PERFORMANCE AND AVAILABILITY MODELING OF REPLICATED DATABASE SERVERS

Daniel A. Menascé
Dept. of Computer Science, MS 4A5
George Mason University
Fairfax, VA 22030-4444
menasce@cne.gmu.edu

Nagesh Kakarlamudi
Performance Engineering Corporation
3949 Pender Dr.
Fairfax, VA 22030
nkakarla@pec.com

Fault tolerance in DBMSs is achieved via DB server replication. Audit logs are used to synchronize the two copies of the DB. Several schemes can be used to provide various degrees of reliability and performance at different cost levels. This paper uses a two-level modeling approach to obtain combined availability and performance metrics for various design architectures, including (un)balanced capacity and/or (un)balanced load. The modeling techniques used here include Markov chains and mixed-class Queuing Networks. Numerical results demonstrate the use of these modeling techniques.

1. Introduction

The modeling efforts discussed in this paper were motivated by the type of replicated databases found in a large global financial company. To reach a system availability goal of close to 100%, the main database is replicated in a "synchronous" mode so that the secondary database can be activated with minimal service interruption. The primary and secondary database servers are located across a large geographic region and are connected by high-speed links. This type of replication is employed for disaster recovery purposes. The application is basically a store-and-forward messaging system. The requirements include very high availability and strict performance targets. The basic architecture comprises of multiple front-end processors that perform user authentication and message validation. Incoming messages are then routed to a primary database server, which hosts the main data store and also performs message switching. The main data store is comprised of a message file and an audit file. This main data store is replicated in a disaster backup site, called the secondary database server, which is an exact mirror of the primary. Replication is accomplished by sending audit packets over the high-speed links connecting the two nodes, and rebuilding the database from the audit packets.

In more general terms, this paper identifies and studies scalability issues for distributed fault tolerant database systems (DFTDBSs). Using analytical models of a reference architecture, we combine availability, performance, and cost into a single metric called *effectiveness*. This metric is then used to assess different configurations.

Fault tolerance is primarily achieved through redundancy [DELM95, BKGMP]. Redundancy can be achieved in several forms. The most common is the *active/standby* scheme, in which a standby component becomes active when an active component fails. Redundancy can also be implemented with multiple simultaneously active components; if one of them fails, processing continues in degraded mode with the remaining components. In all cases, after a failure of a component, the system as a whole is less fault tolerant. Redundancy can be applied at all levels including processors, networks, memory subsystems, and storage devices.

Fault tolerance is desired primarily for high availability. Highly critical systems which require close to 100% availability, need to be fault tolerant. The type of system dictates what kind of fault tolerance is needed. In this paper we concentrate on distributed fault tolerant databases. Distributed systems are generally implemented on multiple servers interconnected through high-speed networks. These systems are primarily used to distribute the processing load among multiple servers and to provide for fault tolerance. Distributed fault tolerant systems implemented over WANs can also be used as a disaster backup mechanism. In this case, the networking requirements become quite critical.

High availability normally impacts the performance and cost of the system in an inverse manner; higher availability requirements usually result in lower performance and higher cost.

Availability, defined as the percentage of time the system responds to user requests during a particular interval, is computed as

$$\frac{\text{UpTime}}{\text{UpTime} + \text{DownTime}} \quad (1)$$

where UpTime is the total duration of intervals during which the system was operational and responding to users and DownTime is the total interval during which the system did not provide service [SIEW92].

Performance may be described by several metrics including throughput and response time. For the purpose of this paper we concentrate on response time. Response time is normally defined as the time elapsed from the submission of a request to the receipt of the response.

System cost includes total cost of ownership defined as the sum of acquisition and operational costs.

Using these three metrics, a new composite metric, called *effectiveness* of an architecture is defined here as

$$\text{Effectiveness} = \frac{\text{Availability}}{\text{Response Time} \times \text{Cost}} \quad (2)$$

The higher the availability and the lower the response time and cost, the higher the effectiveness of an architecture. For illustration purposes, assume that system A has an availability of 0.995, a response time of 2 sec and a cost of $1.5 million. System B has an availability of 0.9995, a response time of 2.3 sec and a cost of $1.8 million. System A has an effectiveness of 0.332 per (sec x $million) while system B has an effectiveness of 0.241 per (sec x $million). Even though system B has a higher availability, it achieves it at a higher cost and imposes higher response times to all users given that more resources are used to provide the redundancy needed for the increased availability. The absolute value of the effectiveness is not relevant because different units may be used to measure response time and cost. However, when systems are being compared, their effectiveness values can be used for comparison purposes provided that response time and costs are measured in the same units for both systems.

The rest of this paper is organized as follows. Section two describes a reference architecture for the type of distributed fault tolerant database systems considered in this study. Section three presents the analytic model used to compute the availability and response time for the various possible configurations considered here. Section four uses the models developed in section three to analyze some specific architectures. Finally, section five presents some concluding remarks.

2. Model Architecture

The reference architecture shown in figure 1 consists of *primary* and *secondary* nodes, connected to each other. A Front End Processor (FEP), which handles the interface to the clients, is normally only communicating to the primary node (or the secondary node when the primary is unavailable). The primary and secondary have a database and an *audit log* storage space. Changes made to the database are "packetized" and stored in the audit space. These packets are sent to the secondary node, and stored in the secondary's audit space. The audit logs are used for recovery purpose.

A batch process runs in the secondary to update the secondary's database from the information stored in the audit log.

The system is subject to two external workloads: query and update. The arrival rates of these two types of requests are λ_q and λ_u, respectively. These workloads are modeled as open workloads. A *DB synchronization* daemon runs in the background to synchronize the secondary DB with the primary one using the audit packets. When at least one of the nodes is recovering, a *recovery* process is also active. The recovery and DB synchronization processes are closed workloads.

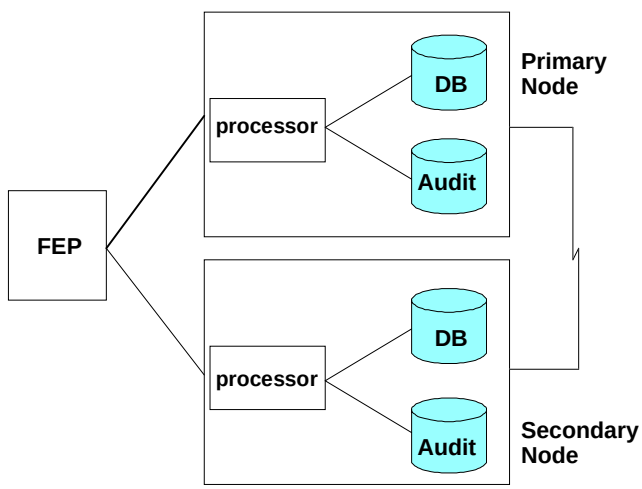

Figure 1 Reference Architecture.

The capacity of the primary and secondary need not be the same. Thus, different scenarios can be evaluated based on the cost of providing a certain capacity to the primary and secondary nodes. The heterogeneity of the primary and secondary is also an important factor when determining the load distribution strategy between them.

3. Analytical Model

We have built analytical models of the architecture at two levels to study the system behavior under the four

different scenarios resulting from the two possible states (up or down) of the primary and secondary. The various states of the system are defined. The system states and its transitions are represented by a Markov chain [KLEI75]. For each state, a mixed class queuing network (QN) model [MAD94] is defined since there are open and closed workloads. The queuing network models are solved using the Mean Value Analysis (MVA) technique for mixed models. Using these results and the Markov chain analysis of the state transition diagram, the probability of each state is computed. The hierarchical approach used in the model is based on the Decomposition principle [COUR75]. Other examples of hierarchical modeling can be found in [MA94, MNN97]. Using these results, the availability and response times are derived and the efficiency computed.

3.1 Systems States

Each of the two nodes: primary and secondary can be in two possible states: up or down. Table 1 represents the combination of all states. A state will be denoted by *xy* where *x* is the state of the primary (*u* or *d*) and *y* is the state of the secondary (*u* or *d*).

The system transitions among these four states according to two types of events: failures of the primary or secondary, or recovery of the primary or secondary.

Table 1 - System States

	Primary Up	Primary Down
Secondary Up	**uu:** System offers required level of performance.	**du:** Level of performance determined by secondary.
Secondary Down	**ud:** Level of performance determined by primary.	**dd:** Level of performance = 0. The system is unavailable.

3.2 State Transition Diagram

Figure 2 depicts the state transition diagram for the entire system. Note that transitions that would be generated by the simultaneous occurrence of recovery and failures (e.g., both primary and secondary going down at exactly the same time, or the primary recovering at exactly the same time as the secondary goes down) are not indicated in the diagram because the events that would generate these transitions are assumed to have a negligible probability of occurrence. This type of assumption is similar to the "one-step behavior" assumption used in operational analysis of queuing networks [AGRA85].

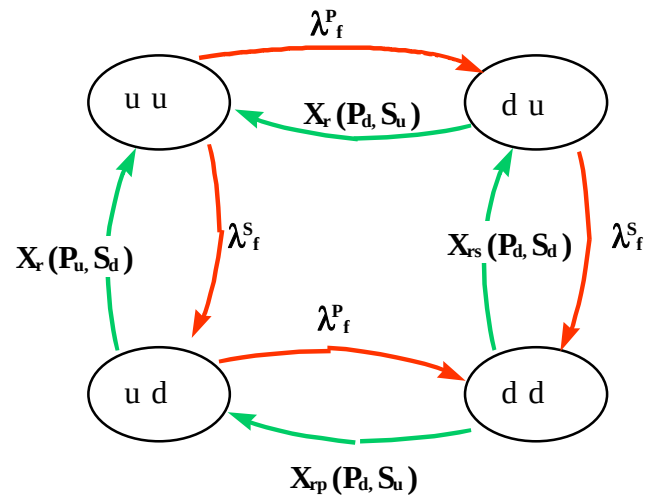

Figure 2 - State Transition Diagram

The following notation is used to represent the state transition rates:

λ^p_f : rate at which the primary node fails; this rate is the inverse of the Mean Time Between Failures for the Primary node ($MTBF_p$).

λ^s_f : rate at which the secondary fails; this rate is the inverse of the Mean Time Between Failures for the Secondary node ($MTBF_s$).

X_r (P_d, S_u): rate at which the primary node recovers when the primary is down and the secondary is up; this rate is the throughput of the recovery process obtained by solving the QN for state *du*.

X_r (P_u, S_d): rate at which the secondary recovers when the primary is up and the secondary is down; this rate is the throughput of the recovery process at the secondary and is obtained by solving the QN for state *ud*.

$X_{rs}(P_d, S_d)$: rate at which the secondary recovers when both primary and secondary are down; this rate is the throughput of the recovery process at the secondary and is obtained by solving the QN for state *dd*.

$X_{rp}(P_d, S_d)$: rate at which the primary recovers when both primary and secondary are down; this rate is the throughput of the recovery process at the primary and is obtained by solving the QN for state *dd*.

The probability of finding the system in state *xy* is denoted by P_{xy}. So, for example, P_{ud} is the probability that the primary is up and the secondary is down.

The four state probabilities can be obtained by writing the flow balance equations for the Markov chain of

figure 2 by applying the *flow-in=flow-out* principle [KLEI75, MAD94] to each state. One of the four equations thus obtained is linearly dependent on the others and should be replaced by the conservation of probability equation. Hence, the resulting set of equations is:

$$(\lambda_f^p + \lambda_f^s) P_{uu} = X_r(P_d, S_u) P_{du} + X_r(P_u, S_d) P_{ud} \quad (3)$$

$$[\lambda_f^s + X_r(P_d, S_u)] P_{du} = \lambda_f^p P_{uu} + X_{rs}(P_d, S_d) P_{dd} \quad (4)$$

$$[\lambda_f^p + X_r(P_u, S_d)] P_{ud} = \lambda_f^s P_{uu} + X_{rp}(P_d, S_u) P_{dd} \quad (5)$$

$$P_{uu} + P_{du} + P_{ud} + P_{dd} = 1 \quad (6)$$

To solve Eqs. (3)-(6) and obtain the four probabilities, we need the values of the throughputs $X_r(P_d,S_u)$, $X_r(P_u,S_d)$, $X_{rs}(P_d,S_d)$, and $X_{rp}(P_d,S_d)$. These throughputs are obtained by solving QN models for each of the four states, as discussed in section 3.4.

3.3 Performance Metrics

3.3.1 Availability

The system is available if at least the primary or secondary are up; therefore, the system is available if it is not in state *dd*. Thus, the availability is given by

$$Availability = 1 - P_{dd} \,. \quad (7)$$

3.3.2 Performance

We are interested in the average response time for queries and updates, R_q and R_u, respectively. Using the QN models for each state, we can compute the response time for queries and updates for that state. Then, the average response time can be computed as

$$R_q = R_q^{uu} \times P_{uu} + R_q^{du} \times P_{du} + R_q^{ud} \times P_{ud} \quad (8)$$

$$R_u = R_u^{uu} \times P_{uu} + R_u^{du} \times P_{du} + R_u^{ud} \times P_{ud} \quad (9)$$

where R_t^{xy} stands for the average response time at state *xy* for transaction of type *t* (*t* = *q* or *u*).

3.4 Queuing Network Models

This section presents the QN models for each of the four states indicated in figure 2. Each node, primary or secondary, can be modeled by a QN such as the one shown in figure 3.

The devices of the QN are:

FEP : Front End Processor

CPU_p : CPU at the primary node

DB_p : database disk at the primary node

AU_p : audit disk at the primary node

NET: network connecting the primary to the secondary

CPU_s : CPU at the secondary

DB_s: database disk at the secondary node

AU_s: audit disk at the secondary node

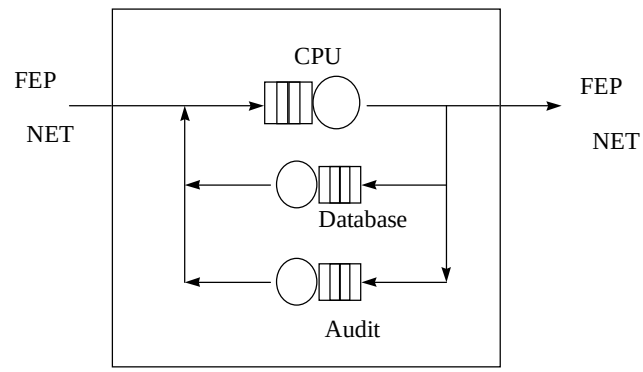

Figure 3 - QN model of a node.

3.4.1 QN Model for state *uu*

The QN for state *uu* (both primary and secondary are up) is shown in figure 4 and has three types of workloads or classes:

Update: this class represents update transactions and is modeled as an open class. When an update transaction arrives at the FEP from an external client, it is routed to the primary node. The database at the primary is updated and an audit packet is written to the audit space at the primary. Then, an audit packet is sent to the secondary node via the network. The secondary then writes the audit packet to its audit space. Following this action, an acknowledgment packet is sent back to the primary. The primary updates its audit, and sends a response back to the external client via the FEP. The resources involved in an update request are then: the CPUs, database and audit disks at both nodes, and the network.

Query: this class represents query transactions and is also modeled as an open class. It needs to be processed only by either the primary or secondary node, and impacts only the DB disk. When a query arrives at the FEP from an external client, it will get routed to the primary or the secondary, depending on the load balancing policy used. The node in charge of processing the transaction replies directly to the client via the FEP. The resources involved in this workload are the CPU

and DB disk for the node that processes the request.

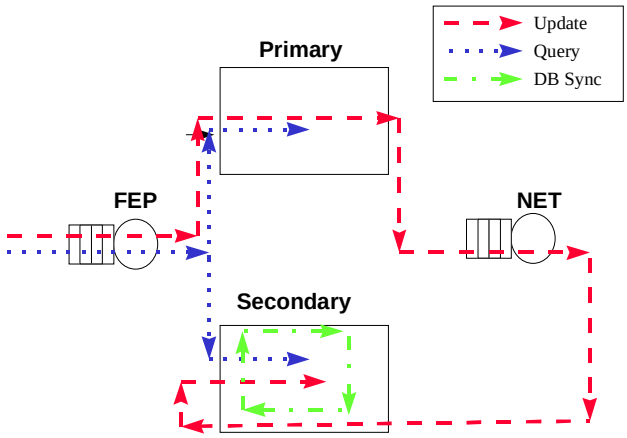

Figure 4 QN model for state *uu*.

DB Synch: this is an internal process executed at the secondary. It reads the audit packets from the secondary's audit space and updates the DB at that node. This workload is modeled as a closed class in the QN model and uses the CPU, audit and DB disks at the secondary.

The QN model for state *uu* is then a mixed class QN model since two classes are open (query and update) and one is closed (DB synch).

The service demands and workload intensity parameters for the three classes are given in Table 2. For open classes, one must provide arrival rates and for closed classes one must provide the level of multiprogramming (MPL).

Table 2 - Service Demands for state *uu*.

Device	Update (open)	Query (open)	DB Synch (closed)
FEP	$D_{FEP, U}$	$D_{FEP, Q}$	0
CPU_p	$D_{CPUp, U}$	$D_{CPUp, Q}$	0
DB_p	$D_{DBp, U}$	$D_{DBp, Q}$	0
AU_p	$D_{AUp, U}$	0	0
NET	$D_{NET, U}$	0	0
CPU_s	$D_{CPUs, U}$	$D_{CPUs, Q}$	$D_{CPUs, D}$
DB_s	$D_{DBs, U}$	$D_{DBs, Q}$	$D_{DBs, D}$
Au_s	$D_{AUs, U}$	0	$D_{AUs, D}$
Workload intensity parameters			
Arrival Rate	λu	λq	
MPL			1

3.4.2 QN Model for state *du*

In this state, the primary is down and the secondary is up. The model for this state is also a mixed class QN as shown in figure 5. There is a new class identified in this model, called Recovery. The two open classes, update and query are the same as in the previous model. The DB Synch class is not present in this model.

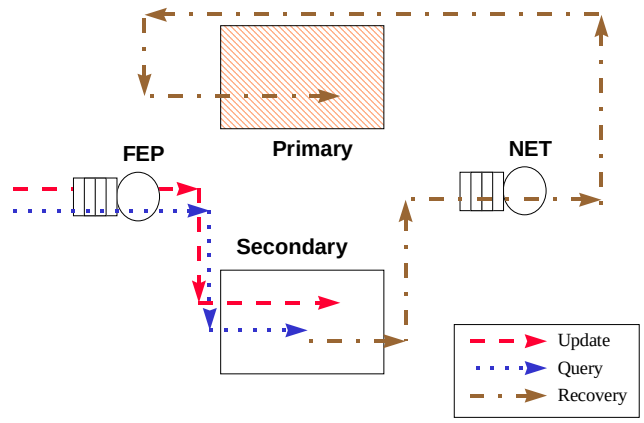

Figure 5 QN model for State *du*.

Recovery: This is the process of rebuilding the primary database from the audit packets of the secondary audit. This is modeled as a closed class.

The service demands for the three classes in this model are given in Table 3.

Table 3 - Service Demands for State *du*.

Devices	Update (open)	Query (open)	Primary Recovery (closed)
FEP	$D_{FEP, U}$	$D_{FEP, Q}$	0
CPU_p	0	0	$D_{CPUp, PR}$
DB_p	0	0	$D_{DBp, PR}$
AU_p	0	0	$D_{AUp, PR}$
NET	0	0	$D_{NET, PR}$
CPU_s	$D_{CPUs, U}$	$D_{CPUs, Q}$	$D_{CPUs, PR}$
DB_s	$D_{DBs, U}$	$D_{DBs, Q}$	$D_{DBs, PR}$
AU_s	$D_{AUs, U}$	0	$D_{AUs, PR}$
Workload intensity parameters			
Arrival Rate	λu	λq	
MPL			1

3.4.3 QN Model for State *ud*

This is the case in which the primary is up and the secondary is down. The model for this state is quite similar to that of state *du*. The service demands may

be different, if primary and secondary are unbalanced nodes.

Recovery: This is the process of rebuilding the secondary database from the audit packets of the primary audit. This is modeled as a closed class.

The QN model for state *ud* is given in figure 6 and the service demands and workload intensity parameters in Table 4.

3.4.4 QN Model for state *dd*

This is the case in which both primary and secondary are down and in the process of recovering. The model for this case is composed of two separate single class QN models for each node. Both reflect the recovery processes, but are 'cold' since the other node is also down, hence cannot support in rebuilding the DB:

Primary Recovery Cold: This is the process of starting up the primary node recovering its database. This is modeled as closed class.

Secondary Recovery Cold: This is the process of starting up the secondary node and recovering its database. This is also modeled as closed class.

The QN model for state *dd* is depicted in figure 7. The service demands for the two classes are given in Table 5.

Table 4 - Service Demands for State *ud*.

Devices	Update (open)	Query (open)	Secondary Recovery (closed)
FEP	$D_{FEP, U}$	$D_{FEP, Q}$	0
CPU_p	$D_{CPUp, U}$	$D_{CPUp, Q}$	$D_{CPUp, SR}$
DB_p	$D_{DBp, U}$ 0	$D_{DBp, Q}$	$D_{DBp, SR}$
AU_p	$D_{AUp, U}$	0	$D_{AUp, SR}$
NET	0	0	$D_{NET, SR}$
CPU_s	0	0	$D_{CPUs, SR}$
DB_s	0	0	$D_{DBs, SR}$
AU_s	0	0	$D_{AUs, SR}$
Workload intensity parameters			
Arrival Rate	λu	λq	
MPL			1

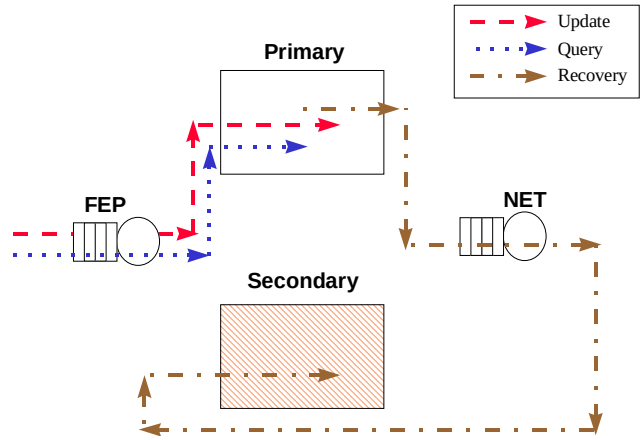

Figure 6 QN model for State *ud*.

Table 5 - Service Demands for State *dd*.

Devices	Primary Recovery Cold (closed)	Primary Recovery Cold (closed)
FEP	0	0
CPU_p	$D_{CPUp, PRC}$	0
DB_p	$D_{DBp, PRC}$	0
AU_p	0	0
NET	0	0
CPU_s	0	$D_{CPUs, SRC}$
DB_s	0	$D_{DBs, SRC}$
AU_s	0	0
Workload intensity parameters		
MPL	1	1

3.5 Recovery time computation

The recovery time in states *du* and *ud*, depends on the number of audit packets at the up node when the recovery of the failed node is about to begin. Assume that there are *n* packets to be replayed at the beginning of the recovery period. These packets accumulated at the up node during the time it took the failed node to come up. This time is equal to the MTTR (Mean Time to Recover) of the node. So, $n = \lambda_u$ x MTTR. The recovery process is said to be complete when no packets are left to be replayed.
During recovery, new update transactions continue to arrive at the up node, these update transactions will result in more packets to be replayed. The average number of new packets that arrive during the recovery period T is λ_u x T. A typical plot of the number of packets to be replayed at the recovering node versus time is shown in figure 9.

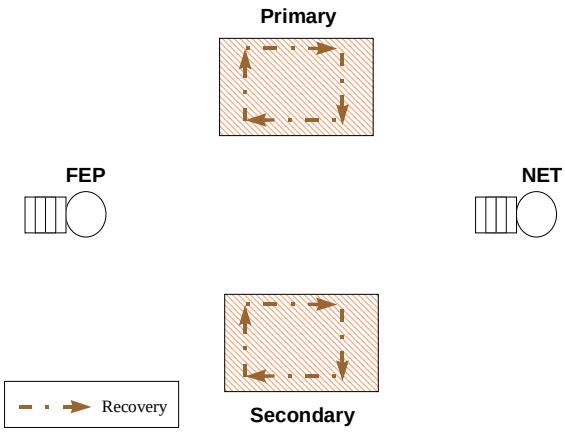

Figure 7 - QN for State *dd*.

Let R be the time needed to reply one packet at the recovering node. This time is just the sum of the service demands at all devices involved in recovery at the recovering node given that there is only one recovery process and there are no other workloads in the recovering mode. So, the total number of packets replayed during time T is equal to T/R. This number must be equal to the number of packets, *n*, present at the beginning of the recovery process plus the number of new packets that arrive during recovery. Thus,

$$n + \lambda_u \times T = T / R. \qquad (10)$$

Using the fact that $n = \lambda_u \times MTTR$ in eq. (10) and solving for T, we get that

$$T = (\lambda_u \times MTTR \times R) / (1 - \lambda_u \times R) \qquad (11)$$

Clearly, $\lambda_u R < 1$ for the recovery process to complete eventually. So, the number of packets replayed during the recovery process, T / R, is equal to $(\lambda_u \times MTTR) / (1 - \lambda_u \times R)$. The service demand of each device involved in the recovery process is then computed as the service time per packet multiplied by the average number of packets replayed.

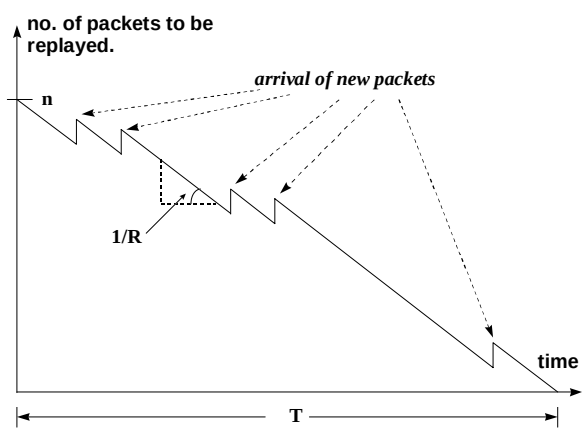

Figure 9 - Number of packets to be replayed vs. time

The following steps must be followed to solve the model:

1. Compute the service demands for the QNs for all four states according to Tables 2-5. The service demands for the recovery workloads in states *du* and *ud* must be adjusted according to the average number of packets to be replayed.
2. Solve the QNs for each state using Mean Value Analysis (MVA) techniques for mixed multiclass Queuing Networks [MAD94] for states *uu, du,* and *ud,* and regular MVA for state *dd*. As a result of this step, the throughputs $X_r\ (P_d, S_u)$, $X_r\ (P_u, S_d)$, $X_{rs}\ (P_d, S_d)$, and $X_{rp}\ (P_d, S_d)$ are obtained.
3. Solve the system of linear equations (3)-(6). We used MS Excel's Solver for that purpose.
4. Compute the availability using eq. (7).
5. Compute the response times for query and updates using eqs. (8) and (9).
6. Compute the effectiveness for queries and updates using eq. (2). In the case of queries, use the query response time and the case of update effectiveness use the response time for updates. The cost in the denominator of eq. (2) should always be the entire system cost, including primary and secondary.

4. Numerical Results

To illustrate the use of the model described in the previous section, we considered a scenario in which the primary is 33.3% faster than the secondary and costs 25% more. Primary and secondary are connected by a 45 Kbps link. For all graphs discussed here, we assume that the update arrival rate is half of the arrival rate of query transactions.

Figure 10 shows the effectiveness of queries and updates for the case when all query traffic goes to the primary when both primary and secondary are up (unbalanced load case). As the picture indicates, the effectiveness of query requests (dashed line) falls at a much faster rate than for updates as the load increases.

Figure 11 is similar to figure 10, except that in this case 50% of the traffic goes to the primary and 50% to the secondary when both are up. As we can see the effectiveness for queries is significantly better than in the no load balancing case at high loads. This fact is better seen in figure 12 that shows the query effectiveness for the unbalanced and balanced (50%) cases.

As seen in figure 12, the query effectiveness decreases at a much slower rate for the balanced case (solid line) as the arrival rate increases.

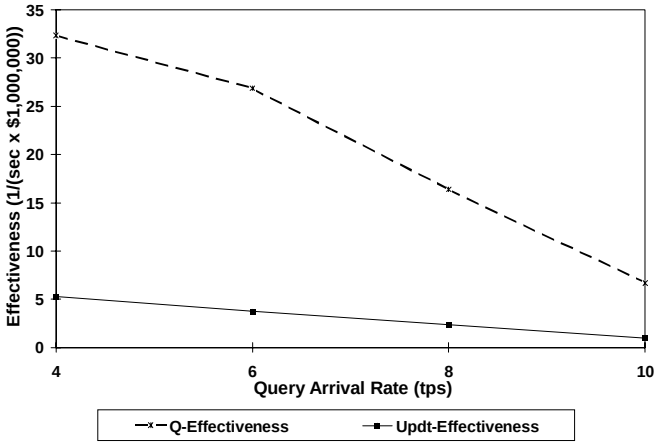

Figure 10 - Query and update effectiveness vs. the arrival rate of query requests (in tps) for the unbalanced load case.

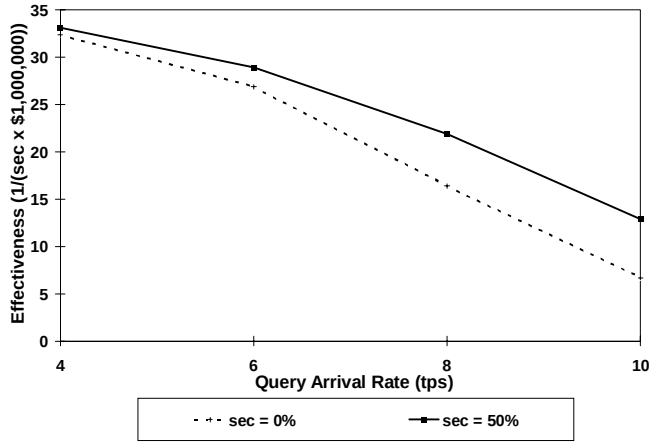

Figure 12 - Query effectiveness vs. the arrival rate of query requests (in tps) for the balanced and unbalanced load cases.

5. Concluding Remarks

There are many ways to assess a fault tolerant database server. Clearly, availability, response time, and cost are three important metrics. In this paper, we suggest a new metric, effectiveness, defined as availability / (response time * cost). We develop an analytic model, based on the principle of decomposition [COUR75], to compute the effectiveness of a fault tolerant DB server. Decomposition is fully applicable in this case, since it is expected that any replicated DB server of interest will spend most of its time in any of the states, especially state *uu*, than transitioning between states. The model combines standard well know techniques such as Markov chains [KLEI75] and Mean Value Analysis for mixed multiclass QNs [MAD94] and may become a valuable tool in sizing fault tolerant systems.

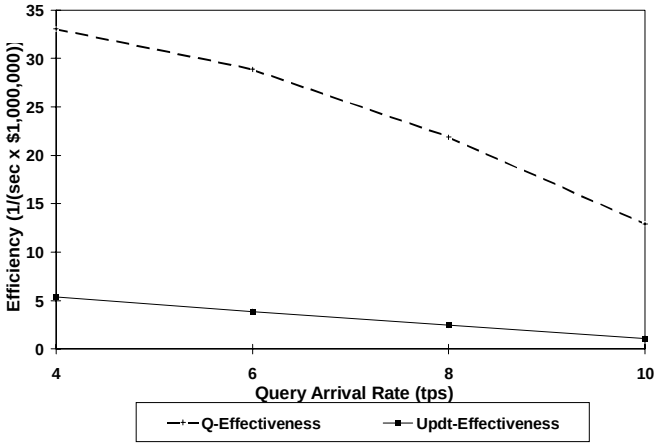

Figure 11 - Query and update effectiveness vs. the arrival rate of query requests (in tps) for the balanced load case.

References

[AGRA85] Agrawal, S. C., *Metamodeling: A Study of Approximations in Queuing Models,* MIT Press, 1985.

[COUR75] Courtois, P., "Decomposability, instabilities and saturation in multiprogramming systems," *Comm. of the ACM*, Vol. 18, No. 7, 1975.

[DELM95] Delmolino, D., "Strategies and Techniques for Using Oracle 7 Replication," Oracle Corporation, 1995.

[BKGMP] Beedubail G., A. Karmarkar, A. Gurijala, W. Marti and U. Pooch, "Fault Tolerant Objects in Distributed Systems Using Hot Replication." Department of Computer Science, Texas A&M University, College Station, TX - 77843.

[Info95] INFORMIX-OnLine Extended Parallel Server for Loosely Coupled Cluster and Massively Parallel Processing Architectures. July 1995. Informix White Paper

[KLEI75] Kleinrock, L., *Queuing Systems*, *Vol. I: Theory*, Wiley-Interscience, New York, 1975.

[MAD94] Menascé, D. A., V. A. F. Almeida, and L. W. Dowdy, *Capacity Planning and Performance Modeling: from mainframes to client-server systems*," Prentice Hall, Upper Saddle River, NJ, 1994.

[MA94] Menascé, D. A. and V. A. F. Almeida, "Two-level Performance Models of Client-Server Systems," Proc. of the 1994 Computer Measurement Group Conference, Orlando, FL, Dec. 1994.

[MNN97] Menascé. D. A., K. Nguyen, and D. Nguyen, "Performance Modeling of a UNIX Communications Server," Proc. of the 1997 Computer Measurement Group Conference, Orlando, FL, Dec. 1997.

[SIEW92] Siewiorek, Daniel P., *Reliable Computer Systems*, Digital Press, 1992.