Submission to DSN 2004

Evaluating the Performance of the Database State Machine*

A. Sousa† J. Pereira L. Soares A. Correia Jr. L. Rocha R. Oliveira F. Moura
Distributed Systems Group - Departamento de Informática - Universidade do Minho
Campus de Gualtar
4710-057 Braga
Phone: +351 253 604 477
Fax: +351 253 604 471
PORTUGAL
als@di.uminho.pt

Abstract

Replication of database servers using the Database State Machine (DBSM) approach has recently been the subject of much attention as it promises both good performance and increased reliability. Fulfilling the promise of performance, however, requires that the impact of environment parameters as well as of design and implementation decisions are evaluated in a realistic setting which allows experimentation with configuration and environment parameters.

This paper introduces a model that combines simulated network and database engine components with real implementations of termination and communication protocols. This allows us to precisely evaluate the protocols' performance when subjected to a realistic load generated by the simulated database engine. It also allows us to evaluate the impact of the protocol overhead on the overall performance of the database system in several environments. Besides describing the design and validation of the simulation model, results obtained with prototype implementations of the protocols are presented.

REGULAR PAPER

* Research funded by FCT, ESCADA project (POSI / 33792 / CHS / 2000).
† Contact author

1 Introduction

Replication of database servers using the Database State Machine approach (DBSM) [25] is done by allowing all replicas to optimistically execute transaction requests without distributed locking. The resulting updates to the data are then atomically multicast to all replicas. Upon their delivery, each replica certifies the transaction, only committing transactions that do not violate serialization guarantees. The total ordering of messages and the determinism of the certification procedure ensure that replicas remain consistent. This approach has recently been the subject of much attention of both theoreticians and practitioners [32, 20, 24, 5, 17, 4, 21] because it ensures consistency and increases availability by relying on the properties of atomic multicast protocols. In addition, by allowing concurrent execution of the transactions, without distributed locking, it provides good performance and scalability.

Fulfilling the promise of performance requires however the evaluation of the impact of design and implementation decisions of both the certification and the communication protocols that are introduced with the approach. In particular, we are interested in evaluating the performance of the termination protocol, including both the certification procedure as well as the underlying group communication protocol.

The performance of the DBSM approach has been firstly evaluated with the simulation of the certification and the communication protocols [23]. This approach allows multiple runs of the same scenario with different configuration settings, thus evaluating the impact of each parameter. On the other hand, it makes it difficult to estimate the resources used by the protocols, which compete with the database engine. In [20], the DBSM was evaluated by implementing it within the PostgreSQL database engine. Although this provides a realistic test environment, the results are tightly related to this single database engine. Moreover, it is also difficult to setup and run multiple tests with slight variations of configuration parameters. This becomes particularly evident if one considers a large number of replicas and wide-area networks.

In this paper we propose a model of a replicated database server that combines simulation and profiling of real code. The ESCADA distributed database model [32] encompasses the replicated database servers, its users and the underlying environment. A real implementation of the certification and communication protocols is used and profiled, as these are the focus of our interest in developing and optimizing, and the database engine and the network are simulated. This allows us, by experimenting with different configuration parameters, to assess the validity of the design decisions, and, in addition, provides us the opportunity to subject real components to fault scenarios which would be unlikely and difficult to replicate in real systems.

The model is instantiated and then validated by comparing runs of a single site with runs of a identically configured real PostgreSQL database [3]. A key issue in the instantiation of the model is the generation of realistic traffic. Therefore, we generate traffic according to the industry standard OLTP benchmark TPC-C [34]. Namely, we use the non-uniform random distribution to populate the database and generate queries. We also use the same mix of transactions, thus balancing fast transactions with longer ones and read-only queries with those that update the database. Results presented in this paper are in fact strongly influenced

by this choice, as the impact of concurrency control conflicts would otherwise not be visible.

The rest of the paper is structured as follows: Section 2 briefly describes the DBSM approach to database replication. Section 3 describes the simulated components of the model and Section 4 the protocol prototypes. Section 5 presents the instantiation and validation of the model. Section 6 presents the results obtained. Section 7 describes related work and Section 8 concludes the paper.

2 Motivation

In this section we briefly describe the Database State Machine [25] approach to replication and the trade-offs that have been proposed and should be evaluated.

2.1 Distributed Databases with the DBSM

The Database State Machine [23] approach to replication takes advantage of group communication [12] and works as follows: Transactions are executed optimistically by any of the replicas without distributed locking. When the transaction is ready to be committed, the resulting read and write-sets are multicast to all replicas which perform a deterministic certification procedure to ensure that the transaction does not conflict with concurrent transactions already committed. During certification, a transaction t is aborted if a concurrent transaction t' has been committed and the read-set of t intersects with the write-set of t'. Total order multicast is used to ensure that the sequence of transactions certified by each replica is the same, thus ensuring consistency.

This approach has several advantages when compared to existing replication schemes. In contrast to widely used lazy replication techniques [22], DBSM provides strong consistency and fault tolerance. When compared with the primary-backup (or master-slave) approach [11], it allows transaction execution to be done in parallel in several replicas. This is ideal when there is a large share of non-conflicting update transactions. By avoiding the requirement for distributed locking used in synchronous replication [15], the DBSM scales to larger number of nodes. And when compared to active replication [31], it allows for better usage of resources because each transaction is executed by a single node. This also allows for non-deterministic execution, as resulting from concurrently processed requests.

Several modifications of the DBSM approach have also been proposed. An alternative proposal [20] avoids multicasting read-sets which can be reasonably large in some kinds of transactions. The trade-off in this approach is two-fold: it reduces the resilience of the protocol as it confines the certification capability to one of the replicas, but increases latency since an additional communication step is required to multicast the outcome of the certification to all replicas. It also suggests the use of a weak consistency criterion [8], called snapshot isolation level, which avoids the reduction of resilience and the additional communication step. Basically, just the write-sets and write-values are multicast and the certification procedure identifies write operations to the same tuple as conflicting operations.

With the aim of improving the protocol scalability, an approach exploiting partial replication appeared in [32]. With partial replication, since the data is fragmented across the replicas a coordinated certification process is required. To achieve this, the DBSM protocol is extended with a final agreement step in order to all replicas agree in the outcome of the transaction.

For an understanding of the advantages and disadvantages of the different approaches, it is important to evaluate these proposals under different scenarios. For instance, it is extremely important to understand the impact of an increase number of clients using a realistic workload on the overall performance.

2.2 Evaluation

Simulation models allow for detailed evaluation of complex systems offering easily controlled and reproducible experiments without the cost of setting up a real system. The simplicity of simulation models of replicated databases using the DBSM [25, 4] has however some drawbacks:

Offered load Existing models have subjected termination protocols to synthetic loads which are not realistic. For instance, the outcome of certification is tightly related with conflicting operations which these loads lack. In addition, it is difficult to assess such results in the context of widely accepted benchmarks and well known performance results.

Concurrency control In previous simulations, the traffic is offered directly to the termination protocols. In fact, there can be a substantial impact of local concurrency control in the outcome of termination protocols.

Resource usage No estimation of resources used by termination protocols (processor and network) is usually done.

System-wide impact No results for system wide impact of DBSM replication such as on overall throughput and latency. Although latency of termination can be estimated, the side-effects of holding locks for some additional time during termination in the execution of local transactions cannot be predicted.

On the other hand, implementing the termination protocol in a concrete database engine and subjecting the resulting system to benchmarking offers the most realistic results. Nevertheless, there are also a number of disadvantages:

Cost of deployment There is a high cost involved in setting up a realistic benchmarking environment. Besides dedicated servers, it is required that users and network infrastructure are also controlled to avoid interference with the results.

Cost of implementation Testing in a real environment requires the implementation of a large portion of the system, in particular, the integration with database engines which do not provide adequate programming interfaces requires dealing with system internals.

Flexibility Cannot be used to evaluate what-if scenarios, in which system parameters are varied, especially if these involve speculation (e.g. will the next year's CPU solve our bottleneck?).

Reproducibility Reproducibility is hard to achieve, especially if a costly dedicated testing environment cannot be deployed. This makes it very hard to determine the causes for interesting phenomena that are observed.

Probe effect and global observation Although coarse grained observation of the system can be done without disturbing its operation, fine grained recording of system behavior is bound to affect the results.

Fault injection It is hard to perform reproducible fault injection to evaluate the behavior. This is especially relevant as testing for dependability is an absolute requirement for evolving prototypes in usable implementations.

The ESCADA distributed database model bridges the gap between the two approaches by combining models of realistic database users, a transaction processing engine, with concurrency control and resource accounting, and the network with the execution of the implementation of the protocols under study. Parts of the system can be used separately. Namely, the database clients and engine, or the communication protocols and network model thus allowing fine-grained evaluation of system components.

Without the possibility of combining real implementations with simulated engines would be extremely difficult to evaluate with clarity and precision the advantages and disadvantages of the proposed approaches under different scenarios. Further, the use of database clients producing realistic traffic is quite important to put the components under real workload. For instance, only with this realist traffic is possible to identify the true impact of the read-sets on the overall performance, being possible to decide if the benefits of using a termination protocol which avoids to send read-sets but reduces resilience surpasses the others.

3 Replicated Database Model

In this section we describe the simulated components of the model which provide a realistic environment for the prototype components under study. These components are depicted as white boxes in Figure 1. The simulation model is developed using the Java platform and the Scalable Simulation Framework (SSF) kernel [13].

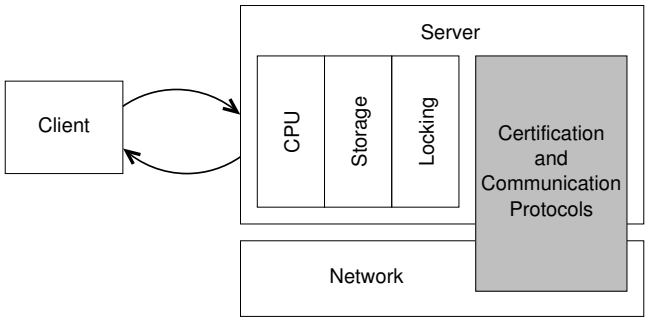


Figure 1: Architecture of the model.

Tables	Cardinality	Row Length	Table Size
Warehouse	1	89	0.089 K
District	10	95	0.950 K
Customer	30 K	655	19650 K
History	30 k	46	1380 K
Order	30 K	24	720 K
New Order	9 K	8	72 K
Order Line	300 K	54	16200 K
Stock	100 K	306	30600 K
Item	100 K	82	8200 K

Table 1: Size of Tables (k is 1000)

3.1 Database Clients

A database client is attached to a database server and produces a stream of transaction requests. After each request is issued, the client blocks until the server replies, thus modeling a single threaded client process. After receiving a reply, the client is then paused for some amount of time (think-time) before issuing the next transaction request.

The content of each request is generated according to a simulated user based on the TPC-C benchmark [34]. The database is populated according to the number of clients. It is worth noting that, our interest to initially evaluate the DBSM prototypes is just in the workload partner produced by this benchmark. Therefore, the constraints of throughput, performance, wait time, response time, screen load and background execution of transactions are not considered here.

The TPC-C benchmark proposes a wholesale supplier with a number of geographically distributed sales districts and associated warehouses as an application. This environment simulates an OLTP workload with a mixture of read-only and update intensive transactions. The database entities are presented in Table 1.

In this benchmark an emulated client can request five different types of transactions. Each transaction is chosen based on a probability distribution presented below. From the beginning of the benchmark run until the end, the warehouse number is constant and each warehouse supports 10 emulated clients and the

database grows with the number of warehouses [34]. The transactions are described as follows:

New Order Transaction (44%) Adds a new order into the system.

Payment Transaction (44%) Updates the customer's balance, district and warehouse statistics.

Order Status Transaction (4%) Returns a given customer's latest order.

Delivery Transaction (4%) Records the delivery of products.

Stock Level Transaction (4%) Determines the number of recently sold items that have a stock level below a specified threshold.

Each transaction is modeled as a sequence of operations, which can be one of: i) fetch a data item; ii) do some processing; iii) write back a data item. An additional operation is the commit marker, that is used by the server when operating as a DBSM node to determine when to submit the transaction to the distributed certification procedure.

During the run of the simulation, the client logs the time at which a transaction is submitted, the time at which it terminates, the outcome (either abort or commit) and a transaction identifier. The latency, throughput and abort rate of the server can then be computed for one or multiple users, and for all or just a subclass of the transactions.

3.2 Database Server

The database server handles multiple clients and is modeled as a scheduler and a collection of resources, such as storage and CPUs, and a locking policy. Upon receiving a transaction request each operation is scheduled to execute on the corresponding resource.

Processor operations are scaled according to the configured CPU speed. Each is then executed in a round-robin fashion by any of the configured CPUs. A processor operation can be preempted, typically, to assign the CPU to a higher priority task, like the network protocol handler.

A storage element is used for fetching and storing items and is defined by its latency and number of allowed concurrent requests. Each request manipulates a storage sector, hence storage bandwidth becomes configured indirectly. A cache hit ratio determines the probability of a read request being handled instantaneously without consuming storage resources.

Operations fetching and storing items are also submitted to the lock manager. Depending on the locking policy being used, the execution of the transaction can be blocked between operations. The locking policy described in this paper is based on PostgreSQL's multi-version [9]. This policy ignores fetched items, while it exclusively locks updated items. When a transaction commits, all other transactions waiting on the same

locks are aborted. If the transaction aborts, the locks are released and can be acquired by the next transaction. In addition, locks are atomically acquired before executing any of the operations, and released, also atomically, when the transaction commits or aborts.

When the database server is operating as a DBSM node and a commit marker is reached, the corresponding transaction is submitted for distributed certification. This involves the identification of items read and written as well as the values of the written items. As certification is handled by real code, the representation of item identifiers and values of updated items must accurately correspond to those of real traffic. This is described in more detail in Section 4.1.

During the simulation run, the usage and length of queues for each resource is logged and can be used to examine in detail the status of the server.

3.3 Network Model

The network is simulated using the SSFNet network simulation tool [14]. The SSFNet model includes physical network components, such as hosts, links and routers, as well as protocol layers, such as IP, UDP and TCP. Complex network models can be configured using such components to mimic existing networks or to explore particularly large or interesting networks.

In addition to the configuration of the network, it is also possible to add application components to generate realistic background traffic. Components can also be attached to routers and hosts to log packets. The resulting format is the same used in real networks and thus the log files can be examined using a variety of existing tools.

3.4 Simulation Kernel and Centralized Simulation

A key element of the model proposed in this paper is the ability to combine simulated environment components using discrete-event simulation model with real code for those components that are under study. This is the centralized simulation model of [7].

The interaction with the real code for centralized simulation is performed by additional code layered on SSF primitives. Briefly, this works as follows. An event that is to be handled by real code, such as the submission of a transaction for certification or the reception of a network message by the group communication protocol is delayed (in simulation time) until a simulated CPU becomes available. The execution is then timed using a profiling timer and the result used to mark the CPU busy during the corresponding period, thus preventing other real code events or simulated processing to be attributed concurrently to the same simulated CPU.

During the execution of real code, interaction with simulated components happens when reading the clock or scheduling simulated events through the exchange of network messages. The interception of such primitives is achieved by reimplementing network communication and event scheduling primitives used by

the protocol code on top of the simulation kernel. When one call is intercepted, the profiling timer is stopped and used to calculate the elapsed time since the event began executing. This value is then added to the delay of events being scheduled, thus keeping the illusion of a monotonically increasing clock and preserving causality.

The clock can also be explicitly stopped and restarted, thus allowing execution of real code without implicit consumption of simulation time as happens in simulated components. This is useful, for instance, for detailed logging from within real code without disturbing the normal execution.

4 Protocol Prototypes

We now examine each of the real components used in the model, namely, certification and the communication protocols depicted as the shaded box in Figure 1. Being real code, it can be run stand-alone, using a real network, or within the centralized simulation model. Our aim is to evaluate and improve its performance and robustness, but also study its impact in the overall replicated database system.

4.1 Distributed Certification

The distributed certification procedure [32] runs in two stages. First, just after a transaction has been executed and is ready to be certified, its associated data is gathered and atomically multicast to the group of replicas. Then, upon delivery, the second stage of the certification procedure is run by each replica to decide whether the transaction commits or aborts.

In detail, when a transaction enters the committing stage, identifiers of read and written tuples are obtained. Our prototype assumes that each of these is a 64-bit integer. The values of the written tuples are also obtained (in the simulation the tuples size is used to calculate the amount of padding data that should be putted in messages so its size resembles the one obtained in a real system). All this information, along with the identifiers of the last transaction that has been committed locally, are marshaled into a message buffer. In practice, most protocols will avoid copying the contents of buffers that are already marshaled (i.e. have been written to disk) thus improving performance. Our prototype implements this optimization.

The size of the read-set may render its multicast impractical. In this case, a single identifier for each complete table can be sent. This is similar to the common practice of upgrading individual locks on tuples to a single table lock.

Upon delivery, the message is unmarshaled. The sequence number of the last transaction committed is used to determine which transactions were executed concurrently and thus can result in conflicts. The read-set is then compared with the write-set of all concurrent transactions that have been committed. If they intersect, the transaction is aborted. Otherwise, there are no conflicts and the transaction can be committed.

Notice that this involves comparing not only individual tuple identifiers but also comparing identifiers of individual written tuples with those of the table. This is simplified by including the table identifier as the

highest order bits of each tuple identifier. The run time is minimized by keeping tuple identifiers ordered in both lists, thus requiring only a single traversal to conclude the procedure.

4.2 Atomic Multicast Protocol

The atomic multicast protocol is implemented in two layers. A view synchronous multicast protocol and a total order protocol. The bottom layer, view-synchronous multicast, works in two phases. First, messages are disseminated, taking advantage of IP multicast in local area networks and falling back to unicast in wide-area networks. Then, reliability is ensured by a receiver initiated mechanism [29] and a scalable stability detection protocol [16]. Flow control is performed by a combination of a rate-based mechanism during the first phase and a window-based mechanism during the second phase. View synchrony uses a consensus protocol [30] and imposes a negligible overhead during stable operation.

Total order is obtained with a fixed sequencer protocol [10, 18]. In detail, one of the sites issues sequence numbers for messages. Other sites buffer and deliver messages according to the sequence numbers. View synchrony ensures that a single sequencer site is easily chosen and replaced when it fails.

By implementing total order within our prototype, it becomes possible to later explore several optimizations of atomic multicast are possible in the context of transaction processing. Namely, semantic reliability [28] and optimistic total order [26, 33]. Semantic reliability improves throughput stability in heterogeneous and wide area networks by discarding messages that become obsolete while still in transit, for instance, because a transaction is known to have aborted. Optimistic total order recognizes that it is possible to exploit the spontaneous order of messages which happens with high probability to optimistically start certifying transactions. If the order turns out to be wrong, the certification is redone in the correct order.

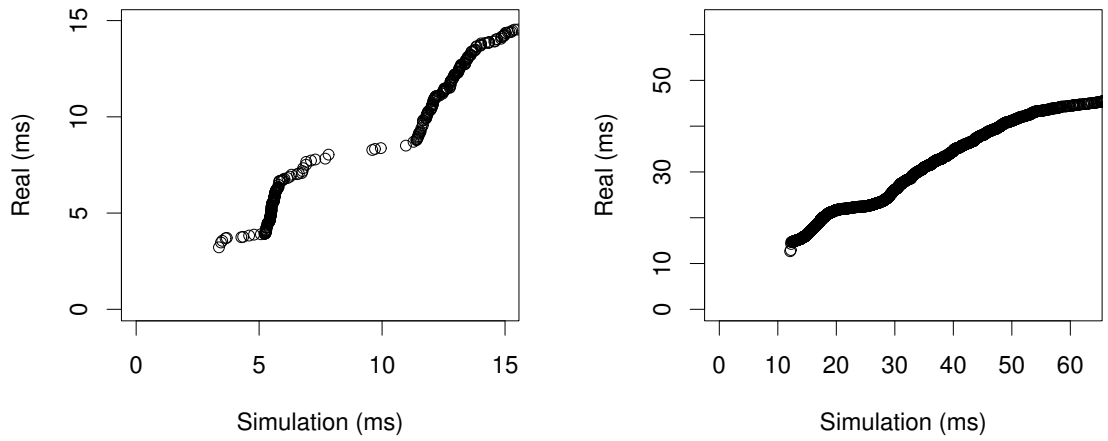
5 Model Instantiation and Validation

We validate our implementation by comparing it to a real database server. In this section we describe how the model is instantiated with a single host to reproduce the behavior of a real system and then compare the results.

5.1 Parameters

For validation we configure our model according to the equipment used for testing. This corresponds to a server with 2 Pentium III 1GHz processors and with 1GB of RAM. As the cache hit ratio observed has always been above 98%, we configure the simulation hit ratio to 1. This means that read items do not directly consume physical resources (CPU or storage), as this is already accounted for in the CPU times as profiled in PostgreSQL.

For storage we used a fiber-channel attached box with 4×36GB SCSI disks in a RAID-5 configuration.



(a) Read-only transactions.

(b) Update transactions.

Figure 2: Validation results (Q-Q plots).

The file system used to hold the database (executable and data) is ext3 (Linux version 2.4.21-pre3). The latency parameter for writing an item is determined by observing the commit times of transactions writing single items and it is set to 5ms. Throughput for the storage was determined by running the IOzone disk benchmark [2] on the target system with synchronous writes of 4KB pages and a variable number of concurrent process. This resulted in a maximum throughput of 9.486MBps. This is never a bottleneck for results presented in this paper.

The only remaining parameter is the amount of CPU consumed by the execution of each transaction. This is tightly related with the database system used and with the size of the database although not significantly affected by concurrency. We therefore chose to profile PostgreSQL [3], as each process handles a single transaction from start to end. This reduces the problem of profiling a transaction to that of profiling a process in the host operating system.

In detail, we used the CPU timestamp counter which provides accurate measure of elapsed clock cycles. By using a virtualization of the counter for each process [1] we also obtain measurements of process virtual time (i.e. the time elapsed when the process is not scheduled to run is not accounted for). To minimize the influence in the results, the elapsed times are transmitted over the network only after the end of each query (and thus out of the measured interval), along with the text of the query itself.

The time consumed by the transaction's execution is then computed from the logs. By examining the query itself, each transaction is classified. Interestingly, the processor time consumed during commit is for all transactions negligible (i.e. less than 2ms). In read-only transactions the real time of the commit

operation equals processing time, meaning that no I/O is performed. This does not happen in transactions that update the database.

After discarding aborted transactions and the initial 15 minutes, the resulting histogram allows an empirical distribution to be obtained and used later for simulation. However, some transactions classes (i.e. payment and orderstatus) perform some work conditionally and thus result in bimodal distributions. Therefore, we split each of these in two different classes. The resulting transaction classes can therefore be approximated by an uniform distribution.

5.2 Validation Results

The simulation with a single host is validated by comparing its results with real results. Validation is however limited by the number of clients that can be deployed and handled by the server. We have therefore used a run of TPC-C with 20 clients only.

In order to perform a fair comparison between a real and a simulated run, a quantile-quantile plot (Q-Q plot) is presented in Figure 2. The Q-Q plots are used to determine if the two data sets originate from a similar distribution. If so, the plot must be close to a 45^o line, indicating that there is no great departure from one another.

Analyzing the Q-Q plots one finds that the behavior of a simulated execution follows quite closely the real one. This difference is mainly due to the larger transactions which take a little more time to execute in the simulated run. Variance, in the read only transactions, is also greater in the simulation run. However, these two aspects do not degrade the simulation global performance when facing the real run.

6 Results

In this section we present results obtained with our model. Our aim is to determine the impact of distribution in resource usage and overall performance with an large number of clients.

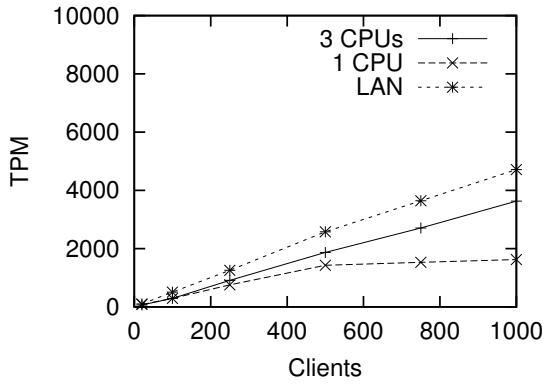
6.1 Performance

Starting with the same configuration parameters used for validation, we set up three different scenarios:

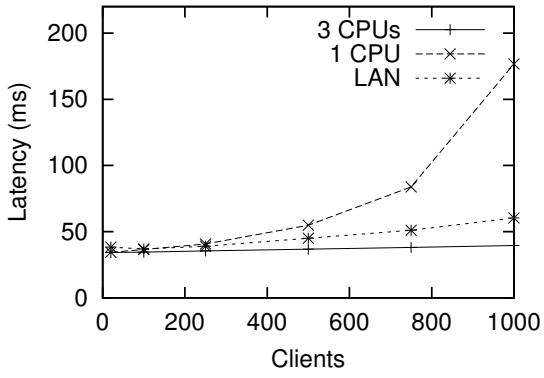
1 CPU This is a centralized system with a single processor and otherwise identical to the reference machine. This is the baseline for comparison.

3 CPUs This is a centralized system with three processors. The results represent near perfect scalability that should be approximated by a 3 node replicated DBSM system.

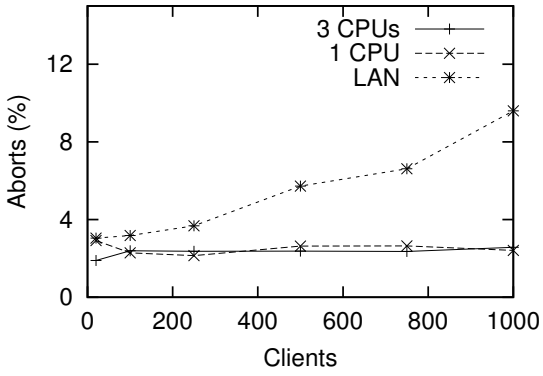
LAN This is a distributed system with 3 nodes connected by a 100Mbps local area network using the ESCADA protocol.



(a) Throughput.



(b) Latency.



(c) Abort rate.

Figure 3: Performance results.

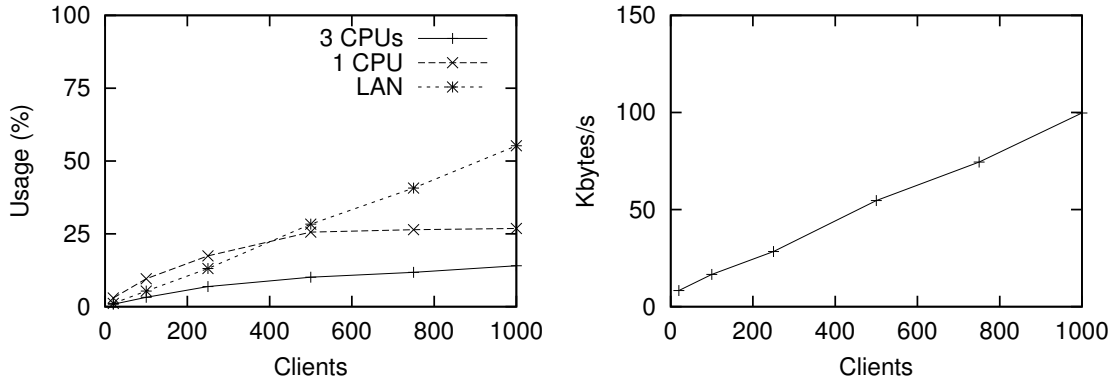
For each of these scenarios, we run simulations with an increasing number of clients from 20 to 1000 and compute the resulting latency, throughput and abort rate.

Figure 3(a) presents the number of transactions per minute observed. Notice that the 1 CPU system handles less than 2000 tpm, regardless of the offered load by an increasing number of clients. As a consequence latency, as shown in Figure 3(b) grows due to queuing. In contrast, the 3 CPU system scales linearly and thus there is no increase in latency.

The replicated system also allows for a linear increase in throughput. Surprisingly, throughput is even larger than with the centralized system with 3 CPUs. This is explained by Figure 3(c) that shows that an increasing number of the transactions processed are in fact aborted due to serialization conflicts during the termination protocol. The higher latency indicates also that the replicated system is getting some congestion. It is interesting to notice that, with a small number of clients, the latency overhead due to the replication protocol is very small.

Transaction	1 CPU	3 CPUs	LAN
delivery	0.00	0.00	1.28
neworder	1.09	1.05	1.72
payment-01	7.03	7.12	22.70
payment-02	0.00	0.00	16.53
All Update	2.48	2.50	10.33
orderstatus-01	6.21	6.82	5.47
orderstatus-02	0.00	0.00	0.00
stocklevel	0.00	0.00	0.00
All Read-only	2.02	2.13	1.65
All	2.44	2.47	9.60

Table 2: Abort rates with 1000 clients (%).



(a) CPU.

(b) Network.

Figure 4: Resource usage.

As we are using realistic traffic, it is also interesting to evaluate the impact of serialization conflicts in different transactions. This is presented in Table 2 for each transaction class alone, as well as for all read-only transactions and all update transactions.

6.2 Resource Usage

The information logged by the simulation runtime allows the precise evaluation of resource usage. Figure 4(a) presents average usage of each of the involved CPUs. This justifies the throughput and latency results of the previous section, showing that a single CPU is a bottleneck at approximately 25% usage, due to interaction of variability and locking mechanisms. A centralized system with 3 CPUs with average usage of 15% each allow for a higher throughput. The results obtained with the replicated database, when compared with the centralized 3 CPU system, have shown that there is a high overhead of termination and atomic multicast protocols.

	Simulation time	Real time
LAN	45:00.00	51:03.89
1 CPU	45:00.00	2:05.75
3 CPUs	45:00.00	2:08.36

Table 3: Simulation performance with 150 clients.

In fact, one can log separately different event handlers within the termination and atomic multicast protocols. This has shown that a large portion of the overhead is due to fragmentation and reassembly of large messages and is being addressed by current work. Nevertheless, average network output of replicas as presented by Figure 4(b) shows a linear increase in transmitted bytes with number of clients and transaction throughput. This shows that the protocol is scalable in terms of bytes transmitted.

6.3 Simulation performance

The usefulness of the model is also dependent on the amount of real time used to run simulations. Table 3 shows the amount of real time consumed for full 45 minutes runs with 150 clients of the 3 scenarios. The LAN scenario is the most time consuming due to running of real protocols which, as presented in the previous section, currently consume a large share of CPU. The optimization of such protocols will allow to keep the close mapping from real to simulated time with increasingly larger models.

Notice that one can obtain results without a full run of TPC-C with 45 simulated minutes, as the simulated database engine quickly achives stable behavior because it has no real caching inside. This makes it feasible to run a large number of simulations to explore the variation of parameters.

7 Related Work

Simulation has previously been used to evaluate DBSM replication [23]. The high level of that model easily allows the experimentation of several variations of the basic DBSM approach. Namely, of the reordering technique before certification. Nevertheless, the simplicity of the traffic generator used severely limits the detail of the results regarding conflicts.

An implementation of the DBSM using the PostgreSQL database engine is available and has been previously evaluated [20]. Although such an implementation provides results for a real running system, it is much harder to evaluate. For instance, it would be very difficult to set up and run the same experiments presented in this paper. This possibility is also invaluable when optimizing and debugging certification and communication protocols, as one can generate unlikely but possible environment scenarios to stress the implementation [7]. The usage of a high-level simulation model for the database engine allows us to easily experiment with different concurrency control models. In fact, although we have not presented it in this paper, we have already implemented different locking policies and are evaluating its impact in DBSM

replication. This would be very hard to do using PostgreSQL.

The use of simulation models has been used frequently to evaluate the performance of database processing techniques. In fact, our model of database processing is close to that of [6]. Our work differs mostly in the configuration of the model according to a real database engine and the consequent ability to validate the results experimentally as well as on the integration of real code. Combining simulated and real components in a single model has been described previously in the context of fault-tolerant distributed systems [7]. By using a standard simulation API we are however able to reuse an existing simulation model, SSFNet [14].

We have decided to use a high level model of the CPU time consumption by transaction processing, namely, by modeling it after the results of profiling. If we were interested in studying this parameter in more detail, we could extend the proposed simulation model with a low level model of access to items as has been previously described [19].

8 Conclusions

This paper presents a simulation model to evaluate the DSBM technique with unprecedented detail and realism by combining simulated and real components. This provides a coarse grained model of environment components with a very fine grained model of the components under study. The results obtained pinpoint possible problems in the current prototype or when processing specific transactions classes, which have not been identified in previous work. Nonetheless, results allow us to conclude that the DBSM replication technique is viable and cost-effective in real environments if one can optimize the communication protocol and then ensure that the number of aborts due to conflicts can be kept low despite an increasing load.

The same model can easily be applied to other problems. We are currently working on evaluating the DBSM in wide area networks and the interaction of locking policies with DBSM replication. We have also used subsets of the model to test specific protocols [33, 27] not directly related to database replication. In addition, the proposed model allows for extensive testing of certification and group protocols. In fact, it has been possible to assemble a set of tests which are run upon modification of protocol code to test its properties. This will be very useful in maturing the prototype into a solid implementation.

References

[1] High-resolution timers for the linux kernel. http://www.cs.wisc.edu/~paradyn/libhrtime.

[2] IOzone filesystem benchmark. http://www.iozone.org.

[3] PostgreSQL. http://www.postgresql.org.

[4] D. Agrawal, A. El Abbadi, and R. C. Steinke. Epidemic algorithms in replicated databases (extended abstract). In *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles*

of database systems, pages 161–172. ACM Press, 1997.

[5] D. Agrawal, G. Alonso, A. El Abbadi, and I. Stanoi. Exploiting atomic broadcast in replicated databases. In *Proceedings of EuroPar (EuroPar'97)*, Passau (Germany), 1997.

[6] R. Agrawal, M. Carey, and M. Livny. Concurrency control performance modeling: alternatives and implications. *ACM Transactions on Database Systems (TODS)*, 12(4):609–654, 1987.

[7] G. Alvarez and F. Cristian. Applying simulation to the design and performance evaluation of fault-tolerant systems. In *IEEE International Symposium on Reliable Distributed Systems*, 1997.

[8] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil. A critique of ANSI SQL isolation levels, 1995.

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

[10] K. Birman and R. van Renesse. *Reliable Distributed Computing with the Isis Toolkit*. IEEE Computer Society Press, 1994.

[11] N. Budhiraja, K. Marzullo, F. Schneider, and S. Toueg. The primary-backup approach. In S. Mullender, editor, *Distributed Systems*, chapter 8. Addison Wesley, 1993.

[12] G. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: a comprehensive study. *ACM Computing Surveys*, 33(4), December 2001.

[13] J. Cowie. *Scalable Simulation Framework API Reference Manual*, Mar 1999.

[14] J. Cowie, H. Liu, J. Liu, D. Nicol, and Andy Ogielski. Towards realistic million-node internet simulation. In *Proc. of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, Las Vegas, Nevada, Jun 1999.

[15] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, San Francisco, CA, 1993.

[16] K. Guo. *Scalable Message Stability Detection Protocols*. PhD thesis, Cornell University, Computer Science Department, May 1998.

[17] J. Holliday, D. Agrawal, and A. El Abbadi. The performance of database replication with group multicast. In *Proceedings of IEEE International Symposium on Fault Tolerant Computing (FTCS29)*, pages 158–165, 1999.

[18] M. Kaashoek and A. Tanenbaum. Group communication in the Amoeba distributed operating system. In *Proc. the* 11^{th} *Int'l Conf. on Distributed Computing Systems ICDCS*, pages 222–230, Washington, D.C., USA, May 1991. IEEE CS Press.

[19] W. Keezer. Array-driven simulation of real databases. In *Proc. of the 1998 Winter Simulation Conference*, Washington, DC, 1998.

[20] B. Kemme and G. Alonso. Don't be lazy, be consistent: Postgres-R, a new way to implement database replication. In *Proceedings of 26th International Conference on Very Large Data Bases (VLDB 2000)*, pages 134–143. Morgan Kaufmann, 2000.

[21] B. Kemme, F. Pedone, G. Alonso, and A. Schiper. Processing transactions over optimistic atomic broadcast protocols. In *Proceedings of 19th International Conference on Distributed Computing Systems (ICDCS'99)*, 1999.

[22] R. Ladin, B. Liskov, and L. Shrira. Lazy replication: Exploiting the semantics of distributed services. *ACM SIGOPS Operating Systems Review*, 25(1), January 1991.

[23] F. Pedone. *The Database State Machine and Group Communication Issues*. PhD thesis, Département d'Informatique, École Polytechnique Fédérale de Lausanne, 1999.

[24] F. Pedone, R. Guerraoui, and A. Schiper. Exploiting atomic broadcast in replicated databases. In *Proceedings of EuroPar (EuroPar'98)*, Southampton, England, September 1998.

[25] F. Pedone, R. Guerraoui, and A. Schiper. The database state machine approach. *Journal of Distributed and Parallel Databases and Technology*, 2003 (to appear).

[26] F. Pedone and A. Schiper. Optimistic atomic broadcast: A pragmatic viewpoint. *Theoretical Computer Science Journal*, 291:79–101, 2003.

[27] J. Pereira and R. Oliveira. A mutable protocol for consensus in large groups. In *Ws. Large Scale Group Communication (with SRDS'2003)*, 2003.

[28] J. Pereira, L. Rodrigues, and R. Oliveira. Semantically reliable multicast: Definition implementation and performance evaluation. *Special Issue of IEEE Transactions on Computers on Reliable Distributed Systems*, 2003. to appear.

[29] S. Pingali, D. Towsley, and J. Kurose. A comparison of sender-initiated and receiver-initiated reliable multicast protocols. In *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, May 1994.

[30] A. Schiper and A. Sandoz. Uniform reliable multicast in a virtually synchronous environment. In *IEEE International Conference on Distributed Computing Systems*, May 1993.

[31] F. Schneider. Replication management using the state-machine approach. In S. Mullender, editor, *Distributed Systems*, chapter 7. Addison Wesley, 1993.

[32] A. Sousa, F. Pedone, R. Oliveira, and F. Moura. Partial replication in the database state machine. In *IEEE Int'l Symp. Networking Computing and Applications*. IEEE CS, Oct 2001.

[33] A. Sousa, J. Pereira, F. Moura, and R. Oliveira. Optimistic total order in wide area networks. In *Proc. 21st IEEE Symposium on Reliable Distributed Systems*, pages 190–199. IEEE CS, October 2002.

[34] Transaction Processing Performance Council (TPC). TPC Benchmark™ C standard specification revision 5.0, February 2001.