Analysis of Replicated Data with Repair Dependency

ING-RAY CHEN AND DING-CHAU WANG

Institute of Information Engineering, National Cheng Kung University, Tainan, Taiwan
Email: irchen@iie.ncku.edu.tw

Pessimistic control algorithms for replicated data permit only one partition to perform update operations at any time so as to ensure mutual exclusion of the replicated data object. Existing availability modelling and analyses of pessimistic control algorithms for replicated data management are constrained to either site-failure-only or link-failure-only models, but not both, because of the large state space which needs to be considered. Moreover, the assumption of having an independent repairman for each link and each site has been made to reduce the complexity of analysis. In this paper, we remove these restrictions with the help of stochastic Petri nets. In addition to including both site and link failures/repairs events in our analysis, we investigate the effect of repair dependency which occurs when many sites and links may have to share the same repairman due to repair constraints. Four repairman models are examined in the paper: (a) independent repairman with one repairman assigned to each link and each node; (b) dependent repairman with FIFO servicing discipline; (c) dependent repairman with linear-order servicing discipline; and (d) dependent repairman with best-first servicing discipline. Using dynamic voting as a case study, we compare and contrast the resulting availabilities due to the use of these four different repairman models and give a physical interpretation of the differences. We show that ignoring concurrent site and link failures/repairs events or repair dependency can very unrealistically overestimate the availability of replicated data.

Received May 15, 1996; revised February 15, 1997

1. INTRODUCTION

Pessimistic control algorithms for replicated data permit only one partition to perform update operations at any given time so as to ensure mutual exclusion of the replicated data object. Over the past few years, various pessimistic algorithms based on voting (e.g. [1, 2]) and quorum consensus (e.g. [3, 4, 5, 6]) have been proposed in the literature and many recent works discuss how to distribute replicated data to improve the performance of the system [7, 8]. These past works mainly emphasized the algorithmic aspects and correctness proofs of their approaches with little or inadequate availability modelling and analyses. To date, the availability analysis of these pessimistic algorithms is constrained to either site or link models, but not both, possibly because of the large state space that needs to be considered. Moreover, an independent repairman associated with each link or site has always been assumed to reduce the complexity of analysis. Ironically, although pessimistic algorithms were designed to deal with network partitioning problems, these past works were content with availability analyses based on site-failure-only system models (e.g. [9]). The only exception is the work by Jajodia and Mutchler [2] who have considered the site and link models separately, but not altogether in one model.

This paper removes these inadequate modelling restrictions with the help of stochastic Petri nets. In particular, we examine the effect of repair dependency where all sites and/or links may share the same repairman due to repair resource constraints. Using dynamic voting as a case study, we compare and contrast the resulting availabilities under different repairman models. Our objective is to provide a more informative, realistic estimation of the availability metric in the presence of both site/link failures and repair dependencies. Although we have chosen to test the modelling techniques developed in the paper with dynamic voting, it will become clear later that these techniques can be generally applicable to other pessimistic algorithms. We pick dynamic voting as a case study simply because analytical results under site- or link-failures/repairs only models (but not both together) are available in [2], against which we can validate our Petri net models at the boundary conditions. We extend our previous work [10] in this paper by investigating various strategies for repairing failed sites and links and analysing the effect of repair dependency on data availability.

The rest of the paper is organized as follows. Section 2 gives an overview of dynamic voting; states the assumptions and notation used; and develops a stochastic Petri net (SPN) model for dynamic voting that considers both site and link failures/repairs for the case when each site or link has its own independent repairman. Section 3 considers three dependent repairman models in which all sites/links share the same repairman: (a) First In First Out (FIFO), (b) linear-order and (c) best-first. In the linear-order repairman model, we always repair failed sites or links in a prespecified order, whereas in the best-first repairman model, we intelligently select a failed site or link whose repair will result in the largest improvement in availability. Section 4 compares the availabilities obtained under various repairman models for a 5-site ring topology and provides a physical interpretation. Section 5 summarizes the paper and outlines some future research areas.

2. SYSTEM MODEL AND ASSUMPTIONS

We illustrate the key modelling techniques developed in this paper by an example pessimistic control algorithm called dynamic voting developed by Jajodia and Mutchler [2]. A detailed description of dynamic voting can be found in [11, 12, 2]. Here, we only give a background overview of the algorithm.

2.1. Background

It is instructive to view voting as a form of quorum consensus which must satisfy the quorum intersection property, i.e. let R and W be the cardinalities of any two read and write quorums on the same data item, respectively, then R+W as well as $2W$ must be greater than the total number of copies to ensure mutual exclusion. Taking this view, dynamic voting is different from static voting in that its quorum set can be dynamically adjusted in reaction to system state changes, as opposed to a fixed quorum set as in the static voting scheme. The principle concept of dynamic voting is that instead of using a static quorum set based on the original set of copies, it uses the current up to date copies at any time for deriving its dynamic quorum set. This way, the system can dynamically adjust its quorum set in response to state changes and it results in an increase in availability when compared with static voting because of a higher probability of finding a quorum to serve an operation.

Suppose that a data item (e.g. a file) f is replicated to n copies, stored at sites $S_1, S_2, \ldots, S_n$. The essence of dynamic voting is that it must keep track of the number of up to date copies involved in the last update, and also which copies are up to date. To achieve this, consider a version of the dynamic voting algorithm where each copy is associated with three variables, VN, SC and DS, such that VN stores the version number of the local copy and can serve as an indicator to tell whether the local copy is current; SC records the number of current copies that participated in the last update of f; and DS stores the ID of the highest linearly ordered site among all sites that presently store current copies. A copy bearing the highest version number is called a current copy. Let (VN_i, SC_i, DS_i) be the set associated with the copy stored in site S_i. Initially, all copies are current and $VN_i = 0$, $SC_i = n$, $DS_i = n$ for all S_is, where DS_i is initialized to n because S_n is the highest linearly ordered site among all initially. When receiving an update operation, a site C (called the coordinator) requests all sites that have a copy of f to send their values of (VN_i, SC_i, DS_i). Let $\mathcal{P}$ denote the set consisting of the coordinator C and all subordinates that responded to the request; each site in set $\mathcal{P}$ locks its copy of the data item f during the process. Site C then inspects the largest version number, VN_p, found in set $\mathcal{P}$. Let $\mathcal{I}$ denote the subset consisting of only those sites containing a copy with version number VN_p. Let SC_p and DS_p be the values stored in SC_i and DS_i of any site in set $\mathcal{I}$, respectively. If one of the following two conditions is true, then site C is in the major partition: (a) the cardinality of $\mathcal{I}$ is larger than one half of SC_p; (b) the cardinality of $\mathcal{I}$ is exactly equal to one half of SC_p and set $\mathcal{I}$ contains site DS_p. If neither condition is true, site C aborts the update, sends ABORT messages to subordinates and releases the lock on its local copy. The update operation can retry at a later time. Otherwise, site C is in the major partition. It thus commits the update locally, sends COMMIT messages to subordinates along with the new update and new value of (VN, SC, DS) and releases the lock on its local copy. The new value of (VN, SC, DS) is set to $(VN_p + 1$, the cardinality of set $\mathcal{P}$, the ID of the highest linearly ordered site found in $\mathcal{P})$. A termination protocol can be used to terminate correctly the execution of the update process if it is interrupted by failures. A possible terminal protocol can be found in [2] and is not described here. To simplify our presentation, this paper only considers update operations, although the SPN models developed in the paper can easily be applied to handle read operations.

2.2. Assumptions and availability metric

We first state the system assumptions. We assume that there are n sites connected by a topology to be specified. Each site is assigned a single vote and a unique site ID, numbered $1, 2, \ldots, n$ where n is the total number of sites in the system. We use the subscript i to refer to the site ID of site i. All sites in the system are linearly ordered in descending order of their site IDs. That is, site n is the highest linearly ordered site. We assume independent failure modes for sites and links, with λ_s and λ_l being the failure rates of sites and links, respectively. A repairman repairs a failed site with rate μ_s and a failed link with rate μ_l. For the case when many failed sites and links shared the same repairman, the repairman can only fix one failed entity at a time. All times between these events are assumed to be exponentially distributed. Other than the independent repairman model in which each site or link has its own designated repairman, we consider three dependent repairman models in this paper: (a) a shared repairman with FIFO servicing discipline, i.e. all sites and links share the same repairman who repairs failed sites or links in a FIFO order; (b) a shared repairman with a fixed servicing discipline, i.e. the shared repairman repairs failed sites or links in accordance with a fixed linear order; (c) a shared repairman with a best-first servicing discipline, i.e. all sites and links share the same repairman who will pick a failed site or link whose repair afterward will result in the largest availability improvement to the system. We assume frequent updates, i.e. there is an update between two consecutive failure or repair events. This assumption is justified for any data item that is being used frequently enough to justify data replication.

As for the availability evaluation of dynamic algorithms, we consider both the system and site availability metrics. The site availability is defined as the probability that an update arriving at an arbitrary site will succeed [2, 6], while the traditional system availability metric is defined as the probability that a major partition exists. The site availability metric normally has a smaller value than the system availability metric because for an arriving operation to succeed at a site not only must a major partition exist but also the site must be a member of the major partition and be in the state of 'up'.

2.3. Stochastic Petri net for the independent repairman model

2.3.1. Site subnet

Figure 1 shows an SPN subset that describes the behaviour of a particular site (site i) upon an update action. Site i may or may not be in the major partition when the update occurs. A site can be in one of four states, namely, 'up and current', 'up and out-of-date', 'down and current' or 'down and out-of-date'. We use four places, $upcc_i$, $upoc_i$, $downcc_i$, and $downoc_i$, to represent these four states. Since a site can only be in exactly one of these four states, we allow only one of these four places to contain a non-zero number of tokens, thus identifying the state of site i.

Initially, site i is in the state of up and current. However, instead of putting only one token into place $upcc_i$ we deliberately put i tokens initially into place $upcc_i$. Therefore, the number of tokens used in the subnet for site 1 will be one, and will be two for site 2, and so on. The reason for using i tokens specifically for site i is to easily identify the ID of site i. This is a useful modelling technique to keep track of the site ID of the distinguished site after a state change, i.e. the site ID of the distinguished site after a state change can easily be remembered by copying the number of tokens used in the subnet for the resulting distinguished site to a place holder called ds in the system subnet (to be described later in the next subsection). As the state of site i changes in response to update actions, the i tokens in the subnet will be moved in one unit from one place to another among the four places. For example, suppose that site i initially is in the state of up and current, i.e. $\#(upcc_i) = i$, $\#(upoc_i) = 0$, $\#(downcc_i) = 0$ and $\#(downoc_i) = 0$. Now suppose an update operation occurs and site i is found not in the major partition (due to network partitioning), then i tokens will flow from place $upcc_i$ to place $upoc_i$, resulting in $\#(upcc_i) = 0$, $\#(upoc_i) = i$, $\#(downcc_i) = 0$ and $\#(downoc_i) = 0$, which means that its new state becomes up and out of date since it did not participate in the last update.

Note that in Figure 1 we use the tuple (transition-name, priority-level, enabling function) to label a transition. For example, the transition from place $upcc_i$ to $upoc_i$ is labelled by $(t5_i, 3, \bar{g}_i)$ in which $t5_i$ is the name of the transition, 3 is the priority of transition $t5_i$ relative to other transitions in the net, and $\bar{g}_i()$ is the associated *enabling function* which returns TRUE if site i is found not in the major partition. In our SPN, a transition is enabled when (a) each of the transition's input places contains a number of tokens greater than or equal to the *multiplicity* of the associated input arc; and (b) the associated enabling function (if specified) returns TRUE. If multiple transitions are enabled at the same time, the transition with the highest priority level will fire first. Also, when a transition is fired, one or more tokens, depending on the multiplicity of the associated input arc, will be removed from the input place, and one or more tokens, depending on the multiplicity of the associated output arc, will be added to each output place. In Figure 1, only one transition can be enabled at a time and all transitions are of the same priority level. We use the label '*#i*' to indicate that the multiplicity of some arcs in Figure 1 is i instead of the default value of 1. As an example of these rules, consider the case when place $ready_i$ contains one token, place $upcc_i$ contains i tokens, and the enabling function $\bar{g}_i()$ returns TRUE. In this case transition $t5_i$ will be enabled and, after it is fired, the token in place $ready_i$ and the i tokens in place $upcc_i$ will be removed and i tokens will be added to the output place $upoc_i$, signifying that site i is now in the state of up and out of date.

Figure 1 is part of a larger net to be explained later. Below, we explain in detail how we construct it.

1. When an update operation arrives and a major partition exists, a token will be put into place $ready_i$ by the 'system subnet' (to be described later) so that site i can start a local status update. There are six immediate transitions competing to fire at site i at this moment. However, only one out of the six transitions can fire, which will consume the token in place $ready_i$. These six transitions are $t0_i$, $t1_i$, $t2_i$, $t3_i$, $t4_i$ and $t5_i$, all of which have the same priority level (i.e. 3).
2. Transition $t0_i$ will fire if site i is in a state in which it is down and current. In this case, i tokens will flow from place $downcc_i$ to place $downoc_i$. This means that if an update occurs and site i is in the state of down and current, then the new state of site i will be down and out of date since site i is down and will not be able to participate in the update.
3. Transition $t1_i$ will fire if site i is in the state of down and out of date. In this case, site i remains in the state of down and out of date.
4. Transition $t2_i$ will fire if site i is in the state of up and out of date, as well as in the major partition. In this case, i tokens will flow from place $upoc_i$ to $upcc_i$ and the new state of site i will become up and current, since site i will participate in the update. Transition $t2_i$ is associated with an enabling function $g_i()$ which returns TRUE if site i is in the major partition at the moment. See Table 4 later for the meaning of $g_i()$ and its counterpart $\bar{g}_i()$.
5. Transition $t3_i$ will fire if site i is in the state of up and current, as well as in the major partition. In this case, site i will remain in the state of up and current. The enabling function $g_i()$ is associated with transition $t3_i$.
6. Transition $t4_i$ will fire if site i is in the state of up and

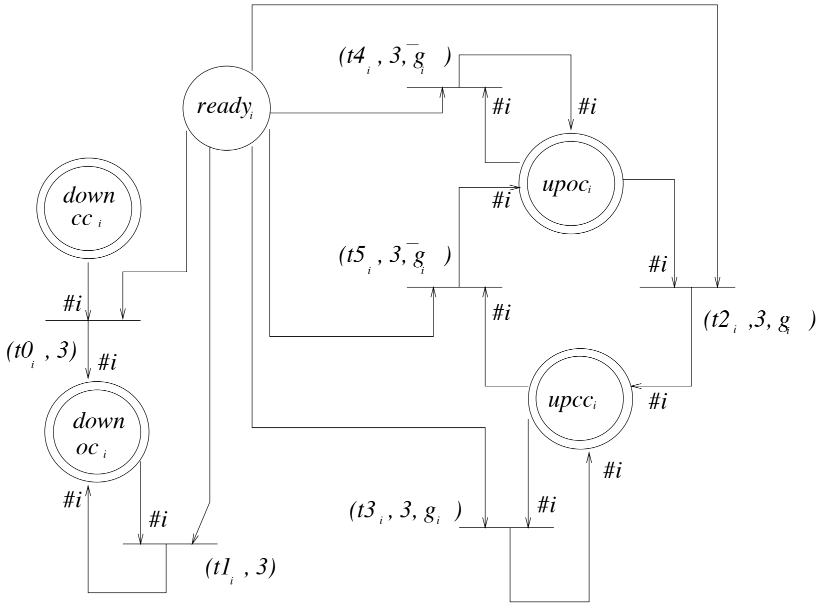

FIGURE 1. Local status update actions by site i.

out of date, as well as *not* in the major partition (say, due to network partitioning). In this case, site i will remain in the state of up and out of date since site i will not be able to participate in the update. Transition $t4_i$ is associated with an enabling function $\bar{g}_i()$ which returns TRUE if site i is *not* in the major partition at the moment.

7. Transition $t5_i$ will fire if site i is in the state of up and current, as well as *not* in the major partition. In this case, the new state of site i becomes up and out of date since it will not be able to participate in the update. The enabling function $\bar{g}_i()$ is associated with transition $t5_i$.

Note that all transitions in Figure 1 are immediate transitions, thus modelling that the state change at a site in response to an update event consumes little time, as opposed to an exponentially distributed time for a site and link failure/repair, or for an update arrival.

2.3.2. *System subnet*

Figure 2 shows an SPN subnet that describes the system behaviour as it responds to the arrival of an update operation. At the bottom of Figure 2, each of the boxes labelled site i is the SPN subset shown in Figure 1. There are several priority levels shown in Figure 2. The two transitions, tf and $tfbar$, at the top of Figure 2 are given the highest priority levels (i.e. 5 and 4). When an update event arrives, a token will be put in place $update\_event$ (by 'state-affecting subnets' to be described later) to initiate an update action. Transition tf is evaluated prior to transition $tfbar$ since it has the highest priority. If a major partition exists at the moment, the enabling function $f()$ associated with tf shall return TRUE and tf will subsequently fire, resulting in the removal of the token from place $update\_event$ and the placement of one token each into places sc, ds and $ready_i$ for all is. If a major partition does not exist at the moment, $tfbar$ will fire instead, thus removing the token in place $update\_event$.

The next highest priority level (i.e. 3) is assigned to each of the boxes which are evaluated as described previously in the subsection 'site subnet'. After all the sites have been evaluated and each site's status has been updated, transitions tds and tsc which have lower priority levels (i.e. 2 and 1) will subsequently execute. Here, transition tsc is used to update the site cardinality. The input arc multiplicity from place sc to transition tsc is set to $\#(sc)$, meaning that all the tokens originally stored in place sc will be removed. On the other hand, the output arc multiplicity from transition tsc to place sc is defined by a multiplicity function which returns the number of sites with $mark(upcc) > 0$ in the major partition (see Table 2 later). This way, whenever an update changes the status of the system, the site cardinality can be properly updated and the new value can be simply stored as the number of tokens in place sc. In a similar way, transition tds is used to update the distinguished site. The input arc multiplicity from place ds to transition tds is $\#(ds)$ while the output arc multiplicity from transition tds to place ds is determined by a multiplicity function which returns the maximum $\#(upcc)$ value among all the sites in the major partition at the moment (also see Table 2). This way, the site ID of the new distinguished site after the update will be stored as the number of tokens in place ds. Recall that we used i tokens to identify the ID of site i (Figure 1). Therefore, by merely looking at the maximum $\#(upcc)$

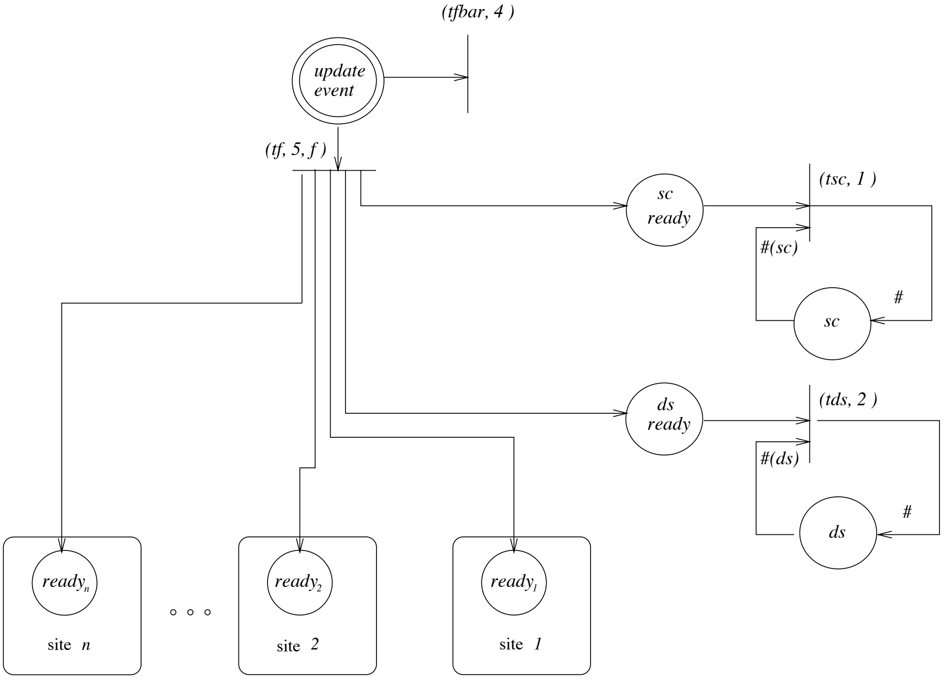

FIGURE 2. Global status-update actions triggered by an update operation.

value among all the sites in the major partition, we can easily determine the site ID of the new distinguished site.

2.3.3. Site failure/repair subnets

Figure 3 shows two subsets for describing the effect of site i's failure and repair on the system state, for the independent repairman model. At the top, state transitions are between the following two states: 'up and current' and 'down and current', while at the bottom, state transitions are between 'up and out of date' and 'down and out-of-date'. Again, since site i can only be in one state at a time, only one transition out of these two subnets is possible at a time. For the failure events, if site i is in the state of 'up and current' and subsequently fails, then its new state will be 'down and current'. Similarly, if site i is in the state of 'up and out of date' and subsequently fails, then its new state will be 'down and out of date'. These are modelled by associating transitions $tccf_i$ and $tocf_i$ each with a rate of λ_s, meaning that the firing time of these transitions is exponentially distributed with rate λ_s. For the repair events, if site i is in the state of 'down and current' and is subsequently repaired, then its new state will be 'up and current'. Similarly, if site i is in the state of 'down and out of date' and is subsequently repaired, then its new state will be 'up and out of date'. These are similarly modelled by associating transitions $tccr_i$ and $tocr_i$ each with a site repair rate of μ_s. The multiplicity of all the arcs in Figure 3 for site i is i.

Under the 'frequent updates' assumption, an update will always arrive at the system after a failure/repair and will always be processed before the occurrence of the next failure/repair. Since processing an update by the system has the effect of updating the system state, this means that the system will update its status upon every failure or repair event, not after two or more failure/repair events. We describe this status update behaviour upon every failure or repair event by also putting a token into place $update\_event$ when site i fails or is repaired. The system thus behaves as if an update operation were received after a failure or repair event. The system subnet described earlier will respond to this update event and properly update the states of all sites, including site i, after the failure or repair event of site i. This also ensures that the system state is updated before another failure or repair event occurs.

2.3.4. Link failure/repair subnets

Figure 4 shows a subset for describing a link's failure and repair, for the independent repairman model. Here we use the subscript ij to refer to the (bidirectional) link between sites i and j. For the ring topology, for example, there will be n such subnets, one for each link. For example, for a five-site ring topology as in our case study, there will be links 12, 23, 34, 45 and 51. A token in place $uplink_{ij}$ means that the link between sites i and j is in the state of 'up', while a token in place $dwlink_{ij}$ means that the link between sites i and j is in the state of 'down'. Failures and repairs of links occur independently with failure and repair rates of λ_l and μ_l, respectively. Similar to the subnet for site failure/repair, we note that in order to model frequent updates, we also put

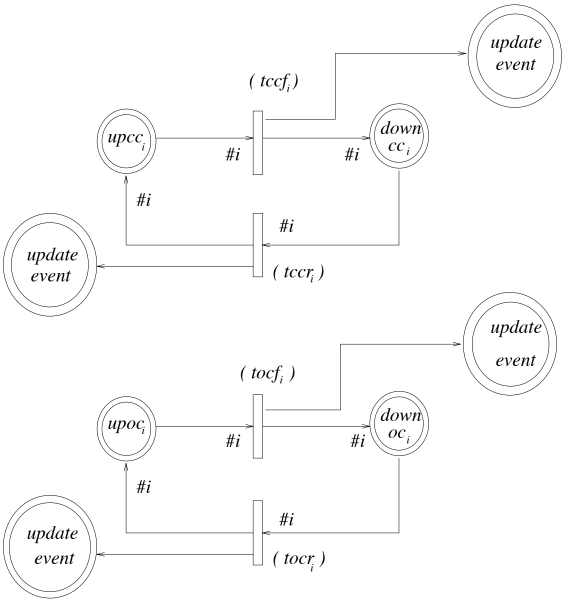

FIGURE 3. Site failure/repair events.

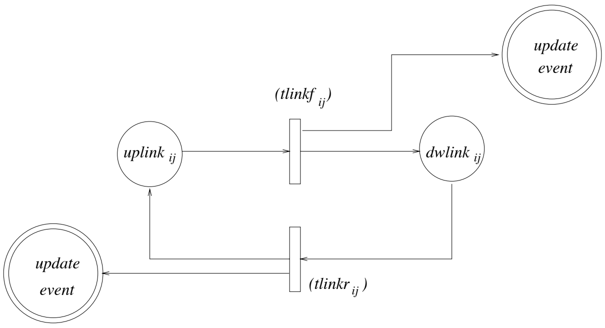

FIGURE 4. Link failure/repair events.

TABLE 1. Meanings of places.

Place	Meaning
$upcc_i$	Copy$_i$ is up and current
$downcc_i$	Copy$_i$ is down and current
$upoc_i$	Copy$_i$ is up and out of date
$downoc_i$	Copy$_i$ is down and out of date
$uplink_{ij}$	Link$_{ij}$ is up
$dwlink_{ij}$	Link$_{ij}$ is down
$update\_event$	An update is initiated
sc_ready	An SC is initiated
sc	$\#(sc)$ indicates the SC
ds_ready	A DS change is initiated
ds	$\#(ds)$ indicates the ID of the DS
$ready_i$	A local update at site i is in process

TABLE 2. Arc multiplicity functions.

Arc	Multiplicity
$sc \rightarrow tsc$	$\#(sc)$
$tsc \rightarrow sc$	# of sites in the major partition with mark$(upcc) > 0$
$ds \rightarrow tds$	$\#(ds)$
$tds \rightarrow ds$	Max $\#(upcc)$ among all sites in the major partition

a token into place $update\_event$ when a link is failed or repaired.

Combining Figures 1, 2, 3 and 4 together, we obtain a composite SPN for the independent repairman model. In the

TABLE 3. Rates of timed transitions.

Timed transition	Rate value
$tccf_i$	λ_s
$tccr_i$	μ_s
$tocf_i$	λ_s
$tocr_i$	μ_s
$tlinkf_{ij}$	λ_l
$tlinkr_{ij}$	μ_l

TABLE 4. Enabling functions.

Tr.	Enabling function
tf	$f()$ {IF $\exists$ a partition $\mathcal{M}$ with sum equal to # of sites in $\mathcal{M}$ with mark$(upcc) > 0$; AND IF (sum $> \#(sc)/2$) OR (sum $= \#(sc)/2$ AND mark$(upcc_{\#(ds)})$ AND site $\#(ds) \in \mathcal{M}$) THEN RETURN 1; ELSE RETURN 0}

$t2_i$	$g_i()$
	{Look at mark$(uplink_{jk})$ $\forall j \forall k$ to
	determine site i's partition;
	IF site i's in the major partition $\mathcal{M}$
	THEN RETURN 1;
	ELSE RETURN 0}

$t3_i$	$g_i()$
$t4_i$	$\bar{g}_i()\{1 - g_i()\}$
$t5_i$	$\bar{g}_i()$

TABLE 5. Rates of timed transitions for FIFO repair.

Timed Tr.	Rate value
$tccf_i$	λ_s
$tccr_i$	$\dfrac{\mu_s}{\sum_{j,k,k\neq j} \#(downcc_j + downoc_j + dwlink_{jk})}$
$tocf_i$	λ_s
$tocr_i$	$\dfrac{\mu_s}{\sum_{j,k,k\neq j} \#(downcc_j + downoc_j + dwlink_{jk})}$
$tlinkf_{ij}$	λ_l
$tlinkr_{ij}$	$\dfrac{\mu_l}{\sum_{j,k,k\neq j} \#(downcc_j + downoc_j + dwlink_{jk})}$

composite SPN, a common place among all the subnets is labelled with double circles. Tables 1–4 help annotate this composite SPN.

3. SPNS FOR DEPENDENT REPAIRMAN MODELS

In this section, we consider the case in which many sites and links may have to share the same repairman due to repair resource limitations. To illustrate the modelling techniques used in addressing this issue, we focus on the case where all sites and links share a repairman. Three repairman models are considered: (1) FIFO: failed sites and links are repaired in a first in first out order; (2) linear-order: failed sites and links are repaired in a prespecified linear order; and (3) best-first: among all failed sites and links, we choose the one whose repair will yield the largest improvement in availability relative to the current state. Apparently, among the three repair models, best-first may incur the highest overhead in deciding who to repair next; in return, it can provide the best achievable availability under a single repairman constraint. In this paper, we ignore the overhead issue and focus on availability modelling.

To model the FIFO repairman model, we note that by queueing theory [13] the performance characteristics (e.g. queueing delay) of a client under the FIFO service discipline for a single server whose service time is exponentially distributed would be the same as that under the processor sharing (PS) discipline where the processing power of the server is divided equally among all clients in the queue. Therefore, without having to introduce any extra places in the SPN to keep track of the queueing order of failed sites and links in a state, we simply make use of the SPN for the independent repairman model described earlier and modify the repair rates to account for repair dependencies. A similar approach has been adopted in [14]. Specifically, Table 5 replaces Table 3 with each repair rate now becoming a function of the total number of failed sites and links that depends on the current state of the system. For example, if a particular state has two failed sites and one failed link, then instead of using repair rates of μ_s, μ_s and μ_l, respectively, for these three failed entities based on the independent repairman model, we now use repair rates of $\mu_s/3$, $\mu_s/3$ and $\mu_l/3$, respectively, to model the fact that the same repairman is being shared among these three failed entities. In general, if a particular state has M failed sites and links, then the respective repair rates of these failed entities in that state are 'deflated' by a factor of M to account for the effect of repair resource sharing.

As for the linear-order repairman model, there is a prespecified order among failed entities. Determining which order to use is a search problem itself and requires availability studies in its own right. Since a linear order already exists in dynamic voting for the purpose of selecting the distinguished site (DS), a natural way is to follow that order for repair as well. The intuition is that it is more likely that a higher linearly ordered site will become a distinguished site in the event that the number of sites within the major partition is an even number. Therefore, giving a higher repair priority to a higher linearly ordered site increases the chance of finding the majority partition. An example is a state in which there are only two sites D and E left in the major partition with E being the distinguished site. Suppose sites D and E subsequently fail. Then, repairing D would not result in a major partition being found while repairing E would. It is less clear whether site repairs should take precedence over link repairs or vice versa. However, for the site availability metric considered here, given the fact that a single site can still provide services

TABLE 6. Enabling functions for linear-order repair.

Transition	Enabling function

$tccr_i, tocr_i$	$h1_{site}(i)$
	{IF site i failed and the repair rank of site i is higher than those of other failed sites or links in the linear order
	THEN RETURN TRUE;
	ELSE RETURN FALSE}

$tlinkr_{ij}$	$h1_{link}(i, j)$
	{IF link ij failed and the repair rank of link ij is higher than those of other failed sites or links in the linear order
	THEN RETURN TRUE;
	ELSE RETURN FALSE}

if it is the distinguished site regardless of the network status, it seems intuitive to give site repairs a higher priority than link repairs. For the same reason, links that connect higher linearly ordered sites should be given a higher priority than links that connect lower linearly ordered sites.

We model the linear-order repairman model by directly modifying the SPN for the independent repairman model. We create a new enabling function associated with the transition of each site or link repair event. All such enabling functions shall execute the same code, with each enabling function knowing the ID of the site or link it is designated to. Only one enabling function at any state shall return TRUE based on the prespecified linear order and all others shall return FALSE. For example, suppose that the linear order is (sites 5,4,3,2,1; links 54,51,43,32,21) for a five-site ring topology and sites 4, 2 and link 51 are down in a particular state. In this case, the enabling functions associated with sites 4, 2 and link 51 will return the boolean values of TRUE, FALSE and FALSE, respectively, meaning that site 4 will be repaired next over site 2 and link 51. Note that in this state all other enabling functions are not activated since the input places of the corresponding transitions would be empty. This technique prevents concurrent firing of multiple repair transitions. Consequently, in any state at most one repair event can occur with the correct repair rate based on the specified repair order. Table 6 shows a description of the set of additional enabling functions needed for the linear-order repairman model.

While the linear-order repairman model applies a fixed repair order which is not changed at the run time, the best-first repairman model changes the repair order on-the-fly, with the preference always given to the site or link which can most improve the site availability of the system after its repair with respect to the current state. To do so, the best-first repairman model executes the following repair strategies.

1. If there exists a distinct failed site or link whose repair would lead to the existence of a major partition in which the number of current copies is the largest among all possible repair choices in that state, then the distinct failed site or link will be selected to be repaired next.
2. If there exist more than one failed site or link whose repair would lead to the existence of a major partition in which the number of current copies is the largest among all possible repair choices in that state, then a tie-breaker rule will be applied to select one distinct member of the group to be repaired next. We consider the following tie-breaker rules in this paper, with the reasons explicitly given.
    (a) If the group contains both failed sites and failed links as repair candidates, then the preference is given to the failed sites if $\mu_s/\lambda_s \geq \mu_l/\lambda_l$; otherwise, the preference is given to the failed links. Then, the tie-breaker rule (described next) for just either failed sites or links applies. The reason behind this rule is that preference is given to the failed entity which can be repaired fastest, which means fastest recovery given the same availability improvement outcome.
    (b) If the group contains just failed sites, then preference is given to the highest linearly ordered site in the group. The reason behind this rule is that a higher linearly ordered site is more likely to become a distinguished site and thus repairing a higher linearly ordered site has the effect of moving more available sites toward the current distinguished site. This creates a higher probability of uniting more sites in the major partition in the future. Intuitively, we like to repair sites or links closer to the current distinguished site so that the system will be in a better position to unite more sites at a later time. To see this, consider a particular state of a five-site ring system[1] in which the major partition contains only the distinguished site, say site 4, while all other sites (sites 1, 2, 3 and 5) and links (links 12, 23, 34, 45 and 51) have failed. This is a scenario in which repairing any of the failed sites or links yields the same availability improvement, i.e. no improvement afterward therefore the preference is given to repairing sites over repairing links. Then, repairing site 5 will put the system in a better position to unite sites 4 and 5 (after a future repair of link 45), while repairing a lower linearly ordered site, say site 2, has no such benefit. One may argue that another scenario would give a totally different result but the point is that site 4 has a higher probability than site 2 to become the distinguished site.
    (c) If the group contains just failed links, then the preference is given to the link that connects the highest linearly ordered sites among all links in the group. The reason follows that for the tie-breaker rule for sites except that we now like to repair links around the current distinguished site. Consider the same scenario mentioned above

[1] A five-site ring consists of sites 1, 2, 3, 4 and 5 and links 12, 23, 34, 45 and 51.

TABLE 7. Enabling functions for best-first repair.

Transition	Enabling function

$tccr_i, tocr_i$	$h1_{site}(i)$
	{IF site i failed and the hypothetical site availability after repairing site i is higher than those of other failed sites or links in the system
	THEN RETURN TRUE;
	ELSE RETURN FALSE}

$tlinkr_{ij}$	$h1_{link}(i, j)$
	{IF link ij failed and the hypothetical site availability after repairing link ij is higher than those of other failed sites or links in the system
	THEN RETURN TRUE;
	ELSE RETURN FALSE}

except that $\mu_s/\lambda_s \geq \mu_l/\lambda_l$ is false, so that repairing links takes precedence over repairing sites. In this case, repairing link 45 would be able to unite sites 4 and 5 after a future repair of site 5, while repairing any other link, say link 12, will not result in a major partition containing more than one site after any subsequent repair. Again, since a higher linearly ordered site has a higher probability of becoming the distinguished site, repairing a link connecting higher linearly ordered sites has the effect of putting the system in a better position to unite more sites after some future selective repairs.

For the analysis of the best-first repairman model, we also modify the SPN for the independent repairman model. Similar to the linear-order repairman model, we again associate an enabling function with the repair transition of each site or link; let all enabling functions execute the same code, with each enabling function knowing the ID of the site or link it is designated to; and let only one enabling function at any state return TRUE while all others return FALSE. The difference is that the encoding of the enabling function ranks all failed sites and links in a particular state based on the hypothetical site availability values (which would result if they were chosen to be repaired next). Only the enabling function designated to the highest-ranked site or link for that particular state returns TRUE and all others return FALSE. Table 7 shows a description of the set of additional enabling functions needed for the best-first repairman model.

4. EVALUATION

We test the modelling techniques developed in the paper with a five-site ring topology. Sites were labelled by 1, 2, 3, 4 and 5 with site 5 being the highest linearly ordered site among all initially; links were labelled with 12, 23, 34, 45 and 51 (bidirectional)[2]. Four different repairman models were examined. For the independent repairman model, sites and links fail and repair independently of each other. The SPN for the independent repairman model consists of Figures 1, 2, 3 and 4 with Tables 1, 2, 3 and 4. The underlying Markov chain contains 8674 states. For the remaining three dependent models, we let all sites and links share the same repairman. In particular, for the linear-order repairman model, we assume that failed sites and links are repaired in the order of site 5, site 4, ..., down to site 1 and then link 54, link 54, ..., down to link 12, so that a higher repair rank is assigned to a higher linearly ordered site (among sites) and a link connecting higher linearly ordered sites (among links). The SPN for the dependent repairman model with FIFO consists of Figures 1, 2, 3 and 4 with Tables 1, 2, 4 and 5, with the number of states in the underlying Markov chain being 8674. The SPN for the dependent repairman model with linear-order consists of the same four figures with Tables 1, 2, 3, 4 and 6, with the underlying Markov chain containing 5429 states. Finally, the SPN for the dependent repairman model with best-first consists of the same four figures with Tables 1, 2, 3, 4 and 7; the number of states in this case is 3821. The time it takes to solve these models using SPNP [16] is in the order of a few minutes on a SUN Ultra 1 machine.

The site and system availability metrics considered in the case study were obtained by assigning proper 'rewards' with states of the system [16]. The system availability metric, i.e. the steady-state probability that a major partition exists, was obtained by associating a reward rate of 1 with those states in which the enabling function $f()$ is evaluated to TRUE and a reward rate of 0 otherwise. The site availability metric, i.e. the probability that an update arriving at an arbitrary site will succeed, was obtained by associating a reward rate of $1 \times k/n$ with those states in which the enabling function $f()$ is evaluated to TRUE and a reward rate of 0 otherwise, with k representing the number of 'up' sites in the major partition (if it exists) in a particular state and n is the total number of sites (copies) in the system.

4.1. Availability for the independent repairman model

Figure 5 shows the site availability values for the independent repairman model under various combinations of site and link repair-rate/failure-rate ratios. While the repair-rate/failure-rate ratios can be arbitrarily large, in this paper we consider a range of ratio values from 1 to 20 so as to facilitate the comparison of our results with those published in [2] which had previously considered only either site or link failures, but not both. The reader can refer to [10] for the case when the ratios are larger and also when the assumption of frequent updates is relaxed. Here, the x-axis is the ratio of the mean time to failure per link to the mean time to repair per link or, equivalently, the ratio of link repair rate to

[2] The modelling techniques developed in the paper can be generally applicable to all types of network topologies. For the five-site ring, we explicitly include every site and every link in defining the SPN model. It may be possible to exploit the symmetric property of the five-site ring and apply model reduction techniques [15] to define a smaller yet symmetric SPN model without losing the solution exactness. This aspect is not explored in this paper.

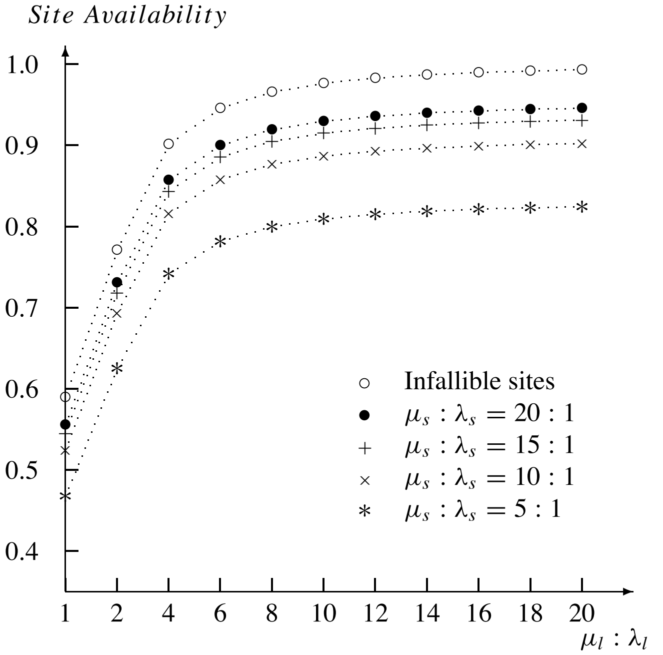

FIGURE 5. Site availability of five-site ring under independent repairman model.

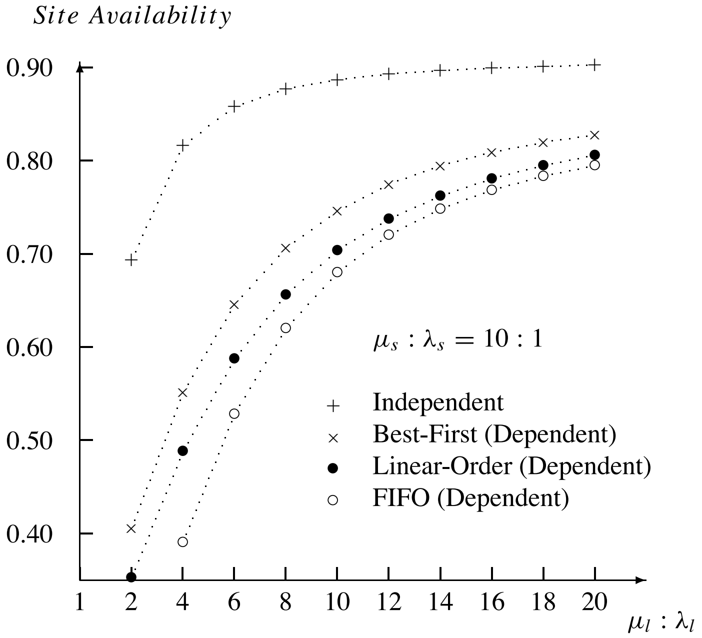

FIGURE 6. Site availability of five-site ring under various repairman models.

link failure rate. The asymptotic case where only links are fallible as obtained in previous studies [2] is also shown for comparison reasons. From the significant difference in site availability shown in the figure, we conclude that models which consider either only fallible sites or fallible links, but not both, in previous studies can very unrealistically overestimate the site availability metric.

4.2. Comparing availability metrics under various repairman models

Figure 6 shows the site availability values obtained under four different repairman models, with the ratio of the site repair rate to the site failure rate fixed at 10:1. Figure 7 shows the same data except using the system availability as the comparison metric. The x-axis is again the ratio of the mean time to failure per link to the mean time to repair per link. From the figure, we observe the effect of repair dependency: the site availability of dynamic voting with independent repair is much higher than those with dependent repair. This is because in the latter cases only one repairman is available and all failed sites and links must compete for the same repair resource, while in the former case every site or link has its own designated repairman. Only when the ratio of repair rate to failure rate is relatively high can the availability of the best-first repairman model approximate that of the independent repairman model.

Among the three dependent repairman models, however, the best-first repairman model can always provide a better site availability value than the other two. These are two reasons by which best-first improves the availability when compared with other dependent repairman models.

1. The best-first repairman model can avoid unfavourable

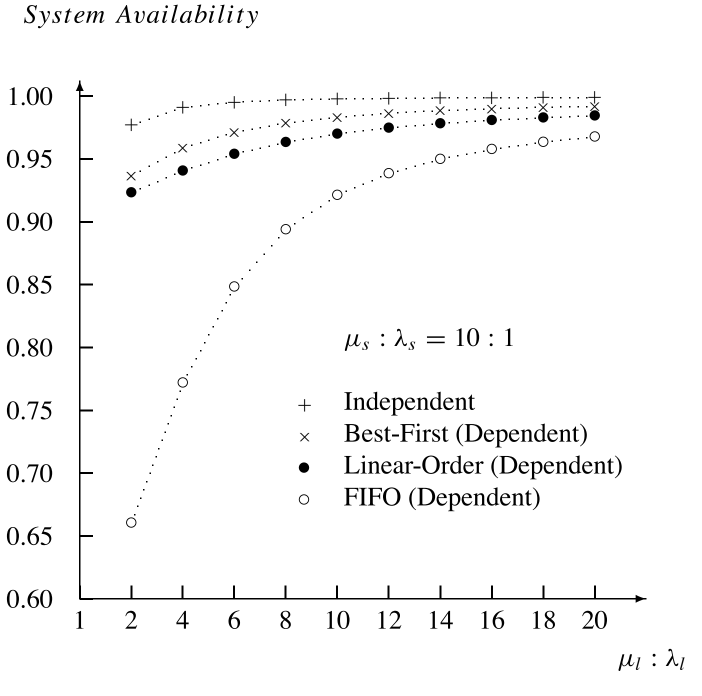

FIGURE 7. System availability of five-site ring under various repairman models.

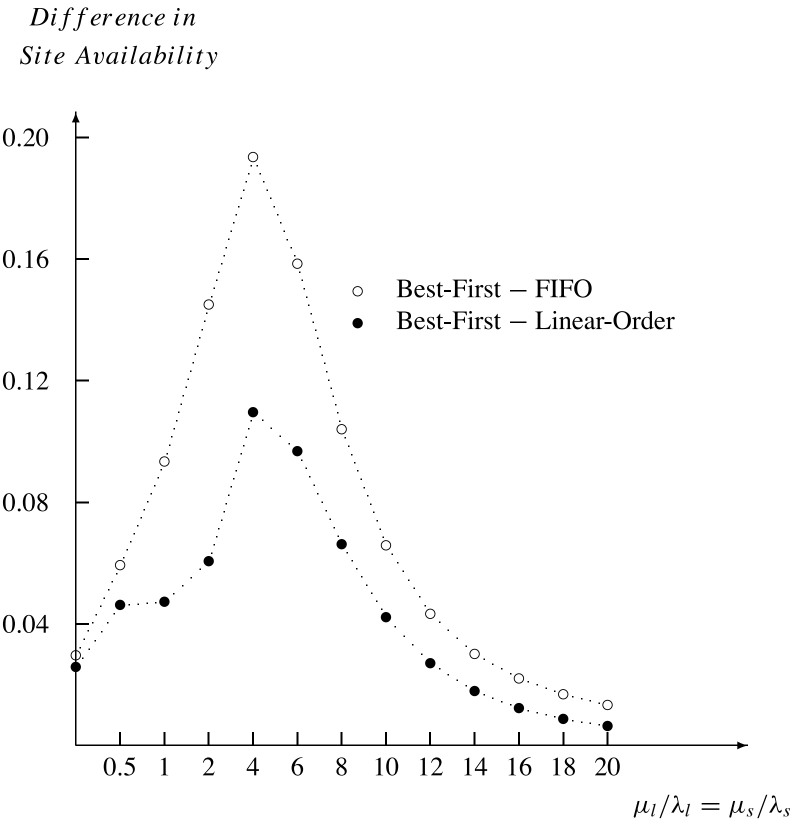

FIGURE 8. Difference in site availability among dependent repairman models.

states which other dependent repairman models cannot. These unfavourable states occur when the system contains either (a) one or more up but out of date copies or (b) one up and current copy which is not the distinguished copy, and one or more up but out of date copies. These unfavorable states result from ineffective repair activities which occur during a period in which a major partition does not exist. For the FIFO and linear-order repairman models, it is possible that a major partition still could not result even after several repairs. For the best-first repairman model, however, if the system is in such a state (in which a major partition does not exist), it will avoid going into these unfavourable states (in which no major partition exists again after some repairs) by repairing the distinguished site first such that after a selective repair a major partition always results.[3]

2. If the system is in a state in which a major partition exists and some failed sites and links need to be repaired, then best-first will choose a failed site or link to repair next such that the size of the major partition after the repair will become the largest among all possible choices. This eliminates the bad effect of ineffective repairs. As an example, consider a system being partitioned into two parts, with one part containing two up sites and one other part containing two up sites and one down site. An ineffective repair in this case can repair the down site (as could be done by FIFO and linear-order) instead of the failed link which partitions the five sites, thus resulting in a major partition of size 3. Best-first would select the failed link to repair and after the repair the size of the major partition is 4 instead. In addition, in cases in which repairing any member of a group results in the same availability improvement outcome, it will select the one with the shortest repair time so that for most of time the system can stay in those states in which it has a high availability.

Figure 8 shows the difference in site availability between best-first and linear-order, and also between best-first and FIFO, as a function of the ratio of the repair rate to the failure rate, assuming that the repair rates for sites and links are identical and the failure rates for sites and links are also identical, i.e. $\mu_s = \mu_l = \mu$ and $\lambda_s = \lambda_l = \lambda$. Figure 9 shows the same except using the system availability as the metric. These two figures show the impact of 'effective repairs' employed by the best-first repairman model (and to some extent by the linear-order repairman model). When the ratio of the repair rate over the failure rate (μ/λ) is very

[3] We should mention that these arguments are valid only under the assumption of frequent updates. Without frequent updates, even the best-first repairman model cannot guarantee the existence of a major partition after a repair.

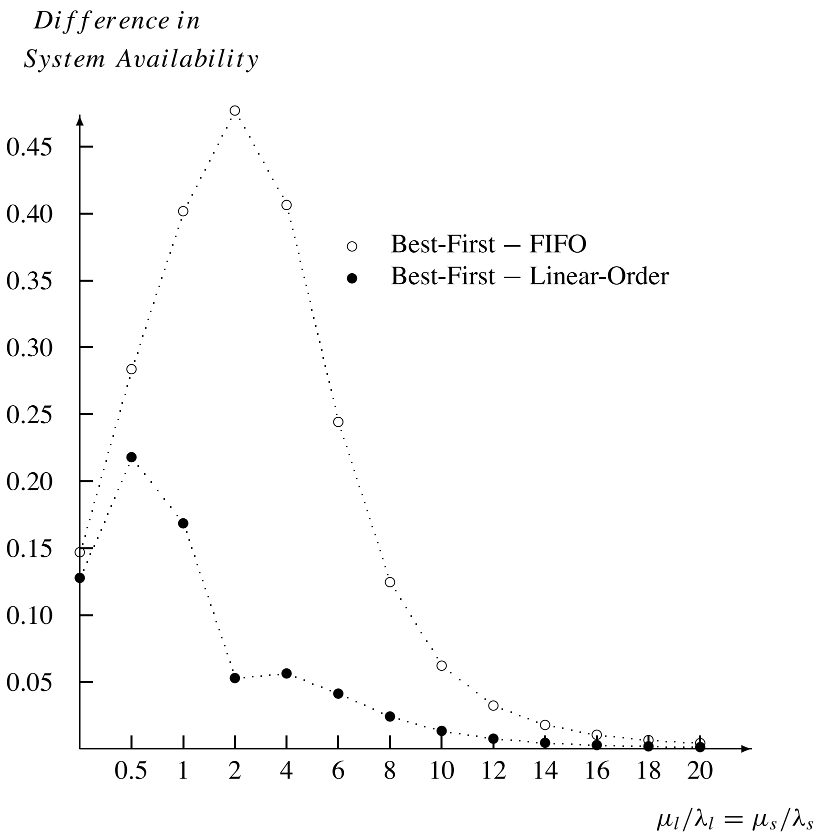

FIGURE 9. Difference in system availability among dependent repairman models.

high, i.e. greater than 20, the difference among these three dependent repairman models is small since sites/links can be repaired much faster than they can fail, so it matters little which repairman model is being used. However, as the ratio of μ/λ becomes lower (say between 5 and 15), the impact of effective repairs manifests itself more (because at any time there can be many failed entities waiting to be repaired) and the difference in availability becomes higher. This also reflects the condition under which the best-first repairman model can benefit the system most when compared with the other two repairman models. Finally, when the ratio becomes very low (i.e. approximately 1) in which case sites/links can fail faster than they can be repaired, the site or system availability metric is so low anyway that the effect of effective repairs becomes less significant. The concave curves shown in Figures 8 and 9 indicate that there exists a data point at which best-first has the largest edge over the other two repairman models.

5. CONCLUSIONS

In this paper, we proposed and investigated four repairman models which can exist in replicated data management. Using dynamic voting as a case study, we developed modelling techniques based on Petri nets to analyse the effect of these repairman models on availability. We discovered that (a) an availability model that considers only site or link failures/repairs, but not both, can give a very unrealistic, overestimated value of the availability metric; (b) a significant difference in availability exists between two systems with independent and dependent repairman models, except when the repair rate is much higher than the failure rate (i.e. when μ/λ is greater than 20); (c) when several sites and links have to share the same repairman, the best-first repairman model can always provide a better availability than the FIFO and linear-order repairman models; the difference in availability becomes more pronounced as the site/link failure rate increases relative to the site/link repair rate; and there exists a region in which the difference in availability is significant. From these results, we conclude that ignoring concurrent site/link failure modes or repair dependency can unduly overestimate the availability of replicated data. We urge that tools such as those developed in the paper be used to estimate realistically the resulting availability metric of replicated data, since after all the main reason for using replicated data is to improve the data availability of the system.

Some possible future research areas include (a) extending and applying the modelling techniques to analysing other pessimistic algorithms such as those based on coteries [17, 18] and (b) developing modelling techniques to study the trade-off between the reduction in data processing overheads (say due to the use of more constrained quorum sets based on a hierarchical or tree structure) and the sacrifice in availability (due to the more constrained way of finding a quorum) for database environments where a combined performance/availability design goal must be met.

ACKNOWLEDGEMENTS

This work was supported in part by the National Science Council, Taiwan, under Grant NSC 86-2745-E006-020.

REFERENCES

[1] Adam, N. R. (1994) A new dynamic voting algorithm for distributed database systems. *IEEE Trans. Knowl. Data Engng*, **6**, 470–478.

[2] Jajodia, S. and Mutchler, D. (1990) Dynamic voting algorithms for maintaining the consistency of replicated database. *ACM Trans. Database Systems*, **15**, 230–280.

[3] Chang, H. K. and Yuan, S. M. (1995) Performance characterization of the tree quorum algorithm. *IEEE Trans. Parallel Distrib. Systems*, **6**, 658–662.

[4] Cheung, S. Y., Ammar, M. and Ahamad, M. (1992) The grid protocol: a high performance scheme for maintaining replicated data. *IEEE Trans. Knowl. Data Engng*, **4**, 582–592.

[5] Kumar, A. (1991) Hierarchical quorum consensus: A new algorithm for managing replicated data. *IEEE Trans. Computers*, **40**, 996–1004.

[6] Rangarajan, S., Setia, S. and Tripathi, S. K. A fault-tolerant algorithm for replicated data management. *IEEE Trans. Parallel Distrib. Systems*. **6**, 1271–1282.

[7] Triantafillou, P. and Taylor, D. J. (1995) The location-based paradigm for replication: Achieving efficiency and availability in distributed systems. *IEEE Trans. Soft. Engng*, **21**, 1–18.

[8] Wolfson, O. and Jajodia, S. (1995) An algorithm for dynamic data replication in distributed systems. *Info. Process. Lett.*, **53**, 1113–1119.

[9] Dugan, J. B. and Ciardo, G. (1989) Stochastic Petri net analysis of a replicated file system. *IEEE Trans. Soft. Engng*, **15**, 394–401.

[10] Chen, I. R. and Wang, D. C. (1996) Analysing dynamic voting using Petri nets. *15th Symposium on Reliable Distributed Systems (SRDS'96)*, Niagara Falls, Ontario, Canada, October, pp. 44–53.

[11] Jajodia, S. and Mutchler, D. (1989) A pessimistic consistency control algorithm for replicated files which achieve high availability. *IEEE Trans. Soft. Engng*, **15**, 39–45.

[12] Jajodia, S. and Mutchler, D. (1989) A hybrid replica control algorithm combining static and dynamic voting. *IEEE Trans. Knowl. Data Engng*, **1**, 459–469.

[13] Kleinrock, L. (1975) *Queueing Systems. Volume 1: Theory*. John Wiley and Sons, New York.

[14] Ciardo, G., Muppala, J. K. and Trivedi, K. S. (1992) Analyzing concurrent and fault-tolerant software using stochastic reward nets. *J. Parallel Distrib. Comput.*, **15**, 255–269.

[15] Sanders, W. H. and Malhis, L. M. (1992) Dependability evaluation using composed SAN-based reward models. *J. Parallel Distrib. Comput.*, **15**, 238–254.

[16] Ciardo, G., Muppala, J. K. and Trivedi, K. S. (1989) SPNP: stochastic Petri net package. *Proc. 3rd Int. Workshop Petri Nets and Performance Models,* Kyoto, Japan, December, pp. 142–151.

[17] Fu, A. W. (1997) Delay-optimal quorum consensus for distributed systems. *IEEE Trans. Parallel Distrib. Systems*, **8**, 59–69.

[18] Garcia-Molina, H. and Barbara, D. (1985) How to assign votes in a distributed system. *JACM*, **32**, 841–860.