Information Processing Letters 75 (2000) 247–253

Information Processing Letters

www.elsevier.com/locate/ipl

Response time behavior of distributed voting algorithms for managing replicated data

Ing-Ray Chen [a,*], Ding-Chau Wang [b], Chih-Ping Chu [b]

[a] *Department of Computer Science, Virginia Tech, Northern Virginia Graduate Center, Falls Church, VA 22043, USA*

[b] *Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan, Taiwan*

Received 14 December 1999; received in revised form 25 May 2000

Communicated by F.B. Schneider

Abstract

Voting is a simple and yet effective way of managing replicated data in distributed systems. In this paper we analyze its response time behavior. We investigate a technique for obtaining the access time distribution for requests that access replicated data maintained by the distributed system. The technique is based on Petri net modeling and can be used to estimate the reliability of real-time applications which must access replicated data with a deadline requirement. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Voting; Replicated data management; Real-time; Quorum; Reliability; Distributed systems; Stochastic Petri nets

1. Introduction

Voting is an effective way to maintain replicated data in distributed systems and has been widely applied to many practical applications. Whether distributed voting can be applied directly to real-time applications, however, remains as an open issue since there is yet an effective way to determine if the imposed access time constraint of the real-time application in accessing the replicated data can be satisfied. One main problem is that many design and environmental parameters can attribute to the access time, e.g., number of copies, how the copied are connected topologically, failure and recovery rates of nodes and links in the distributed system, real-time constraints, etc. Previous researches on voting or quorum schemes for replicated data management concentrate mostly on replica data algorithms developments, mostly using availability as the comparison metric and mostly considering only site failures [1,6,7,11]. More recently, some researches begin to address the performance issue of replica control protocols, notably in [9,10]. Both site and link failures are considered for dynamic voting under various repairman models in [2]. No research yet has been done to address the applicability of replica control algorithms for applications subject to a deadline or deadline distribution constraint.

The problem is interesting and challenging because the normal performance metric such as the average availability or the average response time cannot be used to measure the probability that the access time will be less than the deadline. This requires that an access time distribution rather than the average access time be computed. In this paper, we investigate a

* Corresponding author.

E-mail addresses: irchen@cs.vt.edu (I.-R. Chen), wangdc@csie.ncku.edu.tw (D.-C. Wang), chucp@csie.ncku.edu.tw (C.-P. Chu).

technique for obtaining the access time distribution for requests that access replicated data maintained by the distributed system. The technique stems from Petri net modeling [2,8] and considers both the site and link failure/recovery cases for replicated data management.

2. System model

We assume that majority voting is the replicated data management scheme used for maintaining the consistency of multiple copies. We assume that there are n sites connected by a topology to be specified in the distributed system. Each site contains a copy carrying a single vote. Under majority voting, it requires that a read or a write to the replicated data be allowed if and only if more than one half of the copies are available at the time of access. We shall call such a set a quorum. For example, for three sites (copies) labeled with 1, 2 and 3. A read or write quorum under majority voting can be $\{1, 2, 3\}$, $\{1, 2\}$, $\{1, 3\}$ or $\{2, 3\}$. For a ring topology, we assume that a failed site on the ring can be bypassed but a failed link will block the communication between two sites connected by the failed link.

When a site or link fails, we assume fail-stop behaviors. Due to site/link failures, at the time an access is made the distributed system may be in a state in which a quorum is not available. If it is in this case, then we say the system is unavailable and the access request will have to wait until a quorum is available by means of node/link recovery activities. We assume independent failure modes for sites and links, with λ_s and λ_l being the failure rates of sites and links, respectively. Failed sites and links will be recovered independently with rate μ_s and μ_l, respectively. This independent recovery assumption is not required, e.g., they could share a same recovery source if needed, e.g., see [2]. All times for these failure and recovery events are assumed to be exponentially distributed. This assumption is made to allow us to use Stochastic Petri Nets (SPNs) to describe the system and to use software packages such as SPNP [3] to solve the model based on reward assignments to yield our performance measures of interest. The approach described here can be extended to Markov Regenerative Stochastic Petri Net (MRSPN) [4] or Extended Stochastic Petri Net (ESPN) [5] models in which firing times can be general distributions.

3. Method

Our method of obtaining the access time distribution of replicated data consists of three steps. The first step is to obtain the steady-state probability of the system, that is, to obtain the percentage of time the system stays at a particular state. Of course, the system consists of a finite number of states. In some of these states a quorum is available and therefore the access can be granted immediately, while in the others a quorum does not exist and thus a delay is incurred. This first step is to obtain the steady-state probability that the system stays in each of these states. This step must consider all possible combinations of site and link failures/recoveries as addressed in our previous work [2].

The second step is to collect the access time contribution from each of the system states identified in step one. Let P_i be the steady-state probability of state i determined in step 1 and let $F_i(t)$ be the *per state* access time distribution, given that the system is in state i at the time of access. We assume that only the delay due to a quorum not available at the access time needs to be considered for real-time applications. Therefore, the access time delay is considered zero for a state in which a quorum exists. For a state in which a quorum does not exist, we trace the recovery time of the system starting from the state until it again reaches a state in which a quorum exists. Section 4 will illustrate this technique with a detailed example.

The third step is to compute the overall access time distribution by summing the individual *per state* access time distributions weighted by their respective state probabilities, i.e.,

$$\begin{aligned} F_{system}(t) &= Pr\{\text{access time} \leqslant t\} \\ &= \sum_i P_i F_i(t). \end{aligned} \tag{1}$$

This result can be used to compute the system reliability of the distributed system for supporting a real-time application with a hard deadline, t_d, i.e.,

$$R_{system}^{deadline} = F_{system}(t_d). \tag{2}$$

If the application is characterized by a deadline distribution, $F_{t_d}(t)$, then we compute the system reliability as

$$R_{system}^{deadline} = \int_0^\infty F_{system}(t)\,\mathrm{d}F_{t_d}(t). \tag{3}$$

4. Example

In this section we describe an example to illustrate the three steps described in the last section. We utilize SPNP [3] as a tool to obtain the steady-state probability of the system. Specifically, for each node or link in the system, we create an SPN subnet to keep track of its status as failure and recovery events occur to the node or link. Each site and link thus has two states: up or down. Therefore, the underlying Markov chain is characterized by an n-component state description vector $(s_1, s_2, \dots, s_n)$ where s_i is a binary quantity standing for the status of the ith entity (site or link). Each node or link is labeled in the SPNP program and thus the program knows exactly whether the system has a quorum available in a particular state, by merely inspecting the status of all the nodes and links in that state (to see if a majority of connected, alive sites can be found in that state). We classify all the system states into two classes: good and bad. Good states are those in which a quorum is available and therefore the system can satisfy the access request immediately; conversely, bad states are those in which a quorum is not available. For examples, for 5 sites connected by a ring structures under majority voting, there are altogether 1024 states, of which 186 are "good" and 838 states are "bad" states.

Next we note that the access time of the system, given that the system is in a particular bad state, is the time taken for the system to recover failed sites or links from that bad state such that a quorum (a majority of copies for majority voting) can become available again. Of course, during the recovery period, a site or link that is originally alive can fail and this must also be considered in the model. A quorum is available when the system reaches one of the good states from the initial bad state. Note that since all failed sites and links can do independent recovery, there are more than one way to recover from a bad state. We obtain the per state access time distribution for each "bad" state i, $F_i(t)$, as follows:

(a) the SPN developed earlier in step 1 is modified such that all good states now become *absorbing states*;
(b) the initial state of the system is set to "bad" state i, e.g., for a 3-site ring under majority voting if in a bad state i, sites 1 and 3 have failed and all others are alive, then we initialize the site and link subnets in the SPN according to this condition;
(c) "rewards" are assigned to states in the modified SPN, with 1 being assigned to absorbing states and 0 being assigned to all other states;
(d) the per state access time contribution from bad state i, i.e., $F_i(t)$, can be computed as the expected instantaneous reward.

Here we note that the system evolves over time from the initial bad state i till one of the good (and absorbing) states is reached as recovery events occur. Therefore, the SPN can properly account for the recovery time of the system given that the system starts from the initial bad state i. Moreover, because of the way we assign rewards to bad and good states, the average instantaneous reward also correctly computes the probability that one of the good states will be reached as a function of time.

Once we have collected all per state access time distributions for all bad states, we can compute the system access time distribution based on Eq. (1). It should be noted that in cases where the recovery rate is relatively higher than the failure rate (site or link), then most of the time the system will likely be staying in good states and, consequently, the probability that the system will stay in a bad state, namely, P_i in Eq. (1) for a bad state i, is very low. Thus the contribution to the access time from such a bad state (via the $P_i F_i(t)$ term in Eq. (1)) is also very low, even though the recovery time (access time) itself may be long.

4.1. SPN implementation

The methodology described in the paper is generally applicable to any voting schemes and topologies. Here we illustrate its utility with a 3-site ring topology in which three sites are labeled as 1, 2 and 3, and three (bidirectional) links are labeled as 12, 23, and 13.

We use an SPN subnet to describe the state of each site in the topology. Fig. 1 shows such a subset

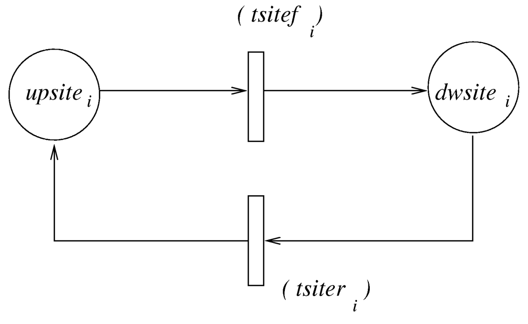

Fig. 1. SPN subnet for describing site failure/recovery events.

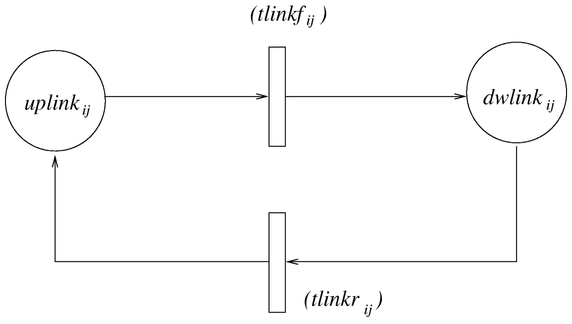

Fig. 2. SPN subnet for describing link failure/recovery events.

describing the failure/recovery activities of a site, say, site i, $1 \leqslant i \leqslant 3$. This subset has two places, namely, *upsite*$_i$ and *dwsite*$_i$. Initially, a token is put in place *upsite*$_i$. This token is used to specify the state of site i. When the token is in place *upsite*$_i$, it means that site i is in the state of "up"; conversely, when the token is in place *dwsite*$_i$, it means that site i is in the state of "down". There are two transitions in the subnet, namely, *tsite* f_i and *tsite* r_i. The first transition is associated with a rate of λ_s to specify the failure rate of site i, while the second transition is associated with a rate of μ_s to specify the recovery rate of site i after it fails. Thus, when site i is the state of "up" (with the token in place *upsite*$_i$), transition *tsite* f_i will be allowed to fire with rate λ_s, after which the token will be removed from place *upsite*$_i$ and put in place *dwsite*$_i$, meaning that a failure of site i has occurred and the new state of site i now is "down". Similarly, when site i is the state of "down" (with the token in place *dwsite*$_i$), transition *tsite* r_i will be allowed to fire with rate μ_s, after which the token will be removed from place *dwsite*$_i$ and put in place *upsite*$_i$, meaning that a recovery of site i has occurred and the new state of site i is again "up". Thus, the state of site i oscillates between "up" and "down" as failure and recover activities occur to site i during its lifetime.

We also use an SPN subnet to describe a link's failure and recovery activities. Fig. 2 shows such a subnet to describe a particular link, say, link ij. Here we use the subscript ij to refer to the (bidirectional) link between sites i and j. Initially, a token is put in place *uplink*$_{ij}$. A token in place *uplink*$_{ij}$ means that link ij is in the state of "up", while a token in place *dwlink*$_{ij}$ means that link ij is "down". Transition *tlink* f_{ij} is associated with a rate of λ_l to specify the failure rate of link ij, while transition *tlink* r_{ij} is associated with a rate of μ_l to specify the recovery rate of link ij. The meanings of places, transitions and tokens in Fig. 2 are similar to those in Fig. 1, except that they are used to describe the failure and recover activities of a link rather than of a site.

For a 3-site ring topology, there are three subnets for sites 1, 2 and 3, and three subnets for links 12, 23, and 13. Combining these 6 subnets together, we obtain a composite SPN which would allow us to compute the steady-state probability of the system in the 3-site ring topology under majority voting. There are altogether 64 states, of which 22 states are "good" states and 42 states are "bad" states.

Next we calculate the per state access time distribution for each "bad" state i, $F_i(t)$. To do this, we transform the composite SPN obtained above into an SPN with absorbing states in two steps.

(1) We add a new SPN subnet which allows us to determine if the system has reached a good state due to recovery events, given that the system initially starts from state i. Note that when a good state is reached, the system reaches an absorbing state. Fig. 3 shows this new SPN subnet. Initially, we put a token in place *no quorum*. The notation (tf, f) is used here to label the only transition tf in this subnet, denoting that transition tf is allowed to fire only if function f enables it. This enabling function f checks to see if a quorum exists whenever a recovery event occurs; it enables transition tf to fire only if a quorum exists, i.e., there exists a partition in which the number of "up" sites is a majority of the total number of sites. When transition tf fires, the token in place *no quorum* will be removed and put into place

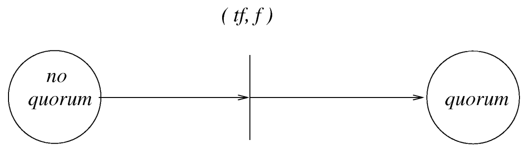

Fig. 3. SPN subnet for testing state changes.

quorum, which means that the system has reached an absorbing state from which no further activities will occur. Initially, the system is in state i in which function f disables transition tf from firing because no quorum exists in state i. Note that transition tf here is of the "immediate" type (represented by a solid vertical line as shown in Fig. 3) instead of the "timed" type (represented by a vertical bar as shown in Figs. 1 and 2), meaning that transition tf takes zero time to fire as soon as it is enabled by function f.

(2) We add a new enabling function with every failure or recovery transition shown in Figs. 1 and 2. This function disallows all failure/recovery transitions from firing if there is a token found in place *quorum*. Thus, all failure/recovery transitions will be disabled when the system reaches a good state (an absorbing state) in which a quorum exists. This step ensures that all failure/recovery activities stop when an absorbing state is reached.

The modified SPN model described above is now an SPN with absorbing states; it consists of 7 subnets, i.e., 3 subnets for sites, 3 subnets for links, and 1 subnet as shown in Fig. 3. To calculate the per state access time distribution for a "bad" state i, $F_i(t)$, we initialize this modified SPN model with state i. For example, suppose that state i is a bad state in which sites 1 and 2 are down and links 12 and 13 are also down (so a quorum does not exist), then $dwsite_1$, $dwsite_2$, $upsite_3$, $dwlink_{12}$, $dwlink_{13}$, $uplink_{23}$ each will be placed with a token initially (see Figs. 1 and 2).

After we initialize the modified SPN with absorbing states described above with a particular bad state i, we then run the SPN until a good state is reached, i.e., when a quorum exists. This is achieved by checking function f whenever a recovery event occurs in the SPNP program. When a good state is reached, the token in place *no quorum* will flow to place *quorum*, thus disallowing all further recovery/failure activities from occurring. Essentially the SPN runs into an absorbing state when one of good states is reached as a result of failed sites or links being recovered starting from state i. The program stops when such a good state is reached. Since the system evolves over time from the initial bad state i to one of the good states, we can compute the per state access time distribution $F_i(t)$ as the expected instantaneous reward rate by associating a reward rate of 1 with those markings in which place *quorum* (in Fig. 3) contains a token, and a reward rate of 0 with all other markings. This initialization and run procedure is repeated to collect the perstate access time distribution $F_i(t)$ for each of the 42 bad states in the system. The system access time distribution is then calculated based on Eq. (1).

4.2. Evaluation

Fig. 4 shows the access time distribution for a 3-site ring structure under majority voting as a function of different ratios of μ to λ, assuming $\lambda_s = \lambda_l = \lambda$ and $\mu_s = \mu_l = \mu$ for simplicity. The data in Fig. 4 are obtained by using the original SPN model described by Figs. 1 and 2 to first obtain the steady-state probability vector P_i's, then calculating $F_j(t)$ for each bad state j based on the modified SPN model described by Figs. 1, 2 and 3, and finally computing $F_{system}(t)$ based on Eq. (1). We purposely selected a higher λ value in Fig. 4 (i.e., λ is 1 per day) to show the effect of various deadlines and μ/λ ratios on response time distribution and system reliability. A lower value of λ will show the same trend except that the probability values will be closer to 1 and the effect will be less obvious. In Fig. 4, we observe that when $\mu \gg \lambda$, the probability that the access time is less than 1 second is high. It also correctly shows that when μ is high relative to λ, the average access time is low since most of the time the system will stay in one of the good states. To see how much improvement in response time distribution by using a 3-copy ring under majority voting instead of a single copy, we also show the response time distribution of a single copy in Fig. 5. The legend of that figure is the same as in Fig. 4. The response time distribution of a single copy is computed by

$$F_{system}(t) = \frac{\mu}{\mu+\lambda} + \frac{\lambda}{\mu+\lambda} \times (1 - e^{-\mu t}).$$

By comparing Figs. 4 and 5, we clearly see that a system with 3 copies based on majority voting can

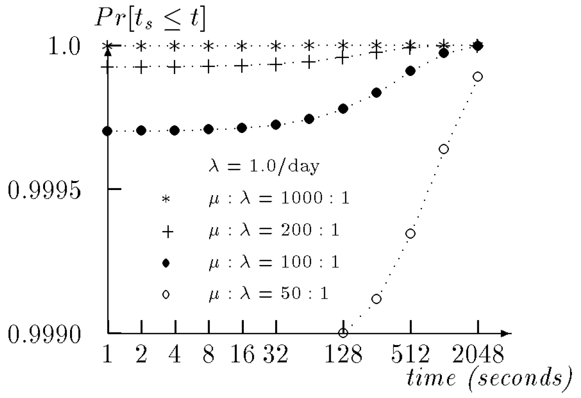

Fig. 4. Access time distribution $F_{system}(t)$ for 3 sites under majority voting.

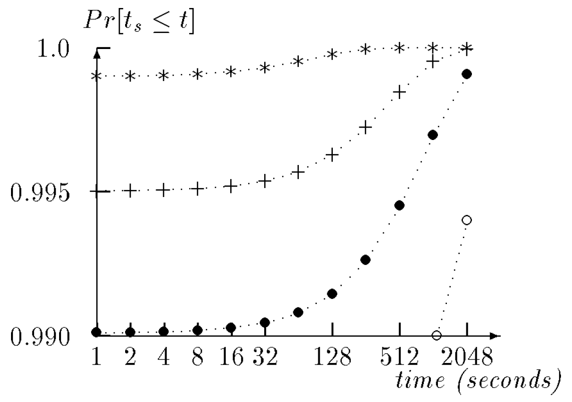

Fig. 5. Access time distribution $F_{system}(t)$ for a single site.

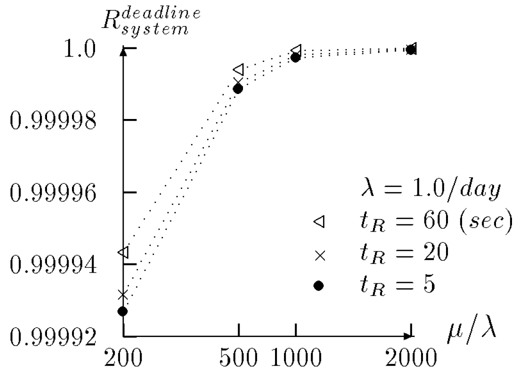

Fig. 6. System reliability $R_{system}^{deadline}$ under various deadlines t_R for 3 sites under majority voting.

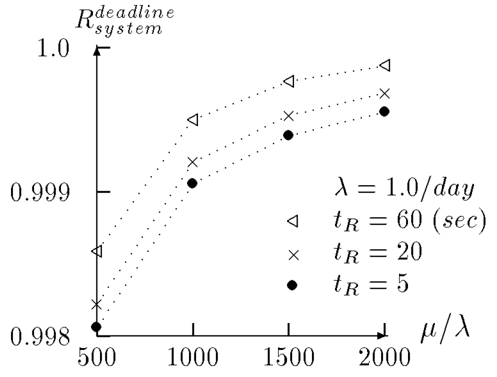

Fig. 7. System reliability $R_{system}^{deadline}$ under various deadlines t_R for a single site.

indeed greatly improve the access time over that with a single copy, based on the assumption that the failure rates of sites and links are the same.

Of course, whether the real-time requirement is satisfied or not cannot be judged by the access time distribution alone; it is largely dependent upon the real-time deadline requirement of the applications. Fig. 6 shows the application of the data in Fig. 4 to a real-time application subject to various deadlines t_R by computing $R_{system}^{deadline}$ based on Eq. (2) for the 3-site ring under majority voting. For comparison reasons, Fig. 7 shows the same for the single copy case. Again by comparing Figs. 6 and 7, we can see that the value of $R_{system}^{deadline}$ essentially depends on the combination of several conditions, including the ratio of the recovery rate to failure rate, the magnitude of the deadline, the number of copies in the system, and also the structure of the network connecting these copies together. It is clear that the reliability of voting schemes in distributed real-time systems can span a large range of values depending on these design and environmental parameters. The technique developed in this paper is generic, allowing us to easily analyze the effect of these design and environmental parameters.

References

[1] N.R. Adam, A new dynamic voting algorithm for distributed database systems, IEEE Trans. Knowledge Data Engrg. 6 (6) (1994) 470–478.
[2] I.R. Chen, D.C. Wang, Analysis of replicated data with repair dependency, Comput. J. 39 (9) (1996) 767–779.
[3] G. Ciardo, J.K. Muppala, K.S. Trivedi, SPNP: Stochastic Petri Net Package, in: Proc. 3rd Internat. Workshop on Petri Nets and Performance Models, Kyoto, Japan, 1989, pp. 142–151.
[4] H. Choi, V.G. Kulkarni, K.S. Trivedi, Markov regenerative stochastic Petri nets, Performance Evaluation 20 (1–3) (1994) 337–357.

[5] J.B. Dugan et al., Extended stochastic Petri nets: Applications and analysis, in: E. Gelenbe (Ed.), Performance 84, Elsevier Science Publishers, Amsterdam, 1984.

[6] A.W. Fu, Delay-optimal quorum consensus for distributed systems, IEEE Trans. Parallel Distrib. Systems 8 (1) (1997) 59–69.

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

[8] J.K. Muppala, S.P. Woolet, K.S. Trivedi, Real-time systems performance in the presence of failures, IEEE Comput. 24 (5) (1991) 37–47.

[9] D. Saha, S. Rangarajan, S.K. Tripathi, An analysis of the average message overhead in replica control protocols, IEEE Trans. Parallel Distrib. Systems (1996) 1026–1034.

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

[11] O. Wolfson, S. Jajodia, Y. Huang, An adaptive data replication algorithm, ACM Trans. Database Systems 22 (2) (1997) 255–314.