View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Infoscience - École polytechnique fédérale de Lausanne

Predicting Replicated Database Scalability from Standalone Database Profiling

Sameh Elnikety
Microsoft Research
Cambridge, UK

Steven Dropsho
Google Inc.
Zurich, Switzerland

Emmanuel Cecchet
University of Massachusetts
Amherst, USA

Willy Zwaenepoel
EPFL
Lausanne, Switzerland

Abstract

This paper develops analytical models to predict the throughput and the response time of a replicated database using measurements of the workload on a standalone database. These models allow workload scalability to be estimated before the replicated system is deployed, making the technique useful for capacity planning and dynamic service provisioning. The models capture the scalability limits stemming from update propagation and aborts for both multi-master and single-master replicated databases that support snapshot isolation.

We validate the models by comparing their throughput and response time predictions against experimental measurements on two prototype replicated database systems running the TPC-W and RUBiS workloads. We show that the model predictions match the experimental results for both the multi-master and single-master designs and for the various workload mixes of TPC-W and RUBiS.

Categories and Subject Descriptors D.4.5. [**Reliability**]: Fault-tolerance; D.4.8. [**Performance**]: Measurements, Modeling and prediction, Operational analysis, Queuing theory; H.2.4 [**Systems**]: Distributed databases, Transaction processing.

General Terms Measurement, Performance, Design, Reliability, Experimentation.

Keywords *database replication; single-master systems; multi-master systems; generalized snapshot isolation.*

1. Introduction

Predicting the performance of replicated databases is important for their wide adoption. Performance models are employed for capacity planning [Lazowska 1984] and for dynamic service provisioning [Urgaonkar 2005b] as in data centers that host several e-commerce applications and receive external loads that vary with the diurnal cycles and seasonal effects. To the best of our knowledge, it is not possible yet to know how an application is going to scale on a replicated database without actually building the replicated system and running the application with a scaled workload.

In this paper, we develop analytical models that predict application workload scalability on a replicated database system. Performance of such replicated systems depends on the workload parameters. We demonstrate that measurements of the workload running on a standalone system capture sufficient information for our models to predict the performance as more replicas are added. The models are designed for middleware-based replicated systems in a LAN environment employing snapshot isolation and running transactional workloads from e-commerce.

Our models borrow from prior work in calculating abort rates by Gray et al. [Gray 1996] and from modeling update propagation by Jiménez-Peris et al. [Jiménez-Peris 2001], but go beyond these works by predicting throughput and response time estimates that combine update propagation overhead and conflicts, rather than calculating upper bounds on system scalability.

We model both multi-master systems (in which each replica handles both read-only and update transactions) and single-master systems (in which the master replica executes update transactions and slave replicas execute read-only transactions). We validate the models by comparing their predictions against the measured performance of prototypes for both the multi-master and single-master systems. While we are aware of the many complexity and availability tradeoffs between single- and multi-master replication, in this paper we focus only on performance prediction.

The contributions of this work are the following: (1) We derive analytical models that predict the performance of multi-master and single-master replicated databases running snapshot isolation. (2) We show how to use the analyt-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
EuroSys'09, April 1–3, 2009, Nuremberg, Germany.
Copyright 2009 ACM 978-1-60558-482-9/09/04...$5.00.

ical models to predict the performance of the two designs from workload measurements of a standalone system. (3) We validate the models by comparing their predictions against experimental measurements of prototype implementations.

The remainder of the paper is structured as follows: Section 2 presents the necessary background to follow the analytical models, which are derived in Section 3. Section 4 shows how to estimate model parameters. We describe the implementation of the prototype systems in Section 5 and experimentally validate the models against the prototype systems in Section 6. We discuss related work in Section 7, and present the conclusions in Section 8.

2. Background

Snapshot isolation (SI). Snapshot isolation (SI) [Berenson 1995] is an optimistic multi-version database concurrency control model for centralized databases. SI achieves high concurrency with low conflicts at the cost of using more space (employing multiple versions of data items) and aborting (rather than blocking and reordering) few update transactions. When a transaction begins, it receives a logical copy, called snapshot, of the database for the duration of the transaction. This snapshot is the most recent version of the committed state of the database. Once assigned, the snapshot is unaffected by (i.e., isolated from) concurrently running transactions. A read-only transaction can always commit after reading from its snapshot. An update transaction commits if it does not have a write-write conflict with any committed update transaction that ran concurrently. When an update transaction commits, it produces a new version of the database. The granularity of conflict detection is typically a row in a database table (i.e., a tuple in a relation).

SI has attractive performance properties. Most notably, read-only transactions do not get blocked or aborted, and they do not cause update transactions to block or abort, making SI particularly suitable for read-dominated workloads with short updates as in e-commerce environments.

Many database vendors use SI, e.g., PostgreSQL, Microsoft SQL Server and Oracle. SI is weaker than serializability, but in practice many applications run serializably under SI [Elnikety 2005, Fekete 2005b, Fekete 1999], including the widely used database benchmarks TPC-C, TPC-W and RUBiS.

Generalized snapshot isolation (GSI). Generalized snapshot isolation (GSI) [Elnikety 2005] extends SI to replicated databases such that the performance properties of SI in a centralized setting are maintained in a replicated setting. In addition, workloads that are serializable under SI are also serializable under GSI. When a transaction starts, it receives the most recent snapshot at the replica where it executes. Although this local snapshot might be slightly older than the globally latest snapshot, it is available immediately without the need for additional communication. A read-only transaction executes entirely locally at the receiving replica. An update transaction executes first locally at the receiving replica. Then at commit, the writeset of the transaction is extracted and a certification service is invoked. The certification service detects system-wide write-write conflicts. If no conflict is detected the transaction commits, otherwise it aborts. The writeset [Kemme 2000] of an update transaction captures the transaction effects and is used both in certification and in update propagation.

Multi-master replication. In a multi-master (MM) system [Elnikety 2006, Elnikety 2007, Kemme 2000, Lin 2005, Patiño-Martínez 2005], each replica executes both read-only and update transactions. The replication middleware resolves conflicts by aborting conflicting update transactions. The MM system consists of a load balancer, several database replicas and a certifier that certifies update transactions to prevent write-write conflicts.

Single-master replication. In a single-master (SM) system [Daudjee 2006, Gray 1996, Plattner 2004] (also called master-slave), the master database executes all update transactions and several slave replicas execute read-only transactions. Restricting the execution of update transactions to the master makes SM systems less flexible but simpler to build compared to MM systems. A SM system is simpler to build because it does not need a certifier. The master handles all write operations, and therefore its concurrency control subsystem can abort transactions that introduce write-write conflicts.

Conflict window. The conflict window [Elnikety 2005] captures the time interval during which an update transaction is vulnerable to write-write conflicts, which result in aborting the update transaction. For a standalone database, the conflict window of an update transaction is its execution time on the database. For a single-master system, the conflict window is the execution time on the master. For a multi-master system, the conflict window has three components: (1) the age (staleness) of the snapshot the transaction receives, (2) transaction local execution time on the database replica, and (3) time for the certification service to certify the transaction.

3. Analytical Models

The analytical models aim to predict the performance of e-commerce workloads on replicated snapshot-isolated databases. Our aim is to capture the essential system features, including update propagation and aborts, while keeping the models sufficiently simple to be analytically tractable.

3.1 Workload

The transaction workload for a standalone database consists of R read-only transactions per second and W update trans-

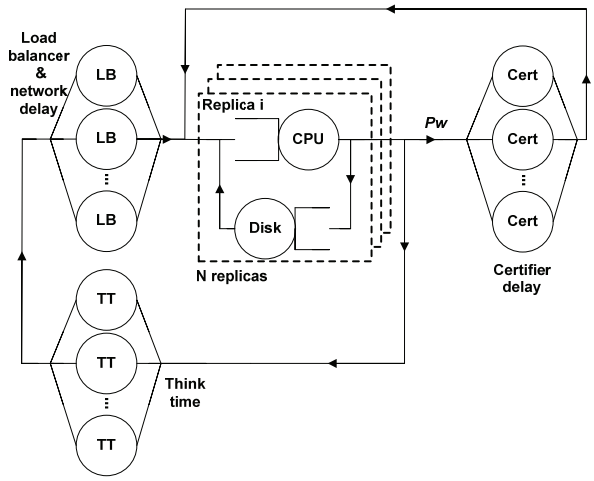

Figure 1. Multi-master queuing model (MM system).

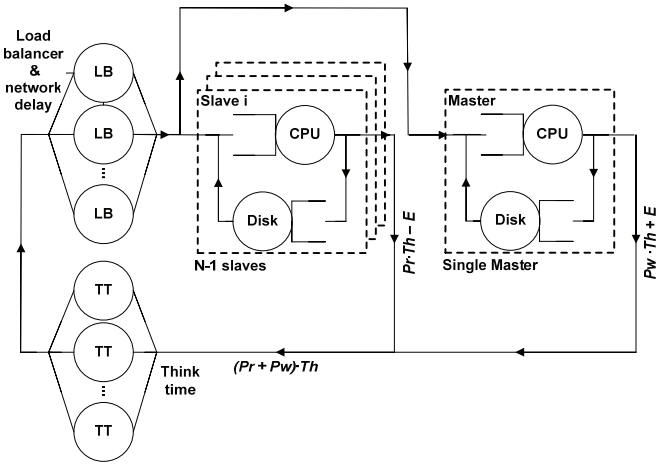

Figure 2. Single-master queuing model (SM system).

actions per second generated by a fixed number of clients. Each client submits a transaction, waits for the database response, examines the response during the think time, and then submits the next transaction, following a closed-loop model [Schroeder 2006]. The fraction of read-only transaction is *Pr* and the fraction of update transactions is *Pw*, such that $Pr + Pw = 1$.

We scale the workload with the number of replicas in the replicated database such that a replicated database system that has *N* replicas receives requests from *N* times the number of clients in a standalone database.

3.2 Queuing models

In this section we construct queuing models that compute the throughput and response time of each replica using transaction service demands that we derive in Section 3.3.

3.2.1 Multi-master

Figure 1 depicts a separable closed queuing network that captures the components of the multi-master system. We model the CPU and disk on the database replica as regular service centers with queues. We model the delays introduced by the load balancer and local area network as a single delay center called load balancer delay. The certification time is almost constant, insensitive to the number of concurrent certification requests. We, therefore, approximate the certifier as a delay center rather than a service center to make the model tractable. We provide more details in Section 6.3 to justify these assumptions.

Model inputs. The model inputs are the following: service demands at each service center, D_{MM} for CPU and disk; delay time at each delay center for think time, load balancer, and certifier delays; number of replicas (*N*); and number of clients per replica (*C*).

Model outputs. The model computes average throughput and response time. The model produces additional data for each resource such as utilization and residence time.

Solving the model. We use the mean value analysis (MVA) algorithm [Lazowska 1984] which is a standard algorithm to solve closed-loop queuing models. All replicas are identical and they contribute the same throughput assuming perfect load balancing. MVA iterates over the number of clients, adding *N* clients (one client per replica) in each iteration. MVA computes the residence time at each service center, system throughput and queue lengths.

3.2.2 Single-master

Figure 2 depicts a separable closed queuing network for the single-master system. The single-master model has similar inputs and outputs as the multi-master model. Solving the model, however, requires balancing the load among the master and the slaves.

Intuitively at steady state when the system is balanced, the ratio of the slaves throughput : master throughput should be *Pr* : *Pw*. Here the queuing model is not symmetric because updates are handled by the master only, raising two unbalanced cases: First, the master becomes underutilized if the transaction workload is dominated by read-only transactions which make the slaves the bottleneck. Since the master has excess capacity, it should process extra read-only transactions as well as all update transactions.

Second, when the master is the bottleneck, the total system throughput becomes limited by the master, forcing clients to queue at the master and reducing the load on the slaves.

Figure **3** presents a load balancing algorithm that balances the load and accounts for these two cases. If the system is balanced, the algorithm terminates immediately and reports balanced read and update throughputs.

```
Sub-routine:
Center.MVA()
   Inputs: readClients, writeClients
   Outputs: readThoughput , writeThoughput

Balancing Algorithm:
masterClients = Pw·C·N
slaveClients = Pr·C·N/(N-1)
( -,writeThput ) = Master.MVA( 0, masterClients )
( readThput, - ) = Slave.MVA( slaveClients, 0 )

//is the system balanced?
if( readThput : writeThput ≈ Pr : Pw )
    return (readThput , writeThput )

//the system is not balanced.
//either master has exceess capacity or it is the bottleneck.
if( readThput : writeThput < Pr : Pw ) {
   //master has excess capacity: add reads to master
   j = 0;
   loop {
      j++;
      ( extraReadThput , writeThput ) =
               Master.MVA( j·(N-1) , masterClients )
       ( readThput , - ) = Slave.MVA( slaveClients - j , 0 )
   } until ( (extraReadThput +readThput) : writeThput ≈ Pr : Pw )
   return (extraReadThput +readThput , writeThput )
} else {
   //master is the bottleneck, more clients queue at master
   j = 0;
   loop {
      j++;
       ( - , writeThput ) = Master.MVA( 0, masterClients + j·(N-1) )
       ( readThput, - ) = Slave.MVA( slaveClients – j , 0 )
   } until ( readThput : writeThput ≈ Pr : Pw )
   return ( readTh , writeTh )
}
```

Figure 3. Balancing throughput of master and slaves.

If the system is not balanced, the algorithm uses two properties to rebalance the solution: (1) the constant ratio of read-only to update transactions *Pr* : *Pw* provided as an input property of the workload, and (2) the fixed number of clients in system, who are distributed among centers proportional to residence times. We use MVA as a building block. The system has *1* master and *N-1* slaves, and the total number of clients is *N·C*. We distribute clients among the master and slaves, and solve the queuing model. We compare the ratio of the resulting throughputs (slaves throughput : master throughput = *Pr* : *Pw*).

If the master throughput is too high, indicating there is excess capacity at the master, we move read-only transactions from the slaves to the master until the ratios are balanced. If the slaves' throughput is too high, we reduce the number of clients on the slaves because more clients queue on the master. Both cases terminate quickly because they iterate over a finite number of clients. The system response time is computed using Little's law [Kleinrock 1975].

3.3 Service demands

We estimate the CPU and disk service demands at the replicas for executing transactions. We first model resources in a standalone database system and then extend the discussion to replicated databases. Table 1 lists the symbols used in the model.

3.3.1 Standalone database

The average service demand of a read-only transaction is *rc* (e.g., CPU time or disk time), and the average service demand of an update transaction is *wc*. Some update transactions are aborted due to write-write conflicts under snapshot isolation. Those transactions are retried. Let A_1 be the probability that an update transaction aborts in a standalone database. To successfully commit *W* update transactions, $W/(1 - A_1)$ update transactions are submitted, of which *W* commit and $W \cdot A_1/(1 - A_1)$ abort. Therefore, the resources required for *R* committed read-only transactions and *W* committed update transactions are:

$$Load(1) = R \cdot rc + \frac{W}{(1 - A_1)} \cdot wc$$

The average the service demand for one transaction is then:

$$D(1) = Pr \cdot rc + \frac{Pw}{(1 - A_1)} \cdot wc$$

To quantify A_1, we run the workload and measure A_1 directly on the standalone database system. However, here we derive the abort probability for a standalone database because we later use a similar derivation to relate the abort rate of a replicated database to that of the standalone database.

Assume the database has *DbUpdateSize* objects that can be modified by update transactions. Each update transaction modifies *U* objects. The probability that an update operation conflicts with another update operation is $p = 1/DbUpdateSize$.

Conversely, an update transaction *succeeds* if it conflicts with *none* of the concurrent updates. The probability of success is $(1 - p)$ to the power of the total number of update operations by the concurrent transactions. The execution time of the update transaction is $L(1)$, which is its conflict window. The number of concurrent updates during the conflict window of the update transaction is $L(1) \cdot W \cdot U$. Since each of the *U* update operations of an update transaction must succeed for the transaction to commit, the probability of success is:

$$Success(1) = (1 - p)^{L(1) \cdot W \cdot U^2}$$

The probability of abort is $A_1 = 1 - Success(1)$:

$$A_1 = 1 - (1 - p)^{L(1) \cdot W \cdot U^2}$$

3.3.2 Multi-master replicated database

Each replica in a multi-master (MM) system processes its local input transactions *plus* the writesets from remote update transactions. Depending on the workload, the cost of a propagated writeset can be less than the cost of fully processing the original update transaction. We, therefore, assign it a cost, *ws*, the average service demand required to process a propagated writeset. In an *N*-replica system, each replica commits *R* read-only transactions, *W* update transactions and $(N\text{-}1)\cdot W$ propagated writesets.

The abort probability is A_N. Aborts affect only local update transactions, and do not affect propagated writesets, which partially cause higher A_N. The resources needed to process the transaction workload at the replica are:

$$Load_{MM}(N) = R \cdot rc + \frac{W}{(1 - A_N)} \cdot wc + W \cdot (N-1) \cdot ws$$

The average service demand for one transaction is:

$$D_{MM}(N) = Pr \cdot rc + \frac{Pw}{(1 - A_N)} \cdot wc + Pw \cdot (N-1) \cdot ws$$

Next, we relate A_N to A_1. The same formula used to derive *Success(1)* in Section 3.3.1 can be used to derive the success probability $Success_{MM}(N)$ for the MM system. Approximately, the *N*-replica multi-master system has *N* times the throughput and a different conflict window, *CW(N)*. The total concurrent update operations is $CW(N)\cdot N\cdot W\cdot U$, and the success probability is:

$$Success_{MM}(N) = (1-p)^{N \cdot CW(N) \cdot W \cdot U^2}$$

which can also be written as:

$$Success_{MM}(N) = \left[(1-p)^{L(1) \cdot W \cdot U^2} \right]^{N \cdot \frac{CW(N)}{L(1)}}$$

$$Success_{MM}(N) = Success(1)^{N \cdot \frac{CW(N)}{L(1)}}$$

Since the probability of success is (1– prob(abort)), then:

$(1 - A_1) = Success(1)$ and $(1 - A_N) = Success_{MM}(N)$

Therefore,

$$(1 - A_N) = (1 - A_1)^{N \cdot \frac{CW(N)}{L(1)}}$$

The final service demand equation is the following:

$$D_{MM}(N) = Pr \cdot rc + \frac{Pw}{(1 - A_1)^{N \cdot \frac{CW(N)}{L(1)}}} \cdot wc + Pw \cdot (N-1) \cdot ws$$

Table 1. Model parameters and symbols.

Symbol	Meaning
A'_N	Abort rate of an update transaction in single-master system having 1 master and *N-1* slaves (§ 3.3.3)
A_1	Abort rate of an update transaction in a standalone database system (§ 3.3.1)
A_N	Abort rate of an update transaction in multi-master system having *N* replicas (§ 3.3.2)
CW(N)	Conflict window of an update transaction on a multi-master system having *N* replicas (§ 3.3.2, § 4.1.1)
DbUpdateSize	Number of objects in the database that can be modified by update transactions (§ 3.3.1)
$D_{master}(N)$	Average service demand to execute one transaction on a master in a single-master system having *1* master and *N-1* slaves (§ 3.3.3)
$D_{MM}(N)$	Average service demand to execute one transaction in a multi-master system having *N* replicas (§ 3.3.2)
$D_{slave}(N)$	Average service demand to execute one transaction on a slave in a single-master system having *1* master and *N-1* slaves (§ 3.3.3)
E	Read-only transactions per second that are executed on the master of a single-master system (to balance the load among the master and slaves) (§ 3.3.3)
GSI	Generalized snapshot isolation (§ 2)
L(1)	Execution time of an update transaction on a standalone database (§ 3.3.1)
MM	Multi-master (§ 3.3.2)
N	Number of replicas in a replicated system (§ 3.1)
p	Probability that one update operation in an transaction conflicts with an update operation in another concurrent transaction (§ 3.3.1)
Pr	Fraction of transactions that are read-only (§ 3.1)
Pw	Fraction of transactions that are update (§ 3.1)
R	Read-only transactions per second in the input workload (§ 3.1)
rc	Service demand of executing a read-only transaction (§ 3.3.1)
SI	Snapshot isolation (§ 2)
SM	Single-master (§ 3.3.3)
U	Number of update operations in each update transaction (§ 3.3.1)
W	Update transactions per second in the input workload (§ 3.1)
wc	Service demand of executing an update transaction (§ 3.3.1)
ws	Service demand of applying a writeset (§ 3.3.1).

3.3.3 Single-master replicated database

An *N*-replica SM system has two components, *1* master and *N-1* slaves. We derive the service demand on each component, assuming a total system load equivalent to an N-replica MM system. The SM system commits $N \cdot W$ update and $N \cdot R$ read-only transactions.

Master Service Demand. The master processes all update transactions in the system. Aborts increase the number of submitted update transactions. $N \cdot W/(1 - A'_N)$ update transactions are submitted to commit $N \cdot W$ update transactions under the master abort rate of A'_N. The master's resource consumption is:

$$Load_{master}(N) = N \cdot \frac{W}{(1 - A'_N)} \cdot wc$$

The average service demand per update transaction is:

$$D_{master}(N) = \frac{wc}{(1 - A'_N)}$$

The difference between the MM system abort probability A_N and the master-slave abort probability A'_N is that the master resolves all conflicts locally, like a standalone system, but at a higher rate of update transactions than the standalone system.

Slave service demand. The slaves process $N \cdot R$ read-only transactions and $N \cdot W$ propagated writesets from the master. Thus, each of the $(N-1)$ slaves must process $N / (N-1)$ read-only transactions and all remote writesets. The slaves process only committed writesets; there are no aborts at the slaves. The resource consumption at a slave is:

$$Load_{slave}(N) = \frac{N}{N-1} \cdot R \cdot rc + N \cdot W \cdot ws$$

The service demand per transaction is the following:

$$D_{slave}(N) = rc + \frac{Pw}{Pr} \cdot (N-1) \cdot ws$$

The performance of the single-master system can be limited by either the master or the slaves.

Executing extra read-only transactions at master. If the master has extra capacity, the service demand equations change. The master processes E extra read-only transactions, while the $N-1$ slaves process $N \cdot R - E$ read-only transactions. The average service demand per transaction becomes:

$$D_{master}(N) = \frac{E}{N \cdot W + E} \cdot rc + \frac{N \cdot W}{N \cdot W + E} \cdot \frac{wc}{(1 - A'_N)}$$

$$D_{slave}(N) = rc + \frac{N \cdot (N-1) \cdot W}{N \cdot R - E} \cdot ws$$

3.4 Summary of model assumptions

We summarize the main assumptions of the analytical model.

1. The workload is based on e-commerce applications; i.e., high volume of relatively short lived transactions that can be effectively distributed across replicas. E-commerce workloads are typically read dominated.
2. The concurrency protocol is based on snapshot isolation; therefore, only write-write conflicts occur.
3. Since (generalized) snapshot isolation is a multi-version concurrency control algorithm, the bottleneck is much more likely to be a physical resource rather than a logical resource. GSI trades space (as it maintains multiple versions) to achieve fewer conflicts; readers never block writers and writers never have to wait for readers. The model assumes that the bottleneck is a physical resource rather than a logical resource. The model, therefore, does not directly capture logical resources such as semaphores and lock contention. However, their effects are partially reflected on the physical resources.
4. The abort probability of update transactions in the standalone database as well as in the replicated database is small. Updatable data items are updated uniformly, i.e., the database does not have a hotspot.
5. The database server is scalable; resource consumption is linear with the server throughput. Modern server operating systems are unlikely to thrash because they employ mechanisms that prevent over-subscription of physical resources, such as the O(1) thread scheduler and admission control policies. The model does not apply in overload regions that are not linear if they exist.
6. The model uses perfect load balancing among identical machines and inherits the assumptions employed by the MVA algorithm [Lazowska 1984], such as having exponential distributions of the service demands.
7. The database is replicated in a LAN environment rather than a WAN or geographically distributed environment.

When these assumptions are not valid, the model in general predicts an upper bound on performance. This is the case for example, when the abort probabilities are high. We investigate the sensitivity of the model to some of these assumptions in Section 6.3.

4. Estimating model parameters

We use standard workload characterization techniques [Menasce 1998] to estimate model parameters. These techniques gather information using measurement on an offline system. Online methods can, however, be employed if the underlying live database system provides the required resource utilization for each transaction.

4.1.1 Multi-master parameters

We take a backup of the database and capture the transaction workload from the standalone database system using the database log file. The log must contain the full SQL statements, a client or session identifier and a start timestamp at which the statement was executed. Our experience is that these values can be generated by most database logging facilities. For example, in PostgreSQL 7, this information is generated by turning on log_statement, log_pid, log_connection and log_timestamp. In PostgreSQL 8, a log line prefix string such as '%r %p %m %c %x' captures the necessary information.

Additionally, we need to generate the writesets corresponding to the update transactions. This is done by defining triggers on all tables to extract and record the transaction writeset.

The service demand equation $D_{MM}(N)$ in Section 3.3.2 requires the following parameters: Pr, Pw, A_1, $CW(N)$, $L(1)$ as well as rc, wc, ws for the CPU and disk.

***Pr, Pw,* and A_1.** We count the number of read-only and update transactions in the captured log to determine the fractions Pr and Pw. We count the number of aborted update transactions to calculate the abort probability A_1.

***rc, wc, ws* and *L(1)*.** We instrument a standalone database with triggers and play the log to capture the writesets. We play read-only transactions from the log against the database and collect CPU and disk utilization to compute the service demands rc_{CPU} and rc_{disk} using the Utilization Law [Lazowska 1984]. The average service demand at a resource is the resource utilization divided by the throughput. Next we play update transactions against the database to compute wc_{CPU} and wc_{disk}. We also play the writesets to compute ws_{CPU} and ws_{disk} in a separate run. We finally replay both read-only and update transactions to measure $L(1)$, the average response time for update transactions in a standalone system.

A_N, CW(N). We can derive A_N from A_1 using the formula below, but this requires $CW(N)$, the conflict window, which is not available from standalone measurements.

$$(1 - A_N) = (1 - A_1)^{N \cdot \frac{CW(N)}{L(1)}}$$

We approximate the conflict window by the sum of CPU residence time, disk residence time and certification time for update transactions [Elnikety 2005]. However, to compute the latter three terms we still need $CW(N)$. Since the MVA algorithm iterates over the number of clients, we approximate $CW(N)$ at iteration i+1 by the sum of CPU, disk residence time and certification time at iteration i. This slightly underestimates the abort probability. We investigate higher abort rates in Section 6.3.3.

Finally, the model requires the think time (Z) and the number of clients (N). In the experimental validation section these values are inputs. However in a practical deployment, well-known approaches [Jain 1991, Urgaonkar 2005] can be used to estimate the think time and predict the number of clients. The certifier delay time is 12 ms as discussed in Section 6.3.2.

4.1.2 Single-master parameters

We estimate the model parameters for SM in the same way as for MM, except for A'_N.

***A'_N* (Master-Slave Abort Rate).** Because all updates are processed at the master, the abort rate for any level of replication can be measured directly by loading a database with a scaled update transaction workload.

5. Implemented Systems

5.1 Multi-master system

The MM system is based on Tashkent [Elnikety 2006]. Each replica has two main components: a database system and a proxy which intercepts all incoming requests to the database, as shown in Figure 4.

Transaction processing. When the load balancer receives a transaction T, it forwards T to the proxy of the least loaded replica. The proxy executes T on the database. All T's read and write operations, e.g., the SELECT, UPDATE, INSERT and DELETE SQL commands, are executed locally on the replica. The proxy intercepts the SQL commit command at which point the proxy examines the writeset of T. If the writeset is empty (T is read-only), the proxy commits T immediately. Otherwise, the proxy invokes the certification service, sending the writeset of T and the version of its snapshot to the certifier. The certifier decides whether to commit or abort the update transaction and performs the update propagation functionality. When the proxy receives the certifier response, it either commits or aborts T, and forwards the outcome to the client.

Replica database. We use a database engine on each replica that processes client read-only and update transactions.

Replica proxy. The proxy performs two main functions. First, it applies incoming writesets to the database. Second, it intercepts all requests to the database to prevent interference between local update transactions and propagated writesets. The proxy eagerly extracts the writeset of each transaction to perform early certification on partial write-

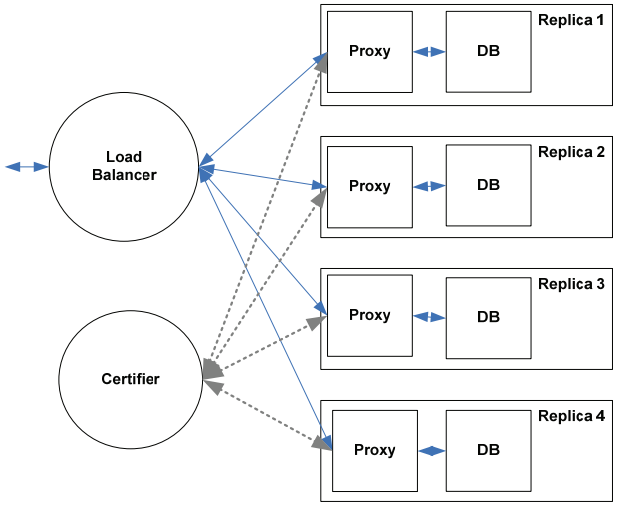

Figure 4. Multi-master (MM) system.

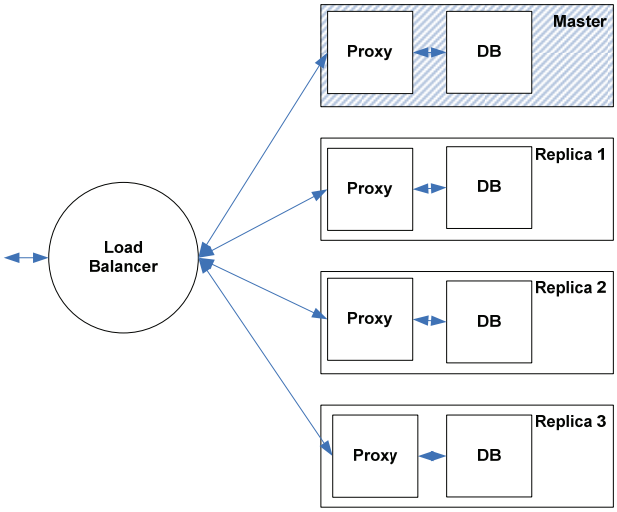

Figure 5. Single-master (SM) system.

sets, obviating the hidden deadlock problem [Lin 2005]. It aborts local update transactions whenever they conflict with a propagated writeset [Elnikety 2006].

Certifier. Certification is a lightweight stateful service that maintains committed writesets and their versions. The request to certify a transaction contains its writeset and version. The certifier detects write-write conflicts by comparing the writeset of the transaction to be certified to the writesets of the transactions that committed after the version supplied in the request. An update transaction is committed when its writeset is made persistent by the certifier. Certification is deterministic and the certifier is replicated using Paxos [Lamport 1998] for fault-tolerance.

5.2 Single-master system

Figure 5 depicts the architecture of a single-master system. The master database executes both read-only and update transactions. The slaves execute read-only transactions and the propagated updates from the master.

Load balancer. The load balancer dispatches all update transactions to the master. When the load balancer receives a read-only transaction, it selects the least loaded replica (among the master and slaves) and forwards the transaction to that replica. When the load balancer receives a writeset from the master, it relays the writeset to the slaves.

Transaction processing. The master executes update transactions and either commits or aborts them. On a commit, the master proxy extracts the transaction's writeset from the master database. This information is forwarded to the load balancer, which forwards the commit to the client. Read-only transactions can execute on master or slaves.

Master database. The master database processes all update transactions. We define triggers on all replicated tables to capture the transaction writeset in main memory.

Master proxy. The master proxy intercepts incoming requests to the master database. When the proxy intercepts the SQL COMMIT, it invokes a trigger to retrieve the writeset of the transaction and then forwards the commit command to the database.

Slave database. The slave database is effectively a cache against which read-only transactions are executed.

Slave proxy. The slave proxy applies incoming writesets to the database. It is the only source of updates to the database.

6. Experimental Validation

6.1 Experimental setup

TPC-W Benchmark. TPC-W is a benchmark from the Transaction Processing Council designed to evaluate e-commerce systems. It implements an on-line bookstore and has three workload mixes that differ in the relative frequency of each of the transaction types. The *browsing mix* workload has 5% updates, the *shopping mix* workload has 20% updates, and the *ordering mix* workload has 50% updates. The shopping mix is the main workload but we report results from all three mixes. The average size of a propagated writeset is 275 bytes. The TPC-W database standard scaling parameters are 100 EBS (emulated browsers) and 10,000 items. The database size is 700 MB.

RUBiS Benchmark. RUBiS [Amza 2002] is a popular e-commerce benchmark. It models an auction site like eBay and has two workloads: the browsing mix (entirely read-only) and the bidding mix (20% update transactions). The average size of a propagated writeset is 272 bytes. The scaling parameters are 1M users, 10,000 active items, and 500,000 old items. The database size is 2.2 GB.

Hardware and Software Environment. Each machine in the database cluster runs the 2.6.11 Linux kernel on a single Intel Xeon 2.4GHz CPU with 1GB ECC SDRAM, and a 120GB 7200pm disk drive. The machines are interconnected using gigabit Ethernet. We monitor the system load with a modified version of the Mercury server management system [Heath 2006]. For the certifier, we use a leader and two backups for fault tolerance. We use the PostgreSQL 8.0.3 database configured to run transactions at the snapshot isolation level (which is the strictest isolation level in PostgreSQL and called the "*serializable transaction isolation level"*). Both TPC-W and RUBiS are serializable under GSI [Elnikety 2005].

To drive the replicated database system, we use many client machines. Each client machine runs Tomcat 5.5 and a remote terminal emulator (RTE) for TPC-W or RUBiS. The RTE is a multithreaded Java program in which each thread represents one client and the application server (Tomcat) executes the requested Java Servlets which access the database using JDBC. If an update transaction is aborted, the Java Servlet retries the transaction.

The client machines are lightly loaded and the processing delay is less than 100 ms per transaction. Client think time follows an exponential distribution with an average of 900 ms. For the analytical models we use 1000 ms as the effective think time to account for processing times on the client machines, think time, load balancer delay and networking delay. Each point in the graphs below represents the result of one experiment. We report sustained average throughput and response time during 15 minutes after a warm-up period of 10 minutes. These intervals are selected such that the measurements are performed while the system performance is in steady state.

6.2 Comparing prediction and measurement

6.2.1 Validation using TPC-W

To evaluate the accuracy of the models, we compare the measured performance of the TPC-W benchmark to the predicted performance across the three workload mixes.

TPC-W parameters are summarized in Table 2. The parameters needed for modeling CPU and disk service demands per transaction are listed in Table 3. The abort rate of TPC-W in the standalone database, A_1, is very small for all mixes below 0.023%. We address the topic of prediction under higher abort rates in Section 6.3.3.

Figure 6 plots the throughput in transactions per second (tps) on the y-axis for TPC-W browsing, shopping and ordering mixes as a function of the number of replicas on the x-axis for the MM system using solid lines. The corresponding throughput curves predicted by the model are shown using dotted lines. The browsing mix scales almost linearly: Its throughput curve starts at 22 tps at one replica and increases to 347 tps at 16 replicas, which is a speedup of 15.7 times. The browsing mix has excellent scalability because it is dominated by read-only transactions.

Table 2. TPC-W parameters.

Mix	Read (*Pr*)	Write (*Pw*)	Clients per Replica (*C*)	Think Time (*Z*)
Browsing	95%	5%	30	1000 ms
Shopping	80%	20%	40	1000 ms
Ordering	50%	50%	50	1000 ms

Table 3. Measured service demands (in ms) for TPC-W.

Mix	Resource	Read(*rc*)	Write(*wc*)	Writeset(*ws*)
Browsing	CPU	41.62	17.47	3.48
	Disk	14.56	8.74	2.62
Shopping	CPU	41.43	12.51	3.18
	Disk	15.11	6.05	1.81
Ordering	CPU	22.46	13.48	4.04
	Disk	12.62	8.34	1.67

In contrast, the ordering mix increases from 45 tps at one replica up to 304 tps at 16 replicas yielding a speedup of 6.7 times due to the high ratio of update transactions in the workload. Notice that read-only transactions are in general more expensive than update transaction. For this reason, the browsing mix starts at 22 tps on one replica, whereas the ordering mix starts at 45 tps. As more replicas are added the cost of processing writesets (during update propagation) limits the scalability in the ordering mix.

The predicted throughput curves from the model match the measured throughputs. We find that the model captures the overhead of processing update transactions in the replicated system.

Figure 7 depicts the average response times for the three TPC-W mixes. The x-axis is the number of replicas and y-axis is the average response time in millisecond (ms). The response time curve for the browsing mix remains almost flat because there are a few update transactions. We see an increase in the response time curve for the ordering mix. The model predicts response times well for the three mixes.

In summary, the multi-master performance estimates match well with the measured results for all workload mixes with an error margin below 15%. Next, we analyze performance measurements and predictions for the single-master system.

Figure 8 plots the throughput of TPC-W browsing, shopping and ordering mixes as a function of the number of replicas for the single-master (SM) replicated database system. In contrast to MM, update transactions are executed on the master only and the system saturates as soon as the master becomes the bottleneck. Both the real system and the model scale linearly with the browsing mix since it is dominated by read-only transactions and the extra capac

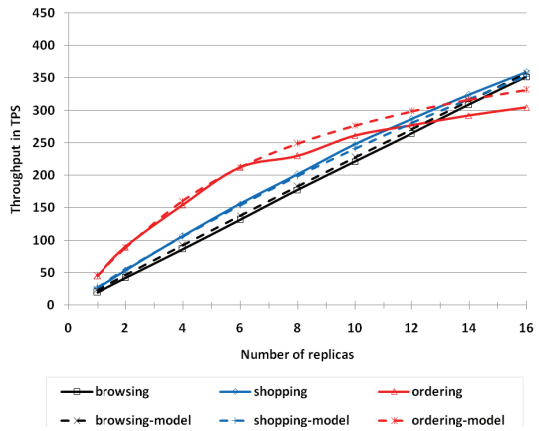

Figure 6. TPC-W throughput on MM system.

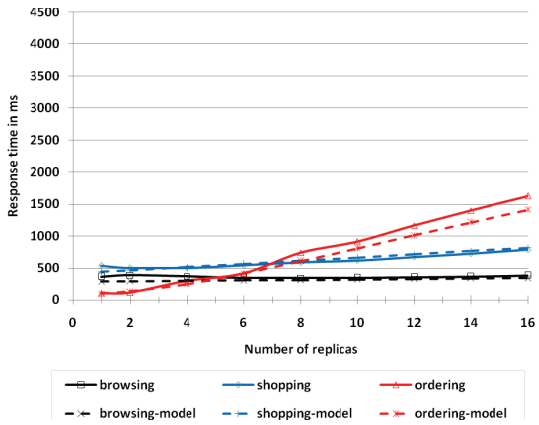

Figure 7. TPC-W Response time on MM system.

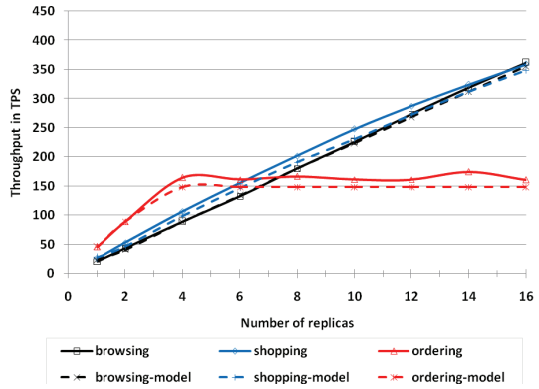

Figure 8. TPC-W throughput on SM system.

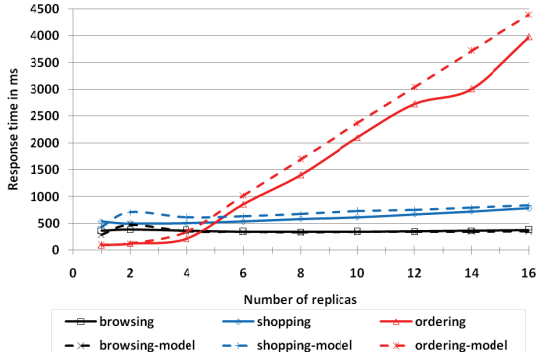

Figure 9. TPC-W Response time on SM system.

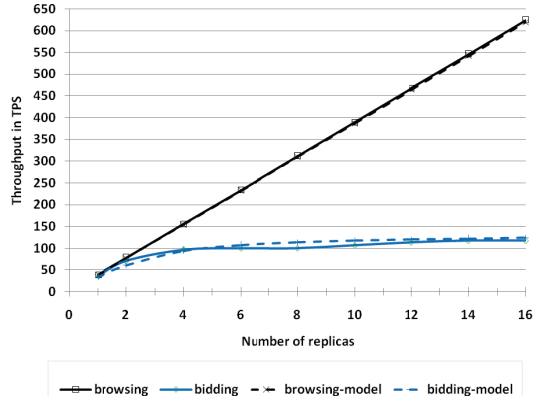

Figure 10. RUBiS Throughput on MM system.

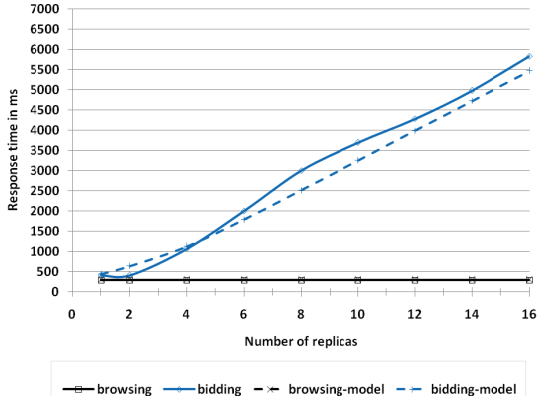

Figure 11. RUBiS response time on MM system.

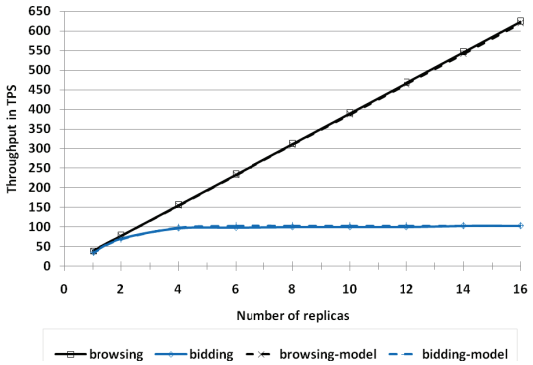

Figure 12. RUBiS throughput on SM system.

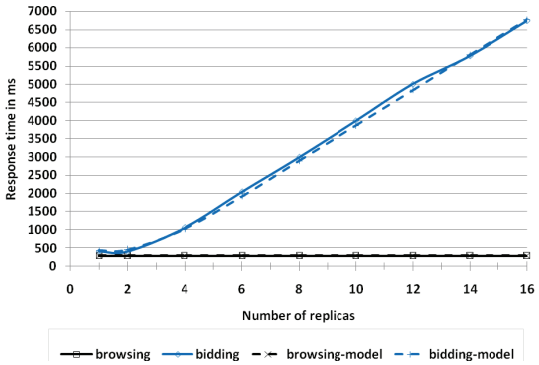

Figure 13. RUBiS response time on SM system.

ity at the master is employed to handle read-only transactions when there are one or more slaves.

With the greater ratio of update transactions in the ordering mix, SM saturates at 4 replicas. The master's CPU becomes the bottleneck, indicating that using a more powerful machine as the master would mitigate this bottleneck and improve system scalability.

Response time graphs are depicted in Figure 9. The response time curves for the browsing and shopping mixes are almost flat. The ordering mix response time increases rapidly after 4 replicas as the master becomes the bottleneck due to processing all update transactions. The predicted throughput and response times for SM match the measure values.

6.2.2 Validation using RUBiS

Table 4 summarizes RUBiS system parameters and Table 5 lists the measured service demands for the benchmark. Note that the browsing mix does not contain any update transactions.

Figure 10 shows the measured and predicted performance of the RUBiS benchmark on the multi-master system. The browsing mix consists of 100% read-only transactions. Hence, the measured and predicted throughput scales linearly with the number of replicas. The scalability of the bidding mix is much more modest with the peak throughput reached at 6 replicas. In this particular mix, update transactions update a small amount of data but incur a high cost due to enforcing integrity constraints and updating indexes. Therefore the cost of applying writesets is only slightly less than the cost of the original update transaction as reflected in the disk service demands for writes and writesets of the bidding mix in Table 5.

Figure 11 depicts the average response times. The model predicts well both the average throughput and the response time for RUBiS in the multi-master system.

Next we turn to the single-master replicated database. Figure 12 and Figure 13 present the throughput and response time for the two RUBiS mixes. The browsing mix scales linearly since the load can be distributed evenly among all replicas. The bidding mix scalability is bounded by the master's performance. The CPU on the master database is saturated and adding more slaves does not improve system performance. For both mixes, the model predicts throughputs that match the measured throughputs.

6.3 Sensitivity analysis

6.3.1 Load balancer and network delays

The model assumes that the combined load balancer delay and network delay are 1 ms. This is a reasonable approximation since the load balancer acts mainly as a router, introducing sub-millisecond delays. Our systems run in a LAN environment. A netperf test verifies that the network links support nearly the full 1 Gbit/s while sending 275 byte packets (writeset size). In contrast, the maximum bandwidth to/from the certifier in the most demanding run is less than 1 Mbit/s, orders of magnitude lower than the available bandwidth. Thus, no queuing or network congestion is observed.

6.3.2 Certifier

The service time at the certifier is dominated by the writes to the disk, which takes 6-8 ms on average. Several writesets are batched in one disk write that is executed in parallel at the leader certifier disk and the two backup certifier disks. The write is committed as soon as it completes at two disks. Waiting for these first two disk writes dominates the certification time.

When a request arrives at the certifier, it waits 12 ms on average (0.5 * 8 (waiting half service time) + 8 (service time)). We therefore model the certifier as a delay center of 12 ms. Prior work [Elnikety 2006] shows that the certifier

is lightweight and can handle more than 3500 tps, a rate much higher than TPC-W and RUBiS transaction rates. In this work, the certifier receives at most 150 requests/s (from TPC-W ordering mix on MM system), which is less than 5% of certifier capacity and therefore queuing delays do not develop, allowing modeling the certifier as a delay center.

6.3.3 Predictions with high abort rates

The TPC-W and RUBiS benchmarks have very low abort rates. We show in Section 6.2 that the models predict well under those conditions. To explore the prediction capability of the models under higher abort rates, we artificially introduce aborts in the TPC-W benchmark such that aborts increase with the system load.

Here we focus on the trends of abort probabilities rather than on accuracy, since the model assumes that abort rates are low. We consider only the MM design, as for SM the abort rates are directly measured on the master and not predicted by the model.

We introduce a replicated heap table (which is stored in main memory only). We instrument each update transaction to include an update operation to randomly selected row. We increase the probability that an update transaction aborts, by controlling the number of rows in the heap table.

We increase the abort probability A_1 to the following three values, A_1=0.24%, 0.53%, 0.90%. The corresponding measured abort rates at 16 replicas for the multi-master system are A_{16}=10%, 17%, 29%, respectively.

Figure 14 depicts the measured MM abort probability A_N for three starting values of A_1 and the predicted abort rates. The predictions closely match for both A_1=0.24%, 0.53%, but consistently under-estimate the probabilities for A_1=0.90%. The predicted trend is good but the difference is noticeable. The real abort rates appear to accelerate faster than the model predicts at large abort rates, likely a compounding effect not accurately accounted for in the model's approximations. The abort rates used in Figure 14 are purposefully high in an attempt to stress the models. These abort rates (e.g., 29% at 16 replicas) are well beyond what application designers and administrators would tolerate. The model captures the essential behavior of how the conflict window grows as the system scales, though, the predictions are better at lower abort rates.

Table 4. RUBiS parameters.

Mix	Read (*Pr*)	Write (*Pw*)	Clients per Replica (*C*)	Think Time (*Z*)
Browsing	100%	0%	50	1000 ms
Bidding	80%	20%	50	1000 ms

Table 5. Measured service demands (in ms) for RUBiS.

Mix	Resource	Read(*rc*)	Write(*wc*)	Writeset(*ws*)
Browsing	CPU	25.29	-	-
	Disk	11.36	-	-
Bidding	CPU	25.29	41.51	9.83
	Disk	11.36	48.61	35.28

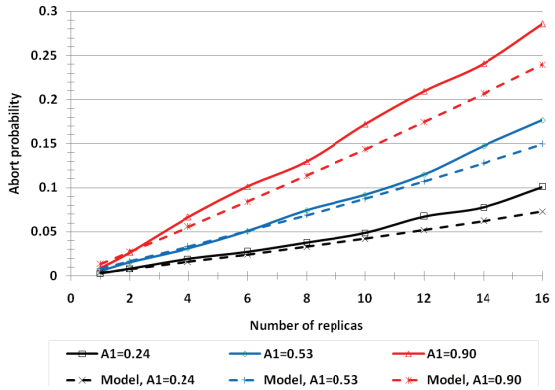

Figure 14. TPC-W shopping MM A_N abort probabilities.

7. Related work

Performance of replicated databases. In the classical work, "Dangers of Replication" [Gray 1996], the authors develop a model to capture the probability of waits and deadlocks in a replicated database under pessimistic concurrency control. They show that replicated databases are subject to a cubic increase in deadlocks. In contrast, our model captures the effects of conflicts under optimistic concurrency control as well as the resources needed to apply the propagated updates. For the class of workloads we study in this paper, the overhead of applying the propagated updates is reached first before the scalability limit due to conflicts.

Prior work [Jiménez-Peris 2001] develops a model to capture the overhead of propagating updates. However, the model does not capture aborts and is used to establish an upper bound on the scalability of replicated databases, rather than to predict their performance. In this paper, we use analytical models that capture both updates and aborts, and we validate their ability to predict performance experimentally.

Modeling the growth of abort rates [Gray 1996] and conflict windows [Elnikety 2005] appears in prior work. A few models [DeWitt 1992, Gray 1991, Gunther 1994] have been developed to capture the speedup of replicated databases. They are used to derive a model after building the system, rather than a priori.

Analytical models [Ferrari 2006, Kelly 2006, Urgaonkar 2005] have been developed to predict throughput response times for multi-tier distributed applications in data centers with the purpose of enabling dynamic provisioning. Our

models are complementary to these models. Combining those models enables modeling all tiers, including the database tier, for Web services.

Replication in SI databases. The implemented systems used in this paper represent some of the most recent works in replicated database literature [Cecchet 2008], matching recent research prototypes. The single-master system is similar to Ganymed [Plattner 2004], but improved to perform group commits at the master. The multi-master system is similar to replicated PostgreSQL prototypes [Elnikety 2006, Lin 2005, Wu 2005] as they propagate updates using writesets.

8. Conclusions

We present analytical models to predict the performance of two middleware-based database replication designs, multi-master and single-master. The analytical models capture the characteristics of these systems in terms of update propagation overheads and abort rates. We describe how to measure the system performance metrics on a standalone database and use these measurements as inputs to the analytical models. For experimental validation, we use prototypes of both systems and show that the models match well with the measured system performance. Performance predictions are within 15%.

Acknowledgments

We thank our shepherd, Fernando Pedone (University of Lugano), for his feedback and suggestions. We also thank Tim Harris, Alexandre Proutiere, Bozidar Radunovic, Eno Thereska, and Milan Vojnovic (Microsoft Research in Cambridge), and the anonymous reviewers for their constructive comments.

This research was partially supported by the Swiss National Science Foundation grant number 200021-121931 and by the Hasler Foundation grant number 2316.

References

[Amza 2002] Cristiana Amza, Emmanuel Cecchet, Anupam Chanda, Alan L. Cox, Sameh Elnikety, Romer Gil, Julie Marguerite, Karthick Rajamani, and Willy Zwaenepoel. Specification and implementation of dynamic Web site benchmarks. The 5th Annual IEEE Workshop on Workload Characterization, 2002.

[Berenson 1995] Hal Berenson, Phil Bernstein, Jim Gray, Jim Melton, Elizabeth O'Neil, and Patrick O'Neil. A critique of ANSI SQL isolation levels. In proceedings of the 1995 ACM SIGMOD international conference on management of data, 1995.

[Bernstein1987] Philip Bernstein, Vassos Hadzilacos, and Nathan Goodman. Concurrency control and recovery in database systems. Addison-Wesley, 1987.

[Cahill 2008] Michael J. Cahill, Uwe Roehm, and Alan Fekete. Serializable isolation for snapshot databases. In proceedings of the 2008 ACM SIGMOD international conference on management of data, 2008.

[Cecchet 2008] Emmanuel Cecchet, George Candea, and Aanastasia Ailamaki. Middleware-based database replication: the gaps between theory and practice. In proceedings of the 2008 ACM SIGMOD international conference on management of data, 2008.

[Daudjee 2006] Khuzaima Daudjee, and Kenneth Salem. Lazy database replication with snapshot isolation. In proceedings of the 32nd international conference on very large data bases, 2006.

[DeWitt 1992] David J. DeWitt, and Jim Gray. Parallel database systems: the future of high performance database systems. Communications of the ACM, 35(6):85-98, 1992.

[Elnikety 2005] Sameh Elnikety, Fernando Pedone, and Willy Zwaenepoel. Database replication using generalized snapshot isolation. In proceedings of the 24th IEEE symposium on reliable distributed systems (SRDS), 2005.

[Elnikety 2006] Sameh Elnikety, Steven Dropsho, and Fernando Pedone. Tashkent: uniting durability with transaction ordering for high-performance scalable database replication. In proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems (EuroSys), 2006.

[Elnikety 2007] Sameh Elnikety, Steven Dropsho, and Willy Zwaenepoel. Tashkent+: memory-aware load balancing and update filtering in database replication. In proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems (EuroSys), 2007.

[Fekete 1999] Alan Fekete. Serialisability and snapshot isolation. In proceedings of the Australian Database Conference 1999.

[Fekete 2005a] Alan Fekete, Dimitrios Liarokapis, Elizabeth O'Neil, Patrick O'Neil, and Dennis Shasha. Making snapshot isolation serializable. ACM Transactions on Database Systems (TODS), 30(2):492-528, 2005.

[Fekete 2005b] Alan Fekete. Allocating isolation levels to transactions. In proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems (PODS), 2005.

[Ferrari 2006] Giovanna Ferrari, Paul Ezhilchelvan, and Isi Mitrani. Performance modeling and evaluation of e-business systems. Annual Simulation Symposium 2006.

[Gray 1991] The Performance Handbook for database and transaction processing systems, Jim Gray editor. Morgan Kaufmann, 1991.

[Gray 1996] Jim Gray, Pat Helland, Patrick O'Neil, and Dennis Shasha. The dangers of replication and a solution. In proceedings of the 1996 ACM SIGMOD international conference on management of data, 1996.

[Gunther 1994] Neil J. Gunther. Issues facing commercial OLTP applications on MPP platforms, Compcon Spring '94, Digest of Papers, 1994.

[Heath 2006] Taliver Heath, Ana Paula Centeno, Pradeep George, Luiz Ramos, Yogesh Jaluria, and Ricardo Bianchini. Mercury and Freon: temperature emulation and management for server systems. In proceedings of the 12th international conference on architectural support for programming languages and operating systems (ASPLOS), 2006.

[Jain 1991] R. K. Jain. The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling. Wiley- Interscience, 1991.

[Jiménez-Peris 2001] Ricardo Jiménez-Peris, Marta Patiño-Martínez, Gustavo Alonso, and Bettina Kemme. How to select a replication protocol according to scalability, availability, and communication overhead. In proceedings of the 20th IEEE international conference on reliable distributed systems (SRDS), 2001.

[Kelly 2006] Terence Kelly, and Alex Zhang. Predicting performance in distributed enterprise applications. HP Tech. report #HPL-2006-76.

[Kemme 2000] Bettina Kemme, and Gustavo Alonso. Don't be lazy, be consistent: Postgres-R, a new way to implement database replication. In proceedings of the 26th international conference on very large data bases (VLDB), 2000.

[Kleinrock 1975] Leonard Kleinrock. Queueing systems, Volume 1: Theory. John Wiley and Sons, Inc., 1975.

[Lamport 1998] Leslie Lamport. The part-time parliament. ACM Transactions on Computer Systems (TOCS), 16(2):133-169, 1998.

[Lazowska 1984] Edward Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik. Quantitative system performance. Prentice Hall, 1984.

[Lin 2005] Yi Lin, Bettina Kemme, Marta Patiño-Martínez, and Ricardo Jiménez-Peris. Middleware based data replication providing snapshot isolation. In proceedings of the 2005 ACM SIGMOD international conference on management of data, 2005.

[Menasce 1998] Daniel Menasce, and Virgilio A. F. Almeida. Capacity planning for Web performance: metrics, models and methods. Prentice Hall, 1998.

[Patiño-Martínez 2005] Marta Patiño-Martínez, Ricardo Jiménez-Peris, Bettina Kemme, and Gustavo Alonso. Middle-R: Consistent database replication at the middleware level. ACM Transactions on Computer Systems (TOCS), 23(4):1-49, 2005.

[Plattner 2004] Christian Plattner, and Gustavo Alonso. Ganymed: Scalable replication for transactional Web applications. In proceedings of the 5th ACM/IFIP/USENIX international conference on middleware, 2004.

[Schroeder 2006] Bianca Schroeder, Adam Wierman, and Mor Harchol-Balter. Open versus closed: a cautionary tale. In proceedings of the 3rd conference on networked systems design & implementation (NSDI), 2006.

[Urgaonkar 2005a] Bhuvan Urgaonkar, Giovanni Pacifici, Prashant Shenoy, Mike Spreitzer, and Asser Tantawi. An analytical model for multi-tier internet services and its applications. In proceedings of the 2005 ACM SIGMETRICS international conference on measurement and modeling of computer systems, 2005.

[Urgaonkar 2005b] Bhuvan Urgaonkar, Prashant Shenoy, Abhishek Chandra, and Pawan Goyal. Dynamic provisioning of multi-tier Internet applications. In proceedings of the second international conference on automatic computing (ICAC), 2005.

[Wiesmann 2000] Matthias Wiesmann, Fernando Pedone, André Schiper, Bettina Kemme, and Gustavo Alonso. Understanding replication in databases and distributed systems. In proceedings of the 20th international conference on distributed computing systems (ICDCS), 2000.

[Wu 2005] Shuqing Wu, and Bettina Kemme. Postgres-R(SI): combining replica control with concurrency control based on snapshot isolation. In proceedings of the 21st international conference on data engineering (ICDE), 2005.