A Stochastic Analysis of the Performance of Distributed Databases with Site and Link Failures

Amitabh Shah*
Dipak Ghosal**

TR 90-1072
January 1990

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

*Supported in part by a grant from Xerox Corporation.
**Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742.

A Stochastic Analysis of the Performance of Distributed Databases with Site and Link Failures

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

Dipak Ghosal
Institute for Advanced Computer Studies
University of Maryland
College Park, MD 20742
ghosal@umiacs.umd.edu

January 22 1990

A stochastic model for analyzing the performance of a distributed database is proposed. The database is prone to site and link failures, possibly leading to a partition of the underlying communication network. The system model is parametrized to support very general assumptions about data replication, transaction access patterns and network connectivity. For concreteness of analysis, a concurrency control protocol based on Thomas' Majority Consensus protocol and the Two-Phase Commit Protocol is used. A detailed queueing network solution is given to evaluate the values of relevant system parameters. A new performance measure called *expected system degradation* is proposed; this measure is a combination of availability of data and the transaction response time; this is the first step towards the ultimate goal of defining the notion of availability for real-time transaction systems. The model allows a database designer to analyse the expected system performance and choose the right input parameters that emphasize the relative importance of availability and response times.

*Supported in part by a grant from Xerox Corporation.

A Stochastic Analysis of the Performance of Distributed Databases with Site and Link Failures

1 Introduction

We develop an analytical model to study the performance of distributed databases in presence of site and link failures. The failures are assumed to be benign (crash failures), and can lead to a partitioning of the underlying communication network. The model that we develop is a stochastic one, with assumptions about transaction arrival and service rates, the rates of site and link failures, and the rates of recovery. For concreteness of analysis, we use a variant of Thomas' Majority Consensus Algorithm [Tho79] and the Two-Phase Commit Protocol [Gra78] for concurrency control.

We make very general assumptions about the system, in particular about the connectivity of sites, about data replication, and about the transaction access patterns. In fact, our model is completely parametrized to give us a whole spectrum of system behavior. Our goal is to study two important system performance measures—*availability of data* and *transaction response time*—and the interplay between the two. Our particular interest is in the assumption of *dominant local access* of data [SM89] and in a hierarchically constructed distributed database with this assumption of locality in mind [MS90]. But the methodology presented in the subsequent sections is applicable to any assumption of locality of access, as well as of the underlying network architecture. Also, while we have assumed a particular concurrency control protocol to show specific details of the performance analysis, the methodology can easily extend to other protocols as well.

Availability of data is an important measure of performance in a database system. It has generally been defined as the amount of time the data is available for access, or the proportion of successful transactions in the total number of transactions submitted [COK86] (over a period of time). We assume this latter definition in this paper. One of the principal reasons for designing distributed databases is to increase availability of data by replicating it at several sites. However, too much replication also hampers availability, since the executions of the concurrency protocols that ensure consistency of the replicated data get more time-consuming [SM89]. The possibility of site and link failures in the communication network further complicate things and contribute towards reducing availability.

Transaction response time is the time that elapses between the instant a user submits a transaction to the system, and the instant the results of that transaction are communicated back to the user. The response time will, in general, depend on the complexity of the concurrency control protocol, as well as on the size of the system. Component failures will also adversely affect the response times. But, what is more interesting is the interplay between availability and response times: by waiting long enough (so that failed sites and links recover and participate in the concurrency control protocol) more transactions will be serviced, thus increasing availability, but increasing response times as well. The precise nature of the database applications will in general determine the relative emphasis of one over the other. We propose a new measure of performance, called the *expected system degradation* that captures this trade-off. The stochastic model proposed here allows a database designer to analyze the expected system degradation based on the input characteristics and fine tune the system according to the relative importance of the two measures.

We believe that the methodology presented in this work would be very suitable to analyzing the performance of *real-time* distributed databases. The response time analysis that is presented here forms a basis for defining and analyzing *real-time availability*. We are investigating the extensions of this analysis to this end.

The stochastic model we propose is based on the model used by the first author in [GT89]. Our work differs from that analysis in several ways. First, we consider a hierarchical construction of the database with partial replication of data. Second, we are interested in the system performance under the locality of access assumption, and hence deal with transactions that access different kinds of data (see Section 2). Third, we allow site and link failures, possibly leading to network partitions; only site failures (that do not partition the network) are considered in [GT89] Fourth, the concurrency control protocol considered in [GT89] is the same as that of Thomas; we use a variant of it. Finally, the performance measure that we want to study is system degradation; two different measures, called the system power and lost potential are used in [GT89].

The rest of this paper is organized as follows. In section 2 we present the assumptions about the system model, about failures in the system, and about the stochastic model we propose to analyze. An overall system analysis, based on the mean values of the system variables is performed in section 3. In section 4, the analytical expressions for the response times, and other measures are derived. The expression for expected system degradation is derived in section 5. We conclude with a discussion in section 6. Details of the analysis are presented in the Appendix.

2 Assumptions about the System Model

The underlying communication network of the database is a Harary Network [MS90] consisting of S sites divided into k groups. The sites in a group are connected via a network of *regular* degree and the groups themselves are connected in a *hypergraph*, also of a graph of regular degree. A Harary Network is characterized by high connectivity within the groups, and low connectivity across the groups. A typical Harary Network consisting of 5 groups and 32 sites is shown in Figure 1.

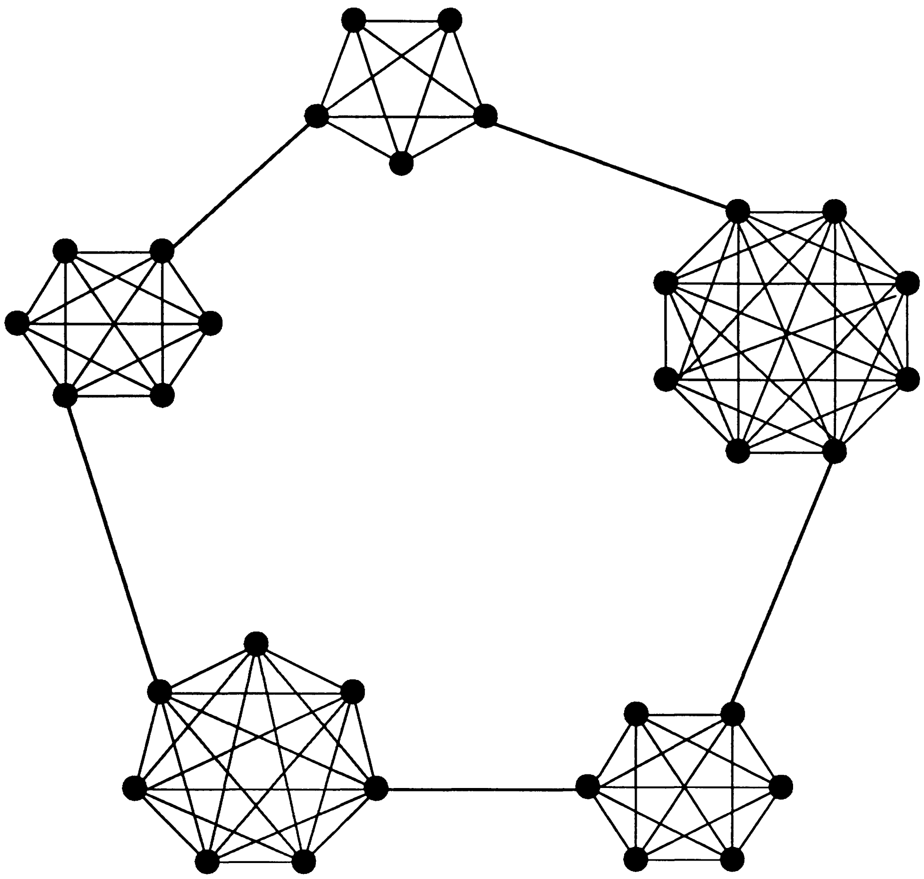

Figure 1: A Typical Harary Network

The set of data in the database is assumed to be *partially* replicated. By this we assume that the set of data is divided into $k+1$ disjoint sets, k of these sets are *local* data sets for the k groups, and one set is that of *global* data. Thus, for any site, there are three kinds of data—local, global, and non-local. Local data for a site is the data that is replicated at every site in the group that the site belongs to. Global data is replicated at every site in the system. The last is the union of the sets of data that are local to other groups.

Transactions are submitted to individual sites in the system by the *users*. Transactions in this system can be

- local transactions—those that access only data local to the site,
- global transactions—those that access only global data,
- non-local transactions—those that access only non-local data, or
- mixed transactions, those that access more than one type of data.

We assume that the system is *homogeneous*, i.e., all the sites, and all the links are identical in their expected behavior. We have already assumed connectivity via regular graphs; homogeneity further implies that the data access pattern of the transactions arriving at a site (i.e., the proportion of local, global, non-local, and mixed transactions in the set of transactions) is the same for all the sites. Also identical are the expected failure behaviors of the sites and the links in the system (see also section 2.3).

The grouping of the sites in a Harary Network is based on the locality of transaction access. If the set of transactions can be divided into several groups, each predominantly accessing a different set of data, then it makes sense to divide the set of sites into several groups, each responsible for a set of data and the transactions that access that data. (A similar model is also used in Bakunin Data Networks [KGM87].) On the other hand, if most of the transactions in the system are global, then the set of sites would be divided into very few groups, reflecting the fact that they need to be densely connected for more reliable communication (and hence, better availability).

Our particular interest is in the assumption of *dominant local access* [SM89]. This assumption says that most of the transactions arriving at a site are for local data, as evidenced in most real-life distributed databases. Thus, the access pattern of transactions is as shown in figure 2(a). To conform with the assumption of homogeneity, we assume that the access pattern is as shown in figure 2(b). This issue is discussed in detail in [SM89,MS90].

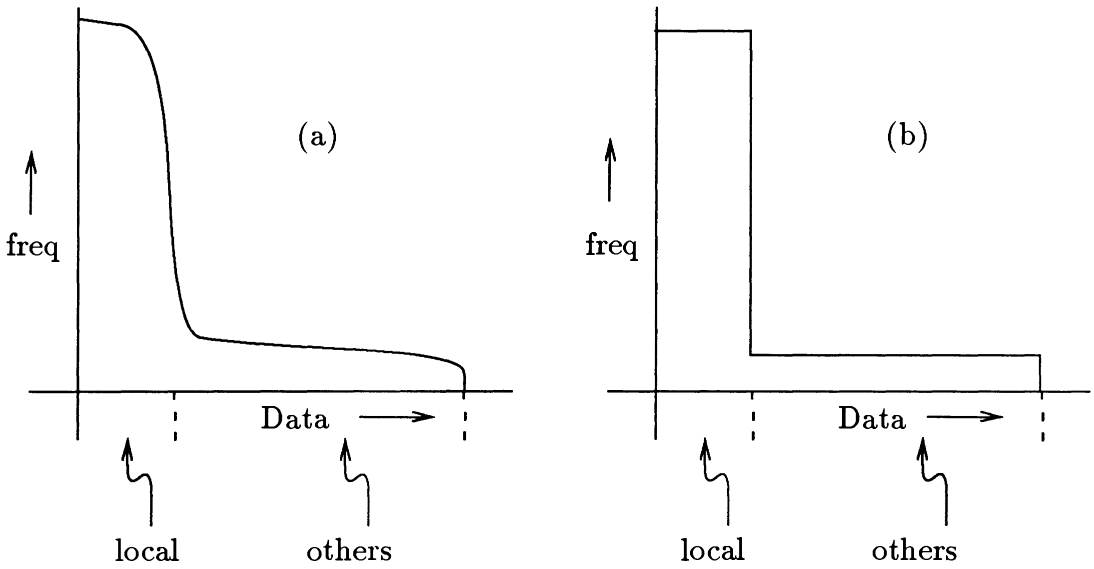

Figure 2: Data Access Patterns

2.1 Concurrency Control Protocol

As a protocol for concurrency control, we present below a variant of Thomas' majority consensus algorithm [Tho79]. It is actually a combination of the *two-phase commit* protocol [Gra78] and Thomas' algorithm, appropriately augmented to cope with site and link failures. The concurrency control protocol is used to service the transactions that arrive at the sites in the system. The service includes resolving conflicts between transactions and installing the effects (if any) of the successful transactions.

A transaction can be either a *read* transaction, or an *update* transaction. With every data item is associated a *timestamp* that reflects the time the data was last updated; for a data item d, let $ts(d)$ denote this timestamp. Also, associated with every arriving transaction T is another timestamp, denoted $ts(T)$.

A local, or a global read transaction arriving at a site s is serviced as simply a read of the local copy of the data. A non-local read of a data item d is serviced at s by sending the read request to a site in the group where d is local. It is serviced there as a local read and the result communicated back to s. It is easy to see that the local and global read transactions always complete, whereas the non-local reads may not complete due to failures. With the assumption of dominant local access, this loss is not very much, and we will not concern ourselves with the availability of read transactions any further.

The protocol for update transactions is much more elaborate and requires five rules to be defined:

1. *voting rule* for concurrency control,

2. *communication rule* for stipulating how the communication between sites takes place,

3. *request resolution rule*, along with the voting rule, for deciding the fate of a transaction, as well as for resolving the conflicts between transactions,

4. *update application rule* for applying the results of a successful update transaction to the replicated data, and

5. *timestamp generation rule* for generating timestamps for the transactions that arrive from the users, as well as determining new timestamps for data that have been updated by the update application rule above.

For our purpose, we need to describe only the first three of these; for the other two, see [Tho79]. First, we describe the update transaction in more detail, as well as establish some terminology. An update transaction is a sequence of read and write operations, accessing several database variables. Two transactions are said to *conflict*, if they have at least one accessed variable in common, and at least one of them is an update transaction. The site where an update transaction is submitted, is called the *initiator* site, and the other sites in that group are called the *cohorts* of the initiator.

Recall that there are two kinds of timestamps: those associated with data items, and those associated with the transactions. We will also assume that the latter timestamps define a total priority order on the arriving transactions, i.e., a transaction has higher priority than another one if the timestamp associated with the first one is greater than the timestamp associated with the second one. We say that an update transaction T is *late* if the timestamp of any variable that it is accessing is greater than the timestamp of the T itself, i.e., $\exists d{:}T$ accesses d and $TS(d) > TS(T)$.

We first describe the protocol rules for local transactions.

2.1.1 Concurrency Control for Local Transactions

1. *Voting Rule*: Associated with every site are two lists, a *pending* list, and a *deferred* list. The former is the list of all the update transactions that the site has voted on, but are not yet resolved. The latter is the list of the update transactions that the site could not vote on due to conflicts with unresolved transactions, and are deferred for later voting. Recall that an update request consists of the list of variables that are read and written. Let an update transaction T arrive at a site s. Then s votes

   (a) a *REJECT* if T is late. The user is then notified that the transaction is **rejected**.

   (b) an *OK* if T is not late, and there is no unresolved conflicting transaction (in the pending list) that s has voted (*OK*) on. The request is then broadcast to the cohorts, as well as put in the pending list of s and later resolved when the votes from the cohorts are received.

   (c) a *PASS* if T is not late but there exists a conflicting transaction P in the pending list of s, and $ts(T) < ts(P)$. T is then broadcast to the cohorts for their votes.[1]

   (d) a *DEFER* otherwise, i.e., T is not late, and there exists a conflicting transaction P in the pending list, such that $ts(T) > ts(P)$ and the request is put in the deferred list at s for later consideration for a vote.

2. *Communication Rule*: The communication used in the protocol is a *broadcast* with time-outs. Thomas [Tho79] describes a broadcast scheme without time-outs, and two other schemes based on daisy chaining, with and without time-outs; for his protocol he uses the daisy chaining without time-outs. We believe that for high performance in terms of faster transaction response times, a broadcast is much more appropriate. Also, since sites and links fail, a time-out is necessary.

   For a local transaction, the initiator broadcasts the request to its cohorts. The latter, on receiving such a request, vote on it (according to the voting rule described above) and communicate the votes back to the initiator. The

[1]The *PASS* vote is not necessary for the (partial) correct execution of the concurrency protocol, but is necessary to guarantee that no deadlocks occur due to sites waiting for one another's votes before voting on their own requests. See [Tho79] for a discussion on this issue.

initiator then decides the fate of the transaction based on the request resolution rule and communicates the result to all the cohorts. All the sites then apply the result to their local copies according to the update application rule.

We assume that the broadcast are *atomic* in that if a site fails during a broadcast, the message is delivered to none of its cohorts. However, note that if the site was up during the broadcast, it does not mean that every cohort that is up will receive the message—link failures may prevent this from happening.

3. *Request Resolution Rule*: As described above, the transaction receiving site broadcasts the transaction for votes only if its own vote was an *OK* or a *PASS*. It collates the votes of other sites for the request. The transaction is resolved as

   (a) a **reject** if at least a majority of sites voted a *REJECT* or a *PASS* for it (and hence no majority of *OK* is possible). The initiator broadcasts a **reject** to all the sites whereby the sites that voted *OK* on the transaction remove the it from their pending lists.

   (b) an **accept** if there was a majority of *OK* votes. The coordinator then broadcasts an **accept** to all the other sites all of which apply the update application rule to install the new values, and update the corresponding timestamps.

   In either case, a transaction (if any), that was deferred due to a conflict with the transaction just resolved, is considered for voting as before.

2.1.2 Concurrency Control for Other Types of Transactions

In the previous section, we described in detail the concurrency control protocol for local transactions. For a non-local transaction, the initiator simply forwards the request to some site in another group to which the accessed data is local. The transaction is then treated as a local transaction to that group, is processed as such, and the initiating site is notified of the result.

For a global transaction, the initiator broadcasts the request to one site in each of the k groups. These sites, and the receiving site, each act as an initiators for this request for their respective groups. The request is then broadcast to each group, and the votes of each group are collated by each initiator and sent to the original receiving site. The latter then resolves the request and notifies every other site by a similar hierarchical global broadcast. Similar schemes can be employed to deal with mixed transactions as well.

It should be clear from the above discussion that it would be sufficient to solve the queueing model and derive the performance measures for a local broadcast. The others can be derived from it by changing system parameters appropriately. Thus, in what follows, we only describe the rules and the queueing model for dealing with local transactions.

2.2 Assumptions about the Queueing System

We would like to model a collection of sites where transaction arrive independently, transaction requests move between sites, and leave the system on resolution. Clearly, the most natural stochastic model for such a system is an *open Jackson Network.* To model failure and recovery, we let the servers in the network fail with the possibility of server state being flushed-out, and recover later. It turns out that with the assumptions of independent arrival of transactions at different sites and of homogeneity (described below), we can treat this network as a *separable* Jackson network [LZGS84]. This enables us to model each site separately, and obtain the overall system solution by combining the solutions for individual sites.

Again, we reiterate that we are dealing local transactions that are resolved via broadcasts in a single group. Let the group consist of $S = 2N + 1$ (for some N)[2] connected by a communication network of regular degree.[3] We will assume that communication between two functioning sites, if possible at all, takes a constant amount of time $\overline{R}_c$. We further assume that all the sites in a group are homogeneous, i.e., they have the same mean characteristics insofar as the transactions that initiate at those sites, the locality of accesses of these transactions, the failure characteristics, etc. are concerned. Thus, we can completely characterize the system by describing the functioning of one site.

To deal with the arriving transactions (also referred to as *customers*, using the Queueing Theory terminology), a site maintains two queues, Q_1, and Q_2, as well as a pending list *PL*, and a deferred list *DL*. We also refer to Q_i as the "class-i"

[2]The assumption that there are odd number of sites is only to obviate the need for taking "floor"s of the number of sites for computing the majority; there is no loss of generality.

[3]Actually, for an odd size graph, there could be possibly one node with degree one more than the rest [BM76], thus contradicting the assumption of a regular network, but its effect on the system performance, specifically on the probability of communication would be negligible, and hence is ignored.

queue, and its customers as the "class-i customers", $(i = 1, 2)$. The state of a site is determined by the values of the data items residing at the site, along with their timestamps, the states of the two queues, and the states of the two lists.

For a given site, the *arrival* of transactions from the users is assumed to constitute a Poisson process with a mean inter-arrival rate of λ_2. The arriving transactions are inserted in a queue Q_2 on a first-come-first-served basis. Let $\bar{t}$ be the average size of a transaction, i.e., the average number of data items accessed by the transaction. We assume that the transaction is likely to access data items uniformly.

The *service time* for the transactions in Q_2 is assumed to be exponentially distributed with a mean rate of μ_2 $(\mu_2 > \lambda_2)$. The service includes checking of the timestamps of the data items accessed and performing logical and arithmetical operations on the data. Note that the service rate μ_2 does not account for the site failures. Let η_2 denote the *throughput* for Q_2; this is defined as the *effective* rate of service of the class-2 customers, i.e., the proportion of μ_2 when there are class-2 customers in the system, and the site is up; the exact expression for η_2 is given in the next section. For Q_2, let $\overline{R}_2$ be the mean response time, without taking into account any failures; $\overline{R}_2$ is the sum of the time spent waiting to be served and the time for actual service. Finally, let $\overline{C}_2$ be the mean length of Q_2.

A transaction that is served is given a vote as described in the voting rule above. Let p_{ok}, p_{rej}, p_{pass}, and p_{def} be the probabilities that the vote is an *OK*, a *REJECT*, a *PASS*, or a *DEFER* respectively. A transaction that is voted *OK* or *PASS* is broadcast to all the other sites in the group to get their votes. It is inserted in the pending list only if the vote was *OK*. A transaction that is voted *DEFER* is inserted in the deferred list *DL*. A transaction that is voted *REJECT* is **rejected**.

Voting requests that arrive from other sites are inserted in the queue Q_1 on a first-come-first-served basis. These arrivals are assumed to form a Poisson process with a mean inter-arrival rate of λ_1. In general, $\lambda_1 > \lambda_2$, unless the probability of *REJECT* vote is very high, which is not very likely in most real-life scenarios. In absence of failures, $\lambda_1 \approx (1 - p_{rej})S\mu_2$; the exact expression for λ_1, that accounts for the failures is derived in the next section. Moreover, the queue Q_1 is assumed to have a *preemptive priority* over the queue Q_2, i.e., as long as there are customers in Q_1 waiting to be served, no customers of Q_2 are served. Also, if an arrival of a class-1 customer occurs while a class-2 customer is being served, the latter is preempted, and the former is served. Besides the voting requests, the other items that go into Q_1 are the **accept** and **reject** signals from other sites, as well as the time-out messages from the sites to themselves (discussed in detail in section 2.3).

The service time for the customers in Q_1 is assumed to be exponentially dis-

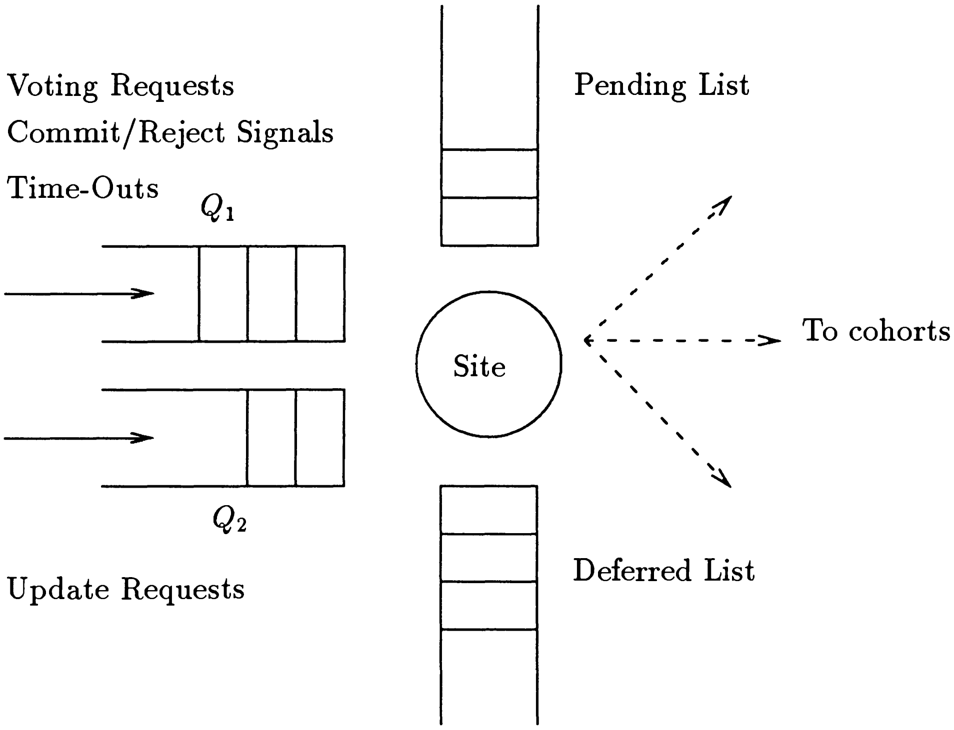

Figure 3: State of a Typical Site

tributed with a mean rate of μ_1 ($\mu_1 > \lambda_1$). The service for Q_1 consists of checking timestamps and voting on the requests from other sites, resolving requests that arrived at the site, applying updates in case of **accept** decisions or signals, and updating pending and deferred lists. Again, similar to η_2, we define η_1 to be the effective service rate, or the throughput for the queue Q_1; the expression for η_1 is given later. Similarly, for the queue Q_1, let $\overline{R}_1$ be the mean response time, without taking into account any failures, and $\overline{C}_1$ be its mean length. Note that the arrival rate λ_1 is not known *a priori*; it in fact depends on the arrival rate λ_2 and other parameters. An expression for λ_1 is derived in the next section. A typical site is shown in figure 3.

2.3 Failure Assumptions

Both sites and links can fail in our model. These failures are assumed to be crash failures. In the worst case, failures can lead to network partitioning where two or more functioning sites are unable to communicate for a period of time. Sites and links recover after failure. The recovery of a site involves recreating (a part of) its state prior to the failure. It is assumed that the input specification of the system includes the specification of the *mean time between failures* (MTBF) for every site. When links fail, they simply recover after some time; there is no state to restore in this case. Thus the links' behavior can be described simply by a probability with which a given link is available. Again, we assume that the input specification gives this probability. Also, the assumption of homogeneity applies to failures as well.

The failures of sites are assumed to occur in a Poisson process with a mean inter-failure rate of γ. Sites recover on failure, and the recovery time r is assumed to be exponentially distributed with a mean of $1/\beta$. When a site fails, both of its queues, Q_1, and Q_2 are lost. Also lost are the pending and the deferred lists, PL and DL. Only data items previously updated are stored on stable storage. On recovery, the site communicates with the other sites in the group to recreate its state. Note that it can not recreate the queue Q_2 since it is the only site that knows about this queue. The transactions queued in Q_2 are **lost** forever. Also, part of the deferred list—the transactions that arrived at the failing site and were not voted on before the failure—will also be **lost**. The **lost** transactions are the first component of *unavailability* in the system.

However, it is possible to recreate Q_1, PL, and the remainder of DL, by exchanging information with the other sites in the group. (Thomas [Tho79] gives a two-pass algorithm to determine this part of the state of the site for the daisy-chaining communication protocol; it can be extended to the broadcast case also using the assumption of atomicity of broadcasts.) The recovery process includes recreating this state, as well as serving the recreated class-1 customers (possibly including some from the recreated pending and deferred lists).

While the failed site is recovering, customers of both classes continue to arrive there. We assume that these customers are not served at all. In particular, the class-2 customers that are not served are said to be **ignored**. The **ignored** transactions are the second source of unavailability. (Depending on the actual application, it may also be valid to assume that no transactions are **ignored**. In this case the class-2 customers that arrive during recovery are also served as a part of the recovery process, thus extending the average recovery time, as well as the average response time. The transaction availability would be higher in this case however. (See also section 5.))

Besides site failures, links can also fail in our model. Link failures can lead to network partitioning in which two functioning sites are unable to communicate until the partition is repaired. To formally model network partitioning, we assume that the probability that any given link is available (for the duration of communication) is known *a priori* and is the same for every link in the system. This, along with the fact that we have a regular network will give us the probability that any pair of functioning sites are able to communicate. Let p_c denote this latter probability.

We shall assume that every site has a built-in time-out period for resolving transactions. Specifically, the initiator, on broadcasting an update request to its cohorts starts a timer for that transaction.[4] It waits for replies from the cohorts until the timer goes off. If the transaction can not be resolved by the replies it receives before the timer then the transaction is **aborted** and the user notified. From the point of view of the queueing model, we assume that the effect of the timer going off is to insert a special time-out customer in the queue Q_1. When this customer is served by the site, either the associated transaction has already been resolved and the customer is discarded, or the transaction can not be resolved and the user is notified of the **abort**. The **aborted** transactions are the third source of unavailability in the system. If there are any more customers for an **aborted** transaction (responses from the cohorts that arrived too late) after the time-out has been served, they are simply discarded.

The time-out period plays a crucial role in the system performance. If it is too large, the initiator will expect to receive responses from all other sites in the group and resolve them. If it is too small, the transactions will be **aborted** most of the time due to lack of sufficient number of responses to determine a majority. The first case leads to an increase in response time of the transaction, the second to reduced availability because of unnecessary **aborts**. The length of a time-out period will depend on the particular application and it will be up to the database designer to decide its value. We leave it as an input parameter and observe its effect on the system performance.

Thus, the fate of a transaction can be an **accept**, a **reject**, an **abort**, **lost**, or **ignored**. While the first two denote successful completion of a transaction, the last three denote unsuccessful completion, and are the sources of unavailability in the system, and hence system (non-)performance. Note that the link failures only con-

[4]It is assumed that every transaction has a globally unique identifier.

tribute towards the **aborts**, while site failures contribute towards all three, **aborted**, **ignored** and **lost** transactions.

As we mentioned before, there is a second aspect to system performance, viz., transaction response time. This depends on the voting probabilities, as well as the time-out period. Increasing the time-out period leads to fewer **aborts**, but increases the average response time, and vice versa. Thus, there is a trade-off between unavailability and the system response time. In section 5, we derive the expressions for unavailability and the increase in response time, as well as expected system degradation.

3 Mean Value Analysis of the System

In this section, we derive the relations between various parameters using their mean values. Such a *mean value analysis* is the standard approach to analyzing queueing systems where an exact solution is not possible [LZGS84]. Before we derive the relations, we need to introduce a few variables.

Let $\overline{R}_v$ be the mean time a site needs to vote on an update request. Without any site failures, this would be $\overline{R}_1 + \overline{R}_{def}$ where $\overline{R}_{def}$ is the mean time a transaction spends in the deferred list. Let p_{fs} the probability that a site fails while serving the queue Q_1 ($p_{fs} = 1 - \eta_1/\lambda_1$). Then

$$\overline{R}_v = p_{fs}(1/\beta) + (1 - p_{fs})\overline{R}_1 + \overline{R}_{def}. \tag{1}$$

Similarly, let $\overline{R}_{ca}$ be the mean time needed for a site to service a **accept**, **reject** or an **abort** issued by the initiator. Then it is easy to see that $\overline{R}_{ca} = \overline{R}_v - \overline{R}_{def}$, i.e.,

$$\overline{R}_{ca} = p_{fs}(1/\beta) + (1 - p_{fs})\overline{R}_1. \tag{2}$$

Let $\overline{R}_{req}$ be the mean response time for a transaction to be resolved, i.e., this is the average time from the instant the user submitted a transaction to a site to the instant that user is informed of the result. The submitted transaction spends, on an average $\overline{R}_2$ time in queue Q_2 by the time it is serviced. If it is not rejected outright (with probability $(1 - p_{rej})$), it is broadcast to the other sites. Let τ be the time-out period associated with any such broadcast, i.e., τ time after the broadcast, a time-out packet (TOP) is inserted in the queue Q_1. If the transaction is not resolved by the time the site gets to TOP (in time $\tau + \overline{R}_1$) then it is aborted.

Let p_{res} be the probability that the transaction is resolved before TOP is serviced by the site, and let $p_{ab} = 1 - p_{res}$ be the probability that it is not resolved, i.e.,

aborted. Finally, a transaction that is not voted *REJECT* by the initiator will also be resolved one way or the other and a **commit** or an **abort** will be sent to all the sites, including the initiator itself; this takes an additional $\overline{R}_{ca}$ time. Then $\overline{R}_{req}$ is estimated by

$$\overline{R}_{req} = \overline{R}_2 + (1 - p_{rej})[p_{res}(\overline{R}_1) + p_{ab}(\tau + \overline{R}_1) + \overline{R}_{ca}]. \tag{3}$$

To find out $\overline{R}_{def}$, recall that an update request is put in the deferred list if the request conflicts with another one in the pending list and the timestamp of the latter is smaller than that of the former. Also, every time a transaction is resolved, the initiator checks DL to see if any other request had blocked because of the transaction just resolved. It then chooses the transaction with the lowest timestamp for service and the others are deferred again with the probability p_{def}. Thus, the number of times a transaction is deferred is given by a geometric distribution with a mean of p_{def}, and hence, the mean number of times a transaction is deferred, $\overline{X}_{def}$, is given by

$$\overline{X}_{def} = \frac{p_{def}}{1 - p_{def}}. \tag{4}$$

Also note that the mean time between two deferrals of a transaction is precisely $\overline{R}_{req}$ since a transaction is resolved at an interval of mean length $\overline{R}_{req}$. Thus, $\overline{R}_{def}$ is given by

$$\overline{R}_{def} = \overline{X}_{def}\overline{R}_{req}. \tag{5}$$

To find out the probabilities of voting, first recall that we have made the assumption that a transaction is equally likely to access a particular data item. Thus since there are $|D|$ data items, the probability that an update accesses any data item is $1/|D|$. Let $\overline{N}_p$ be the number of update requests in PL. Since all these requests must be non-conflicting, they must access disjoint sets of data. Thus, the data items already accessed by the transactions in PL is $\overline{t}\overline{N}_p$, where $\overline{t}$ is the average size of a transaction. Then the mean probability that an incoming transaction conflicts with the pending requests is $\overline{t}^2\overline{N}_p/|D|$.

Let k_1 be the probability that the timestamp of the arriving update is more than the timestamps of the data items accessed, and k_2 be the probability that the timestamp of the arriving update is more than the timestamp of a conflicting pending request. Then we can write down the voting probabilities as:

$$p_{ok} = k_1(1 - \overline{t}^2\overline{N}_p/|D|), \tag{6a}$$

$$p_{rej} = (1-k_1)(1-\bar{t}^2\overline{N}_p/|D|), \quad (6b)$$
$$p_{def} = k_2\bar{t}^2\overline{N}_p/|D|, \quad (6c)$$
$$p_{pass} = (1-k_2)\bar{t}^2\overline{N}_p/|D|. \quad (6d)$$

In general, it is very hard to derive expressions for k_1 and k_2. Our approach is to leave them as parameters to the system and examine their effect on the system behavior by experimenting with their values; this work is currently being done, and their results will appear in the final version of the paper. To estimate $\overline{N}_p$, we note that on an average $\eta_2 S$ transactions arrive at any site (from itself, as well as other sites). With probability p_{ok}, they are voted *OK* and are put in *PL*. A transaction spends on an average $\overline{R}_{req}$ time in the *PL*, since this is the time needed to resolve a request. Thus by Little's Law, we can estimate $\overline{N}_p$ by

$$\overline{N}_p = p_{ok}\eta_2 S\overline{R}_{req} \quad (7)$$

It still remains to derive expressions for the arrival rate λ_1 for the queue Q_1, and the mean recovery rate β. The arrival rate λ_1 can be computed as the sum of the voting requests arriving from other sites and the votes and time-outs generated by requests that arrived at the site itself. Of the λ_2 requests that arrive at any site (per unit time), only η_2 are processed, the rest accounted for by site failures. Of these, only those that are not voted *REJECT* are sent out to other sites, and generate votes. Note that only p_c proportion of the voting requests get through. Thus

$$\lambda_1 = (S-1)\eta_2(1-p_{rej})2p_c + \eta_2(1-p_{rej})(\overline{X}+1) \quad (8)$$

where $\overline{X}$ is the expected number of responses a site gets back in response to its request; it is estimated as

$$\overline{X} = (S-1)pc^2\beta/(\gamma+\beta). \quad (9)$$

The first term in equation 8 is for the update and result notification from other sites; second term is for the traffic generated by the request arriving at the site. The factor of 2 in the first term above is for the voting request and for the notification of a request resolution; the extra term of 1 in the second term denotes the time-out generated for the request.

The recovery time $1/\beta$ consists of some fixed time $\overline{R}_f$ for the site to come back (assumed to be known from the system specification), the time to reconstruct the states of Q_1, PL, and DL, and the time to serve all the requests in Q_1. If $\overline{C}_1$ is the mean number of class-1 customers lost due to a failure, then assuming a linear cost of reconstruction

$$\frac{1}{\beta} = \overline{R}_f + d[\overline{C}_1 + \overline{N}_p + \overline{N}_d] + \frac{\overline{C}_1}{\mu_1}. \tag{10}$$

The part $\overline{N}_d$ of the deferred list that can be recreated is estimated by

$$\overline{N}_d = (S-1)\eta_2(1-p_{rej})p_{def}\overline{R}_{def} \tag{11}$$

4 Deriving Response Times and Throughputs

In the previous section we derived relations between the relevant parameters in terms of the known parameters, viz., λ_2, μ_1, μ_2, and γ, the unknown parameters, viz., $\lambda_1, \beta, \overline{R}_1, \overline{R}_2, \eta_1, \eta_2, \overline{C}_1$, and $\overline{C}_2$, and the input parameters, viz., τ, k_1 k_2. Note that since these relations are recursive, we can not hope to derive analytical expressions for the unknown parameters in terms of only the known ones. So our goal is to derive expressions for the response times and the throughputs, and solve the the system of relations iteratively, starting with arbitrary, but consistent values for the unknown parameters. Certain other parameters, such as the time-out period, the average size of a transaction, etc. will remain as input to the system of relations.

Thus, in this section we derive expressions for the mean response times $\overline{R}_1$, and $\overline{R}_2$, and the mean throughputs η_1 and η_2, for the two queues Q_1, and Q_2, respectively. If $\overline{C}_1$ and $\overline{C}_2$ are the mean number of customers for these two queues, then by Little's Law, the response times are given by

$$\overline{R}_i \triangleq \overline{C}_i/\eta_i, i = 1, 2. \tag{12}$$

Thus, our goal is to derive expressions for the the waiting times and the throughputs.

Let U be a random variable that takes the value 1 if the site is up, and 0 if the site is down. Then, it is easy to see that

$$Pr[U = i] = \begin{cases} \gamma/(\beta+\gamma) & i = 0 \\ \beta/(\beta+\gamma) & i = 1 \end{cases} \tag{13}$$

Let N_1 and N_2 be two random variables, representing the number of class-1 and class-2 customers in the system, respectively. These are the customers either waiting (in the queues, in the pending list or in the deferred list), or being served. Our goal is to find the joint probability distribution of N_1 and N_2. From this joint distribution, we shall, using standard transform techniques, derive the expressions for the mean number of customers. We only describe the salient steps of the derivation; details of the derivations are given in the Appendix.

We are interested in finding out the distribution of the number of customers in the system when the site is up or down. The former will help us find the mean waiting times for each class, as well as the number of customers **lost** due to a site failure. The latter will help us estimate the number of customers **ignored** because they happen to arrive while the site was recovering from a failure. Let $E[w^{N_1}, z^{N_2}]$ be the distribution of the number of customers in the system. Then, we are interested in finding $E[w^{N_1}, z^{N_2}|U = 1]$—the distribution of the customers when the site is up, and $E[w^{N_1}, z^{N_2}|U = 0]$—the distribution when the site is down. We give below the expressions for first of these quantities, the second one is given in the next section; their detailed derivations are shown in the Appendix.

Let $N(w, z)$ be the p.g.f. of N_1 and N_2, i.e., $N(w, z) \triangleq E[w^{N_1}, z^{N_2}|U = 1]$. Then, as shown in the Appendix,

$$N(w,z) = \frac{N_0(z)\Big[(\mu_1 - \mu_2)wz + \mu_2 w - \mu_1 z\Big] + \mu_2 w(z-1)p_{NC} + \frac{\gamma\beta wz}{\gamma+\beta}}{z\Big[w(\sigma - \mu_2 - \lambda_1 w - \lambda_2 z) - \mu_1\Big]}. \tag{14}$$

where

$$p_{NC} = Pr[N_1 = 0, N_2 = 0, U = 1]$$

is the probability that there are *no customers* in the system when the site is up,

$$N_0(z) = \sum_{j=0}^{\infty} Pr[N_1 = 0, N_2 = j, U = 1] z^j$$

is the p.g.f. of N_2 when there are no class-1 customers in the system while the site is up, and

$$\sigma = \lambda_1 + \lambda_2 + \mu_1 + \mu_2 + \gamma.$$

The throughputs for the two classes of customers are defined as the amount of time the site is busy serving each class:

$$\begin{aligned} \eta_1 &= \Big(Pr[U = 1] - Pr[N_1 = 0, U = 1]\Big)\mu_1 \\ \eta_2 &= \Big(Pr[N_1 = 0, U = 1] - Pr[N_1 = 0, N_2 = 0, U = 1]\Big)\mu_2. \end{aligned} \tag{15}$$

The expressions for $Pr[N_1 = 0, U = 1]$ and $Pr[N_1 = 0, N_2 = 0, U = 1]$ are derived in the Appendix. To find the expressions for the mean queue lengths we note (from standard transform theory) that

$$\overline{C}_1 \triangleq \frac{\partial}{\partial w} N(w,1)\bigg|_{w=1} \quad \text{and}$$
$$\overline{C}_2 \triangleq \frac{\partial}{\partial z} N(1,z)\bigg|_{z=1} . \tag{16}$$

Using the above definition, and simplification, we get

$$\overline{C}_1 = \frac{\beta\lambda_1}{\gamma(\gamma+\beta)} - \frac{\eta_1}{\gamma}. \tag{17a}$$
$$\overline{C}_2 = \frac{\beta\lambda_2}{\gamma(\gamma+\beta)} - \frac{\eta_2}{\gamma}. \tag{17b}$$

From equations 12, 15, 17a, and 17b we can now find the mean response times for each class.

Finally, note that the mean number of customers lost due to site failures is precisely the number of customers in the system. Thus, $\overline{C}_1$ customers of class-1 and $\overline{C}_2$ customers of class-2 are lost in every site failure. Note in particular that the number of transactions lost per failure is $\overline{C}_2$. $\overline{C}_1$ will be used to find β, the mean time for recovery.

5 System Performance

Our goal is to find the expected system degradation for the system described in section 2. We compute unavailability, denoted $\mathcal{U}_T$, for a given set of transactions T, by finding the expected number of transactions that are **lost** and **aborted** and **ignored**. We also compute the expected increase in response times, denoted $\mathcal{R}_T$, for a set of transactions T by computing the response times expressions with and without the assumptions of failures. Then the expected system degradation, $\mathcal{D}_T$, for T is given by

$$\mathcal{D}_T \triangleq c_1\mathcal{U}_T + c_2\mathcal{R}_T, \tag{18}$$

where c_1 and c_2 are non-negative constants adding up to 1. The choice of c_1 and c_2 gives the system designer the freedom to decide the relative importance of unavailability and response times for a particular application at hand.

The expected system degradation, $\mathcal{D}$, for the overall system can be computed as the weighted sum of the degradations for the different classes of transactions, i.e., for a system with local, non-local, and global transaction sets LT, NT, and GT, respectively, $\mathcal{D}$ is given by

$$\mathcal{D} \triangleq p_L\mathcal{D}_{LT} + p_{NL}\mathcal{D}_{NT} + p_G\mathcal{D}_{GT}, \tag{19}$$

where p_L, p_{NL}, and p_G are the proportions of local, non-local and global transactions respectively, in the set of transactions.

Recall that our goal is to study the expected system degradation, which is defined as the weighted sum of unavailability and the expected increase in the transaction response time. Below we outline how to find each of the two for local transactions. Note that we will find the two only for a single site. By our assumption of homogeneity, we can find out the degradation for the whole system. To find the degradation for a system with multiple types of transactions, we find the degradation for each type separately and then compute the weighted sum.

5.1 Unavailability

Unavailability at a given site is the proportion of the number of transactions lost, **aborted** and **ignored** out of the total number of transactions presented to the site over a (sufficiently long) period of time. Consider a period of time T for which we wish to compute unavailability. On an average, a site would receive $\lambda_2 T$ transactions in that time. Let L, A, and I denote the number of transactions lost, **aborted**, and **ignored** over T respectively. Then, the unavailability is given by

$$\mathcal{U} = \frac{L + A + I}{\lambda_2 T}. \tag{20}$$

From the previous section, we know the number of transactions lost per failure, viz. $\overline{C}_2$. We can also find I_f, the mean number of transactions **ignored** per failure, by computing the distribution of the arrivals of customers during failure recovery. As shown in the Appendix, this distribution is given by

$$E[w^{N_1}, z^{N_2} | U = 0] = \frac{\beta(1 - R^*(\lambda_1(1-w) + \lambda_2(1-z)))}{\lambda_1(1-w) + \lambda_2(1-z)}, \tag{21}$$

where $R^*(s) = \beta(\beta + s)$ is the Laplace Transform of the recovery time distribution. Thus, its mean is given by

$$\begin{aligned} I_f &\triangleq \frac{\partial}{\partial z} E[w^{N_1}, z^{N_2} | U = 0]\Big|_{w=1,z=1} \\ &= \frac{\lambda_2}{\beta}. \end{aligned} \tag{22}$$

Let f be the number of failures that occur over the time period T. Then it is easily seen that

$$f = \frac{\gamma\beta T}{\gamma + \beta}. \tag{23}$$

Thus, $L = \overline{C}_2 f$ and $I = I_f f$. Finally, to compute A, we find the probability p_{ab} of a transaction being **aborted** due to lack of sufficient responses. Then $A = p_{ab}(1 - p_{rej})\lambda_2 T$.

5.2 Increase in Response Time

To find the expected increase in the response time due to failures, we consider a similar system of stochastic processes without any failures. The average transaction response time in such a system would be given by[5]

$$\overline{R}^c_{req} = \overline{R}^c_2 + (1 - p_{rej})\left[2\overline{R}_c + \overline{R}^c_v + S\overline{R}^c_1\right], \tag{24}$$

where $\overline{R}^c_1$ and $\overline{R}^c_2$ are the mean response times for queues Q_1 and Q_2 respectively in this modified system, and $\overline{R}^c_v = \overline{R}^c_1 + \overline{R}_{def}$ is the mean time required to vote on an update request. To find $\overline{R}^c_{req}$ we can solve a similar stochastic model, without failures, and can show that the distribution of the number of customers is given by

$$N^c(w, z) = \frac{N^c_0(z)\left[(\mu_1 - \mu_2)wz + \mu_2 w - \mu_1 z\right] + \mu_2 w(z-1)p_{NC}}{z\left[w(\sigma' - \mu_2 - \lambda_1 w - \lambda_2 z) - \mu_1\right]} \tag{25}$$

where

$$p_{NC} = Pr[N_1 = 0, N_2 = 0]$$

[5]A superscript "c" is attached to the relevant parameters of the failure-free system.

is the probability that there are no customers in the system,

$$N_0^c(z) = \sum_{j=0}^{\infty} Pr[N_1 = 0, N_2 = j] z^j$$

is the p.g.f. of N_2 when there are no class-1 customers in the system, and

$$\sigma' = \lambda_1 + \lambda_2 + \mu_1 + \mu_2.$$

(The proof parallels the one given in the Appendix for a system with failures; it includes solving another bulk-arrival system for class-2 customers; the details are left to the reader.)

The mean queue lengths for the queueing system without failures is defined by

$$\overline{C}_1^c \triangleq \left.\frac{\partial}{\partial w} N^c(w,1)\right|_{w=1} \quad \text{and}$$

$$\overline{C}_2^c \triangleq \left.\frac{\partial}{\partial z} N^c(1,z)\right|_{z=1}. \tag{26}$$

Using the above definition, we get

$$\overline{C}_1^c = \frac{\mu_1 \lambda_1 Pr[N_1 = 0]}{(\mu_1 - \lambda_1)^2} \tag{27a}$$

$$\overline{C}_2^c = \frac{\mu_2 \frac{\partial}{\partial z} N_0^c(z)(1) - \eta_2}{\lambda_2}. \tag{27b}$$

Note that if $\rho_1 = \lambda_1/\mu_1$ is the server utilization for class-1 customers then, $\overline{C}_1^c = \rho/(1-\rho)$, as one would expect from a classical M/M/1 queue behavior (and this is true for class-1 customers since they do not observe any class-2 customers).

From this, as before, we can derive the mean response times $\overline{R}_1^c$ and $\overline{R}_2^c$, for the queues Q_1 and Q_2 respectively. Then the expected increase in response time is given by the ratio $\overline{R}_{req}/\overline{R}_{req}^c$.

6 Conclusions and Future Work

We have presented a very general stochastic model for evaluating the performance of distributed databases. The model is parametrized and supports very general assumptions of data replication, of transaction access patterns, and of underlying connectivity. While we have chosen a concurrency control protocol for specific system performance evaluation, we are currently exploring how to parametrize this aspect of the analysis as well. We believe that the group paradigm for concurrency control [ET89] will be a good candidate for this step.

Using standard mean value analysis of a separable queueing network, we showed how to evaluate various system parameters. From this, we showed how to evaluate availability of data and the average response times for transactions, observed over a sufficiently long period of time. We are currently experimenting with the parameters to see how they interact. The effect of the choice of input parameters, such as the time-out period, as well as the constants in the probabilities of voting is also being studied. The details appear in the full version of the paper.

We also gave a new measure, expected system degradation, that captures the interplay between availability and response times for such fault-tolerant database systems. We believe that this is the first step towards measuring real-time availability—availability of data for transactions in a real-time setting. We are exploring the definition of real-time availability further.

Finally, we would like to extend the solution to a database system that has transactions with priorities. Of course, this would entail solving a queueing model with several levels of priority. We would like to explore if the standard methods for dealing with priority queueing systems [Jai68] apply here.

Acknowledgement

We would like to thank Keith Marzullo, Pat Stephenson, and Satish Tripathi for their helpful comments on this work.

References

[BM76] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications.* North Holland, 1976.

[COK86] Brian A. Coan, Brian M. Oki, and Elliott K. Kolodner. Limitations on database availability when networks partition. In *Proceedings of the Fifth ACM Symposium on Principles of Distributed Computing*, pages 187–194, Calgary, Alberta, August 1986. ACM SIGOPS-SIGACT.

[ET89] Amr El Abbadi and Sam Toueg. Maintaining availability in partitioned replicated databases. *ACM Transactions on Database Systems*, 14(2), June 1989.

[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*. Springer-Verlag, 1978. Also appears as IBM Technical report RJ2188.

[GT89] Dipak Ghosal and Satish Tripathi. Analysis of a distributed database system with site failures. Technical Report UMIACS–TR–89–70, Institute for Advanced Computer Studies, University of Maryland, 1989.

[Jai68] N. K. Jaiswal. *Priority Queues.* Academic Press, 1968.

[KGM87] Boris Kogan and Hector Garcia-Molina. Update propogation in Bakunin data networks. In *Proceedings of the Sixth ACM Symposium on Principles of Distributed Computing*, pages 13–26, Vancouver, British Columbia, August 1987. ACM SIGOPS-SIGACT.

[Kle75] Leonard Kleinrock. *Queueing Theory*, volume I: Theory. John Wiley and Sons, First edition, 1975.

[LZGS84] Edward Lazowska, John Zahorjan, G. Scott Graham, and Kenneth L. Sevcik. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models.* Prentice Hall, 1984.

[MS90] Keith Marzullo and Amitabh Shah. Harary Networks: Connectivity for highly available real-time distributed databases. Technical Report TR 90-1070, Department of Computer Science, Cornell University, January 1990.

[SM89] Amitabh Shah and Keith Marzullo. Trade-offs between replication and availability in distributed databases. Technical Report TR 89-1065, Department of Computer Science, Cornell University, 1989.

[Tho79] Robert H. Thomas. A majority consensus approach to concurrency control for multiple copy database. *ACM Transactions on Database Systems*, 4(2):180–209, June 1979.

[Tri82] Kishor S. Trivedi. *Probability & Statistics with Reliability, Queueing, and Computer Science Applications.* Prentice Hall, 1982.

[TT89] Don Towsley and Satish Tripathi. A single server priority queue with server failures and queue flushing. Technical Report UMIACS–TR–89–25, Institute for Advanced Computer Studies, University of Maryland, 1989.

Appendix

In this section, we prove several of the results used in section 4. The derivations are based on the results in [GT89,TT89].

Proof of equation 14: To derive the expression for $N(w,z)$, we consider a continuous time Markov process whose state at any given time is given by the triple (N_1, N_2, U), where N_1 (N_2) is the number of class-1 (class-2) customers in the system, and U denotes whether the site is up or down, i.e., $U = 1$ if the site is up, and 0 if the site is down.

Let $p_{i,j} = Pr[N_1 = i, N_2 = j, U = 1], i,j = 0,1,\ldots$ and $p_d = Pr[U = 0]$—the probability that the site is down. (Note that $p_d + \sum_i \sum_j p_{i,j} = 1$.) When the system is ergodic, the stationary probabilities satisfy the following (obtained from the flow conservation equations for the system shown in Figure 4):

$$p_d = \frac{1}{\beta}\Big[\gamma(1-p_d)\Big] \tag{28a}$$

$$p_{0,0} = \frac{1}{\sigma}\Big[\mu_1(p_{0,0}+p_{1,0}) + \mu_2(p_{0,0}+p_{0,1}) + \beta p_d\Big] \tag{28b}$$

$$p_{0,j} = \frac{1}{\sigma}\Big[\lambda_2 p_{0,j-1} + \mu_1(p_{0,j}+p_{1,j}) + \mu_2 p_{0,j+1}\Big];\ \ j \geq 1 \tag{28c}$$

$$p_{i,j} = \frac{1}{\sigma}\Big[\lambda_1 p_{i-1,j} + \lambda_2 p_{i,j-1} + \mu_1 p_{i+1,j} + \mu_2 p_{i,j}\Big];\ \ i > 0, j \geq 0 \tag{28d}$$

where $\sigma = \lambda_1 + \lambda_2 + \mu_1 + \mu_2 + \gamma$. Recall that we have assumed that on recovery, the site enters the state (0, 0, 1). From the above, it is clear that $p_d = \gamma/(\gamma+\beta)$ as expected from equation 13. To derive the distribution of N_1, and N_2, we use standard *z-transform* Theory; let

$$N_i(z) \triangleq \sum_{j=0}^{\infty} p_{i,j} z^j \text{ and} \tag{29}$$

$$N(w,z) \triangleq \sum_{i=0}^{\infty} N_i(z) w^i \tag{30}$$

be two generating functions. Then, using simple algebra, we get

$$\begin{aligned} N_0(z)\Big[(\sigma-\mu_1)z - \mu_2 - \lambda_2 z^2\Big] &= \mu_2(z-1)p_{0,0} + \mu_1 z N_1(z) + \beta z p_d \\ N_i(z)\Big[\sigma - \mu_2 - \lambda_2 z\Big] &= \lambda_1 N_{i-1}(z) + \mu_1 N_{i+1}(z);\ \ i > 0. \end{aligned} \tag{31}$$

Using equations 30 and 31, we get

$$N(w,z) = \frac{N_0(z)\Big[(\mu_1-\mu_2)wz + \mu_2 w - \mu_1 z\Big] + \mu_2 w(z-1)p_{0,0} + \beta w z p_d}{z\Big[w(\sigma - \mu_2 - \lambda_1 w - \lambda_2 z) - \mu_1\Big]}. \tag{32}$$

from which equation 14 follows. (Note that $p_{0,0} = p_{NC}$.) Check that $N(1,1) = \beta/(\gamma+\beta) = Pr[U=1]$ as expected, since $N(w,z)$ is the distribution of the customers in the system when the site is up.

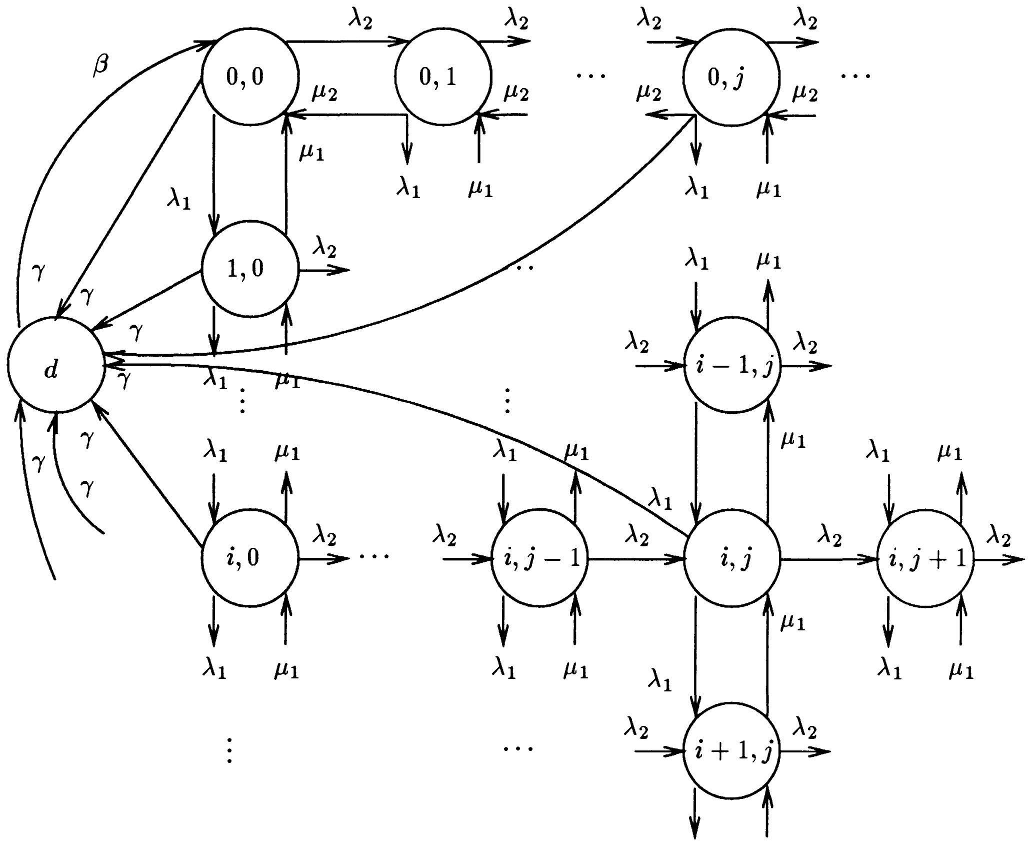

Figure 4: State Transitions at a site

Next, we show how to derive the expressions for $N_0(z)$ and p_{NC}. Now $N_0(z)$ can also be written as

$$N_0(z) = E[z^{N_2}|N_1 = 0, U = 1]Pr[N_1 = 0, U = 1]. \tag{33}$$

Note that $Pr[N_1 = 0, U = 1] = N_0(1)$. Thus, to compute $Pr[N_1 = 0, U = 1]$, we look at the expression $N(w, 1)$. Since this function is analytic in the region $w \in [0, 1)$, its numerator would be zero whenever the denominator is zero in that region. The latter occurs when w^* is the root of the quadratic equation $\lambda_1 w^2 - (\lambda_1 + \gamma + \mu_1)w + \mu_1 = 0$ in the region [0, 1), i.e.,

$$w^* = \frac{(\lambda_1 + \gamma + \mu_1) - \sqrt{(\lambda_1 + \gamma + \mu_1)^2 - 4\lambda_1\mu_1}}{2\lambda_1}. \tag{34}$$

This gives us, from equating the numerator to zero,

$$Pr[N_1 = 0, U = 1] = \frac{\gamma w^* \beta}{(1 - w^*)\mu_1(\gamma + \beta)}. \tag{35}$$

To compute $E[z^{N_2}|N_1 = 0, U = 1]$, we consider a modified queueing system consisting of only the class-2 customers. This is done by treating the arrivals of class-2 customers in the original system, when some class-1 customer was being served, as *bulk* arrival of class-2 customers in the modified system. Note that a site can be busy in two ways when it is up: it could be serving either a class-1 customer (called a class-1 busy period) or serving a class-2 customer (class-2 busy period).

To get the modified queueing system, consider shrinking all the class-1 busy periods in the original system to a point that corresponds to the time the class-1 busy period ends. All the class-2 customers that arrived during the class-1 busy period in the original system are now considered to arrive in bulk at that point. (See figure 5.)

So consider this $M^{[G]}/M/1$ queue with with the arrival bulk size G. Let $G(z)$ be the distribution of the bulk size, i.e.,

$$G(z) = \sum_{j=1}^{\infty} Pr[G = j]z^j$$

with mean $G'(1)$. We will denote the parameters associated with this system as λ', μ', γ', and so on.

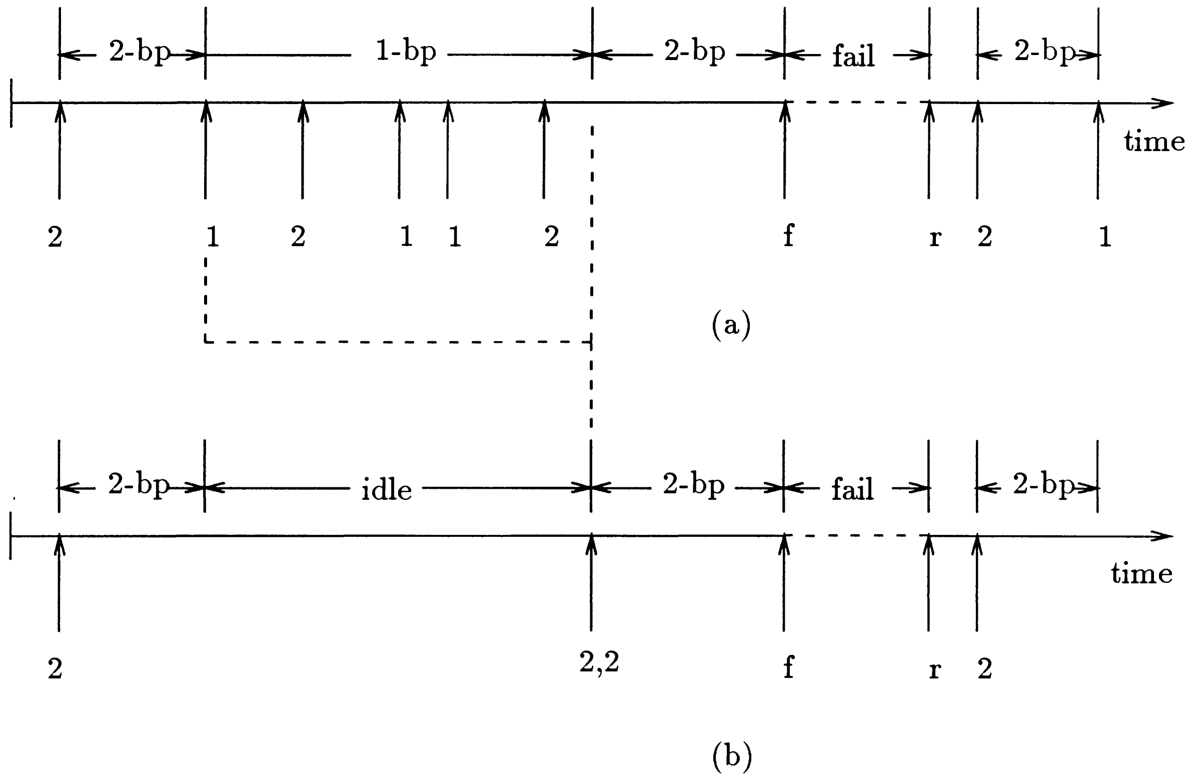

Figure 5: Modified Queueing System for Class-2 Customers

To derive the distribution $N^c(z) \triangleq E[z^{N_2}|N_1 = 0, U = 1]$ of (class-2) customers in this system, we follow the steps for solving a standard bulk-arrival system [Kle75] except that now we add the possibility of a site failure in any state, and the recovery leading to the state 0, of no customers. Then we get the following:

$$N^c(z) = \frac{\mu' Pr[N_2 = 0, U = 1|N_1 = 0](z-1) + \frac{\gamma'\beta' z}{\gamma'+\beta'}}{\left[\lambda'(1 - G(z)) + \mu' + \gamma'\right]z - \mu'}. \tag{36}$$

Again, using the analytic properties of $N^c(z)$, we get that

$$Pr[N_2 = 0, U = 1|N_1 = 0] = \frac{\gamma'\beta' z^*}{\mu'(\gamma' + \beta')(1 - z^*)}, \tag{37}$$

where z^* is the root of the equation

$$\left[(\lambda'(1 - G(z)) + \mu' + \gamma'\right]z - \mu' = 0 \tag{38}$$

in the region [0, 1). Check again that $N'(1) = \beta'/(\gamma' + \beta')$, the probability of the site being up in this modified system. It remains to derive the expressions for the parameters of this system, as well as an expression for $G(z)$.

It is easy to see that the arrival rate, the service rate, and the failure rate of this modified system remain the same as the original system, i.e., $\lambda' = \lambda_2, \mu' = \mu_2$, and $\gamma' = \gamma$. To derive $G(z)$ and β', we look at the class-1 busy periods (C1BP's) in the original system. It can be seen ([Kle75],§5.8) that the Laplace Transform $B^*(s)$ of the busy period $b(t)$ is given by

$$B^*(s) \;=\; \frac{(\lambda_1 + \mu_1 + s) - \sqrt{(\lambda_1 + \mu_1 + s)^2 - 4\lambda_1\mu_1}}{2\lambda_1}. \tag{39}$$

Now, in general, for the class-2 customers, the mean recovery period will remain the same or get longer depending on whether the C1BP immediately after a recovery completed without any failure or not, i.e.,

$$\begin{aligned} \frac{1}{\beta'} &\triangleq \frac{1}{\beta}p' + \left(\frac{1}{\beta} + \overline{X}_{BR}\right)(1 - p') \\ &= \frac{1}{\beta} + \overline{X}_{BR}(1 - p'), \end{aligned} \tag{40}$$

where $\overline{X}_{BR}$ is the mean *backward recurrence time for recovery* and p' is the probability that a random C1BP completes before a failure. It is easy to see that $p' = B^*(\gamma)$: let x_f and x_b be two random variables denoting the inter-failure time for the site and the length of C1BP, respectively. Then,

$$\begin{aligned} p' &\triangleq Pr[x_f > x_b] \\ &= 1 - Pr[x_f \leq x_b] \\ &= 1 - \int_0^\infty Pr[x_f < t | x_b = t] f_{x_b}(t)dt \\ &= B^*(\gamma) \text{ (by definition of a Laplace Transform).} \end{aligned}$$

Using standard transform techniques [Kle75,Tri82], it is easy to see that the mean backward recurrence time for recovery, $\overline{X}_{BR}$ is given by

$$\overline{X}_{BR} = \frac{\overline{BP^2}}{2\overline{BP}}, \tag{41}$$

where $\overline{BP}$ is the mean of the distribution of C1BP's and $\overline{BP^2}$ is its second moment; both can be obtained by taking the appropriate partial derivatives of $B^*(s)$.

To find the p.g.f. of the bulk size, note that $G(z)$ can also be written as

$$G(z) = {g'}_1 z + \sum_{i=0}^{\infty} {g''}_i z$$

, where ${g'}_1$ is the probability that a class-2 arrival occurred (singly) when the site was idle, and ${g''}_i$ is the probability that a class-2 bulk arrival (of size i) occurred during a class-1 busy period. Let

$$\overline{FP} = \frac{1}{\lambda_1 \frac{\beta}{\gamma+\beta} B^*(\gamma)}$$

be the mean time between two complete C1BP's. Then ${g'}_1 = \overline{FP}/(\overline{BP} + \overline{FP})$ is the required probability that class-2 arrival did not occur during a C1BP. Let $p = 1 - {g'}_1 = \overline{BP}/(\overline{BP} + \overline{FP})$ be the probability that the arrival occurs during a C1BP. Now

$$\begin{aligned} {g''}_i &= Pr[\text{bulk arrival during C1BP}]\ Pr[\text{bulk size} = i] \\ &= p \int_0^{\infty} \frac{e^{-\lambda_2 t}(\lambda_2 t)^i}{i!} f_b(t)dt \end{aligned}$$

Thus,

$$\begin{aligned} G(z) &= {g'}_1 z + \sum_{i=0}^{\infty} p \int_0^{\infty} \frac{e^{-\lambda_2 t}(\lambda_2 t)^i}{i!} f_b(t) z^i dt \\ &= {g'}_1 z + (1 - {g'}_1) B^*(\lambda_2(1-z)). \end{aligned} \tag{42}$$

Note that $G(1) = 1$, as expected.

Finally, we are left with deriving p_{NC} which is given by

$$\begin{aligned} p_{NC} &= Pr[N_1 = 0, N_2 = 0, U = 1] \\ &= Pr[N_2 = 0 | N_1 = 0, U = 1] Pr[N_1 = 0, U = 1] \\ &= N'(0) Pr[N_1 = 0, U = 1] \\ &= \frac{(\gamma' + \beta')}{\beta'} Pr[N_2 = 0, U = 1 | N_1 = 0] Pr[N_1 = 0, U = 1] \\ &= \frac{\gamma' \beta' z^*}{\mu'(\gamma' + \beta')(1 - z^*)} \frac{\gamma w^* \beta}{(1 - w^*)\mu_1(\gamma + \beta)}. \end{aligned} \tag{43}$$

which concludes the derivation of $N(w, z)$. ■

Proof of Equation 21: Again, if r is the random variable denoting the recovery period, then from *Renewal Theory* [Kle75,Tri82], it can be shown that

$$f_x(t) = \frac{1 - F_r(t)}{E[r]} \tag{44}$$

where x is a random variable denoting an arbitrary point in the recovery period, and $E[r] = 1/\beta$ (Figure 6).

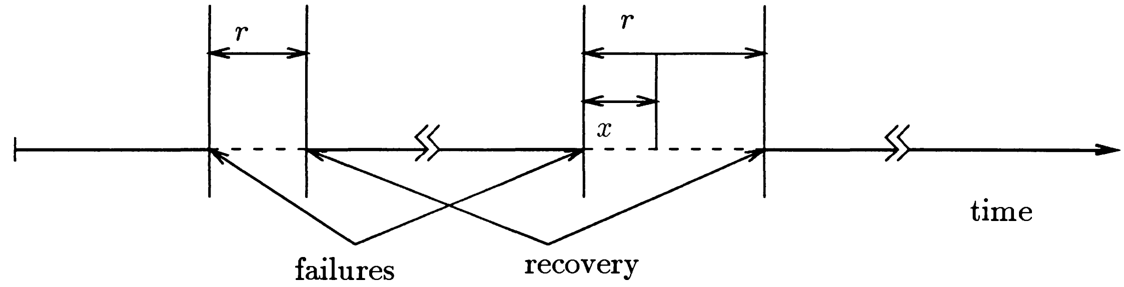

Figure 6: Recovery of a site

Now, we have assumed that the independence of the arrivals in Q_1 and Q_2. Thus,

$$\begin{aligned} Pr[N_1 = i, N_2 = j | x = t, U = 0] &= Pr[N_1 = i] Pr[N_2 = j] \\ &= \Big(\frac{e^{-\lambda_1 t}(\lambda_1 t)^i}{i!}\Big)\Big(\frac{e^{-\lambda_2 t}(\lambda_2 t)^j}{j!}\Big) \end{aligned}$$

Thus,

$$\begin{aligned} E[w^i, z^j | x = t, U = 0] &= \sum_{i=0}^{\infty}\sum_{j=0}^{\infty}\Big(\frac{e^{-\lambda_1 t}(\lambda_1 t)^i}{i!}\Big)\Big(\frac{e^{-\lambda_2 t}(\lambda_2 t)^j}{j!}\Big) w^i z^j \\ &= e^{-\lambda_1 t} e^{-\lambda_2 t} e^{\lambda_1 t w} e^{\lambda_2 t z} \\ &= e^{-[\lambda_1(1 - w) + \lambda_2(1 - z)]t} \end{aligned}$$

Hence, by the Law of Total Expectation,

$$E[w^{N_1}, z^{N_2} | U = 0] = \int_0^{\infty} e^{-[\lambda_1(1 - w) + \lambda_2(1 - z)]t} f_x(t) dt$$

$$
\begin{aligned}
&= \int_0^\infty e^{-[\lambda_1(1-w)+\lambda_2(1-z)]t}\beta(1-F_r(t))dt \\
&= \beta\Big(\int_0^\infty e^{-[\lambda_1(1-w)+\lambda_2(1-z)]t}dt \\
&\quad - \int_0^\infty e^{-[\lambda_1(1-w)+\lambda_2(1-z)]t}F_r(t)dt\Big) \\
&= \beta\Big(\frac{1-R^*(\lambda_1(1-w)+\lambda_2(1-z))}{\lambda_1(1-w)+\lambda_2(1-z)}\Big)
\end{aligned}
$$

which proves equation 21. ■