Calculating Performability Measures of Responsive Systems

Tatsuhiro Tsuchiya, Chang Chen, Yoshiaki Kakuda, and Tohru Kikuno

Department of Information and Computer Sciences
Faculty of Engineering Science, Osaka University
1-3, Machikaneyama-cho, Toyonaka-shi, Osaka 560, Japan
Phone: +81-6-850-6566, Fax: +81-6-850-6569
E-mail: { t-tutiya, kakuda, kikuno} @ics.es.osaka-u.ac.jp

Key Words: Responsive system, Multiprocessor system, Reconfiguration, Markov reward model

Abstract

Generally speaking in responsive systems, component processors may be isolated or added dynamically due to their faults and repairs. Such an action to change system configuration is called reconfiguration.

A reconfiguration action incurs loss of performance due to deadline violations since the action may entail an overhead for recovery. In order to exactly evaluate performability measures of such systems, this factor of reconfiguration should be taken into account. In most of the previous approaches, however, the overhead is not explicitly considered.

In this paper, we propose a framework for modeling and evaluating responsive systems which uses extended Markov reward models. The proposed model can represent the loss caused by a reconfiguration action. Then we give an example of modeling and performability evaluation. Finally based on condition for executing a reconfiguration action, we develop a good reconfiguration strategy to improve the performability of the system.

1 Introduction

For the most real-time applications such as factory automation systems, not only timing correctness but also fault-tolerance are indispensable features. Recently, along with development of such computer systems, the integration of both aspects has emerged as an important issue. Especially, for the purpose, the concept of responsive systems was introduced[1]. Responsive systems are defined as systems which integrate real-time systems and fault-tolerant systems. In this paper we concentrate on real-time multiprocessor systems whose component processors may be isolated out or added dynamically due to their faults and repairs. Such an action to change system configuration is called reconfiguration.

For dependability analysis of systems that are reconfigured due to failures and repairs of their components, Markov models have been usually used. Reconfiguration can have noticeable influence on not only dependability but also performance, since it depends on the system configuration such as the number of working processors. Meyer has introduced a framework, called *performability*, which incorporates system performance in the various configurations into dependability modeling[5]. This approach results in a Markov reward model. Smith et al. have defined a variety of performability measures based on it[8].

Muppala et al. has studied, for the first time, performability modeling of a reconfigurable real-time system[6]. However, this work does not consider the following natures of reconfiguration action. The reconfiguration action entails interruptions in the system's operation for data assurance, reassignment of tasks, synchronization of processors, etc. Occasionally, it may be necessary to reboot the system. Thus reconfiguration action itself affects task execution.

In traditional fault-tolerant systems such that no timing constraint is required, however, the effects of reconfiguration can be considered negligible since interruption caused by reconfiguration is very short with respect to the times the system being working stably. On the contrary, in real-time systems, interruption delays the completion times of tasks and may result in missing deadlines. Hence for modeling and analysis of real-time multiprocessor systems, the effects of reconfiguration have to be taken into consideration.

In the performability analysis[6] it is assumed that each task is completed in the same system configuration. On the other hand, the latest work studied by Meer et al. shows a new modeling including such effects[3]. They assume that non-negligible reconfiguration delay exists. Increasing the number of processors does not always enhance dependability or performance in this situation, since it also increases the total failure rate and the number of times of reconfiguration. It is thus important to make a proper decision whether to perform reconfiguration due to a repair or not. In [2, 3] the decision is made dynamically accord-

ing to the remaining mission time so as to optimize system performance. Such a flexible reconfiguration procedure is one of characteristic features of responsive systems.

In this paper, unlike [2, 3], however, we discuss a system whose mission time is not defined and expected to run as long as possible. Then we study the time when reconfiguration is performed in response to changes of computational environment such as the task arrival rate and influences of violating deadlines. We introduce a unified model which includes both failure-repair behavior and changes of the environment. On the unified model, we find the optimal strategies for responsive systems with respect to these two criteria. In addition, the trade-off between two strategies is also discussed.

The rest of the paper is organized as follows. In Section 2, the task and system model we studied in this paper are described. In Section 3, the extended Markov reward model is briefly introduced, and a failure-repair behavior model is proposed using it. Section 4 discusses changes of the environment. In Section 5, we propose a unified model which can represent both the failure-repair behavior and changes of the environment. In Section 6, we study performability evaluation using the proposed model. In addition, reconfiguration strategies that optimize two criteria are considered. Section 7 summarizes the paper and explains future work.

2 System and Task Model

In this paper we consider a reconfigurable real-time multiprocessor system, which consists of n identical processors. We assume that the processors in the system are subject to failure, and that the other components except processors are failure free. If a processor fails, then the failed processor is isolated from the system sequentially. Once repair of a failed processor is completed, it will be possible to connect the processor to the system again for operation by performing reconfiguration.

We assume that tasks arrive at the system according to a Poisson process. When a task arrives, it is queued until a processor becomes free. Then it is allocated to the processor on a first-come-first-served basis. When a free processor exists, the task is allocated to the processor and begin its execution immediately. Arriving tasks are required to be completed within deadline d. The execution times of the tasks are assumed to be exponentially distributed with rate μ. If there are i working processors, the system can be regarded as an M/M/i queueing system. When the task arrival rate is beyond the total service rate of the system, $i\mu$, the system cannot contribute to task execution positively. We assume that tasks that arrive at the system in such a state are lost.

In addition, we assume that a reconfiguration action entails interruption of operation. Then tasks that arrived in the midst of reconfiguration are also considered lost.

3 Modeling of Failure-Repair Behavior

A Markov reward model is obtained by associating reward rates with the states of the continuous-time Markov chain which represents the failure-repair behavior of the system. The expected accumulated reward or its variants are used as performability measures.

Let A be the state space of the continuous-Markov chain and r_i be the reward rate associated with state i. The expected accumulated reward until time t, $E[Y(t)]$, is defined as

$$E[Y(t)] = \sum_{i \in A} \int_0^t r_i P_i(t) d\tau$$

where $P_i(t)$ is the probability of being in state i at time t.

Though Markov reward models are a useful tool, they cannot assess the effects of actions or phenomena that take zero or very short time. The reason for this is that the reward is accumulated according to the sojourn time in the current state. However, we wish to evaluate the effect of a reconfiguration action, which takes very short time compared to the intervals the system staying in a same system configuration.

For this purpose, we use an extended Markov reward model introduced in [2], called *Extended Markov Reward Model*(EMRM). In the graphical representation in Figure 1, a normal state is drawn as a circle, while a new type of state is represented by a dot. Unlike normal states, no time is spent in the new states. The reward rate assigned to each of these states is accumulated instantly by passing through it. By using such states to represent reconfiguration actions, we can assess their effects.

Let A be a set of normal states and B be that of such special states in an EMRM. The expected accumulated reward until time t is thus obtained as follows.

$$E[Y(t)] = \sum_{i \in A} \int_0^t r_i P_i(\tau) d\tau + \sum_{j \in B} r_j E_j(t)$$

where $E_j(t)$ is the expected number of visits to state j until time t. We will use this measure as a criterion for evaluation of the reconfigurable real-time multiprocessor system.

Figure 1 shows a failure-repair behavior of the multiprocessor system. In this figure, circles and dots represent states, and arrows represent transitions. The number attached to each arrow indicates a rate with which its corresponding transition occurs. State NA_n means that the system comprises n working processors, and it is the initial state. When a processor fails in state $NA_i (1 \leq i \leq n)$, a reconfiguration action takes place and the current state changes to NA_{i-1} through RA_{i-1} immediately. State $RA_{i-1} (1 \leq i \leq n)$ represents reconfiguration due to a failure, while state $NA_i (0 \leq i \leq n-1)$ represents that i processors are working and one failed processor is being repaired. In the model we assume that failed components

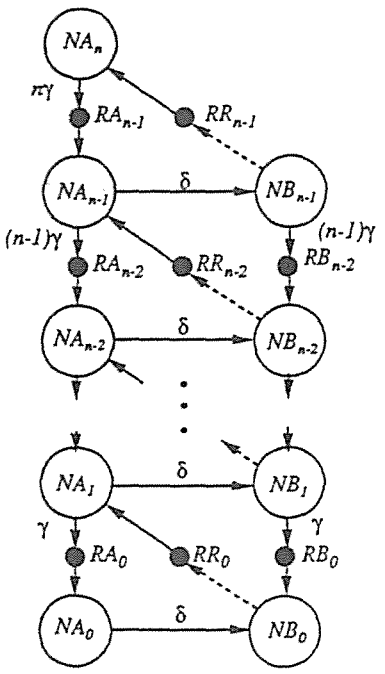

Figure 1: Failure-repair behavior of the system

share a single repair facility, and that it is occupied by only one failed processor until it is brought back to operation.

As stated above, we are interested in timing control of reconfiguration for adding a repaired processor to the system. To express the possibility of making a decision of whether to perform it or not, we introduce state $NB_i (0 \leq i \leq n-1)$ and a dotted arrow from NB_i to RR_i. Unlike NA_i, NB_i means that i processors are working properly and one failed processor has already been repaired but is not yet in operation. The decision of reconfiguration corresponds to the transition represented by the dotted arrow. If the decision of reconfiguration is made for adding a repaired processor, then the transition occurs instantly and the state changes from NB_i to NA_{i+1} through RR_i. RR_i represents reconfiguration action. The detailed procedure represented by dotted arrows will be explained in Section 5.

The question how to assign a reward rate to each state is here. By appropriate assignments, we can calculate several useful performability measures from an EMRM. We may select two measures for evaluation of responsive systems and will discuss them with concrete reward assignments.

4 Changes of Computational Environment

In most of previous works on dependability and/or performability evaluation, it is assumed that the computational environment — which is characterized by one or more elements, such as the task arrival rate, the effects of missing deadlines, etc — is constant. However [4] and [7] consider the changes of such an environment. In this paper, we also wish to model such changes. Though here we study only changes of the task arrival rate for ease of explanation, this approach can be applied to other elements.

As an example, we consider changes of computational environment represented by the Markov model illustrated in Figure 2. In this figure, circles represent phases among the environment varies. Arrows represent transitions between two phases. The number attached each arrow is a rate with which the transition occurs. The task arrival rate changes depending on the current phase. Let $\lambda_i (i = 1, 2, 3)$ be an arrival rate that corresponds to phase i. In the next section, we discuss control of reconfiguration depending on the current environment.

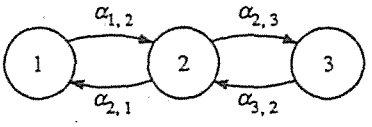

Figure 2: Changes of the environment

5 Unified Model and Reconfiguration Control

In this section, we present a new model that unifies the failure-repair model shown in Figure 1 with the environment model shown in Figure 2. On the new model, furthermore, we discuss control of reconfiguration.

This new unified model is illustrated in Figure 3. Intuitively, it is the product of these two models. State $NA_{i,j}$ in the figure means that the system is in state NA_i in Figure 1 and the environment is in state j in Figure 2.

A reconfiguration action itself incurs interruption of operation and increasing the number of processors also rises the probability of a failure occurring. We are interested in controlling reconfiguration for adding a repaired processor depending on failures of processors and the computation environment.

The key idea is as follows. It is obvious that the effects caused by performing reconfiguration depends on the current computational environment. Hence, when the environment is strict, namely, the task arrival rate is high, reconfiguration is deferred until the environment becomes slacker or the number of failed processors exceeds a certain prespecified number.

Now we define the control method of reconfiguration using a vector $\theta = (\theta_1, \theta_2, \theta_3)$. Each $\theta_j (j = 1, 2, 3)$ denotes a threshold value for performing reconfiguration in phase j, which represents the number of failure-free processors when the computation environment is in phase j. That is, for each (i, j) where i is less than θ_j, a dotted arrow from

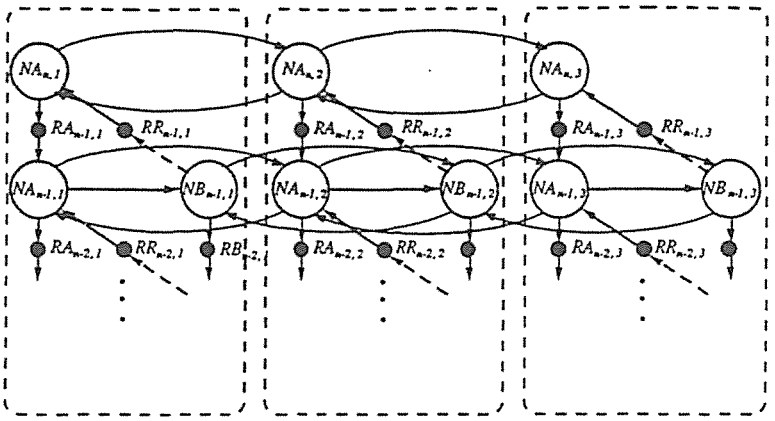

Figure 3: Unified model

$NB_{i,j}$ to $RR_{i,j}$ is activated. In other words, when the current state reaches $NB_{i,j}$ such that $(i,j) \in \{(i,j)|i < \theta_j\}$, a transition to $RR_{i,j}$ (which means a reconfiguration) occurs instantly. On the contrary, if being in state $NB_{i',j}$ such that $(i',j) \in \{(i,j)|i \geq \theta_j\}$, no reconfiguration action for adding a processor is performed.

6 Case Study

In this section, we calculate concrete values of a few criteria (that is, the number of tasks missing deadlines, average response time) and try to find an optimal reconfiguration strategy for each criterion. As an example, we take $n = 10$, $\delta = 0.001$ per hour, $\gamma = 0.1$ per hour, $\alpha_{1,2} = \alpha_{2,1} = \alpha_{2,3} = \alpha_{3,2} = 0.1$ per hour, $\mu = 7.0$ per second, $\lambda_1 = 10$ per second, $\lambda_2 = 20$ per second, $\lambda_3 = 30$ per second and $d = 1.0$ second. In addition, we assume that a reconfiguration action (either for recovering from a failure or for adding a processor) involves 30 seconds' interruption of operation on average. The system is assumed to be in state $NA_{10,1}$ at time 0.

6.1 Tasks missing deadlines

Here we select the expected number of tasks missing deadlines as a criterion to be optimized. To calculate the value using the model shown in Figure 3, the reward rates are assigned as follows. To each of states $NA_{i,j}$ and $NB_{i,j}$, which imply that the system is working stably, average number of tasks missing deadlines per time unit when the system is in the state is assigned as a state-dependent reward rate. The product of the arrival rate λ_j and the probability of a task exceeding its deadline when the system is in the state yields the reward rate. The probability, which is shown in Figure 4, can be obtained by using closed form solutions.

When the system is in state $NA_{i,j}$ or $NB_{i,j}$ and $\lambda_j < i\mu$, arriving tasks are lost since the system is overloaded. Hence, as a reward rate, λ_j is assigned to such states.

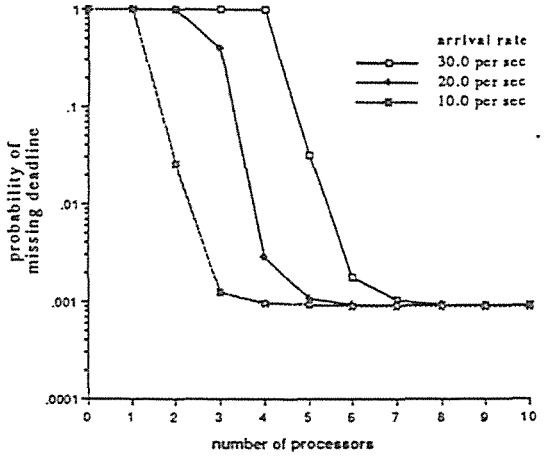

Figure 4: Probability of tasks missing deadline

Next, reward rates of $RA_{i,j}$, $RB_{i,j}$ and $RR_{i,j}$ in Figure 3 correspond to the number of tasks that miss deadlines due to reconfiguration. Since we have assumed that each reconfiguration action takes 30 seconds, the reward rates of these states are (30 seconds) $\times \lambda_j$.

We have investigated the relation between $E[Y(t)]$ and θ by calculating $E[Y(t)]$ for each θ. As a result, $\theta = (7,9,9)$ is found optimal for sufficiently long time t. Figure 5 shows the time-averaged number of tasks missing deadlines as a function of time t. These results are for the optimal control $(7,7,9)$ and $(10,10,10)$, which means that a repaired processor is added in all cases.

6.2 Trade-off between schedulability and response time

In responsive systems, not only satisfying timing constraints but also providing quick responses may be

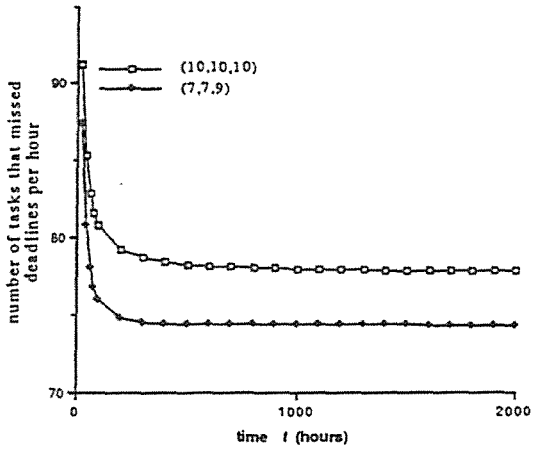

Figure 5: Time-averaged number of tasks missing deadlines

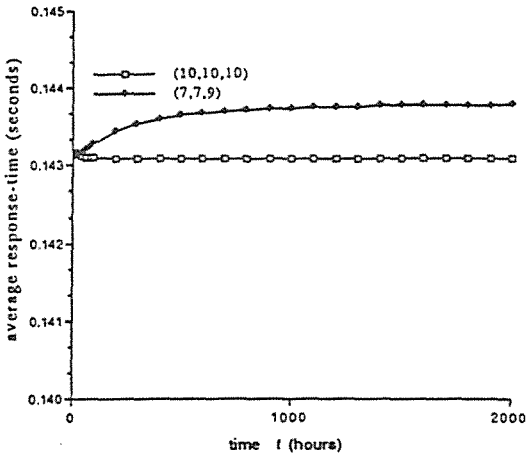

Figure 6: Average response time

strongly required. It is obvious that the reconfiguration strategy that minimizes average response time of executed tasks is $(10,10,10)$. On the other hand, the control method $(7,7,9)$, which minimizes the number of tasks missing deadlines, provides longer response time than that of $(10,10,10)$.

In order to investigate such a trade-off, we calculate average response time for each method as follows. The average response time of tasks that are executed until time t equals the quotient of the total time that these tasks stayed in the system divided by the total number of the tasks. The number of tasks that are executed until t can be calculated in the same way as that of tasks missing deadlines. After assigning the task arrival rate to a *up* state, namely, $NA_{i,j}(NB_{i,j})$ such that $(i,j) \in \{(i,j)|\lambda_j < i\mu\}$ as a reward rate, the value of $E[Y(t)]$ gives that number. On the other hand, to get the total time we assign to each up state the product of the task arrival rate and the average response time in the state. Then the value of $E[Y(t)]$ gives the total time. Average response time in each state can be calculated by using closed form solutions. The results are depicted in Figure 6.

7 Conclusion

In this paper, we have proposed a baseline model for performability evaluation of responsive multiprocessor systems, in which reconfiguration due to faults and repairs of processors has effect on task execution. Using the proposed model, we have shown how to derive a control method of reconfiguration that optimizes each performability measure. As future work, we plan to remove restrictions on both systems and tasks and to develop their modeling.

References

[1] M. Malek, "Responsive systems (A challenge for the nineties)", Proc. 16th Symp. on Microprocessing and Microprogramming, Keynote Address, Amsterdam, The Netherlands, North-Holland, Microprocessing and Micro programming 30, Aug. 1990, pp.9-16.

[2] H.de Meer, H.Mauser, "A modeling approach for dynamically reconfigurable systems", Proc. 2nd International Workshop on Responsive Computer Systems, Oct. 1992, pp.149-158.

[3] H.de Meer, K.S.Trivedi, M.D.Cin, "Guarded repair of dependable systems", Theoretical Computer Science, Vol.128, No.1-2, Jun. 1994, pp.179-210.

[4] R.G.Melhem, "Bi-level reconfigurations of fault tolerance arrays in bi-modal computational environments", Proc. FTCS-19, Jun. 1989, pp.488-195.

[5] J.F.Meyer,"On evaluating the performability of degradable computing systems", IEEE Trans. Computers, Vol.C-29, No.8, Aug. 1980, pp.720-731.

[6] J.K.Muppala, S.P.Woolet, K.S.Trivedi, "Real-time-systems performance in the presence of failures", COMPUTER Magazine, Vol.24, No. 5, May 1991, pp.37-47.

[7] K.G.Shin, C.M.Krishna, Y.H.Lee,"Optimal dynamic control of resources in a distributed system", IEEE Trans. Software Engineering, Vol.15, No.10, Oct. 1989, pp.1188-1197.

[8] R.M.Smith, K.S.Trivedi, A.V.Ramesh, "Performability analysis : measures, an algorithm, and a case study", IEEE Trans. Computers, Vol.C-37, No.4, Apr. 1988, pp.406-417.