A Simulator for Exploring Replication and Locality of Access in a Distributed Database

Amitabh Shah*
Daniel Sturman

TR 90-1175
February 1991

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*Supported in part by a grant from Xerox Corporation.
A Simulator for Exploring Data Replication and Locality of Access in a Distributed Database

Amitabh Shah*  
Department of Computer Science  
Upson Hall, Cornell University  
Ithaca, New York 14853  
shah@cs.cornell.edu

Daniel Sturman  
Department of Computer Science  
Upson Hall, Cornell University  
Ithaca, New York 14853  
sturman@cs.cornell.edu

February 1991

Abstract

In fault-tolerant distributed databases, one method of increasing data availability is through increased data replication. However, with increased data replication comes an increase in the time to perform transactions. What is needed is a way in which this and other problems involved with configuring distributed databases can be explored. This paper describes a simulator which provides an accurate model of many different distributed environments when given a set of parameters describing failures, data replication, and site organization. Such a simulator is a useful tool for exploring the behavior of possible distributed database configurations as well as for verify theoretical results.

*Supported in part by a grant from Xerox Corporation.
1 Introduction

There is a constant struggle within the field of computer science to improve the throughput of computer systems. In the ’80s and ’90s, many have turned to distributed systems of computers as a solution to this problem. However, how to build high performance distributed system is a question with no clear answer and the subject of much current research. In [SG90], a stochastic model for studying the performance of a distributed database is proposed. The authors of [SG90] attempted to create a general, highly parametrized model that can be used to analyze a broad spectrum of database systems.

By implementing a simulator based on their mathematical results, we hope to confirm their theoretical results as well make further explorations into which configurations of the system are best suited in various conditions. In an experimental environment, information can sometimes be gained that would otherwise be missed. By varying the parameters in a simulator, one can see where the greatest improvements can be made and, if it can be understood which improvements in the components of a system lead to the greatest gains in the system’s throughput, then science can better direct its efforts. In addition, the simulator will be used to validate the theoretical results of [SG90]. In particular, since the Shah and Ghosal assume certain simplification in order to keep the mathematical model tractable, the simulator will show whether or not these assumptions were valid.

2 The System Model

The model used for the system was chosen for its general adaptability to a variety of distributed database problems. [SG90] The system is divided into several groups, each containing some number of nodes, connected by links. The links can be faulty and do not guarantee delivery, i.e. there is a probability that any message may be lost. Nodes in the system are also subject to failures. Node failures are assumed to be simple crash failures which result in the loss of the state at that node.

Each node contains a set of data objects divided into local and global data. All nodes within one group share a set of local data. Each node contains a copy of every data object in its local data and each set of local data is mutually exclusive from all other sets. There is also a set of data common to all groups. A copy of each object in this global data is maintained at each site. Data which is local to one group is referred to as being non-local to other groups. By varying the ratio
of local to global data, the relationship between availability verses replication can be studied.

Nodes communicate by exchanging messages. Each node receives messages on one of two prioritized message queues. The class-1 messages, which are comprised of the voting requests and result notifications, enter the higher priority queue \( Q_1 \); those messages requiring relatively little processing by the receiving node. Class-2 messages are full transactions received from the outside world or from another group and would require fairly extensive processing in a real-world system. Class-1 messages are always processed before any class-2 messages. However, a newly received class-1 message will not preempt a class-2 message which is already being processed. Transactions which have been received, but are currently not being processed, are stored on one of two lists: the pending list or the deferred list. The pending list consists of all transactions which have been voted on by the host node, but have yet to be resolved by the system. The deferred node is comprised of those transactions which the node was unable to vote on due to conflicts with other transactions currently on the pending list. Voting on these jobs is performed once the blocking transactions are resolved.

Every node also uses a local clock and a timer. The local clock is incremented one tick every time an event is received from the outside world or when a message is received from another node with a time stamp greater than the local clock. The timer is based on a real-world clock, varying slightly based on a normal distribution.

3 Concurrency Control

3.1 Voting Rule

The concurrency control protocol used in the simulation is a combination of Thomas' consensus algorithm [Tho79] and the standard two-phase commit protocol. [Gra78] When a node receives a transaction from the outside world (referred to as an arrival), it increments the local clock and gives this transaction a time stamp. The initiator (receiving node) then votes on the transaction using the following voting rule:

1. **REJECT** if \( T \) is late, i.e. if there exists a datum being operated on by \( T \) with time stamp \( p \), such that \( p > ts(T) \).

2. **OK** if \( T \) is not late and there are no transactions on the pending list which have been voted **OK** on and reference a data object that \( T \) references.
3. **PASS** if T is not late, but there is a transaction on the pending list P which conflicts with T and \( ts(P) > ts(T) \).

4. **DEFER** if T is not late, but there is a transaction on the pending list P which conflicts with T and \( ts(T) > ts(P) \).

If the initiator votes **REJECT**, the transaction is simply discarded and no further action is taken (except notifying the user of the action). If the vote is **OK**, the transaction is marked as being **OK**, added to the pending list, and broadcast to the cohort nodes, provided the transaction contains only local data. A PASS vote is handled similarly except that the vote on the transaction is noted as being **PASS**. If the transaction is deferred, the transaction(s) which must first be resolved are noted and the transaction is put on the deferred list.

Upon receiving a transaction from the initiator, the node votes on the transaction using the above rules. If the vote is not **DEFER**, the vote is sent back to the initiator where it is stored until the initiator acts on the transaction. A deferred transaction is put on the deferred list until all conflicts have been resolved.

### 3.2 Transaction Resolution

Because of the system's ability to exhibit both link and node failures, it is possible that the initiator will never receive enough votes to resolve the transaction. To alleviate this problem, a timer is set for time \( \tau \) for every transaction that is placed on the pending list. A node will wait at most this time \( \tau \) before resolving the transaction. By using this timer, the protocol is able to guarantee liveness for transaction resolutions. However, in the interest of efficiency, the node need not wait for this timer. If, at some point, the initiator has enough votes to resolve the transaction, it may do so.

When either the timer expires or a node has received enough votes, the transaction is resolved. If more than half the possible votes are **OK**, then the transaction is accepted. If more than half the possible votes are **REJECT** or **PASS**, the transaction is rejected. If neither of these conditions hold, then the node cannot resolve the transaction, so a **ABORT** is issued. In any case, the transaction is removed from the pending queue and the initiator broadcasts the results to its cohorts.

If the transaction contains global or non-local data, the handling of a **OK** or **PASS** vote is slightly more complex. Besides broadcasting the message to its cohorts, for each group whose local data will be operated on, the initiator sends the transaction to a randomly selected node in that group.
of global data, every group receives the transaction. These nodes which received the transaction are referred to as forwarders. These forwarders act on the transactions as if they were an initiator. However, rather than broadcasting the resolution to its cohorts, each forwarder reports its results to the initiator.

The report consists of the number of OK, and REJECT or PASS votes received for global transactions and the status of any local data in the transaction. If a forwarder resolves that, based on the local data for its group, the transaction should be rejected or aborted, the initiator will immediately reject or abort the entire transaction. If the forwarder reports the transaction is OK with respect to its local data, the initiator tallies the global votes (if any). Once the initiator resolves the transaction, its notifies the original forwarders who then pass this information on to their cohorts.

4 Simulator Implementation

4.1 Control Program

The heart of the simulator is the control program. This program keeps track of the global time in the system. This global time represents the real-world time in the simulation and all delays, as well as measurements, are made in terms of this global time. Global time is used solely to give consistency and validity to the results measured. Therefore, the nodes do not need, nor do they have, access to this global time, although their timers follow the global time within a tolerance.

At each tick, several operations are performed by the control program on each node. To facilitate the control program in these operations, each node record contains a busy timer, a failure timer, a time of next arrival, and a time of next failure. All of these values are kept in global ticks since they correspond to outside events and are not available to the system being simulated. Each of these time-based events, as well as the processing times, are given values based on an exponential distribution with a mean value supplied by the user. These parameters are entered at the start of the simulation and are referred to as:

- $\lambda_2$ is the arrival rate of new transactions (class-2).
- $\mu_1$ is the processing rate of class-1 messages.
- $\mu_2$ is the processing rate of class-2 messages.
- $\gamma$ is the rate of failures.
- $\beta$ is the crash recovery rate.
• $\tau$ is the duration of the timer for each local transaction.
• $\tau_n$ is the duration of the timer for each non-local transaction.

The following parameters are also specified by the user and are used in a normal distribution:
• $RC$ is the average channel delay for intragroup communication.
• $ARC$ is the average channel delay for intergroup communication.

In addition to the user specified parameters, $\lambda_1$, the arrival rate of class-1 messages is determined by the above parameters.

### 4.2 Arrivals

The first item checked at each tick is the time of the next arrival of a transaction from the outside world. If an arrival should occur, the arrival is generated, put in $Q_2$ of the receiving node, and the next arrival time is generated and stored. The composition of each transaction is determined using a set of six user-specified parameters having normal and uniform distributions:

• $LO$ is the probability of local data in a transaction.
• $GL$ is the probability of global data in a transaction.
• $NL$ is the probability of non-local data in a transaction.
• $NLO$ is the max number of local data objects/transaction.
• $NGL$ is the max number of global data objects/transaction.
• $NNL$ is the max number of non-local data objects/transaction.

The probabilities are the chance of that type of data being contained in any given transaction. At least one local data item will be present in any transaction to avoid null transactions. The other three variables are parameters to a uniform distribution ranging from zero to the value of the parameter. Although almost any scheme could be used to generate the transactions; this one was chosen for its simplicity and versatility.

### 4.3 Failures

After handling any arrivals, the next failure time of each node is checked. If a failure should occur, the failure timer is set and the next failure time is reset. (Note that, since failures occur at the beginning of a tick, all broadcasts are effectively atomic.) When a node fails, all of the messages in its message queues
are removed from the queues and recorded as being lost. For each tick following
the failure, the failure timer of the affected node is decremented and the node is
not allowed to perform any other action. This time represents the time required
to bring the node back up.

When the timer runs out, all of the messages the node has received since
it failed are tallied as being ignored, and each data item stored in the node
receives a timestamp equal to the maximum timestamp in its group. Rather
than sending multiple messages between cohorts to bring the node back up, the
recovery time is instead chosen to realistically represent the amount of time that
would have been required to perform these operations.

4.4 Messages and Queues

Provided that the node has not failed, it will proceed to process one of the
messages or transactions previously received. In the simulator, processing time
is modeled by a "busy" timer much like the timer used to monitor recovery time
after the node fails. The busy timer is set after each response to a message. An
exponential distribution is used with parameter $\mu_1$ or $\mu_2$ for class-1 or class-2
messages, respectively. This value represents the amount of processing time the
node spends on the response. The timer is decremented once every global tick
and, while the timer is active, the node cannot service any of its queues.

If the node is not currently busy, $Q_1$ is checked to see if there any pending
messages. If so, the node will process one and set the busy timer accordingly. If
not, the node will then check $Q_2$ and process one waiting transaction. If there
are no messages on either queue, the node will remain idle.

As specified earlier, the higher priority messages of $Q_1$ will always be serviced
before the messages of $Q_2$. The messages $Q_2$ receives are transactions arriving
from the outside world and transactions from the initiator in another group.
After removing a class-2 transaction from $Q_2$, an initiator or forwarder node
first votes on the transaction rule itself. If the initiator/forwarder votes $OK$ or
$PASS$ to the transaction, it broadcasts the transaction as a vote request to to its
cohorts and, in the case of access to non-local or global data, to any appropriate
groups. The node then places the message on the pending queue and awaits
responses to the broadcast.

The cohorts receive the vote request on $Q_1$. They process this message by
voting on the transaction and sending a vote back to the initiator/forwarder.
This vote is either an $OK$ or a $REJECT$. (For purposes of this vote, $REJECT$
and $PASS$ have the same effect.) If the vote was a $REJECT$, the rejecting node
specifies whether the vote was caused by global or local data. Once a node
receives a majority of OK's or REJECT's, it can act on the transaction. At
the very latest, a transaction will be acted on when its timer expires. If there
are not enough votes to make a decision, the transaction is considered aborted,
otherwise the message is either accepted or rejected.

In the case of the initiator, once a decision has been reached, it sends an up-
date message to all of its cohorts informing them of the result. In the simulator,
results take longer to reach their destination, but are never lost. The additional
time represents a scheme of multiple messages and responses. After broadcasting
the result, the initiator updates itself. If a transaction is accepted, the update
consists of time stamping all of the data operated on by the transaction. In
the case of a reject or abort, the transaction is simply discarded. Aborts are
tallied for statistical purposes. A forwarder does not broadcast the result on its
own, but instead sends the result to the initiator who will, in turn, inform the
forwarder of the final result. This result is then broadcast to the cohorts of the
node.

References

[GRA78] James Gray. Notes on database operating systems. In R. Bayer,
R. M. Graham, and G. Seegmuller, editors, Operating Systems: An
Advanced Course, volume 66 of Lecture Notes on Computer Science.

control for multiple copy database. ACM Transactions on Database

of Distributed Databases with Site and Link Failures. Technical
Report TR 90-1072, Department of Computer Science, Cornell
University, January 1990.