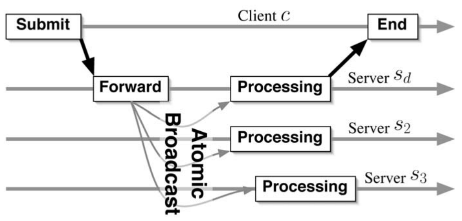


Fig. 2. Active replication scheme.

server s_i serializes the transaction in the same position, and if one server aborts transaction t, then all servers also abort it. Fig. 3 shows the pseudocode for active replication. An early version of this technique is presented in [27] and another variant is presented in [28].

This technique is different from the two following ones in the sense that it requires a fully deterministic execution of transactions, i.e., that the point of determinism [29] is at the beginning of the transaction. This model excludes interactive transactions: All operations must be known at the start of the transaction. Another issue with this technique is that full transactions are sent to all replicas, including both read and write sets—this means that reads are executed by all servers. This negates one of the benefits of replication: load distribution for read operations.

3.2 Certification-Based Replication

This technique is described in [30] and [11] under the "database state machine" name. To avoid ambiguities with active replication technique, we call it *certification-based replication*. Technique A4 described in [25] uses a similar approach.

Fig. 4 illustrates this technique: When the delegate server s_d receives a transaction t from a client c, the server s_d executes transaction t but delays the write operations. When commit time is reached, transaction t is broadcast to all servers using a total order broadcast. Upon delivering the message that contains t, each server executes a deterministic certification phase. Certification decides if transaction t can commit or must abort. Fig. 5 shows the pseudocode.

This technique shares the communication requirements with active replication: Only one total order broadcast is needed per transaction. This technique does not have the drawbacks of active replication: It can handle interactive transactions. On the other hand, this technique is *optimistic*: Conflicting transactions might be processed only to be aborted at the certification phase. This means such techniques are effective in low-conflict situations.

```
task Forward
    {Executed by server $s_i$}
    when receive $t$ from client $c$
        TO-broadcast($\langle t, c, s_d \rangle$) to $S$
    end when
end

task Processing
    {Executed by server $s_i$}
    when TO-deliver $\langle t, c, s_d \rangle$
        process($t$)
        $reply \leftarrow$ try-commit($t$)
        if $s_d = s_i$ then
            send($reply$) to $c$
        end if
    end when
end
```

Fig. 3. Active replication—pseudocode for delegate server s_d (left) and any server s_i (right).

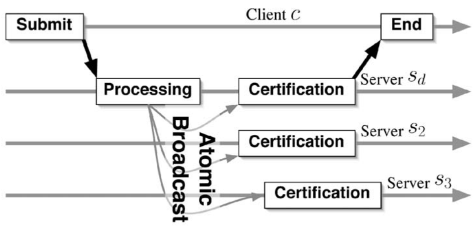


Fig. 4. Certification-based replication scheme.

3.3 Weak Voting Replication

This technique is described in [10], [31] under the "serializability protocol" name. As all techniques in this paper ensure one-copy serializability, we call it *weak voting* replication technique to be consistent with the classification presented in [29]. Technique A3 presented in [25] is similar to weak voting.

Fig. 6 illustrates this technique: when the delegate server s_d receives a transaction t from a client c, the delegate server executes transaction t but delays the write operations. When commit time is reached, the write set of transaction t is broadcast to all servers using a total order broadcast. Upon delivering the message that contains t's write, the delegate sever s_d can determine if conflicting transactions have been committed. Based on this information, the delegate server does a new broadcast containing the outcome of the transaction (commit or abort)—this broadcast may not be totally ordered, but must be reliable. Fig. 7 shows the pseudocode for this technique.

This technique is close to certification-based replication. The main difference is that the deterministic certification mechanism is replaced by a weak voting phase, i.e., the delegate takes the decision to commit or to abort. As it can rely on information available only to the delegate, the certification mechanism can be more accurate, but at the price of an extra broadcast. The voting is said to be weak as only the delegate server can decide on the outcome of the transaction. Other servers cannot influence this decision and must abide by the delegate's decision.

3.4 Primary Copy Replication

Primary copy is a traditional database replication technique [1]; it is presented here for comparison purposes. The technique described here is a *cold standby, nonvoting* primary copy technique [29]. This technique is also called *passive replication* in the distributed system community.

Fig. 8a illustrates the technique: All transactions are routed to a primary server s_p and the other servers (called backups) do not accept transactions. The transaction is processed on the primary and the updates are sent to the backups using a reliable broadcast. The serialization order and the termination (abort or commit) are decided on the primary server. Fig. 9 shows the pseudocode.

3.5 Lazy Replication

Lazy replication has been proposed as an alternative to eager update everywhere techniques, such as distributed locking [1]. When used in an update everywhere setting, lazy replication can violate the ACID properties: Conflicting transactions might be accepted and committed. In this case,

```
task Processing
  {Executed by delegate server $s_d$}
  when receive trans. $t$ from client $c$
    execute trans. $t$
    if aborted($t$) then
      send($aborted$) to $c$
    else
      TO-broadcast($\langle readSet_t,\ writeSet_t,\ c,\ s_d \rangle$)
      to $S$
    end if
  end when
end
```

```
task Certification
  {Executed by server $s_i$}
  when TO-deliver $\langle readSet_t,\ writeSet_t,\ c, s_d \rangle$
    $status \leftarrow$ certify($readSet_t$,$writeSet_t$)
    if $status = commit$ then
      if $s_d \neq s_i$ then
        execute $writeOperations_t$
      end if
      commit($t$)
      if $s_d = s_i$ then
        send($committed$) to $c$
      end if
    else
      abort($t$)
      if $s_d = s_i$ then
        send($aborted$) to $c$
      end if
    end if
  end when
end
```

Fig. 5. Certification-based replication—pseudocode for delegate server s_d (left) and any server s_i (right).

a reconciliation procedure is needed. Here, we ignore the issue of conflicting transactions. Lazy replication is considered only for comparison purposes: Lazy replication without conflict resolution represents the minimal amount of synchronization and communication needed and is optimal from this point of view.

Fig. 8b illustrates the technique: When a transaction reaches a server (s_1, s_2, or s_3), it is first processed on this server. Once this is done, the updates are broadcast to the other servers. Fig. 10 shows the pseudocode.

4 Simulator

All experiments were performed using a discrete event simulator. The simulator is written in C^{++} and relies on the *C-sim* discrete event simulation engine [32].

The overall simulator can be roughly divided into two conceptual parts: the clients and the servers. The clients represent the source of transactions: They generate transactions according to certain parameters, send them to servers, and collect end-to-end performance data. The servers implement the whole replicated database logic, including the local database, the group communication system, and the replication strategies. The simulation concentrates on low-level aspects. High-level issues, like transaction parsing and optimizing, are not considered.

4.1 Server Structure

The architecture of the servers follows the ideas presented in Section 2: Each replica hosts both a local database manager and a group communication stack. A replication

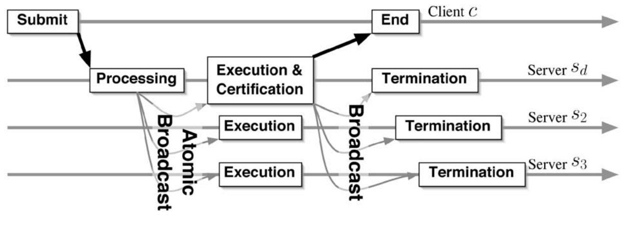


Fig. 6. Weak voting replication scheme.

technique runs on top of these two services. Servers are therefore structured in the following way:

1. the low-level server module,
2. the communication module,
3. the database module, and
4. the database replication module.

4.1.1 Low-Level Server Module

The low-level server module represents the hardware of one server. There is one instance of this module for each server in the system. Each server machine is simulated using two basic resources: the CPU and the disks. Those resources are used by other high-level modules of the simulator. The CPU resource models the processing units. The disk's resources are used by the database module. Basic input/output operations use both the CPU and the disk resources.

These low-level resources (CPUs and disks) are simulated as C-sim resources. High-level operations, like the execution of a network protocol or of a transaction operation, are implemented and executed by the simulator.

4.1.2 Communication Module

The Communication Module models all network interactions. There is one instance of this module for each server. At a low level, both point-to-point and multicast messages are modeled. Group communication primitives described in Section 2 are implemented on top of those low-level messaging facilities. The use of high-level primitives like total order broadcast will therefore result in a simulated execution of a total order broadcast protocol using low-level messages. The algorithms are simulated in failure-free runs—the most common case in normal operating conditions.

The Communication Module relies on two kinds of simulated resources: the CPU and the network. The CPU is the resource exported by the Server Module. The network is a resource shared by the Communication Modules of all servers: It represents the network between the servers.

The sending of a message is modeled in three steps: First, the outgoing message is processed on the sending node, then the message transits through the network, and, finally, the message is processed on the receiving node [33]. The

```
task Processing
  {Executed by delegate server $s_d$}
  when receive transaction $t$ from client $c$
    execute transaction $t$
    if aborted($t$) then
      send($aborted$) to $c$
    else
      TO-broadcast($\langle writeSet_t, c, s \rangle$) to $S$
    end if
  end when
end

task Execution
  {Executed by server $s_i$}
  when $\langle writeSet_t, c, s_d \rangle$
    if $s_d = s_i$ then TO-deliver
      $status_t \leftarrow$ vote(t)
      R-broadcast($status_t$) to $S$
      send($status_t$) to $c$
    else
      execute $writeOperations_t$
    end if
  end when
end

task Termination
  {Executed by server $s_i$}
  when R-deliver($status_t$)
    if $status_t = commit$ then
      commit(t)
    else
      abort(t)
    end if
  end when
end
```

Fig. 7. Weak voting-based replication—pseudocode for delegate server s_d (left) and any server s_i (middle and right).

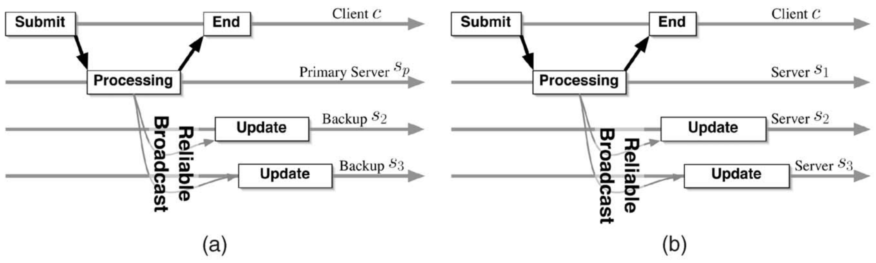


Fig. 8. (a) Primary copy replication technique and (b) lazy replication.

```
task Processing
  {Executed by primary}
  if primary($s$) then
    when receive transaction $t$ from client $c$
      process transaction $t$
      $status_t \leftarrow$ try-commit(t)
      if $status_t$ = committed then
        $update_t \leftarrow$ updates done by $t$
        R-broadcast($update_t$) to $S \setminus s$
      end if
      send($status_t$) to $c$
    end when
  end if
end

task Update
  {Executed by backup}
  if $\neg$ primary($s$) then
    when R-deliver $update_t$
      process($update_t$)
    end when
  end if
end
```

Fig. 9. Primary copy replication—pseudocode for the primary server (left) and the backups (right).

```
task Processing
  {Executed by all servers}
  when receive transaction $t$ from client $c$
    process transaction $t$
    $status_t \leftarrow$ try-commit(t)
    if $status_t$ = committed then
      $update_t \leftarrow$ updates done by $t$
      R-broadcast($update_t$) to $S \setminus s$
    end if
    send($status_t$) to $c$
  end when
end

task Update
  {Executed by all servers}
  when R-deliver $update_t$
    process($update_t$)
  end when
end
```

Fig. 10. Lazy copy replication—pseudocode for the server processing the transaction (left) and all servers (right).

resources queues are handled by a FIFO policy. This means that three resources are involved in simulating the sending of one message: the CPU resource of the sending machine, then the network resource, and, finally, the CPU resource on the receiving machine. This way we can model network and CPU contention between messages, but also contention between the communication system and the database system.

4.1.3 Database Module

The database module simulates a single database system. This modules includes a lock manager and an I/O manager. The lock manager offers lock queues for each item in the database. The lock queues can be used to enforce strict two-phase locking but also support the more complex mode needed by the techniques described in Section 3. The I/O manager handles operations to read and write items of the database. Data items are distributed on the different disks of the machine (each disk holds a partition of the data). The I/O manager also simulates the cache system.

4.1.4 Database Replication Module

This module represents the database replication strategy. There is one instance of this module for each server.

TABLE 1
Simulator Parameters

Parameter	Value
Items in the database	10'000
Number of Servers	9
Number of Clients per Server	2 or 4
Disks per Server	2
CPUs per Server	2
Transaction length	10 – 20 Operations
Operation is a write	50%
Operation is a query	50%

Parameter	Value
Buffer hit ratio	20%
Time for a read	4 – 12 *ms*
Time for a write	4 – 12 *ms*
CPU Time used for an I/O operation	0.4 *ms*
Time for a message on the Network	0.5 or 0.07 *ms*
CPU time to send/receive a message	0.5 or 0.07 *ms*
Time for a broadcast on the Network	0.5 or 0.07 *ms*
CPU time to send/receive a broadcast	0.5 or 0.07 *ms*

Depending on the replication strategy, a different implementation of this module is used. Each replication scheme is represented by a concrete subclass of the abstract database replication class. Each of the techniques described in Section 3 has been implemented. In addition, a distributed locking replication technique has been implemented for comparison.

4.2 Client Module

The clients of the replicated database system were also modeled. Clients are simple sources of transactions. Clients submit one transaction, wait until the transaction is processed, sleep for some time, and start the work cycle again. A client can only submit one transaction at a time—multiple sources are modeled by multiple clients. The most important parameter of clients is the time between submitting transactions. This parameter is the time between the start of two transactions. If a client starts a transaction at time t_1, once the transaction is finished, the client waits until time $t_1 + d$ before issuing the next transaction. The variable d is a random variable with an exponential distribution so that the mean of d matches the requested interval between transactions. By adding some randomness, we create "bursty" load situations.

Typically, a server has many clients attached—each server has the same number of clients attached. Clients do *not* consume any network bandwidth: We consider that the network interface that interconnects the servers is separate from the network interface used to communicate between clients and servers.[3] Clients gather all the performance data and compute statistics. The clients also control simulation runs and experiment settings, typically stopping the simulation when the results obtained fit into a certain confidence interval.

5 Settings

We performed an extensive set of simulations to compare the different replication techniques. All techniques shared the same infrastructure layer and the same operational parameters. The main performance metric is the mean response time observed by clients. Simulation ran until this value was at least within a 95 percent confidence interval with a half width of 5 percent of the mean response time; often, a better confidence interval was obtained. To avoid skewed measures due to initial startup factors [34], the response times associated with the first 500 transactions were discarded.

Simulations were run with the operational parameters in Table 1, with one or two parameters being the variables of the experiment. There are two load settings. The first consists of 18 clients connected to nine servers with two clients per server. We call this the *medium load* setting. The other setting consists in 36 clients connected to nine servers, with four clients per server. We call this the *high load* setting. In the case of primary copy replication, the medium load consists of 18 clients connected to the primary with eight back-ups; the high load consists of 36 clients connected to the primary.

The database settings were based on numbers in the literature [35], [36], [13], [14]. The data set contains 10,000 items. Servers are composed of two CPUs and two data disk units. Each CPU has access to any disk, but only one CPU can access a disk at a time. Data items are distributed on the different data disks. The transaction length is uniformly distributed between 10 and 20 operations. Each transaction is either an update transaction (50 percent) or a query transaction (50 percent). Queries contain only read operations, while updates contain both read (50 percent) and write (50 percent) operations. Read and write operations access each one item of the database (uniform distribution). A write operation might overwrite a read data, with uniform probability $1/database\_size$.

Operating a read or a write operation uses the disk between 4 and 12 ms (uniform distribution). Read operations have a 20 percent chance of hitting the cache and, therefore, cause no disk usage. Each input/output operation (read and write) has a CPU overhead of 0.4 ms.

Network settings are based on observed values on a cluster of PC machines. Each machine is equipped with a 733 MHz processor and a 100 Mbit/s full duplex network interface. The machines are connected using an Ethernet hub. Network performance was estimated using the Neko framework [37] by sending short messages of approximately 256 bytes. Sending a point-to-point message consumes 0.07 ms of CPU at the sender, 0.07 ms of the network resource, and 0.07 ms of CPU at the receiver. We assume a low-level multicast facility (like IP-multicast) with which we can send a multicast in a single operation. The cost is 0.07 ms on the network and 0.07 ms of CPU at both the sender and the receiver. We also did experiments using settings that roughly represent a 10 Mbit/s network with slower network adapters. In this case, the cost of sending and receiving a message is 0.5 ms, and the cost of message transmission is 0.5 ms. We call the first setting (100 Mbit/s)

3. Clients' requests do not consume the CPU resource when being delivered; this is considered part of the transaction parsing and optimizing and is not considered here.

TABLE 2
General Simulator Settings

Setting	Value
Medium load	10 - 20 transactions per second
High load	20 - 40 transactions per second
Slow network	10 *Mbit/s*
Fast network	100 *Mbit/s*

the *fast network* setting and the second the *slow network* setting (10 Mbit/s).

6 Experiments

We performed several experiments to compare the different database replication strategies. This section presents the different experiments and their results. Each experiment explores a different aspect of a replicated database.

6.1 General Performance Comparison

6.1.1 Description of the Experiment

The first experiment aims at comparing the performance of the different replication techniques under moderate and high load, with a medium number of servers (9).

We compared the performance of the different replication techniques by varying the transaction load. This was done by changing the time interval that clients wait before issuing a new transaction (this time is measured between two transaction starts). The time between transactions was between 900 ms and 1,800 ms in 50 ms increments. The experiment was done once with two clients per server and once with four clients per server. With two clients, the load varied between 10 and 20 transactions per second; with four clients, the load varied between 20 and 40 transactions per second. This experiment was done once with the slow network settings (10 Mbit/s) and once with the fast network settings (100 Mbit/s). This gave four basic settings which are summarized in Table 2. The other operational parameters for the experiment are described in Section 5.

6.1.2 Results

Medium Load, Slow Network. Fig. 11a illustrates the result of the experiment with two clients per server and the slow network. The experiment shows the performance of the different replication schemes under the same operating conditions. The X axis represents the load of the system expressed in transactions per second. The Y axis represents the average response time of committed transactions. Each replication technique is represented by one performance curve.

The observed conflict rate in the local database managers changed depending on the replication technique: In a medium load situation, it was below 5 percent, at high-load it would reach 20 percent.[4] The abort rate for all techniques was below 5 percent, and is not shown.

The lazy replication scheme does not enforce consistency. It is presented to give a reference in the form of the best performance that can be achieved in this setting. Because this technique does little extra processing and does no synchronization at all, its performance is very good and is not affected by changes in the load. Basically, the performance of lazy replication with n servers is equivalent to n nonreplicated servers getting $1/n$ of the load.

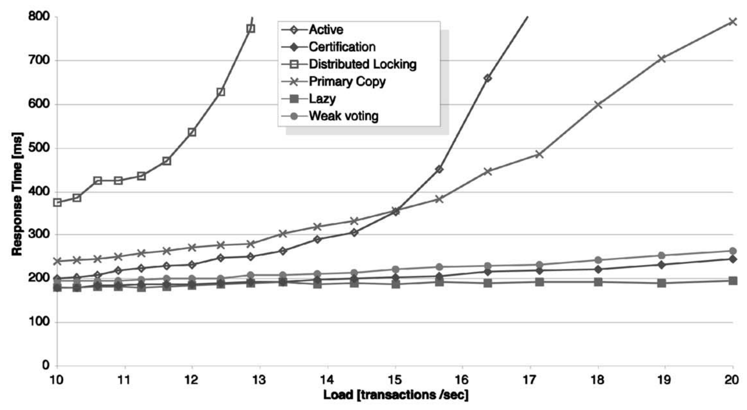
In the left part of the X-axis, most techniques have very similar performance: Certification, weak voting, active, and primary copy have basically the same response time. Lazy replication does outperform those techniques by a rather small margin. The main advantage of lazy replication is very good load balancing, but as the load is limited, this results in a small difference. This explains why there is no noticeable difference between techniques that do load balancing (certification-based replication), and others that do not (primary copy and active replication).

Distributed locking has a response time that is 60 percent higher. This is caused by contention on the network, as distributed locking sends one broadcast per operation. This leads to a situation where the network becomes the bottleneck—in the simulator, the usage rate of the network resource quickly reached 100 percent.

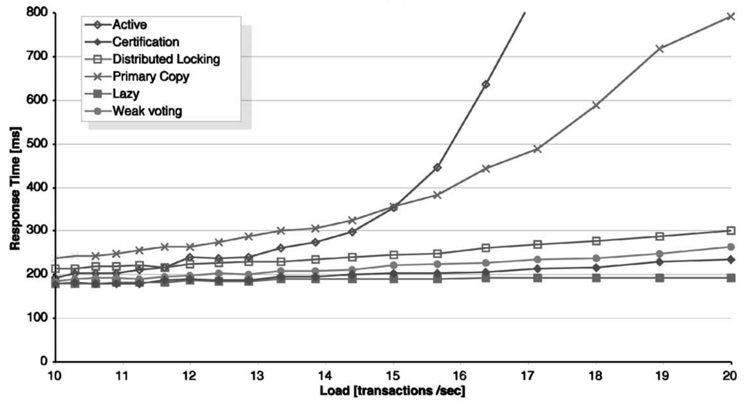
As the number of transactions per second increases, we can see that the response time of all techniques increases, yet the relative performance of the different techniques changes. The response time of distributed locking increases and shows that there is an asymptotic limit to throughput around 13 transactions per second. This is caused by high network contention. A similar phenomenon is observable with active replication, which has a maximum throughput of around 16.5 transactions per second. The reason, this time, is not the network (at the highest load, the network usage rate was around 6 percent), but the cost of getting the locks in total order. With active replication, transactions need to get their locks in the order defined by the total order broadcast primitive. So, if transaction t_1 is delivered before transaction t_2, transaction t_1 must obtain all its locks before t_2 can ask for its locks. If t_1 is blocked due to a lock already held, then t_2 is also blocked. Moreover, active replication does no load balancing: All transactions are executing on all replicas. Those two factors (serialization of locking phase and absence of load balancing) are the bottleneck of active replication: High load causes lock contention, and lock contention slows down the locking phase. Primary copy replication also has a load problem, as most of the work is done on the same server, so performance tends to degrade as the load increases.

The performance of the weak voting technique and the certification-based technique remains very close to lazy replication. It is interesting to note that the response time of the weak voting replication is around 15 ms higher than the response time of the certification-based technique. This is explained by the differences between both techniques. In the weak voting techniques, before a transaction can commit, all replicas must wait for the delegate to decide of the outcome of the transaction. This does not happen in the certification based technique, where all replicas decide deterministically on the outcome of the transaction due to the use of total order broadcast. Still, the difference between certification and weak voting techniques cannot be explained only by the cost of a broadcast which "costs" around 1.5 ms; the 15 ms difference is mostly explained by the cost of coordination: All servers need to wait for the delegate to finish processing

4. The observed conflict rate was calculated by marking transactions that had to wait on a lock held by another transaction during their execution.



(a)



(b)

Fig. 11. Overall performance medium-load (a) slow network and (b) fast network.

t in order to be able to terminate t locally. In other words, the time needed to broadcast the data is negligible, but the time lost waiting is not.

Medium Load, Fast Network. Fig. 11b shows the results of the same experiment but with a fast network (100 Mbit/s). While most techniques behave in a very similar fashion, one technique has a very different performance curve: the distributed locking technique. This makes sense: As the network was the bottleneck of this technique in the previous experiment, a faster network leads to better performance. While performance of distributed locking is much better with a fast network, the response time is still higher than the response time of group communication-based techniques (≈ 50 ms). This difference can only be partly explained by network usage (it takes around 0.2 ms to send a message): With an average of 15 operations per transaction and two messages per operations (request lock and confirm), this gives a network cost of around 6 ms. This means that the major part of the performance penalty is related to the way transactions are processed. This processing overhead is probably also partially responsible for the overhead of the distributed locking technique with the slow network (previous experiment), but the majority of the overhead with the slow network was due to network contention.

Another interesting thing to note when comparing Figs. 11a and 11b is that the difference between the certification-based technique and the weak voting technique stays roughly the same. This shows that the difference between those two techniques is not related to the use of the network.

Network Cost and Distributed Locking. To understand the relationship between the performance of the network and the response time of the distributed locking technique, we measured the response time of the technique while changing the performance of the network. The result of this experiment is illustrated in Fig. 12. In this experiment, we changed the "cost" of sending a message on the network and plotted the response time of the distributed locking technique with an interval between transactions of 1,500 ms, resulting in a load of 12 transactions per second. The X axis represents the cost of sending one message, i.e., a value x means that the CPU needs x milliseconds to send (and to receive) a message and the network needs x milliseconds to transmit the message. The value $X = 0.07$ ms corresponds to the settings of the fast network and $X = 0.5$ ms corresponds to the slow network. The Y axis represents the response time in milliseconds. The graph shows that the response time of the distributed locking technique increases with the cost of networking operations. We also see that this curve is not linear: When the network becomes the bottleneck, the response time increases much more (this can be seen when $X = 0.35$ ms). For this value, the network facility is used at 67 percent and, on average, processes had to wait more than 2 ms to access the network resource.

High Load, Fast and Slow Network. Fig. 13a shows the result of the experiment with four clients and the slow network. Only the curves of the weak voting, certification,

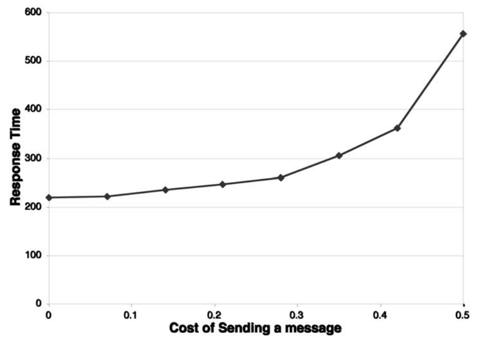


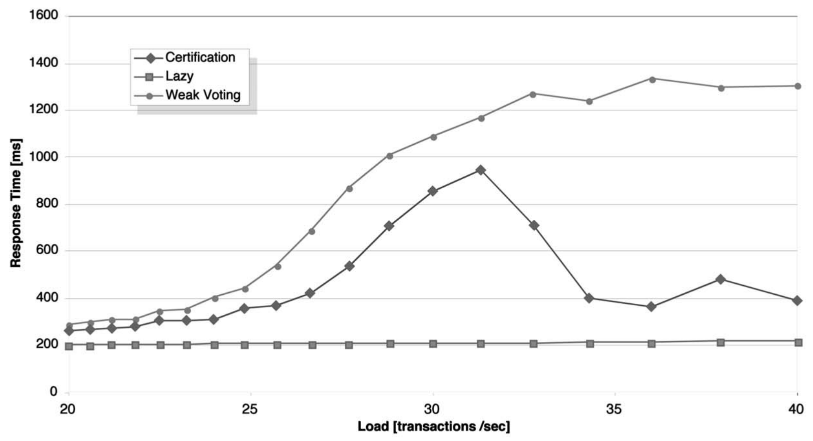
Fig. 12. Influence of network performance on the distributed locking technique.

and lazy techniques were plotted, the others being unable to sustain more than 20 transactions per second. Fig. 13b shows the results of the experiment with four clients and the fast network.

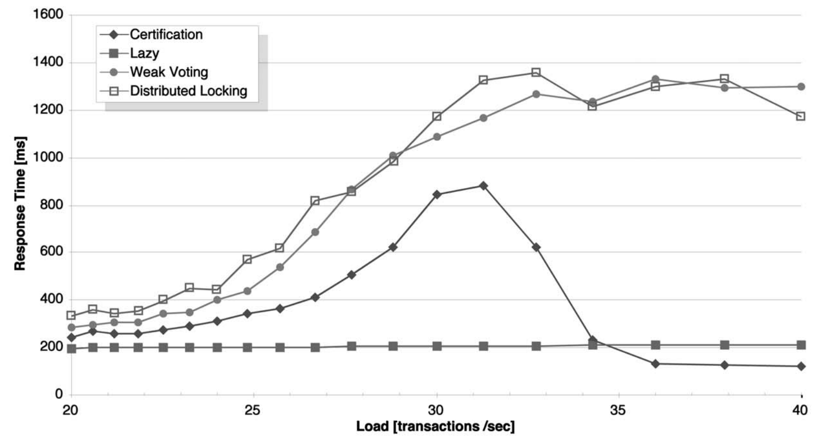
Figs. 13a and 13b are very similar, except for the fact that the distributed locking technique is not present in the slow network case (in this setting distributed locking cannot sustain such high loads). As the main difference between both experiments is the behavior of the distributed locking technique, we will concentrate the discussion on the experiment with the fast network so as to include distributed locking. Except for distributed locking, the influence of network performance on overall performance is weak.

With both the fast and the slow networks, we see that performance degrades steadily when the load is around 30 transactions per second. At first glance, the behavior of the certification technique is much better that distributed locking and weak voting: the response time stays significantly lower even when the system starts to become overloaded. In fact, in the fast network setting, when the load reaches 32 transactions per second, the response time decreases. When the load is above 34 transactions per second, the response time of the certification technique is within the confidence interval of lazy replication.

The difference lies in the abort rate of the different techniques. While in moderate load situations, the abort rate of the different techniques was marginal, in high-load situations, it became significant. Fig. 14 shows the abort rate of certification, weak voting, and distributed locking techniques in high-load situations (fast network)—the parameters are the same as in Fig. 13b. We see that, while the response time of certification-based replication is low, its abort rate is significantly higher. The overload of the system yields a high conflict rate in the certification phase;



(a)



(b)

Fig. 13. Overall performance high-load (a) slow network and (b) fast network.

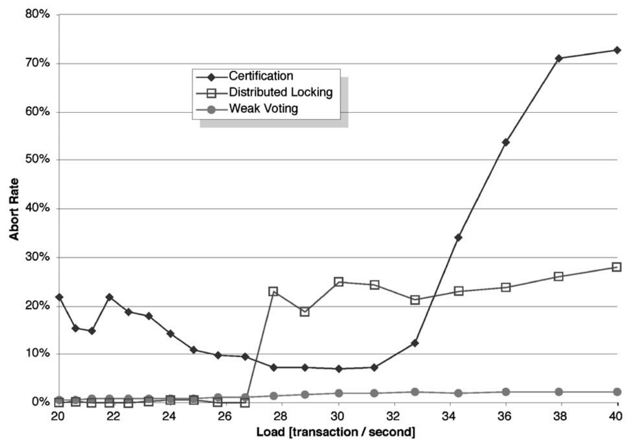


Fig. 14. Abort rate with high-load, fast network.

therefore, a lot of update transactions abort. In fact, most of the aborts are update transactions. We can also see that distributed locking has a sharp increase of aborts once the load reaches 28 transactions per second: At this point, deadlocks start to become significant. It is interesting to note that while the response time of weak voting replication increases, the abort rate stays stable, below 2 percent.

6.1.3 Discussion

The performance measurements show that replication techniques can be split into four categories:

1. network bound replication techniques (distributed locking),
2. limited load techniques (active and primary copy),
3. efficient group communication-based techniques (weak voting and certification) and
4. the lazy technique.

While distributed locking is affected by network performance, most other techniques are not: Their performance is similar with both the slow and the fast network. This is due to the fact that they rely on a single broadcast operation, either a simple broadcast (primary copy and lazy) or a total order broadcast (group communication-based techniques).

The performance of group communication-based techniques depends mostly on their architecture: Techniques that were specifically designed for database replication (certification and weak voting) outperform the basic technique (active replication) significantly. The culprits for the bad performance of active replication are the serial locking phase and the absence of load balancing.

Even in a fast network configuration, distributed locking is outperformed by efficient group communication-based techniques. The reason for this is that the execution of transactions with distributed locking is tightly coupled between the replicas. These synchronization phases cause a serious slow-down. Synchronization is also the cause of the performance difference between weak voting and certification-based replication. The weak voting technique has one synchronization phase (weak voting phase) that "costs" around 50 ms when compared to the certification technique.

The behavior of certification-based replication in high-load situations could be alleviated with flow control techniques. Flow control would help to avoid situations where the conflict rate causes too many aborts, leading to poor performance.

6.2 Scalability Comparison

6.2.1 Description of the Experiment

One aspect of replication techniques that is worth reanalyzing is scalability. A good replication technique must increase its performance when the number of replicas increases. In this experiment, we measured how the system reacts to a changing number of servers. The interval between transactions was fixed at 1,800 ms and the number of clients at 36, thus resulting in a load of 20 transactions per second. We then changed the number of servers and observed the performance of the system.

Note that the scalability experiment does not make much sense for the primary copy technique since the work is done by one single server. With active replication, all servers handle all transactions, so changing the number of replicas will not change 1) the number of transactions that each server handles, and 2) the number of total order broadcast issued. This means that changing the number of servers with active replication has a negligible impact on the performance (this was confirmed experimentally). Therefore, we concentrate below on the scalability of lazy replication, certification, weak voting, and distributed locking. We did the experiment with the fast network setting and two different types of transaction loads. The first is called "mixed load" and contains 50 percent of queries. The second is called "mostly queries" and contains 80 percent of queries.

6.2.2 Results

Mixed Load. Fig. 15a shows the results of the experiment with a query load of 50 percent. Each set of bars represents the performance with a given number of servers, starting with two servers for the leftmost bar. Each technique is represented with a different bar and the height of each bar (Y axis) represents the response time in milliseconds.

The general performance is similar to the experiment presented in Section 6.1: Lazy replication outperforms all other techniques. Group communication-based techniques outperform distributed locking. In general, we see that the response time decreases as the number of replica increases. This shows that, as the number of servers grows, some part of the load (queries) can be distributed on more replicas, thus giving better performance. The most interesting part appears in the extreme case, when the number of replicas is maximal (36 servers): The response time of distributed locking increases significantly. This is not due to network usage (which stays below 5 percent) nor to distributed deadlocks (the abort rate was below 1 percent), but to the cost of coordination inherent to this technique.

While aborts had no significant impact on distributed locking, this was not the case for the certification technique. Fig. 16a shows the abort rates for both the certification technique and the weak voting technique. The X axis represents the number of servers, the left Y axis represents the abort rate in percent. We see that, while the abort rate of the weak voting technique is stable below 2 percent, the

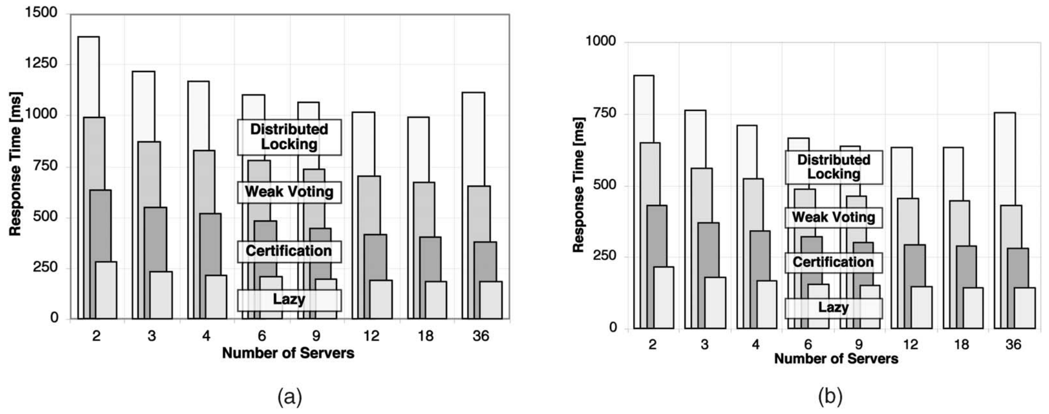


Fig. 15. Scalability with (a) a query rate of 50 percent and (b) a query rate of 80 percent.

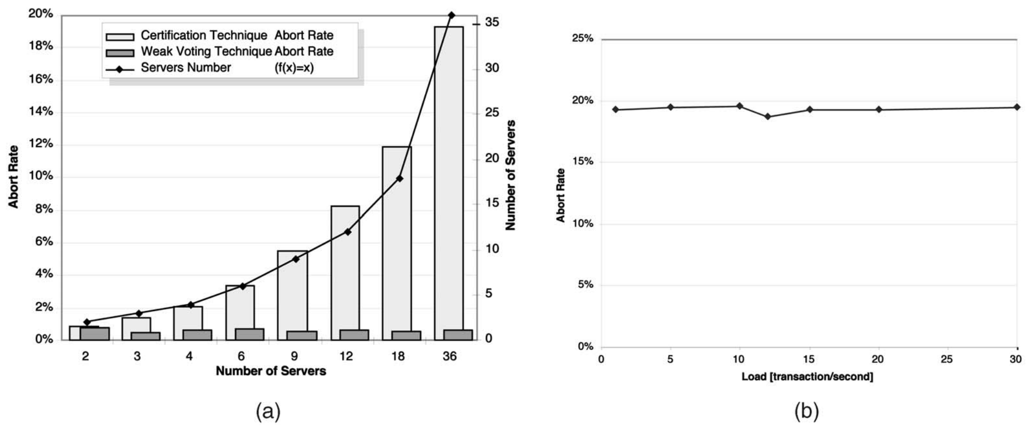


Fig. 16. Abort rates as a function of (a) the number of servers and (b) the load of the system.

abort rate of the certification technique increases with the number of servers. When the number of servers is maximal, the abort rate reaches 20 percent!

A first possible explanation of this behavior is related to the way conflicts are handled. If two transactions originate from the same server (they have the same delegate server), the conflict will be handled by the local locking mechanism. The local locking system will serialize the execution of the conflicting transactions: If no deadlock occurs, there will be no abort. If the two transactions originate from different servers, the conflict will only be detected at certification time and will result in the abort of one transaction. So, increasing the number of replicas increases the chances of abort. If two transactions t_a and t_b conflict, the chances that the conflict results in an abort is proportional to the probability that t_a and t_b are on different servers, which is $1-(1/n)$, where n is the number of servers.

Still, the observed abort rate cannot be explained by this phenomenon (the abort rate would be the same for the weak voting technique and the certification technique). As shown in Fig. 16a, the observed conflict rate is roughly proportional to the number of servers (the black curve represents the function $f(x)=x$ on the right linear Y axis, with 36 servers corresponding to the abort rate with 36 servers). There is a clear correlation between the number of servers and the abort rate with the certification technique, but no correlation with the weak voting technique.

The phenomenon is related to the way the certification algorithm was implemented. At the beginning of the certification phase, when the read and write sets of a transaction are delivered, the transaction is put in a conflict list. This list is used to check for potential conflicts. When a transaction has been committed on some replica, it becomes *stable* on this replica. The information that a transaction is stable is piggybacked on subsequent total order broadcast messages. Once a transaction is known to be stable on all replicas, it is removed from the conflict list. So, a transaction is removed from the conflict list once it has committed on all replicas, *and* all those replicas have issued a total order broadcast.

This means that, even if transactions are executed sequentially (so there should be no conflict), conflicts appear because transactions may stay too long in the conflict list. Fig. 17 illustrates this problem in the case of two servers s_1 and s_2. Initially, the write set of t_0 is processed on s_1 and s_2, and t_0 is committed, i.e., t_0 is stable on s_1 and on s_2. Once committed, transaction t_0 is put in the conflict list of s_1 and s_2. Then, s_1 broadcasts (total order broadcast) the read set and write set of t_1 (message m_1) and s_2 broadcasts the read set and write set of t_2 (message m_2). Messages m_1 and m_2 piggyback the stability of t_0. Let us assume that m_1 is delivered before m_2. When m_1 is delivered, all replicas know that t_0 is stable on s_1, but the status of t_0 on s_2 is unknown, so t_0 stays in the conflict list. All replicas then start to process

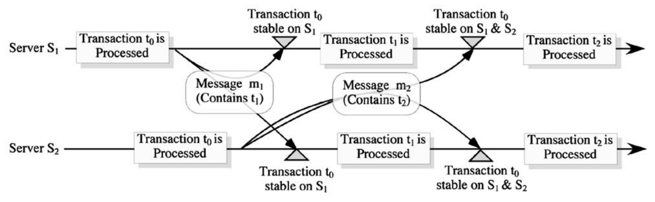

Server S_1
Transaction t_0 is Processed
Transaction t_0 stable on S_1
Transaction t_1 is Processed
Transaction t_0 stable on S_1 & S_2
Transaction t_2 is Processed
Message m_1 (Contains t_1)
Message m_2 (Contains t_2)
Server S_2
Transaction t_0 is Processed
Transaction t_0 stable on S_1
Transaction t_1 is Processed
Transaction t_0 stable on S_1 & S_2
Transaction t_2 is Processed

Fig. 17. Conflict list scenario.

the write set of t_1 (message m_1). As t_0 is still in the conflict list, t_0 can cause the abort of t_1. When m_2 is delivered, the system knows that t_0 is now stable on s_2. As t_0 is stable on all replicas (s_1 and s_2), it can be removed from the conflict list. Therefore, t_0 can cause the abort of t_1, but not t_2.

Now, if we have n servers instead of two, and transactions t_1 to t_n instead of just t_1 and t_2, transaction t_0 will be removed from the conflict list once messages $m_1 \dots m_n$ have been delivered. This means that t_0 can potentially conflict with transactions $t_1 \dots t_{n-1}$, i.e., the number of potential conflicts is proportional to the number of servers n, *regardless* of the load of the system.

To verify this hypothesis, we measured the abort rate of the certification technique with varying loads (between 1 and 30 transactions per second) and a large number of servers (36). Fig. 16b shows the results of this experiment. The X axis represents the load of the system and the Y axis the abort rate. We see that, while the load varies by a factor of 30, the abort rate stays the same, slightly below 20 percent.

"Mostly Queries" Load. Fig. 15b shows the experiment with a query load of 80 percent. The overall graph is similar to the one in Fig. 15a, with better response time—this is expected as there are more queries. We also see that, while the increase in response time is not as significant as with 50 percent of queries, the performance of distributed locking still starts to degrade when the number of servers is high (36).

6.2.3 Discussion

In general, group communication-based replication techniques scale well, assuming a moderate load and a large proportion of queries. The response time does not increase with the number of servers, but diminishes, as one would expect. We showed that the certification technique has a problem that leads to many aborts when the number of servers is high. The weak voting technique has no such problems. Note that the abort problem of the certification technique can be addressed using a version-based database. In this case, the certification test would then rely on the version number of items instead of a conflict list. While this approach solves the problem, its drawback is that it imposes some requirements on the database system (access to version numbers).

The interesting aspect of group communication-based replication is that issues like scalability are resolved outside the replication scheme: If the total order broadcast primitive scales well, then the replication technique will scale well.

6.3 Query Proportion Comparison

6.3.1 Description of the Experiment

The proportion of queries in transactions can have an important effect on the performance in the context of a replicated database. For example, if the read-write policy of a replication technique is ROWA, queries only need to be executed on one replica. In some cases, queries can be executed without requiring any communication. It is interesting to compare the performance of the replication techniques under different query rates.

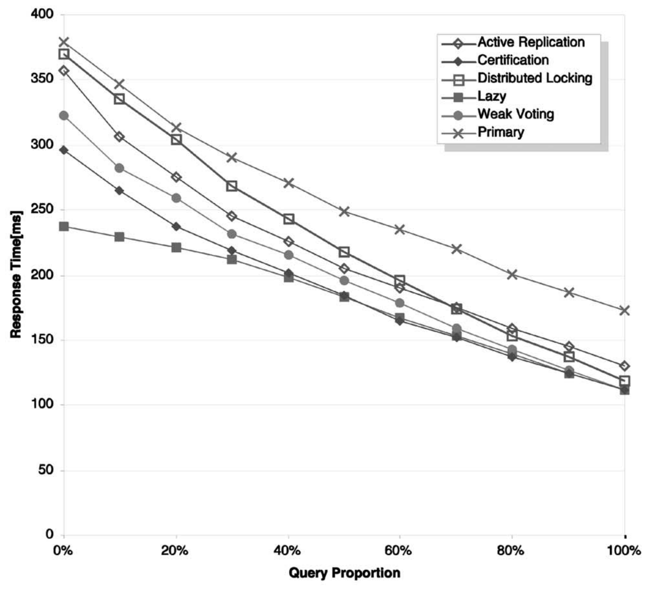
In the experiment, we fixed the number of transactions per second, changed the proportion of queries, and measured the response time. When the query proportion is 100 percent, there are no update transactions; if the query proportion is 0 percent, all transactions are updates. The query proportion was increased from 0 percent to 100 percent in 10 percent increments. We measured the impact of the query load in two settings:

1. **Load = 10 transactions per second**. In this setting, the system is configured with 36 clients and four servers connected by the *fast* network. The interval between transactions is fixed at 3,600 ms. This yielded a load of 10 transactions per second. When the query rate is 50 percent, this setting corresponds to $X = 10$ in Fig. 11b.
2. **Load = 20 transactions per second**. In this setting, the system also consists of 36 clients and four servers connected by the *fast* network. But this time, the interval between transactions is 1,800 ms, leading to a load of 20 transactions per second. When the query rate is 50 percent, this setting corresponds to $X = 20$ in Fig. 11b.

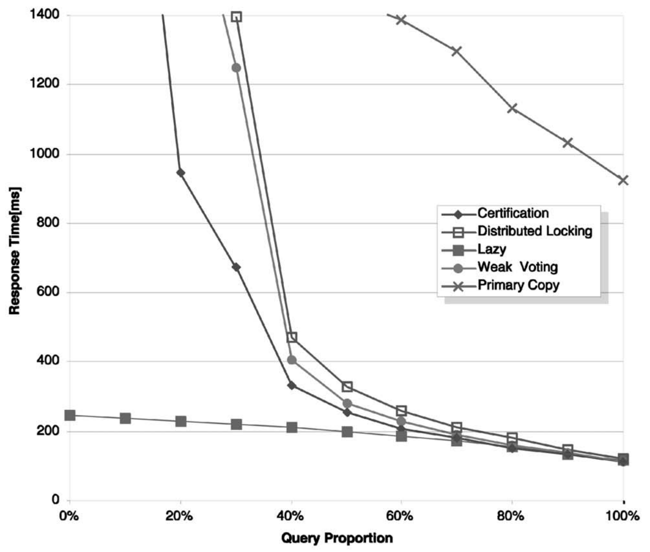
6.3.2 Results

Low Load. Fig. 18a shows the results of the experiment at 10 transactions per second. When the query rate is 100 percent, the load consists of only read operations and transactions can be distributed on all servers. In this situation, certification and weak voting replication have the same performance than lazy replication: The load is perfectly balanced between all servers and no communication occurs. Distributed locking suffers from a slight overhead because of the protocol complexity, but has roughly the same performance.

The two remaining techniques, active replication and primary copy, have worse performances because they do no load balancing. In the primary copy case, all the work is done on one server, the primary. In active replication, the situation is slightly different: all the work is done on *all* replicas, i.e., all replicas do the work of the primary. In both cases, there is no load-balancing. As both techniques handle the load in the same way, one would expect that they have the same performance. In fact, primary copy should outperform active replication by the cost of a total order broadcast. Yet the results show that active replication outperforms primary copy replication by approximately 30 percent. The reason for this lies in the way active replication works. In active replication, the delegate server merely acts as a proxy for all servers: When a transaction is delivered, it is sent to all servers (including itself) using a total order broadcast. Each server processes the transactions



(a)



(b)

Fig. 18. Performance with changing query rate at (a) 10 transactions per second, (b) 20 transactions per second.

and sends the results back to the delegate, the delegate forwards the *first* response to the client. So, in practice, this means that the perceived response time of the client is the response time of the fastest server: The observed response time is the minimum response time of all replicas.

Why are there differences between the response times of servers? The response time of a transaction on one replica depends on multiple factors: actual load of the system, number of items in cache, and time needed for a seek. These factors are, to some extent, random.[5] So, as the number of replicas increases, the observed response time will improve, as we are more likely to get a fast first response. Here, we benefit from the fact that all replicas do all the work. This improvement is similar to the read operation improvement in RAID level 1 systems [38].

As the proportion of queries diminishes, the performance of all techniques degrades. This performance degradation is more noticeable for distributed locking: As the number of writes increases, the overhead of this technique becomes more obvious. It is interesting to note that the performance of the certification-based technique cannot be distinguished from the performance of lazy replication if the query proportion is larger than 40 percent. This shows that, for high query rates and low load, certification-based replication is close to optimum.

Moderate Load. Fig. 18b shows the results for the same experiment with a higher load (20 transactions per second). The first thing we see is that some techniques have very bad performances: active replication (whose curve is outside the graph) and primary copy replication (upper right corner). Most other techniques have nonlinear response curves: If the query proportion is too low, performance drops suddenly.

If we compare this graph with the one in Fig. 18a, it is interesting to see that the relative performance between active replication and primary copy replication is inverted: While both techniques perform poorly, primary copy has better performance. In high-load situations, the advantages of selecting the fastest response is offset by the fact that much more processing is needed. Additionally, the serial locking phase of active replication becomes a bottleneck in high-load situations (see Section 6.1).

Changing load and query rate. To understand how the linear curves in Fig. 18a transform into those of Fig. 18b, we plotted three dimensional graphs for the following techniques: certification (Fig. 19a), weak voting (Fig. 19b), distributed locking (Fig. 19c), and primary copy (Fig. 19d).

We did not plot the remaining two techniques, as their performances in moderate load setting are very bad. For each technique, the X axis represents the query proportion, the Y axis (depth) represents the load of the system in transactions per second, and the Z axis (height) represents the resulting response time. Each technique is represented by a separate surface. On each surface, we marked the curve corresponding to $Z = 200$ ms and to $Z = 300$ ms.

First, we see that all four surfaces are continuous: This shows that all four techniques are stable in the parameter space considered. The most noticeable aspect of those four figures is that distributed locking, certification, and weak voting replication have the same general shape: Performance is good if the load is low or the query proportion high. Response time reaches a "peak" in high-load, low query proportion situations. Primary copy (Fig. 19d) on the other hand, has a very different general shape. While the technique is also sensitive to some extent to the query proportion, the load has a far greater influence on performance. As the load increases, the response time forms a "wall," even if the load is composed only of queries. The reason for this is that primary copy does no load-balancing.

Let us consider the parameter values where the response time reaches 200 ms and 300 ms, respectively. We can see

5. Caching algorithms that are usually used are not random, but deterministic. Classic deterministic caching algorithms would not benefit active replication because all the replicas would cache the same items. In the simulation, caching was simulated as a statistical process. This benefited active replication. Special algorithms could be tailored to active replication and give even better results, for instance by having each replica keep in cache only a subset of the data.

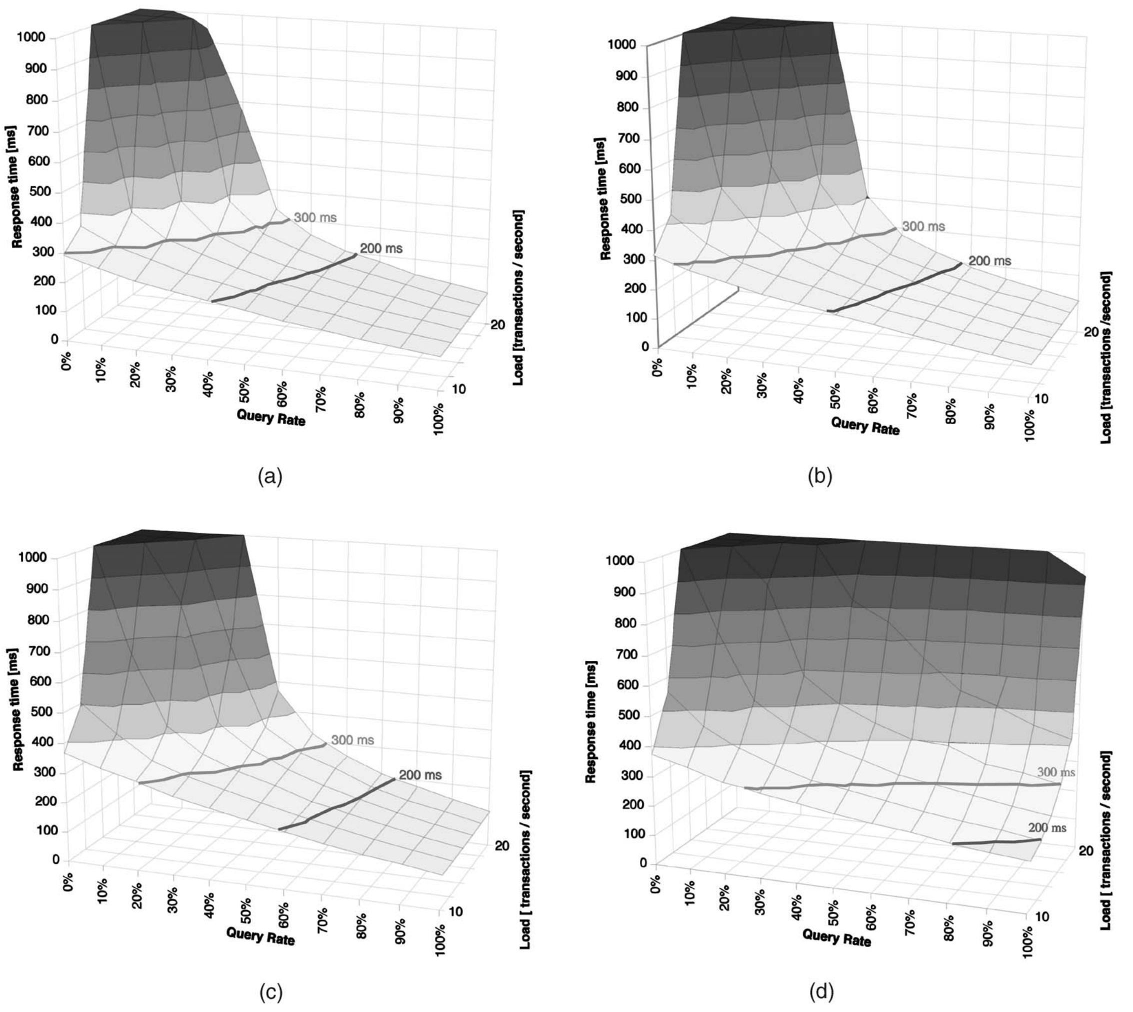


Fig. 19. Performance with changing query proportion and changing loads. (a) Certification based, (b) weak voting, (c) distributed locking, and (d) primary copy.

the performance difference between the distributed locking technique on one hand (Fig. 19c), and the total order-based replication techniques (weak voting and certification, Figs. 19b and 19a) on the other hand. Total order-based replication techniques outperform distributed locking systematically. In general, the difference is roughly equivalent to a difference of 10 percent of the query proportion, e.g., distributed locking behaves with a 50 percent query proportion like total order-based replication with a 40 percent query proportion. This is consistent with the performance difference observed in overall load performance (Section 6.1).

6.3.3 Discussion

The experiments show that the load balancing features of the different replication techniques have an important impact on performance, especially when the load contains a large proportion of queries. We see that total order broadcast-based replication techniques offer good load-balancing. As those techniques are built on optimistic hypotheses (execute first, check for conflicts after), they work best when the proportion of queries is high. In good conditions (moderate load and high query proportion), those techniques have a performance close to the performance of lazy replication.

7 Contributions and Related Work

While there have been other performance evaluations of database replication techniques based on group communication (see below), the paper makes the following original contributions:

- It compares the performance of weak voting and certification-based replication—those two techniques have never been compared.
- It compares the performance of those techniques to 1) distributed locking and 2) lazy replication. This gives us a lower and upper bound for performance.
- While group communication-based database replication has always been presented as a better alternative to lazy replication, the performance of group communication-based replication has never been compared to the optimal solution (from the performance point of view): lazy (update-everywhere) replication. The paper shows that, when the load is reasonable and consists mostly of queries (typically the settings suited for lazy replication), techniques like certification or weak voting have performance that is very close to lazy replication.

- The paper uses a more refined simulation model: Group communication primitives are not simulated, but executed inside the simulator. Previous simulations modeled complex group communication protocol like total order broadcast as a single atomic operation in the simulator.

The results presented confirm the results obtained by previous experimentations [10], [30], [13], [31], [11], namely, that database replication based on total order broadcast offers good performance. We now compare our results with these previous experiments.

Kemme et al. In [10], [14], [16], Kemme et al. present the weak voting technique (called SER-D) with some variants that do not ensure one-copy serializability. Performance evaluation is done using both a simulator and an implementation based on Postgres [16]. The performance of weak voting is compared to distributed locking. The results are consistent with the one presented here. The following behaviors have been observed in both studies:

1. Weak voting significantly outperforms distributed locking.
2. Weak voting is much less affected by slow network performance than distributed-locking.
3. Weak voting scales well, i.e., the throughput of a replicated server can be much higher than that of a single nonreplicated server.
4. Weak voting offers very good performance when the workload is composed mainly of queries.

It is interesting to note that most variants proposed in [14] offer increased performance by relaxing serializability and relying instead on cursor stability. In this context, it is worth noting that certification replication outperforms weak voting replication in most cases, while still enforcing serializability (at the same time, abort rates are an open issue for certification-based replication).

Pedone et al. In [30], [11] Pedone et al. present the certification-based replication technique (called database state machine). An analytical performance evaluation of the technique is also presented. The results are consistent with those presented here. We obtain very similar numbers for the abort rate as a function of the load (Section 6.1) and as a function of the number of servers (Section 6.2). The scalability we observed corresponds to the scalability curve when $k = \infty$, i.e., with an unlimited network. This confirms that communication is not the bottleneck in a LAN setting.

Holliday et al. In [13], Holliday et al. present the performance of four techniques (called A1, A2, A3, and A4) based on group communication and initially described in [4]. The results are not compared to any classical technique. Technique A4 (similar communication pattern as certification) outperforms technique A3 (similar communication pattern as weak-voting) by a small amount (less than 10 percent): Our results confirm this. Scalability experiments for A3 and A4 are also confirmed: When the load includes many queries, increasing the number of replicas decreases the response time. We also echo the conclusion that replication techniques that are specially tailored for group communication (A3 and A4) offer better performance than straightforward approaches (A2, in our case active replication).

The fact that our simulation results are based on a finer network model than the one used in previous simulations, while coming to similar conclusions, validates in a sense the simpler simulation model used in previous experiments. As the performance of group communication-based techniques is generally independent of the type of network (10 or 100 Mb/s ethernet), it remains relevant for faster network (gigabit ethernet).

8 Conclusion

In this paper, we measured the performance of different replication techniques. We focused on techniques based on total order broadcast and compared their performance to each other and to classical replication techniques like primary copy, distributed locking, and lazy update-everywhere replication.

Performance evaluations show that replication techniques based on total order broadcast significantly outperform traditional database replication protocols like distributed locking. In good conditions, performance is very close to lazy replication. The performance difference is large if the network is slow and subject to contention. This is due to the fact that distributed locking uses a lot of messages. Replication techniques based on total order broadcast use less networking resources, typically by restricting interactions to a single broadcast. These techniques are, therefore, very efficient with a slow network. When the network is fast and contention is rare, group communication-based replication still outperforms distributed locking, albeit by a smaller amount. Note that the situation will not change significantly when moving from a 100Mb/s to a 1Gb/s network since distributed locking could not saturate a 100Mb/s network.

The stability of performance measured with different settings, and the lack of significant difference between the experiment with the slow and the fast network, together with the resource usage given by the simulator, confirm a general observation that in a LAN situation, network contention is not a real problem and will become less and less important with the advent of faster networks like gigabit ethernet.

Resources like CPU and disks are much more likely candidates for being bottlenecks in a LAN. This does not mean that communication is not an issue in database replication: Multiple copies still induce several problems like synchronization costs and high abort rates. Techniques that rely more on the network are systematically slower, albeit by a small factor. Those issues are not related to the performance of the network, but to the design of the replication technique. The performance difference between certification-based replication and weak voting replication is not due to the cost of the additional reliable broadcast of weak voting, but to the cost of the additional synchronization between the replicas.

References

[1] J.N. Gray, P. Helland, P. O'Neil, and D. Shasha, "The Dangers of Replication and a Solution," *Proc. 1996 Int'l Conf. Management of Data,* pp. 173-182, 1996.

[2] A. Schiper and M. Raynal, "From Group Communication to Transactions in Distributed Systems," *Comm. ACM,* vol. 39, no. 4, pp. 84-87, 1996.

[3] G. Alonso, "Partial Database Replication and Group Communication Primitives," *Proc. Second European Research Seminar on Advances in Distributed Systems (ERSADS '97),* pp. 171-176, 1997.

[4] D. Agrawal, G. Alonso, A. El Abbadi, and I. Stanoi, "Exploiting Atomic Broadcast in Replicated Databases," *Proc. EuroPar Conf. (EuroPar '97),* 1997.

[5] J.M. Chang, "Simplifying Distributed Database Systems Design by Using a Broadcast Network," *Proc. Ann. SIGMOD '84 Meeting,* B. Yormark, ed., vol. 14, pp. 223-233, 1984.

[6] Ö. Babaoglu and S. Toueg, "Understanding Nonblocking Atomic Commitement," Technical Report UBLCS-93-2, Laboratory for Computer Science, Univ. of Bologna, Italy, 1993.

[7] I. Keidar and D. Dolev, "Increasing the Resilience of Distributed and Replicated Database Systems," *J. Computer and System Sciences (JCSS),* vol. 57, no. 3, pp. 309-224, 1998.

[8] R. Jiménez-Paris, M. Patiño-Martínez, G. Alonso, and S. Arévalo, "A Low Latency Nonblocking Commit Server," *Proc. 15th Int'l Conf. Distributed Computing (DISC 2001),* pp. 93-107, 2001.

[9] F. Pedone, R. Guerraoui, and A. Schiper, "Exploiting Atomic Broadcast in Replicated Databases," *Proc. EuroPar Conf. (EuroPar '98),* 1998.

[10] B. Kemme and G. Alonso, "A Suite of Database Replication Protocols Based On Group Communication Primitives," *Proc. 18th Int'l Conf. Distributed Computing Systems (ICDCS '98),* 1998.

[11] F. Pedone, R. Guerraoui, and A. Schiper, "The Database State Machine Approach," *Distributed and Parallel Databases,* vol. 14, no. 1, pp. 71-98, July 2003.

[12] B. Kemme, F. Pedone, G. Alonso, and A. Schiper, "Processing Transactions over Optimistic Atomic Broadcast Protocols," *Proc. Int'l Conf. Distributed Computing Systems,* 1999.

[13] J. Holliday, D. Agrawal, and A.E. Abbadi, "The Performance of Database Replication With Group Multicast," *Proc. Int'l Symp. Fault Tolerant Computing (FTCS 29),* pp. 158-165, 1999.

[14] B. Kemme, "Database Replication for Clusters of Workstations," PhD dissertation, Swiss Federal Inst. of Technology Zürich, Switzerland, 2000.

[15] F. Pedone and S. Frølund, "PRonto: A Fast Failover Protocol for Off-the-Shelf Commercial Databases," Technical Report HPL-2000-96, Software Technology Laboratory, Hewlett-Packard Laboratories, Palo Alto, Calif., 2000.

[16] B. Kemme and G. Alonso, "Don't Be Lazy, Be Consistent: Postgres-R, A New Way to Implement Database Replication," *Proc. 26th Int'l Conf. Very Large Databases (VLDB),* 2000.

[17] S. Frølund and F. Pedone, "Continental Pronto," *Proc. 20th Symp. Reliable Distributed Systems,* pp. 46-55, 2001.

[18] J. Holliday, D.A.R. Steinke, and A.E. Abbadi, "Epidemic Algorithms in Replicated Databases," *IEEE Trans. Knowledge and Data Eng.,* vol. 15, no. 5, pp. 1218-1238, Sept./Oct., 2003.

[19] J. Holliday, D. Agrawal, and A.E. Abbadi, "Using Multicast Communication to Reduce Deadlocks in Replicated Databases," *Proc. 19th Symp. Reliable Distributed Systems,* pp. 196-205, 2000.

[20] Y. Amir and C. Tutu, "From Total Order to Database Replication," *Proc. 22nd Int'l Conf. Distributed Computing Systems (ICDCS),* 2002.

[21] R. Vandewall, "Database Replication Prototype," master's thesis, Rijksuniversiteit Groningen and École Polytechnique Fédérale de Lausanne, Netherlands and Switzerland, 2000.

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

[23] V. Hadzilacos and S. Toueg, "A Modular Approach to Fault-Tolerant Broadcasts and Related Problems," Technical Report TR94-1425, Cornell Univ., Computer Science Dept., 1994.

[24] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso, "Database Replication Techniques: A Three Parameter Classification," *Proc. 19th Symp. Reliable Distributed Systems (SRDS '00),* pp. 206-215, 2000.

[25] J. Holliday, D. Agrawal, and A.E. Abbadi, "The Performance of Replicated Databases Using Atomic Broadcast Group Communication," Technical Report TRCS99-11, Computer Science Dept., Univ. of California, Santa Barbara, 1999.

[26] F.B. Schneider, "Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial," *ACM Computing Surveys,* vol. 22, no. 4, pp. 299-319, 1990.

[27] F. Pittelli and H. Garcia-Molina, "Reliable Scheduling in a TMR Database System," *ACM Trans. Computer Systems,* vol. 7, no. 1, pp. 25-60, 1989.

[28] I. Keidar, "A Highly Available Paradigm For Consistent Object Replication," master's thesis, The Hebrew Univ. of Jerusalem, Israel, 1994.

[29] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso, "Understanding Replication in Databases and Distributed Systems," *Proc. 20th Int'l Conf. Distributed Computing Systems (ICDCS '00),* 2000.

[30] F. Pedone, "The Database State Machine and Group Communication Issues," PhD dissertation, École Polytechnique Fédérale de Lausanne, Switzerland, 1999.

[31] B. Kemme and G. Alonso, "A New Approach to Developing and Implementing Eager Database Replication Protocols," *ACM Trans. Database Systems,* vol. 25, no. 3, pp. 333-379, 2000.

[32] *CSIM18 Simulation Engine (C++ Version),* Mesquite Software Inc., Austin, Texas 78759, 1994.

[33] P. Urbán, X. Défago, and A. Schiper, "Contention-Aware Metrics for Distributed Algorithms: Comparison of Atomic Broadcast Algorithms," *Proc. Ninth IEEE Int'l Conf. Computer Comm. and Networks (IC3N 2000),* 2000.

[34] R. Ja, *The Art of Computer System Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling.* John Wiley and Sons, 1991.

[35] R. Agrawal, M.J. Carey, and M. Livny, "Concurency Control Performance Modeling: Alternatives and Implications," *ACM Trans. Database Systems,* vol. 12, no. 4, pp. 609-654, 1987.

[36] K. Nørvåg, O. Sandstå, and K. Bratbergsengen, "Concurrency Control in Distributed Object-Oriented Database Systems," *Proc. Advances in Databases and Information Systems Conf.,* pp. 9-17, 1997.

[37] P. Urbán, X. Défago, and A. Schiper, "Neko: A Single Environment to Simulate and Prototype Distributed Algorithms," *Proc. 15th Int'l Conf. Information Networking (ICOIN-15),* 2001.

[38] S. Chen and D. Towsley, "A Performance Evaluation of RAID Architectures," Technical Report UM-CS-1992-067, Dept. of Computer Science, Univ. of Masschusetts, Amherst, 1992.

Matthias Wiesmann received a degree in computer science from the University of Geneva in 1997. He received the PhD degree in computer science at EPFL (Federal Institute of Technology in Lausanne, Switzerland) in 2002. He has been a part-time lecturer at the Technical University of Fribourg, the Technical University of Vaud, and EPFL. His research interests are in the area of fault-tolerance in distributed systems, replicated persistent objects, group communication, and distributed system management. He is currently a research fellow at the European Organization for Nuclear Research (CERN) in Geneva and is working on fault-tolerance on the Atlas data acquisition system.

André Schiper graduated in physics from the ETHZ in Zurich in 1973 and received the PhD degree in computer science from the EPFL (Federal Institute of Technology in Lausanne, Switzerland) in 1980. He has been a professor of computer science at EPFL since 1985, leading the Distributed Systems Laboratory. During the academic year 1992-1993, he was on sabbatical leave at the University of Cornell, Ithaca, New York. His research interests are in the areas of dependable distributed systems, middleware support for dependable distributed systems, replication techniques (including for database systems), group communication, distributed transactions, and, recently, MANETs (mobile ad-hoc networks). From 2000 to 2002, he was the chair of the steering committee of the International Symposium on Distributed Computing (DISC). He has taken part in several European projects. He is currently a member of the editorial boards of the *ACM Distributed Computing* journal and the *IEEE Transactions on Dependable and Secure Computing*, and a member of the IEEE and the IEEE Computer Society.