1988

SETH: A Quorum-Based Replicated Database System for Experimentation with Failures

Bharat Bhargava
Purdue University, bb@cs.purdue.edu

Abdelsalam Helal

Jagannathan Srinivasan

Report Number:
88-784

SETH: A QUORUM-BASED REPLICATED DATABASE SYSTEM FOR EXPERIMENTATION WITH FAILURES

Bharat Bhargava
Abdelsalam Helal
Jagannathan Srinivasan

CSD TR-784
June 1988

SETH : A Quorum-Based Replicated Database System for Experimentation with Failures*

Bharat Bhargava
Abdelsalam Helal
Jagannathan Srinivasan

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

ABSTRACT

To increase our understanding of the behavior and performance of replica control algorithms that deal with network partitions, we have implemented a quorum-based database system called Seth. Seth is designed to be used as a transaction processing system, as well as a flexible experimentation tool. Seth experimentation domain currently includes *Failures* (site/link failure/repair rates), *Transactions* (arrival rate, size, type, read/write ratio), *Quorums* (weights, and thresholds), *Communications* (communication protocols, network type, topology, number of sites), and *Transaction Protocols* (quorum-based RCP's, distributed commitment, and concurrency control). In this paper, we discuss the design and implementation of Seth and present experiments on the Quorum Consensus protocol. We show why a quorum-based transaction processing system is, in general, expensive in terms of the number of messages per transaction. We show that an implementation of such a system on a conventional architectures like a Vax, or a Sequent machine, can not run beyond a relatively small work load. For example, 4 transactions per second is the maximum that can be achieved in Seth in an 8-site configuration that uses UDP sockets to communicate.

*This research is supported by NASA, and AIRMICS under grant number NAG-1-676, UNISYS Corp., and a Purdue University Fellowship.

1. INTRODUCTION

In distributed database systems, replication increases data availability and speeds up query processing. Replication, however, complicates the way updates are processed, specially during site failures and network partitions. A replication control protocol (RCP) ensures that concurrent execution of user transactions over a replicated database is equivalent to some serial execution on a one-copy database. Although many replica control protocols have been proposed in the literature, their evaluation and implementation experience is scarce.

We have selected to experiment with the quorum based methods since they are general enough to deal with the network partitions, site failures, and can also be used for concurrency control during transaction processing. Dealing with failures of network and sites in a distributed system is an important area of research and this effort is directed towards learning more about the performance and implementation of existing algorithms. We decided to implement a separate system to conduct our studies rather than using the Raid system [Bhargava88a] to clearly focus on the limitations and bottlenecks as far as the availability and the message traffic are concerned in quorum based mechanisms. Raid system is a complete database system with facilities for database queries, user interfaces, I/O management in addition to communication and transaction processing facilities. The special characteristics of individual components are not observable in a large system.

Seth is a quorum-based replicated database system that runs on the Sequent machine. Its design emphasizes facilitating experimentation with various patterns of site and communication link failures. Seth virtual *sites* are UNIX processes that process concurrent transactions through three layered protocols which are: a *Replica Control Protocol (RCP), an Atomic Commitment Protocol (ACP), and a Concurrency Control Protocol (CCP).* Database objects are replicated as copies in the sites' virtual memory with a varying degree of replication. A Seth *Configuration* defines the physical connectivity among a certain number of sites. Two axioms constrained the design of Seth:

(1) Seth assumes no particular communication network type. In other words, Seth is designed to be *Network Independent.* This required the implementation of a *Network Interface* called *the Prototype Manager.* The goal here is two-fold. First, we want to separate the design of most of the prototype from the underlying communication network model. For example, Seth sites which run in a LAN (like the Ethernet) can also run in an LHN (like the Arpanet), or a token Ring (like the Crystal network). Second, experimentation with different types of networks is one of our objectives. A major advantage of the prototype manager is that it can interface to a simulator of various network types.

(2) Seth can accommodate for the inclusion of a wide variety of RCP's for experimentation purposes. This is done by providing *Quorum mechanism* which is general enough to implement many other RCP algorithms [El Abbadi86], [Bernstein86], [Bernstein84], [Bhargava87a], [Chan86], [Eager81], [Eager83], and [Gifford79].

In this paper, we present an implementation of Seth that uses the quorum consensus protocol [Gifford 79]. We show how to setup Seth for experimentation and present two experiments. The first shows the behavior of the quorum consensus in terms of transaction commit rate, concurrency control abort rate, Two phase commit abort rate, and quorum consensus abort rate. To study the quorum consensus communication overhead, the experiment measures the average number of messages per transaction. The second experiment studies the effect of communication overhead (traffic) on the system capacity. The experiment measures the traffic against transaction arrival rates, and shows how the system saturates for a relatively small transaction arrival rate.

Implementation of Seth is discussed in section 2. Two experiments are detailed in section 3. Finally the system status and future work is included in section 4.

2. IMPLEMENTATION OF SETH

Seth is written in C programming language and runs on top of Berkeley 4.3 Unix on a 6-processors Sequent Symmetry machine. Seth users are provided with a transaction facility where *read*, and *write* operations are supported in a *flat* (unnested) transaction. Users attach to a Seth site through a user interface. A Seth site can receive transactions from either user interfaces or from a workload generator. To concurrently process several user transactions, Seth sites use a special scheduler that suspends/resumes transactions according to the send/receive patterns incurred by the set of protocols that process the transactions (e.g, Two-Phase Commit). A Seth site provides *location transparency* (where users do not know where the different copies of an object are stored), *replication transparency* (where users deal only with database objects but not copies of objects), *concurrency transparency* (where user transactions are serialized), and *failure transparency* (where user transactions are either atomically executed or aborted) [Traiger 82].

An instance of Seth consists of a number of sites and a *Prototype Manager*. In addition, we can use a workload generator as shown in Figure 1. The Prototype Manager (or simply, the manager) maintains information about the system's configuration. We assume that the manager knows which site is up and which is down, or whether a network partition has occurred. Seth sites communicate only through the manager, and hence need not worry about view information, or view synchronization. Also, sites need not know much about the underlying network type and its characteristics. In essence, the manager acts as a *message forwarder*. It can simulate *timeouts* by sending negative acknowledgements (*NACK*) to compensate replies which it knows will

never arrive. Also, the manager simulates communication delays due to multiple hops. In the following subsections, implementation of the different modules of Seth is discussed in detail.

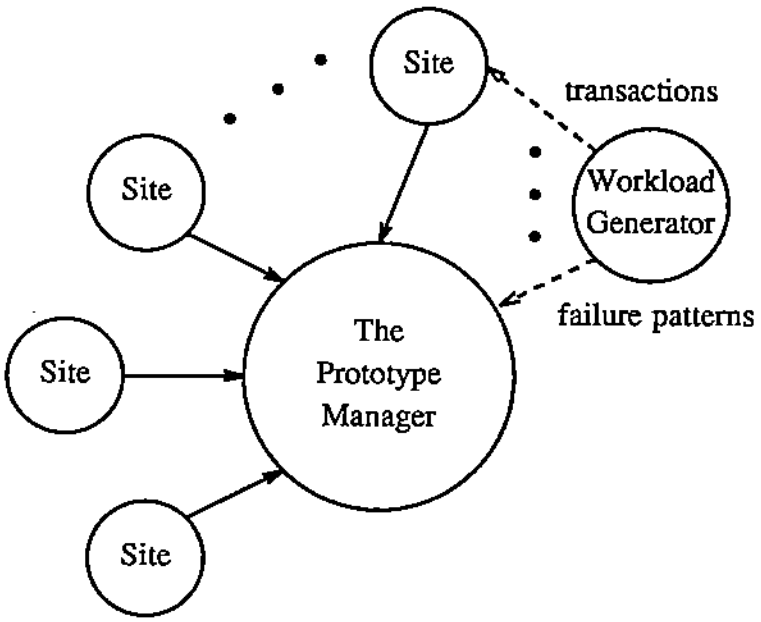

Figure 1: A Seth Instance

2.1. Communication in Seth

Seth sites and the prototype manager are implemented as UNIX processes that communicate through *UDP Sockets*. When a Seth site is initiated comes up, it picks a UDP port on the local host machine and writes it into a well-known file. Other sites can then read this file to discover the port to which to send messages to communicate with this site. Seth communication subsystem is cannibalized from the communication library of Raid [Bhargava88a]. In addition, it includes a *Stub*, and a *Traffic Monitor*. The domain of Seth sites is currently limited to one physical machine for debugging purposes. This limitation, though made debugging possible, reduced the virtual processing power per site since all sites shared the same machine processor(s). As discussed and observed in section 4, one problem that arise due to this limitation is loss of packets when Seth gets highly loaded.

2.2. Seth Sites

Each Seth site maintains a copy of the database in its virtual memory (Figure 2). Each local copy of a database object consists of a version number, a value, a weight, and the read and write thresholds. A site maintains a static transaction table which contains the transaction state and other run-time information (Figure. 3). Similarly, subtransactions are maintained by a static subtransaction table. Associated with each transaction is a unique transaction id, transaction state and information, and an *execution point* (a pointer to the current operation being processed). Each operation has a quorum structure that it builds incrementally. A *Permutation Vector* is part of each operation and it is used by the sites to select the sites to build the quorum. Permutation vectors are computed by the sites. Such a computation involves preferring the neighbor sites, excluding down sites and learning of newly failed/repaired sites. Intuitively, permutation vectors help in the efficient establishment of quorums and in reducing network traffic.

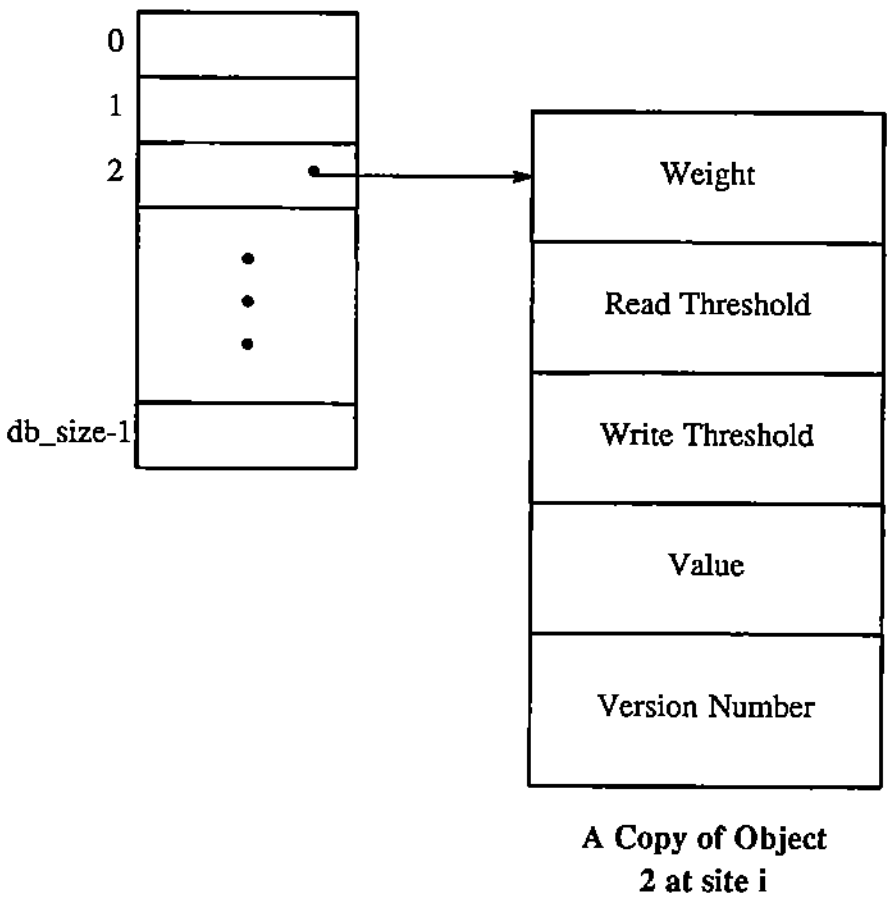

Figure 2. In–Memory Database Copy at Site i

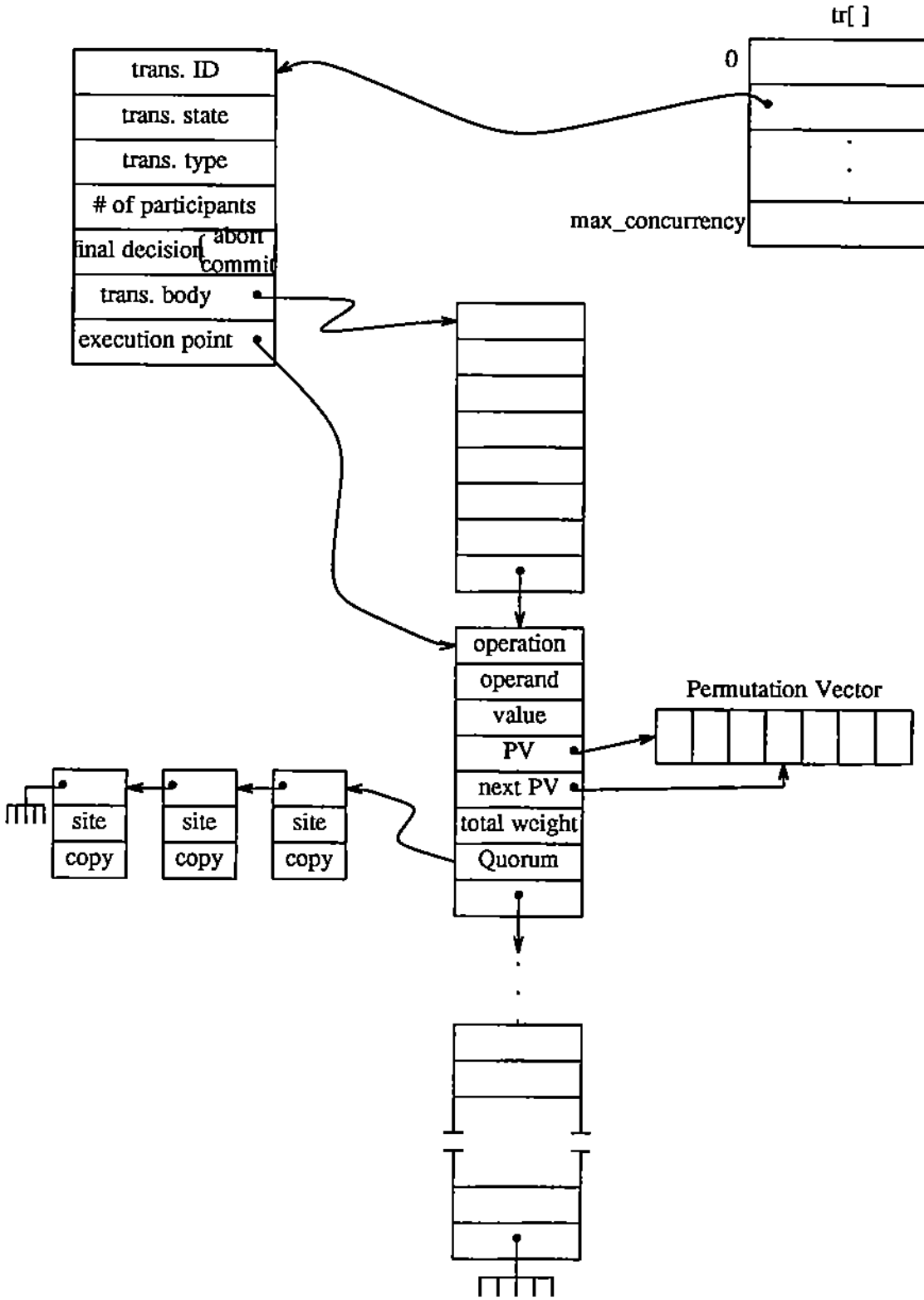

Figure 3. The Transaction Structure

Transactions are executed concurrently using a special *scheduler* that suspends/resumes transactions according to the send/receive patterns of the protocols executing these transactions. Quorum Consensus (QC), Two-Phase Commit (2PC), and Two-Phase Locking (2PL) are the protocols which coordinate the concurrent execution of transactions. Figure. 4 depicts a state transition diagram which describes the interactions between the QC, 2PC, and the 2PL protocols. This state transition diagram which is explained below, is the kernel of the transaction *scheduler*.

2.2.1 QC, 2PC, 2PL Interactions

When a new transaction is submitted to a site, it is added to the transaction table and gets its permutation vectors computed. It immediately invokes the RCP. Since the current version of Seth uses quorum consensus (QC), QC starts building a quorum (read or write) for the first operation of this transaction by sending a request (a subtransaction that inherits its parent transaction's id) to a set of sites for their local copies. In the current version of Seth, sending such quorum requests is done serially. At the remote sites, copies are read (returning their current value) or pre-written (returning their current version number) through the concurrency control protocol (CCP). When a quorum is built for an operation, the next operation is considered. When all operations of a transaction are handled by the RCP, the site initiates a two-phase commit session. When commitment terminates, the transaction is removed from the transaction table. When a site receives a subtransaction from a remote site, it processes its operations through the CCP. If the concurrency control decides to abort the transaction, an ABORT message is sent to the site the subtransaction was issued from. Otherwise, the subtransaction succeeds and is added to the subtransaction table, where it remains till its parent transaction either aborts or commits. In the course of its execution, a transaction can be aborted by any of the three protocols: QC may abort if it can not build a quorum (not enough sites are available). 2PC may abort if, during commitment, one or more participants fail or become unreachable. Finally, CCP may abort due to conflicting operations.

2.2.2 The Transaction Scheduler

The transaction scheduler selects the current transaction for execution based on the semantics of the message type and subtype and the state transition diagram shown in Figure 4. Without a transaction scheduler, transactions may confuse (receive) each others messages and the concurrency may lead to incorrect execution and fatal system errors. The ideas of implementing the scheduler is borrowed from Raid [Bhargava88a] and is described as follow. At each site, the scheduler is the only module which can receive messages addressed to (sub)transactions at that site. When a message is

received, the scheduler can always decode the transaction id(*tid*) out of the message. Using the state transition diagram and the semantics of message types and subtypes, the scheduler computes the state in which the (sub)transaction must have been suspended. Having computed this state, (sub)transaction *tid* is validated against this computed state. If valid, the (sub)transaction resumed and is executed by some or all of the QC, 2PC, and 2PL protocols until the transaction is either completed or one of these protocols needs to send out a REQUEST message. Before the send actually takes place, the scheduler suspends the (sub)transaction after changing its state.

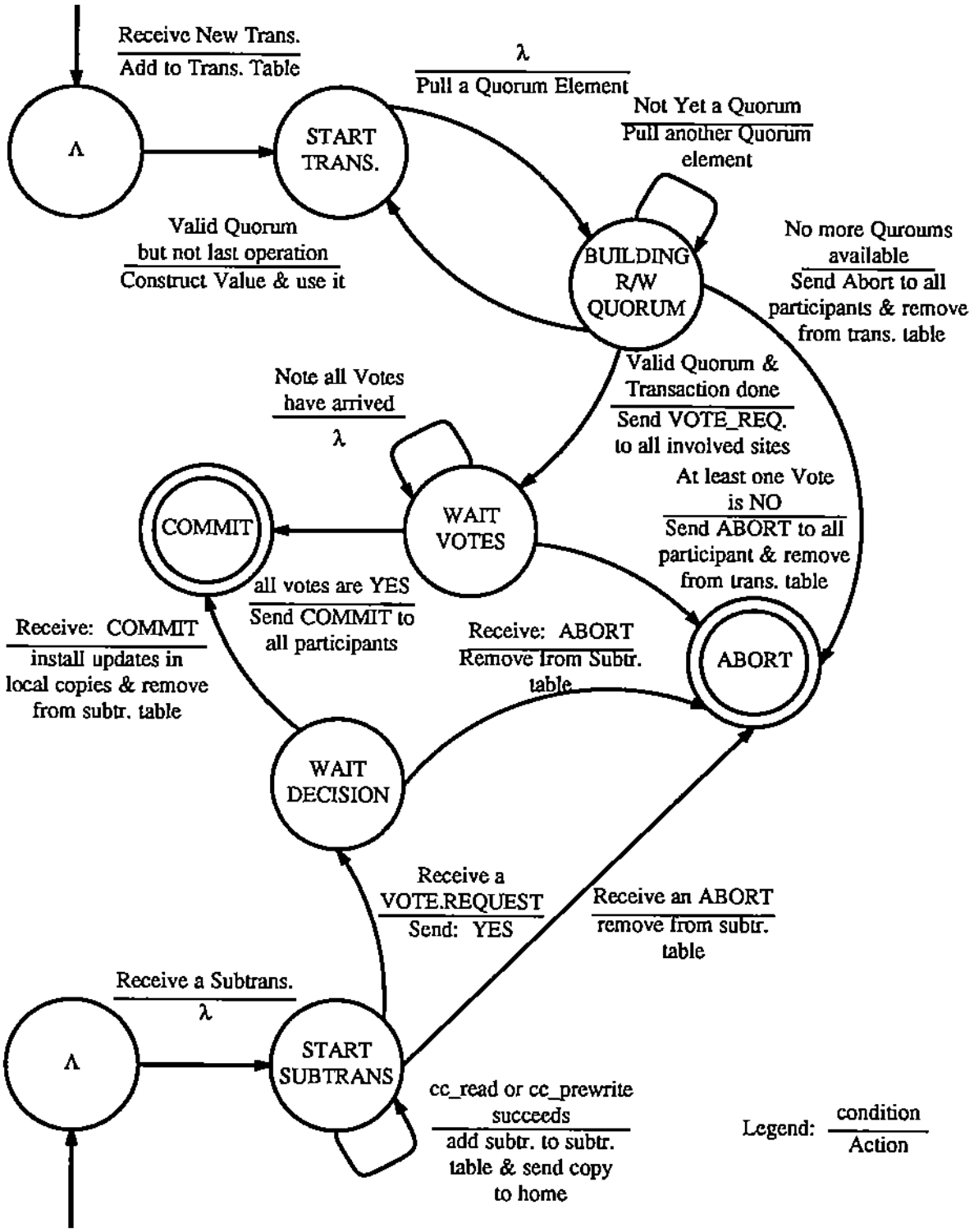

Figure 4. State Transitions Diagram

2.3. The Prototype Manager

Seth prototype manager is implemented as Unix processes. It functions as a message forwarder. All sites communicate through the manager. It maintains the configuration and the status of the network in the form of an adjacency matrix, and an *up-down status* vector. In addition, it computes and maintains a closure matrix (all pairs shortest paths) which contains the least number of hops needed to go from one site to another. When the manager receives a message from a site, it consults the closure matrix to determine the number of hops the message needs to travel. It then relays the message to the destination site after delaying it by the time that is proportional to the number of hops. The manager may receive a message which has to be forwarded to a site which is unreachable. In such a case it either drops the message or sends a NACK reply.

The implementation of hopping delays in the current version of Seth is as follows. Multiple hops through a path of sites is simulated by repeated looping at the manager. For example, if a message has to travel k-hops to reach its destination, the manager will actually send it k-1 times to itself before finally forwarding it to its destination. Unfortunately, this implementation of hopping delay incurs extra messages that can contribute to overload the manager.

In Seth, five types of messages are used: REQUEST, REPLY, ABORT, CONTROL and MONITOR. REQUEST, and REPLY are used more frequently. For example, to build a read quorum, a site may send a read REQUEST message to another site. In response, that site may send its local copy in a REPLY message. MONITOR messages are used to log useful information that is produced at different sites for measurement and debugging purposes. CONTROL messages are used for starting up, shutting down, and simulating failures and repairs. As transactions can get aborted by the *CCP*, *2PC*, or *RCP*. Abort messages have been given a special type (ABORT).

Failures are generated by the workload generator. The workload generator sends a CONTROL message to fail(restart) a site or a link. In response, the manager updates the adjacency matrix (to reflect the change in the network configuration), and recomputes the closure matrix. Thus sites may dynamically become unreachable (because of partition) or may fail. The manager forwards the messages in the following way.

Case 1: The manager receives a REQUEST message (read, write, or vote_request) while the destination site is unreachable. The manager simulates timeout mechanism for the requesting site by sending a faked REPLY message with its NACK field set. On the other hand, the requesting site, after receiving the *NACK* reply acts as if it has timed out on its request.

Case 2: The manager receives a REPLY message to be delivered to a failed site. In such a case, the manager simply drops the message.

Case 3: The manager receives a REPLY message to be delivered to an unreachable site. Since the destination site will be waiting for the reply, the manager builds a NACK reply and sends it to the destination site.

Since every message goes through the manager, overloading the system may bottleneck the manager. This leads to a situation where the system may start losing packets. Therefore, the prototype manager has been carefully designed to do minimal amount of processing on receiving messages. The ease and simplicity provided by the manager, to experiment with failures, has been a major motivation for us to pursue this design.

3. EXPERIMENTS

Two types of experiments can be conducted in Seth. The first are of the type *"real" time* in the sense that there is none or very little simulation. The second type, of experiments are simulation-based experiments where a simulation clock is used. For example, measuring throughput vs. transaction arrival rate is a *type I* experiment. On the other hand, since link/site failure/repair rates are "per month" figures, measuring availability vs.these rates is a *Type II* experiment. In this paper we focus on experiments of *Type I* since the simulation clock is not yet implemented.

In this section we study the behavior of the quorum consensus protocol and the impact of its communication overhead on the system capacity. We consider availability and message traffic to be two important measures in quorum-based systems. In what follows, we describe these measures, and explain the experimental setup. Then, we present two experiments and state the conclusions that we have derived.

3.1. Performance Measures (Availability and Message Traffic)

Availability is an important parameter that can be measured in a replicated database system. One measure of availability is defined in [Jajodia 87]. It is the limit of the probability that a transaction arriving at an arbitrary site at time t will succeed as t goes to infinity. We define the *degree of availability* to be the ratio of this sum of throughput at each partition to the throughput of the same system without the partitioning. Throughput is a parameter that we measure in our experiments.

Message traffic is another important measure for the network protocol's overhead. Measuring traffic is important in quorum-based systems since it can saturate the message processing system. We call this situation, *over running*. Monitoring the message traffic detects such situations and suggests the maximum load tolerable by a Seth instance.

3.2. Seth Experimental Setup

To initiate an experiment in Seth, one has to specify three sets of parameters. These are: network configuration, quorum parameters, and the workload. The current version of Seth assumes a long haul network. Specifying a network configuration involves specifying the number of sites and the sites' adjacency matrix. This information is written into a well-known file called *configuration* which is accessible by the prototype manager.

Quorum data are specified using the *Interactive Quorum-Parameters Design Module*. This automates and facilitates assignment of weights to copies as well as specifying the read and write thresholds for objects. Group operations are supported for cases when a group of copies (or objects) are assigned the same parameters. This is useful in the case of a large database. As its output, this module generates a quorum parameters file for each Seth site.

The workload parameters for our experiments are the transaction arrival rate, (*tar*), maximum transaction size, (*tms*), transaction's read/write ratio(or mix), *tmix*. Other parameters not used by the current experiments of Seth are site failure rate, (*sfr*), site repair rate, (*srr*), link failure rate, (fr), link repair rate, (*lrr*). The transaction arrival process is assumed to be Poisson. Transaction size is uniform over [1..*tms*]. Transaction read/write ratio is specified by the probability of reads. Failure rates are specified as either constant or Poisson. An *Interactive Workload Specification Module* is used to input these parameters. It generates a parameters file which is consulted by the workload generator when the latter is initiated. The workload generator maintains an *event list* in which it schedules events like *Arrival of New Transaction*, *Link(Site) Failure*, *Link(Site) repair*, and *System Shutdown*. The workload generator always *alarms* on the nearest event, pauses till it gets signaled by the timer, generates the event, and then alarms on the next nearest event, and so on. In the current version of Seth, another specification file is used by the workload generator. This file contains information like *the total experiment time* and the *sleep time* based on the interval between the arrival of the last transaction and the shutting down of the system. In addition, this file contains information that determines the initiation of certain types of failure.

3.3. Experiment I: Quorum Consensus Behavior

This experiment was conducted to observe the behavior of the quorum consensus protocol. The transaction commit rate, and abort rate due to the CC, the 2PC, and the RCP were measured against different transaction arrival rates.

3.3.1 Setup of the Experiment I

Seth was run with Configuration 1 (Figure 5-a), first with sites 2 and 3 connected, then with same sites disconnected. The same experiment was repeated with Configuration 2 (Figure 6-a), first with sites 1 and 2 connected, then with same sites disconnected. The transaction arrival rates were varied up to 2.5 transaction/second. Data was collected and averaged over 6 runs of 50 sec and duration each. All copies of an object were assigned a weight of 5, the read threshold was 20, and the write threshold was 25. Maximum transaction size was set to 5 and Read/Write ratio of 0.5 were used. The database size was chosen to be 200 objects.

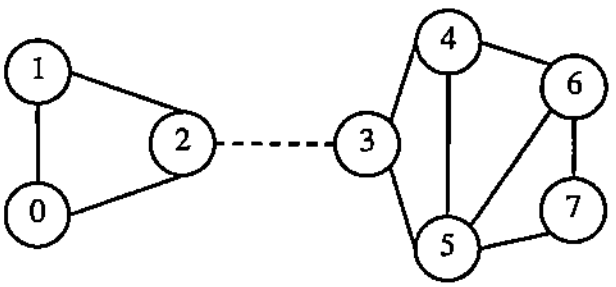

Figure 5-a. Configuration 1.

3.3.2 Analysis and Observations of Data Measurements in Experiment I

As the transaction arrival rate is increased, we observe a linear increase in the commit and the concurrency control (CC) abort rates for Configuration 1 with sites 2 and 3 connected (Figure 5-b). However, when transaction arrival rates are increased beyond 1 transaction/second, commit rate starts to decline. This is due to the increase in CC aborts, and the increase in message traffic. The latter causes the loss of packets as will be discussed in experiment II.

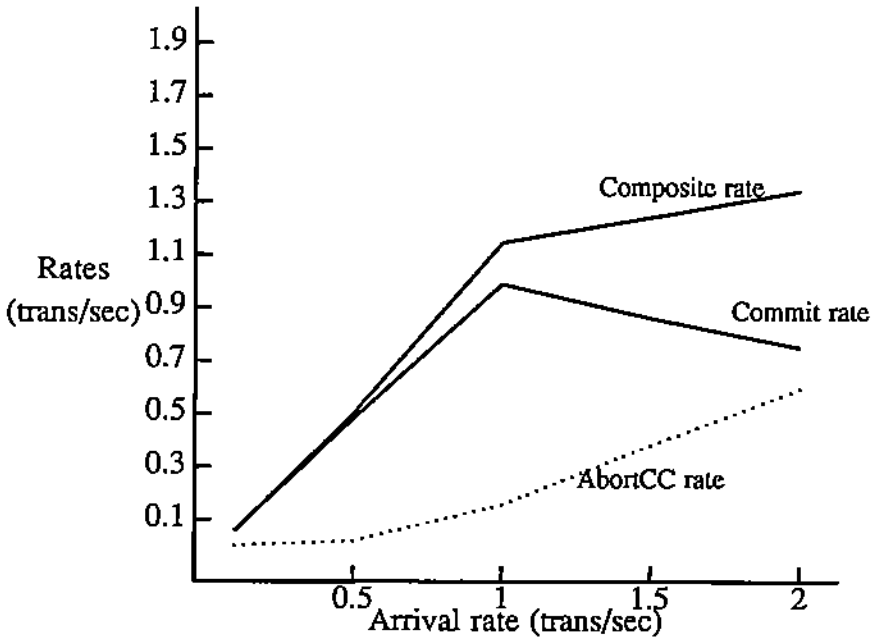

Figure 5-b Commit and Abort rates in Configuration 1
with sites 2 and 3 connected

When the same experiment is repeated with sites 2 and 3 disconnected, we observe a similar behavior (Figure 5-c). The major difference is non-zero aborts due to the RCP caused by network partitioning. As a result, the commit rate is smaller than that in the non-partitioned case. The CC abort rate is smaller compared to that of the non-partitioned case. The aborts due to the CC are reduced as the RCP will abort all transactions in the smaller partition before they reach the CC.

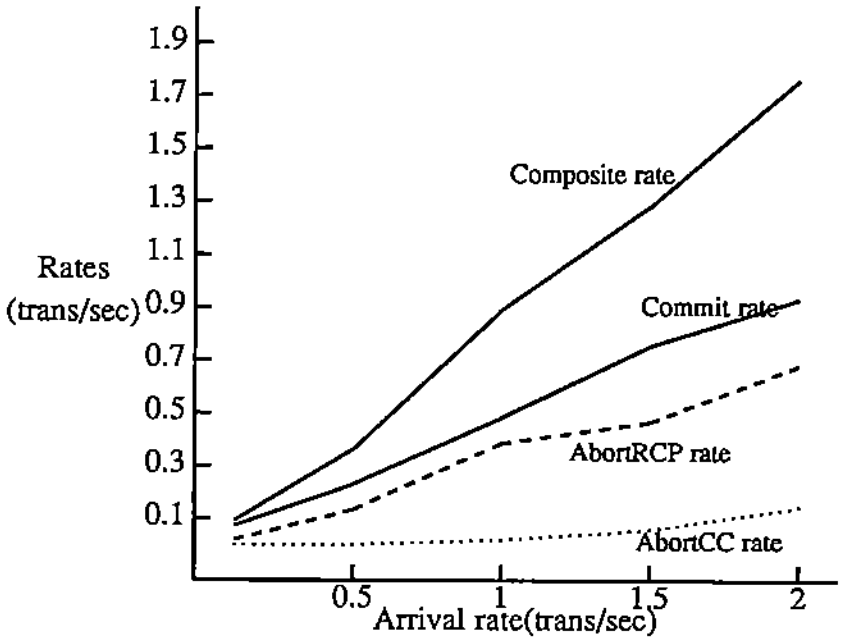

Figure 5-c Commit and Abort rates in Configuration 1
with sites 2 and 3 disconnected

The above experiment was repeated for Configuration 2. The objective was to see the effect of the ratio of the sizes of the two partitions. Figure 6-b, and Figure 6-c show the transaction arrival rate vs. commit rates and abort rates for the non-partitioned and partioned cases. These results are similar to those obtained for Configuration 1. A comparison is shown in (Figure 7) to see the effect of varying the ratio of the two partition sizes. Configuration 1 (with sites 2 and 3 disconnected), had lower commit and higher RCP abort rates as compared to Configuration 2 (with sites 1 and 2 disconnected). This demonstrates that the quorum consensus protocol with uniform weight assignment, has higher RCP abort rates when partitions are of nearly equal sizes. We observe that failure of one link (that causes a partition) results in a lower number of transactions getting aborted by the 2PC. In fact we notice that the maximum number of transaction that are aborted is bound by the transaction arrival rate. This gives a clear indication into the amount of aborts 2PC can contribute as compared to the RCP or the CC protocols.

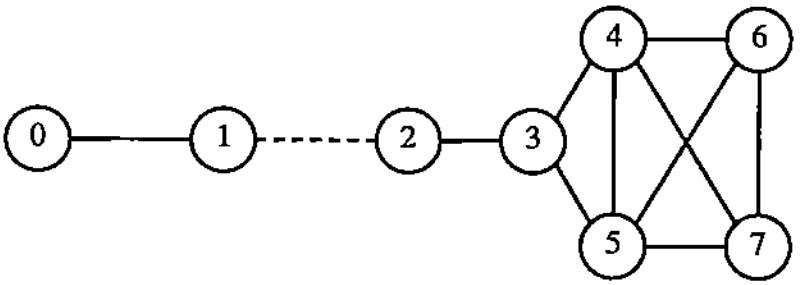

Figure 6-a. Configuration 2

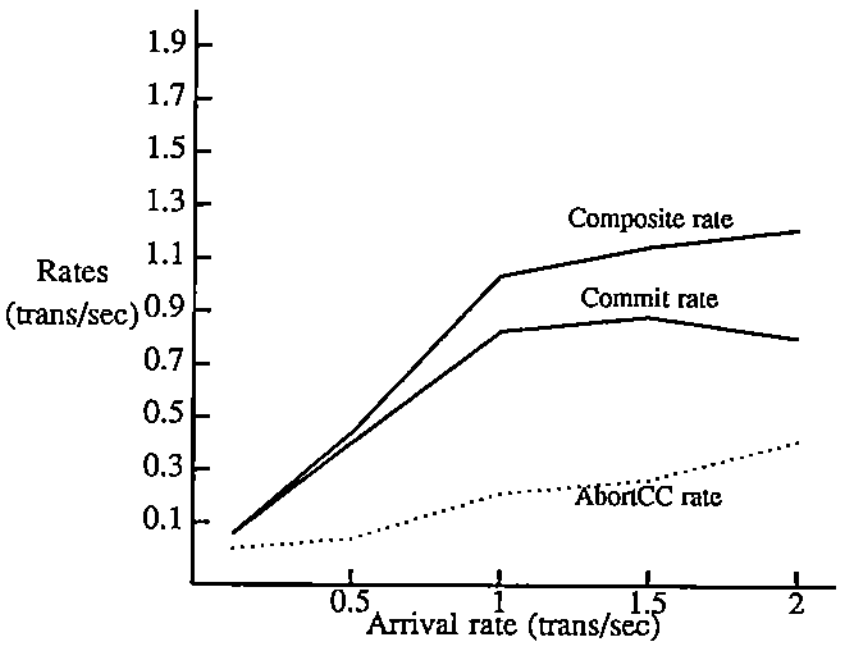

Figure 6-b Commit and Abort rates in Configuration 2
with sites 1 and 2 connected

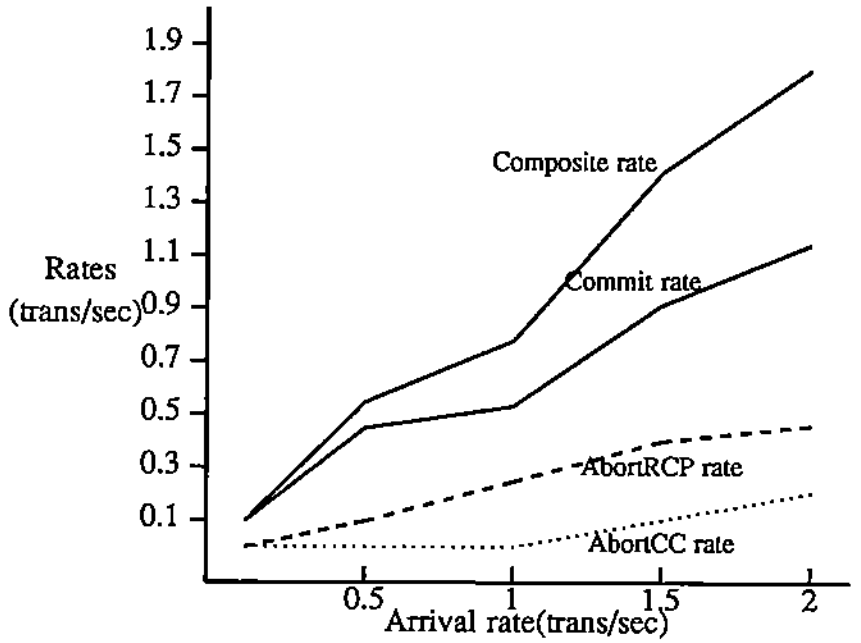

Figure 6-c Commit and Abort rates in Configuration 2
with sites 1 and 2 disconnected

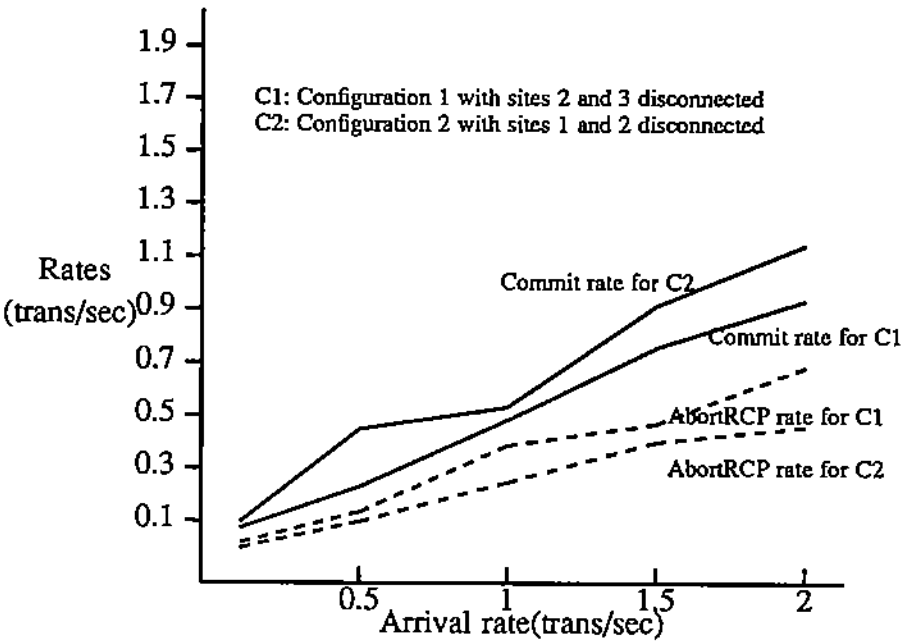

Figure 7 Comparison of Commit and Abort rates
for a 2-6 and 3-5 partitions

3.4. Experiment II: Traffic in Quorum-based Systems

In this experiment, we study the message traffic generated by the quorum consensus protocol in a fully replicated, fully connected 8-site configuration that run on a 6-processors Sequent Symmetry machine. In this experiment we measured the traffic

(message/second) against transaction arrival rate and we studied the side effect of traffic on the system's behavior.

3.4.1 Setup of the Experiment II

A database of 200 fully replicated objects was used in an 8-site fully connected configuration that uses equal weights for their copies. Experiments were conducted for transaction arrival rates varying from 0.5 to 4.0 transactions per second. The transaction maximum size(*tms*) was fixed to 5 operations. The read/write ratio(*tmix*) was fixed to 0.5. For each arrival rate, we measured the traffic, and the commit and the concurrency control abort rate. Each measured point was actually an average over 6 identical experiments.

3.4.2 Analysis and Observations of Data Measurements in Experiment II

Figure 8 shows that the traffic *linearly* increases as the arrival rate goes up to 2.5 transaction/second. After this it saturates and starts to decrease. For example, doubling the arrival rate from 0.5 to 1.0 doubles the traffic from 35 to 70 message/second, while doubling it from 3.5 to 4.0 does not increase the traffic but slightly decreases it instead. To explain Figure 8, we recognize that the maximum rate at which messages can get processed in a machine like the Sequent Symmetry is 200 message/second. The results in Figure 8 are bound to our choice of the unreliable UDP socket communication protocol. The traffic due to the increase in the arrival rate approaches the maximum message processing rate (200) and results in congesting some of the sockets' queues. In this case the sockets start *loosing packets*. This explains the saturation in the traffic curve. When packets start getting lost, some transactions get blocked at their home sites, and hence their further contribution to message traffic is reduced. This side effect explains the small decline in the traffic curve.

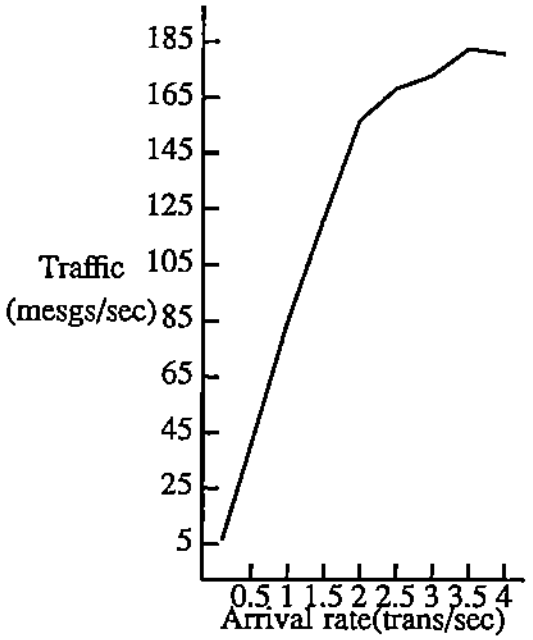

Figure 8 Traffic vs. Arrival rate in a fully connected 8-site Configuration

Figure 9 shows the effect of message traffic on the commit and concurrency control abort rates. The effect of lost packets is obvious from the saturation and decline of the commit rate. The side effect represented in transaction blocking can be seen from the saturation of the concurrency control abort rate, where the reduced interaction between transactions incurs less conflicts and hence less CC aborts. This experiment suggests that the maximum load with which we can conduct an eight site Seth experiments in a machine like the Sequent Symmetry, should not exceed 2.5 transaction/second. As a final comment we emphasize two optimizations in building systems like Seth. The first optimization lies on the communication subsystem and its *Stub* where *message buffering (piggybacking)* can highly reduce message traffic. The second optimization lies on the implementation of standard communication protocols where careful implementations can result in less expensive versions, as was shown in [Bhargava87b].

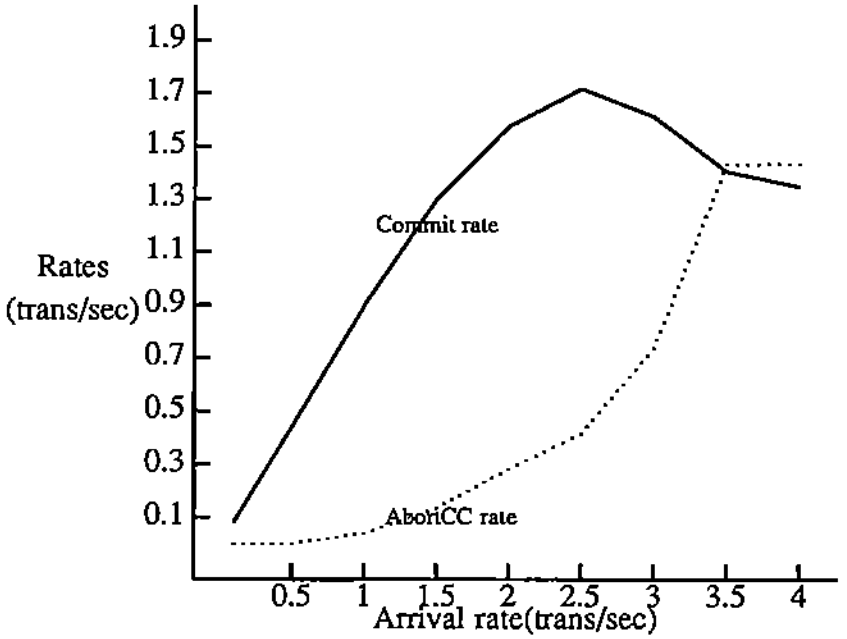

Figure 9 Commit and Abort rates for a fully connected 8-site Configuration

4. CONCLUSION

We have presented the design and implementation of a quorum-based replicated database system called Seth. The contribution of Seth lies on the design and implementation of its network interface which we call the prototype manager. The prototype manager hides the details of the underlying communication network from Seth sites, and therefore the latter do not need any view synchronization information to communicate. The prototype manager is also a flexible means by which various types of failures and repairs can be generated in Seth. The experimentation with quorum consensus has shown that such a protocol is very expensive in terms of the number of messages per transaction. For example, an average of 70 message/transaction was measured in an 8-site fully connected configuration for transaction of at most 5 operations. This figure suggests that even a lightly loaded system may produce very heavy traffic. We have shown how this heavy traffic can slow down the system and finally block transaction processing. For example, 4 transaction/second was the maximum that can be achieved in an 8-site fully connected configuration. In the new version of Seth, we are planning to extend the domain of sites to be multiple machines including SUN workstations and Vaxen. We will try to optimize the number of messages that a transaction generates through optimizing our implementation of Seth sites. We will piggyback manager to reduce message traffic. The current version of Seth consists of 8700 lines of *C* code that runs on Berkeley 4.3 UNIX.

ACKNOWLEDGEMENTS

We would like to thank John Riedl for valuable discussions.

REFERENCES

[El Abbadi86] El Abbadi, A., Toueg, S., Availability in Partitioned Replicated Databases. In Proc. 5th ACM SIGACT-SIGMOD symp. on principles of Database Systems, March 1986.

[Bernstein86] P.A. Bernstein, V. Hadzilacos, N. Goodman, Concurrency Control and Recovery in Database Systems, Addison Wesley, 1986

[Bernstein84] Bernstein B.A., Goodman N., An Algorithm for Concurrency Control and Recovery in Replicated Distributed databases. ACM Trans. on Database Systems 9(4):596-615, December, 1984.

[Bhargava87a] Bhargava B., Transaction Processing and Consistency Control in of Replicated Copies during failures", Journal of Management Information Systems, Oct.1987, vol.4, no. 2.

[Bhargava87b] Bhargava B., Mueller T., Riedl J., Experimental Analysis of Layered Ethernet Software, Proceeding of the ACM-IEEE Fall Joint Computer Conference, Dallas, Texas, Oct.1987.

[Bhargava88a] Bhargava, B., Riedl, J., Implementation of RAID, IEEE Seventh Symposium on Reliable Distributed Systems, Columbus, Ohio, October 1988. (to appear)

[Bhargava88b] Bhargava B., P. Noll, D. Sabo, An Experimental Analysis of Replicated Copy Control during Site Failure and recovery, Proceeding of the 4th Int'l Conference on Data Engineering, Los Angeles, CA, Feb 1988.

[Chan86] Chan A., Skeen, D., The Reliability Subsystem of a Distributed Database Management. Technical Report CCA-85-02, Computer Corporation of America, 1986.

[Eager81] Eager D.L., Robust Concurrency Control in Distributed Databases. Technical Report CSRG#135,Computer Systems Research Group, University of Toronto, October 1981.

[Eager83] Eager D.L., Sevcik, K.C., Achieving Robustness in Distributed Databases Systems. ACM Trans. Database System, 8(3):354-381, September, 1983.

[Gifford79] David K. Gifford, Weighted Voting for Replicated Data, ACM 7th symposium of Principle of Operating Systems, 1979.

[Jajodia87] S. Jajodia, Mutchler, D., Dynamic Voting, Proceeding of ACM SIGMOD 1987 Annual Conference.

[Traiger82] Traiger, I.L., Gray, J., Galtieri, C.A., and Lindsay, B.G., Transactions and consistency in Distributed Database Systems. ACM Trans. Database System, 7(3):323-342, September 1982.