Optimal Reconfiguration Strategy for a Degradable Multimodule Computing System

YANN-HANG LEE AND KANG G. SHIN

The University of Michigan, Ann Arbor, Michigan

Abstract. A new quantitative approach to the problem of reconfiguring a degradable multimodule system is presented. The approach is concerned with both assigning some modules for computation and arranging others for reliability. Conventionally, a fault-tolerant system performs reconfiguration only upon a subsystem failure. Since there exists an inherent trade-off between the computation capacity and fault tolerance of a multimodule computing system, the conventional approach is a *passive* action and does not yield a configuration that provides an optimal compromise for the trade-off. By using the expected total reward as the optimal criterion, the need and existence of an *active* reconfiguration strategy, in which the system reconfigures itself on the basis of not only the *occurrence of a failure* but also the *progression of the mission*, are shown.

Following the problem formulation, some important properties of an optimal reconfiguration strategy, which specify (i) the times at which the system should undergo reconfiguration and (ii) the configurations to which the system should change, are investigated. Then, the optimal reconfiguration problem is converted to integer nonlinear knapsack and fractional programming problems. The algorithms for solving these problems and a demonstrative example are given. Extensions of the optimal reconfiguration problem are also discussed.

Categories and Subject Descriptors: B.2.3 **[Arithmetic and Logic Structures]**: Reliability, Testing and Fault Tolerance; C.2.4 **[Computer-Communication Network]**: Distributed Systems; G.1.6 **[Numerical Analysis]**: Optimization—*integer programming*

General Terms: Performance, Reliability, Verification

Additional Key Words and Phrases: Degradable systems, dynamic failure, fractional programming, performability, reward

1. *Introduction*

Reconfiguration of a system is the process of changing an already existing system organization or the interconnections among its subsystems. In general, the system needs to perform reconfiguration for two reasons. The first reason is the dynamic variations of incoming tasks. The system reconfigures itself to match the special demands made by the incoming tasks and then executes the tasks more efficiently than with the previous configuration. In this case, reconfiguration depends on the

This work was supported in part by NASA under grants NAG-1-296 and NAG-1-492. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of NASA.

Authors' present addresses: Y.-H. Lee, IBM T. J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598. K. G. Shin, Real-Time Computing Laboratory, Division of Computer Science and Engineering, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109.

tasks to be executed. Reconfiguration of a system could be accomplished in many different ways, such as the change of partition [30] or word size [16]. The second reason is to make the system tolerate faults that may occur dynamically and randomly during the mission lifetime. Reconfiguration allows the system to remain operational, perhaps in a degraded mode, even in the case of subsystem failures. Typical examples of reconfiguration for fault tolerance include the handling of an extra stage in permutation networks [1] and the reconfiguration algorithm to maintain the system in a safe state [33].

For the purpose of fault tolerance, several authors have proposed principles and procedures for the reconfiguration of computer systems [2, 15, 17]. These procedures are all intended to make the system operational in the face of subsystem failures. Saheban and Friedman investigated the degradation of computation capability and diagnosability in terms of the number of switches to connect modules [25, 26]. They also proposed a methodology for the design of reconfigurable multimodule systems. Fortes and Raghavendra examined the design of reconfigurable array processors with VLSI chips and analyzed the improved reliability, performability, and computation capability and the additional hardware cost [9].

We classify the conventional system reconfigurations for fault tolerance discussed above as *passive* actions, since they are performed only upon detection of a failure. Moreover, they assume that there is only one configuration that the system will be changed to following each reconfiguration. For instance, the system degrades from an m module-parallel system to an $m - 1$ module-parallel system when a module failure occurs. Thus, there is no choice concerning when the reconfiguration should be performed and what configurations the system should switch to.

In this paper we are concerned with developing a quantitative method for design and analysis of the reconfiguration of a multimodule system. Particularly, using the expected total reward as the optimality criterion, we derive an optimal reconfiguration strategy with which the system can be optimally reconfigured during the entire mission lifetime to maximize the expected total reward.

The term *module* is used here to mean processor, memory, or bus. We assume that the environment and workload of the system are in the steady state throughout the mission lifetime. The system can assign a module to be *functioning*, *redundant*, or *spare*. Functioning modules execute computation tasks. Redundant modules are associated with functioning modules for verifying the correctness of computation results or masking erroneous results. Spare modules do not execute any useful task before they replace failed modules. Although there is no difference between functioning and associated redundant modules in the execution of tasks, they do have different purposes in a logical sense.

It is well known that the goals of reconfiguring a multimodule system are to enhance both *computation capacity* and *system reliability*. In most cases it is easy to see a trade-off between these two goals. For example, if there were no module failures and, therefore, no module redundancy were necessary, then the computation capacity would increase as the number of functioning modules increases. On the other hand, if module failures are allowed, then providing greater module redundancy enhances the system reliability at the expense of the computation capacity. When the number of available modules is finite, it becomes necessary to make a suitable compromise between the system reliability and the computation capacity. It is the optimal reconfiguration that is desired for the most suitable compromise in some sense.

From the standpoints of reliability and performance, it is natural to consider more than one possible way for reconfiguring multimodule systems. An extreme

example is whether the system should be configured to an m-module redundant system or to an m-parallel server system. Obviously, the former offers higher reliability, whereas the latter provides higher throughput. Some criterion is needed to judge the goodness of different configurations. Depending on the criterion, it is possible that the best configuration at a particular moment is no longer best at another moment (see the example in Section 4). In such a case, reconfiguration is needed even if there is no occurrence of failure; we term this *active reconfiguration.* In this case an optimal reconfiguration strategy must specify the optimal configurations for the entire mission lifetime. Active reconfiguration subsumes the conventional passive reconfiguration because reconfiguration can be done at *any* time, and not just at failure instants.

This paper is organized as follows: In Section 2 we introduce necessary concepts, notation, and terminology and define the set of feasible configurations when multimodule systems are to be reconfigured. Then, we develop a criterion, the expected total reward during the mission lifetime, which will be used for judging the goodness of configurations. The need for active reconfiguration is also justified in this section. Section 3 examines the problem of maximizing the expected total reward and the properties of an optimal reconfiguration strategy during the mission lifetime. Also, presented is an algorithm that uses a backward induction to determine an optimal reconfiguration strategy. Actual determination of the optimal configurations is the subject of Section 4, where we develop solution algorithms for two integer nonlinear optimization problems. An example is also provided to demonstrate both active and passive reconfiguration strategies. Concluding remarks are in Section 5.

2. *Reconfiguration Strategy*

Consider a multimodule system that begins its mission with m_0 identical operational modules. Let the mission lifetime be t_0, during which no system repair is allowed—that is, a nonrepairable system. We consider here only the failures that are caused by hardware faults and that, along with the progression of the mission, will trigger system reconfiguration. Transient and intermittent faults from which the system can recover through retry [19, 29] are not considered, since they do not entail reconfiguration.

As we shall see, the optimal configurations are generated *off line* in table form, therefore, the time overhead of performing (on-line) reconfiguration is simply the time required for switching tasks and setting up interconnection or routing. (E.g., the average reconfiguration time of FTMP has been measured to be 82 milliseconds [31].) This on-line time overhead has little impact on the determination of an optimal reconfiguration strategy, since in practice the system has to undergo only a few active reconfigurations during the mission lifetime. Consequently, the overhead of performing reconfiguration is assumed to be negligible in the following discussion.

2.1 Notation and Definitions. Classical reliability analyses treat the system failure probability as a function of the components' failure probabilities using combinatorial mathematics. These approaches neglect the effects of failure recovery overhead on executing tasks. These effects are significant, particularly when real-time applications are considered. For example, a delay in task execution due to failure recovery overhead can lead to increased system operational costs or even a system crash. For these reasons a reliability analysis, including the impact of failure handling on executing tasks, is more realistic and powerful than the classical

methods [18]. In addition, system performance should depend on the tasks completed by the system within its mission lifetime. Thus, it is natural and essential to consider the execution of tasks when reconfiguration strategies have to be determined.

Consider a set of tasks that are to be executed during the mission lifetime t_0. Group the tasks into k classes such that the tasks in the same class will have the same influence on the mission. More specifically, the system will gain the same reward for each task within a class when the task is completed successfully, and will suffer the same penalty if the task execution is unsuccessful or its completion is delayed. Although the pattern of the incoming tasks could change in reality, we assume for simplicity that the combined workload in each task class does not change throughout the entire mission, that is, the task arrival rate and the required computations in each task class are constant.[1]

Because of the failures of individual modules during the mission, when the *remaining mission lifetime* (RML) is $t \in [0, t_0]$, the system may have to operate with only $m(t) \in \{0, \ldots, m_0\}$ modules. Let $m_i(t)$ be the number of modules assigned to the class i tasks. Out of these $m_i(t)$ modules, $n_i(t)$ *computing clusters* are to be constructed. Each computing cluster consists of one functioning module and some redundant modules for reliability reasons. Let $r_i(t)$ be the total number of redundant modules used for the task class i. These redundant modules are used in constructing computing clusters as dyads (with one redundant module each), triads (with two redundant modules each), etc. For notational simplicity, we leave out the time dependency of m_i, n_i, and r_i in the rest of this paper as long as it does not cause any ambiguity. Obviously, $n_i + r_i = m_i$ and $\sum_{i=1}^{k} m_i \leq m$.[2] Since all computing clusters assigned to the same task class have to be homogeneous insofar as their capabilities of computation and fault tolerance are concerned, the r_i redundant modules for the task class i are assumed to be equally distributed over n_i computing clusters. Thus, for the task class i there are $r_i - n_i \lfloor r_i/n_i \rfloor$ computing clusters with $\lfloor r_i/n_i + 1 \rfloor$ redundant modules, and $n_i(1 + \lfloor r_i/n_i \rfloor) - r_i$ computing clusters with $\lfloor r_i/n_i \rfloor$ redundant modules, where $\lfloor x \rfloor$ is the greatest integer that is less than or equal to x.

Let Ω_m be the set of all feasible configurations of m modules and given as

$$\Omega_m \equiv \left\{((n_1, r_1), (n_2, r_2), \ldots, (n_k, r_k)) \mid \sum_{i=1}^{k} (n_i + r_i) \leq m, \right.$$
$$\left. n_i, r_i \in \mathbf{I}^+, r_i = 0 \text{ if } n_i = 0, i = 1, 2, \ldots, k\right\},$$

where $\mathbf{I}^+$ is the set of nonnegative integers. Also, denote the set of all configurations of the system by $\Omega \equiv \bigcup_{m=0}^{m_0} \Omega_m$.

Let γ_m: $\mathbf{R}^+ \rightarrow \Omega_m$ be a *configuration function* where $\mathbf{R}^+$ is the set of nonnegative real numbers. A *reconfiguration strategy* $RS_{t,m}$ is defined as

$$\mathrm{RS}_{t,m} \equiv \{\gamma_{\hat{m}}(\hat{t}) \mid \hat{t} \in [0, t], \hat{m} \in \{0, \ldots, m\}\}.$$

Hence, given a reconfiguration strategy RS_{t_0,m_0}, the system uses the configuration $\gamma_m(t) \in \mathrm{RS}_{t_0,m_0}$ where RML $= t$ and there are m operational modules available.

[1] As we shall discuss in Section 5, this assumption can be easily relaxed.

[2] The "less than" or "equal to" relationship is used to include the case in which the system has standby spare modules.

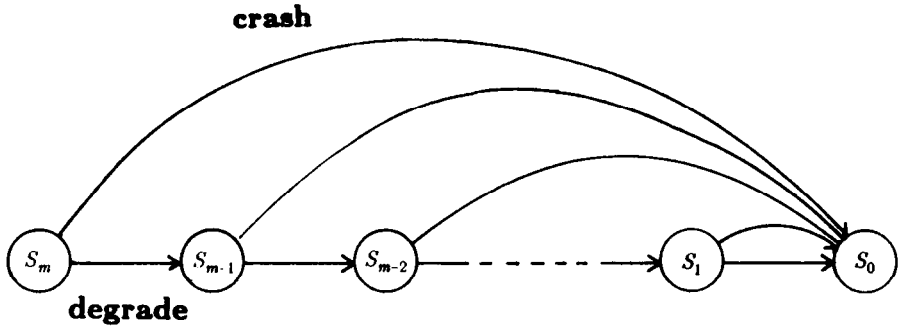

FIG. 1. A model of system degradation.

2.2 RECONFIGURATION MODEL. During the mission lifetime, system degradation is unavoidable owing to module failures. A simple model for system degradation is presented in Figure 1 where state S_m represents the availability of m operational modules. The transition from S_m to S_{m-1}, $m > 1$, implies that a module fails and system operation is recovered subsequently. However, failure of one module could be fatal to the system, for example, *coverage failure* [32] or *dynamic failure* [18], which results in loss of the whole mission. In such a case, the system transfers directly to the total system failure state S_0.

It is assumed that the times to failure for all modules are independently and identically distributed random variables. Distribution of the times to failure is assumed to be exponential with rate λ.

Define a stochastic process $M(t)$ that is equal to (i) the number of available modules when the system is operational with RML $= t$, or (ii) 0 if the system crashed when RML $> t$. This stochastic process is governed by the failure and recovery processes,[3] which in turn depend on the system configurations adopted during the mission lifetime.

Given a reconfiguration strategy RS_{t_0,m_0}, system configurations are represented by another stochastic process, called the *reconfiguration process* and denoted by $\mathrm{RF}_{t_0,m_0}(t)$, $t \in [0, t_0]$. $\mathrm{RF}_{t_0,m_0}(t)$ includes a configuration $\gamma_{M(t)}(t) \in \mathrm{RS}_{t_0,m_0}$ to be used at RML $= t$. The transitions between configurations within $\mathrm{RF}_{t_0,m_0}(t)$ depend on failure and recovery processes, as well as on the reconfiguration strategy RS_{t_0,m_0}. A sample path of RF_{t_0,m_0} is called a *configuration trajectory*, which represents a configuration history of the system. When RML $= t$ and the number of available modules is m, the system reconfigures itself from $\gamma_m(t)$ to $\gamma_{m-1}(t)$ if a recoverable failure occurs, or to $\gamma_m(t - \delta t)$ if there is no failure during δt. Note that, if $\gamma_m(t) = \gamma_m(\hat{t}) = \gamma_m(t - \delta t)$ for all $\hat{t} \in [t - \delta t, t]$ and if there is no failure during this δt, then the system uses the same configuration for the period δt.

For a system that is capable of graceful degradation and reconfiguration, Meyer's *performability* [22] is a useful measure of the system capability. Performability is a composite measure of performance and reliability that automatically takes into account the performance degradation due to component failures. To incorporate the concept of performability, two functions associated with configurations $\omega \in \Omega$ are introduced here. The first is a nonnegative and bounded function, $\rho(\omega)$, called the *reward rate*, which represents the average reward per unit time corresponding to the computation performed by the system with the configuration ω. This function

[3] By recovery process we mean general actions to be taken upon occurrence of a failure, for example, failure detection and masking, task recovery.

is analogous to the *reward structure* defined by Furchtgott [13], and the *reward function* defined by Donatiello and Iyer [8]. Thus, the total reward accumulated during the mission lifetime t_0 is a random variable and is given as

$$W_{t_0,m_0} = \int_0^{t_0} \rho(\gamma_{M(t)}(t))\, dt. \tag{1}$$

Note that the probability measure of W_{t_0,m_0} is the same as Meyer's performability [22].

The reward rates used here can not only express the achievement by the system, but also can be generalized to include the penalty due to the system crashes. A penalty rate, which should be negative, can be regarded as the average loss per unit time when the system is unable to perform any service. Since the reward and penalty rates only represent the relative indices between different configurations and system crashes, we simply let $\rho(\omega) = 0$ for $\omega \in \Omega_0$ and use the reward rate as a general term for the return of the system.

The second function, $\alpha(\omega)$, called the *crash probability*, represents the probability that a module failure causes a system crash. This function indicates the system's vulnerability to a module failure when the system configuration is ω. $\alpha(\omega)$ may be either the coverage failure [32], or the probability of dynamic failure [18] associated with the configuration ω.

The reward rate $\rho(\omega)$ is an implicit function of the number of computing clusters in each task class. We assume that redundancy in each computing cluster does not affect its performance.[4] On the other hand, $\alpha(\omega)$ depends on the number of redundant modules associated with each computing cluster. Since a configuration is completely specified by $\mathbf{n}$ and $\mathbf{r}$, where $\mathbf{n} = [n_1, n_2, \ldots, n_k]$ and $\mathbf{r} = [r_1, r_2, \ldots, r_k]$, the functions $\rho(\omega)$ and $\alpha(\omega)$ are used interchangeably with $\rho(\mathbf{n})$ and $\alpha(\mathbf{n}, \mathbf{r})$, respectively, in the rest of this paper.

Several authors have derived the distribution and the moments of performance variables for gracefully degradable systems under the restriction that system reconfiguration is allowed only upon failure [8, 13, 22]. Moreover, such a system has exactly one new configuration to choose from upon detection of a module failure. This, however, is an unnecessarily limiting factor (it might result in a configuration with less performance and reliability than the actual system's capacity), since there are usually several alternative configurations available for a system with multiple modules. For example, when there are four modules available upon failure, we can construct one 4-module redundant computing cluster, or one triad and one simplex, or two dyads, or four simplexes. Conventional reconfiguration concepts becomes more inappropriate when we consider the fact that the remaining mission lifetime has to play an important role in deciding on a new configuration. This fact can be seen easily with the following two simple cases: In the first the remaining mission lifetime is very short, in which case the probability of having failures is very small. Thus, the computation capability is more important than reliability for higher rewards. In the second case the remaining mission lifetime is long. In this case the probability of having failures becomes large and any good configuration should be able to tolerate module failures and minimize the possibility of a system crash.

In the discussion that follows we consider the problem of determining an optimal reconfiguration strategy, which maximizes the expected reward $E[W_{t_0,m_0}]$. This optimization problem is equivalent to controlling system reconfiguration such that

[4] As discussed in the Conclusion, this assumption can be relaxed.

the system follows a certain configuration trajectory to provide a maximum expected reward, even in the face of random failures.

3. *Derivation of Optimal Reconfiguration Strategy*

Let $\mathrm{RS}^*_{t_0,m_0} \equiv \{\gamma^*_m(t) \mid t \in [0, t_0],\ m \in \{0, \ldots, m_0\}\}$ represent the optimal reconfiguration strategy that maximizes the total expected reward $\mathrm{E}[W_{t_0,m_0}]$. Since the optimal configuration $\gamma^*_m(t)$ is completely specified by the values of n_i^* and r_i^*, the problem is to determine $n_i^*(t)$ and $r_i^*(t)$ for all $t \in [0, t_0]$ and $m \in \{0, \ldots, m_0\}$ that maximize $\mathrm{E}[W_{t_0,m_0}]$.

3.1 PROBLEM FORMULATION. On the basis of the assumption of an exponential distribution of failure occurrence, the probability of having a failure during a small interval δt is approximately equal to $\lambda\,\delta t$. If $\gamma_m(t + \delta t)$ is the configuration used at $\mathrm{RML} = t + \delta t$, then the cumulative expected reward at that time, $W_{t+\delta t,m}$, can be expressed as

$$W_{t+\delta t,m} = \begin{cases} \rho(\gamma_m(t+\delta t))\,\delta t + W_{t,m} & \text{with probability } 1 - m\lambda\,\delta t \\ \rho(\gamma_m(t+\delta t))\,\delta t & \text{with probability } \alpha(\gamma_m(t+\delta t))m\lambda\,\delta t \\ \rho(\gamma_m(t+\delta t))\,\delta t + W_{t,m-1} & \text{with probability } (1 - \alpha(\gamma_m(t+\delta t)))m\lambda\,\delta t, \end{cases} \tag{2}$$

where $m \in \{1, \ldots, m_0\}$ and $t \in [0, t_0]$. Thus, a recursive expression for the expected reward is derived as follows:

$$\begin{aligned} \mathrm{E}[W_{t+\delta t,m}] = {} & (1 - m\lambda\,\delta t)\mathrm{E}[\rho(\gamma_m(t+\delta t))\delta t + W_{t,m}] \\ & + \alpha(\gamma_m(t+\delta t))m\lambda\,\delta t(\rho(\gamma_m(t+\delta t))\delta t) \\ & + (1 - \alpha(\gamma_m(t+\delta t)))m\lambda\,\delta t\mathrm{E}[\rho(\gamma_m(t+\delta t))\,\delta t + W_{t,m-1}]. \end{aligned} \tag{3}$$

On the basis of the exponential distribution assumption, at any moment there is at most one occurrence of failure, implying that the maximum jump in $M(t)$ is one. Because of this, when the system reconfigures itself into a new configuration $\gamma_m(t)$ from $\gamma_m(t + \delta t)$ or from $\gamma_{m+1}(t + \delta t)$, it must be in this configuration for a nonzero mission interval; otherwise, there is no need to move in that configuration at all. Let the optimal strategy $\mathrm{RS}^*_{t^+,m} \equiv \lim_{\delta t \to 0^+} \mathrm{RS}^*_{t+\delta t,m}$ and the configuration $\gamma_m(t^+) \equiv \lim_{\delta t \to 0^+} \gamma_m(t + \delta t)$. Then the following lemma gives a recursive representation of the optimal reconfiguration strategy.

LEMMA 1

$$RS^*_{t^+,m} = \{\gamma^*_m(t^+)\} \cup RS^*_{t,m} \cup RS^*_{t,m-1}.$$

PROOF. Suppose that the configuration chosen at $\mathrm{RML} = t + \delta t$ by a reconfiguration strategy is used at least for the period δt if there is no occurrence of module failure, and also assume that there could be at most one failure occurrence during δt. Let δW be the reward gained during this δt. When there is no failure, $\mathrm{E}[W_{t+\delta t,m}] = \mathrm{E}[W_{t,m}] + \mathrm{E}[\delta W]$. Thus, to have a maximum expected reward in this case, $\mathrm{RS}^*_{t+\delta t,m} \supset \{\gamma^*_m(t + \delta t)\} \cup \mathrm{RS}^*_{t,m}$. Similarly, $\mathrm{RS}^*_{t+\delta t,m} \supset \{\gamma^*_m(t + \delta t)\} \cup \mathrm{RS}^*_{t,m-1}$ when we consider the case of one failure occurrence. On the other hand, by definition, the system does not use any configuration other than (i) $\gamma^*_m(t + \delta t)$ and (ii) the configurations belonging to $\mathrm{RS}^*_{t,m}$ and $\mathrm{RS}^*_{t,m-1}$. Thus, Lemma 1 follows immediately, since the assumption at the beginning of this proof becomes true as $\delta t \to 0^+$. □

The above lemma can be used to determine recursively the optimal reconfiguration strategy. With the knowledge of $RS^*_{t,m}$, $RS^*_{t,m-1}$, and their respective expected rewards $E[W^*_{t,m}]$ and $E[W^*_{t,m-1}]$, the optimal configuration $\gamma^*_m(t^+)$ can be determined by Theorem 1 below.

THEOREM 1. *$\gamma^*_m(t^+)$ maximizes $J_{t,m}(\omega)$ for $\omega \in \Omega_m$, which is defined by*

$$J_{t,m}(\omega) \equiv \rho(\omega) - \alpha(\omega)m\lambda E[W^*_{t,m-1}]. \tag{4}$$

PROOF. Suppose $\gamma_m(t + \delta t)$ is applied at RML $= t + \delta t$ for the period δt and then either $RS^*_{t,m}$ or $RS^*_{t,m-1}$ is used for the remaining mission lifetime t. Then, from eq. (3) we get

$$\begin{aligned} E[W_{t+\delta t,m}] - E[W^*_{t,m}] = {} & \rho(\gamma_m(t + \delta t))\,\delta t - \alpha(\gamma_m(t + \delta t))m\lambda\,\delta t\; E[W^*_{t,m-1}] \\ & - m\lambda\,\delta t\; E[W^*_{t,m} - W^*_{t,m-1}]. \end{aligned} \tag{5}$$

Notice that the only terms in eq. (5) depending on $\gamma_m(t + \delta t)$ are

$$\rho(\gamma_m(t + \delta t)) - \alpha(\gamma_m(t + \delta_t))m\lambda E[W^*_{t,m-1}].$$

Thus, $\gamma^*_m(t + \delta t)$ maximizes the above expression over $\omega \in \Omega_m$ for any $\delta t > 0$. Combining this with Lemma 1 proves the fact that $\gamma^*_m(t^+)$ maximizes $J_{t,m}(\omega)$. □

We define the optimal reconfiguration problem for deriving $\gamma^*_m(t^+)$ in the following form:

$$\begin{aligned} \text{OR Problem:} \quad & \underset{\mathbf{n},\mathbf{r}}{\text{maximize}} \quad J_{t,m}(\mathbf{n}, \mathbf{r}) = \rho(\mathbf{n}) - \alpha(\mathbf{n}, \mathbf{r})m\lambda E[W^*_{t,m-1}] \\ & \text{subject to} \quad \sum_{i=1}^{k} (n_i + r_i) \le m, \\ & \qquad n_i,\ r_i \in \mathbf{I}^+ \\ & \qquad\quad \text{for} \quad i = 1, 2, \ldots, k, \qquad r_i = 0 \quad \text{when} \quad n_i = 0. \end{aligned}$$

Though Lemma 1 and Theorem 1 provide a recursive relationship necessary for obtaining $RS^*_{t_0,m_0}$, the relationship does not yield an acceptable solution. It requires a solution to the OR problem for all $t \in [0, t_0]$ (thus infinitely many times). As a remedy for this difficulty, in the next section we convert the OR problem so that it has to be solved only when $\gamma^*_m(t^+) \neq \gamma^*_m(t)$, instead of for all $t \in [0, t_0]$.

3.2 PROBLEM TRANSFORMATION. Once an optimal configuration at a moment during the mission is determined, it will be used for a nonzero mission interval. Given m, it is therefore natural to look for the *switch times* (in terms of the remaining mission lifetime) at which the optimal configuration is changed to maximize the total expected reward. That is, we only have to solve the OR problem at those switch times instead of for all $t \in [0, t_0]$. Let $s^j_m \in [0, t_0]$, $j = 0, 1, 2, \ldots$, $m = 1, 2, \ldots, m_0$, be the switch times for $RS^*_{t_0,m_0}$, where $s^0_m = 0$. Then, by definition, $\gamma^*_m(s^j_m) \neq \gamma^*_m(s^{j+}_m) \equiv \lim_{\delta t \to 0^+} \gamma^*_m(s^j_m + \delta t)$ for $j \ge 1$. As shown in Figure 2 for the case of no module failure, when the remaining mission lifetime decreases to s^j_m, the system should reconfigure itself from $\gamma^*_m(s^{j+1}_m)$ to $\gamma^*_m(s^j_m)$ and then keep the same configuration until the remaining mission lifetime is reduced to s^{j-1}_m.

From eq. (5), for two confiruations ω_1 and ω_2, both in Ω_m with $\alpha(\omega_1) = \alpha(\omega_2)$, we say that ω_1 is *better* than ω_2 if $\rho(\omega_1) > \rho(\omega_2)$. On the other hand, if $\alpha(\omega_1) \neq \alpha(\omega_2)$, we need a function $A_m(\omega_1, \omega_2)$, which is defined as follows:

$$A_m(\omega_1, \omega_2) = A_m(\omega_2, \omega_1) = \frac{\rho(\omega_1) - \rho(\omega_2)}{m\alpha(\omega_1) - m\alpha(\omega_2)}.$$

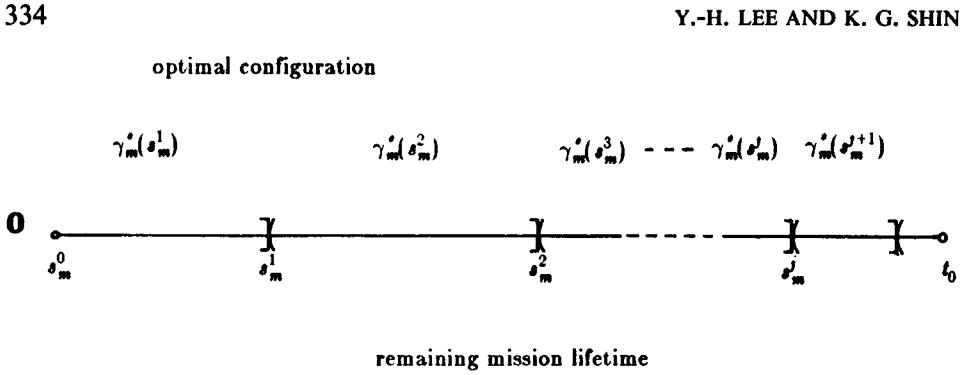

FIG. 2. Switch times and optimal configurations in case of no module failure.

Also, define the following two sets:

$$L_m(s^j_m) \equiv \{\hat{\omega} \in \Omega_m \mid \alpha(\hat{\omega}) < \alpha(\gamma^*_m(s^j_m))\},$$
$$G_m(s^j_m) \equiv \{\hat{\omega} \in \Omega_m \mid \alpha(\hat{\omega}) > \alpha(\gamma^*_m(s^j_m))\}.$$

$L_m(s^j_m)$ and $G_m(s^j_m)$ represent sets of configurations with the crash probabilities less than and greater than that of the optimal configuration at RML = s^j_m, respectively. Then, the following theorem elucidates a useful property of the switch times s^j_m.

THEOREM 2. *For all remaining mission lifetime* $t \in (s^{j-1}_m, s^j_m]$

$$\min_{\omega \in L_m(s^j_m)} A_m(\gamma^*_m(s^j_m), \omega) \geq \lambda E[W^*_{t,m-1}] \geq \max_{\omega \in G_m(s^j_m)} A_m(\gamma^*_m(s^j_m), \omega). \tag{6}$$

PROOF. Since $\gamma^*_m(t) = \gamma^*_m(s^j_m)$ for all $t \in (s^{j-1}_m, s^j_m]$, we have the following relationship from Theorem 1:

$$\rho(\gamma^*_m(s^j_m)) - \alpha(\gamma^*_m(s^j_m))m\lambda E[W^*_{t,m-1}] \geq \rho(\omega) - \alpha(\omega)m\lambda E[W^*_{t,m-1}] \quad \text{for all } \omega \in \Omega_m.$$

Therefore,

$$\frac{\rho(\gamma^*_m(s^j_m)) - \rho(\omega)}{m\alpha(\gamma^*_m(s^j_m)) - m\alpha(\omega)} \geq \lambda E[W^*_{t,m-1}] \quad \text{for all } \omega \text{ satisfying } \alpha(\omega) < \alpha(\gamma^*_m(t)),$$

$$\frac{\rho(\gamma^*_m(s^j_m)) - \rho(\omega)}{m\alpha(\gamma^*_m(s^j_m)) - m\alpha(\omega)} \leq \lambda E[W^*_{t,m-1}] \quad \text{for all } \omega \text{ satisfying } \alpha(\omega) > \alpha(\gamma^*_m(t)). \quad \square$$

Note that $E[W^*_{t,m-1}]$ is a function of t. However, the function $A_m(\gamma^*_m(s^j_m), \omega)$ is dependent upon configurations only. Once the optimal configuration $\gamma^*_m(s^j_m)$ is known, $\min_{\omega \in L_m(s^j_m)} A_m(\gamma^*_m(s^j_m), \omega)$ and $\max_{\omega \in G_m(s^j_m)} A_m(\gamma^*_m(s^j_m), \omega)$ can be determined. The only computation needed to determine s^j_m is to calculate $E[W^*_{t,m-1}]$ recursively by eq. (5) until the inequality (6) becomes invalid, that is, $\gamma^*_m(s^j_m)$ is no longer optimal when RML > s^j_m. Thus, Theorem 2 provides the solution for s^j_m, which is the infimum of $\{t \mid t_0 > t > s^{j-1}_m$ and t that does not satisfy (6)$\}$. If the inequality (6) is valid for all $t \in (s^{j-1}_m, t_0)$ then s^j_m does not exist. Theorem 2 also presents a special property of an optimal configuration; with an optimal configuration $\gamma^*_m(s^j_m)$, the function $A_m(\gamma^*_m(s^j_m), \omega)$ partitions Ω_m into two sets, $L_m(s^j_m)$ and $G_m(s^j_m)$, as shown above. An arbitrary configuration without this property can never be an optimal configuration regardless of the remaining mission lifetime. It is important to note that s^j_m represents the reconfiguration time independent of the presence or absence of a module failure. Moreover, the solution to the OR

problem is embedded in the process of solving for the s_m^j. To explore this property and make full use of it, we present the following lemmas and theorem.

LEMMA 2. *$E[W_{t,m}^*]$ is continuous and nondecreasing in t.*

PROOF. Dividing both sides of eq. (5) by δt and taking limits as $\delta t \to 0$, we can write a first-order differential equation of $\mathrm{E}[W_{t,m}]$ with respect to t as follows:

$$\frac{d\mathrm{E}[W_{t,m}]}{dt} + m\lambda \mathrm{E}[W_{t,m}] = \rho(\gamma_m(t^+)) + m\lambda(1 - \alpha(\gamma_m(t^+))\mathrm{E}[W_{t,m-1}]. \quad (7)$$

Since $\rho(\gamma_m(t))$ is bounded for all $\gamma_m(t) \in \Omega_m$, so is $\mathrm{E}[W_{t,m}]$ for all $t \in [0, t_0]$ and $m \in \{0, \ldots, m_0\}$. Also, $0 \le \alpha(\gamma_m(t)) \le 1$ for all $\gamma_m(t) \in \Omega_m$. Thus, $d\mathrm{E}[W_{t,m}]/dt$ is finite, implying the continuity of $\mathrm{E}[W_{t,m}]$ and, therefore, the continuity of $E[W_{t,m}^*]$.

To prove that $\mathrm{E}[W_{t,m}^*]$ is nondecreasing in t, we must show that

$$\mathrm{E}[W_{t+\Delta t,m}^*] \ge \mathrm{E}[W_{t,m}^*] \quad \text{for all} \quad \Delta t \ge 0.$$

Let

$$\mathrm{RS}_{t+\Delta t,m} \equiv \{\gamma_m(t) \mid 0 \le \hat{t} \le t + \Delta t,\ \gamma_m(\hat{t}) = \gamma_m^*(\hat{t} - \Delta t) \text{ if } \hat{t} \ge \Delta t, \text{ or } \gamma_m^*(0^+) \text{ if } \hat{t} < \Delta t\}.$$

Under the strategy $\mathrm{RS}_{t+\Delta t,m}$, we use the configuration $\gamma_m^*(t)$ when $\mathrm{RML} = t + \Delta t$ and $\gamma_m^*(0^+)$ when $\mathrm{RML} \le \Delta t$. Then, it is easy to see that the expected reward based on $\mathrm{RS}_{t+\Delta t,m}$ is larger than $\mathrm{E}[W_{t,m}^*]$. Thus, $\mathrm{E}[W_{t+\Delta t,m}^*] \ge \mathrm{E}[W_{t,m}^*]$. □

LEMMA 3. *For real numbers a_i and b_i, $i = 1, 2, 3$, where $b_1 > b_2 > b_3$, $(a_1 - a_2)/(b_1 - b_2) \le (a_1 - a_3)/(b_1 - b_3)$ if and only if $(a_1 - a_2)/(b_1 - b_2) \le (a_2 - a_3)/(b_2 - b_3)$. Also, $(a_1 - a_2)/(b_1 - b_2) \ge (a_1 - a_3)/(b_1 - b_3)$ if and only if $(a_1 - a_3)/(b_1 - b_3) \ge (a_2 - a_3)/(b_2 - b_3)$.*

PROOF. The proof is trivial and thus omitted. □

Let

$$\Phi_m(s_m^j) \equiv \{\hat{\omega} \mid \hat{\omega} \in L_m(s_m^j),\ A_m(\gamma_m^*(s_m^j), \hat{\omega}) = \min_{\omega \in L_m(s_m^j)} A_m(\gamma_m^*(s_m^j), \omega)\}.$$

Then, $\Phi_m(s_m^j) \ne \phi$ if $L_m(s_m^j) \ne \phi$. The following theorem gives a recursive relationship between $\gamma_m^*(s_m^j)$ and $\gamma_m^*(s_m^{j+1})$ when $s_m^j < \infty$.

THEOREM 3. *If $s_m^j < t_0$, then $\gamma_m^*(s_m^{j+1}) \in \Phi_m(s_m^j)$, and $\alpha(\gamma_m^*(s_m^{j+1})) \le \alpha(\hat{\omega})$ for all $\hat{\omega} \in \Phi_m(s_m^j)$.*

PROOF. From the definition of switch time, $\gamma_m^*(s_m^{j+1})$ has to be considered if $s_m^j < t_0$. First, we prove that $\alpha(\gamma_m^*(s_m^{j+1})) < \alpha(\gamma_m^*(s_m^j))$. Since $\mathrm{E}[W_{t,m}^*]$ is nondecreasing, we have

$$\max_{\omega \in G_m(s_m^j)} A_m(\gamma_m^*(s_m^j), \omega) \le \lambda \mathrm{E}[W_{s_m^j,m-1}^*] \le \lambda \mathrm{E}[W_{t,m-1}^*],$$

for $t \in (s_m^j, s_m^{j+1}]$. Thus, $J_{t,m}(\gamma_m^*(s_m^j)) \ge J_{t,m}(\omega)$ for all $\omega \in G_m(s_m^j)$. If $\alpha(\gamma_m^*(s_m^{j+1})) > \alpha(\gamma_m^*(s_m^j))$, then $\gamma_m^*(s_m^j)$ is also an optimal configuration at $t \in (s_m^j, s_m^{j+1}]$, which is contradictory to the definition of switch time. Therefore, $\alpha(\gamma_m^*(s_m^{j+1})) < \alpha(\gamma_m^*(s_m^j))$. This result also implies $\Phi_m(s_m^j) \ne \phi$.

Next, we prove $\gamma^*_m(s^{j+1}_m) \in \Phi_m(s^j_m)$. When there is one and only one $\omega \in L_m(s^j_m)$, the theorem automatically holds in case of $s^j_m < t_0$, since $\gamma^*_m(s^{j+1}_m)$ exists and $\alpha(\gamma^*_m(s^{j+1}_m))$ cannot be larger than $\alpha(\gamma^*_m(s^j_m))$. Consider a configuration

$$\omega \in \{\hat{\omega} \mid \alpha(\hat{\omega}) < \alpha(\gamma^*_m(s^{j+1}_m)) < \alpha(\gamma^*_m(s^j_m)),\ \hat{\omega} \in \Omega_m\}.$$

Since $\gamma^*_m(s^{j+1}_m)$ is optimal and $E[W^*_{t,m}]$ is continuous (and so is $E[W^*_{t,m-1}]$), the following order can be obtained from Theorem 2:

$$A_m(\gamma^*_m(s^{j+1}_m), \omega) \geq \lambda E[W^*_{t,m-1}] \geq A_m(\gamma^*_m(s^{j+1}_m), \gamma^*_m(s^j_m)) \quad \text{where} \quad t \in (s^j_m, s^{j+1}_m].$$

Applying Lemma 3, we have $A_m(\gamma^*_m(s^j_m), \omega) \geq A_m(\gamma^*_m(s^{j+1}_m), \gamma^*_m(s^j_m))$.

For $\omega \in \Omega_m$ satisfying $\alpha(\gamma^*_m(s^{j+1}_m)) < \alpha(\omega) < \alpha(\gamma^*_m(s^j_m))$, $A_m(\gamma^*_m(s^{j+1}_m), \gamma^*_m(s^j_m)) \geq E[W^*_{s^j_m,m-1}]$ and $E[W^*_{t,m-1}] \geq A_m(\gamma^*_m(s^{j+1}_m), \omega)$ for all $t \in (s^j_m, s^{j+1}_m]$. The continuity in $E[W^*_{t,m-1}]$ implies $A_m(\gamma^*_m(s^{j+1}_m), \gamma^*_m(s^j_m)) \geq A_m(\gamma^*_m(s^{j+1}_m), \omega)$. From the second part of Lemma 3, we also obtain $A_m(\gamma^*_m(s^j_m), \omega) \geq A_m(\gamma^*_m(s^{j+1}_m), \gamma^*_m(s^j_m))$. Thus, $\gamma^*_m(s^{j+1}_m) \in \Phi_m(s^j_m)$.

If there is only one configuration in $\Phi_m(s^j_m)$, the theorem is proved. Suppose there exist two configurations, ω_1 and ω_2; then $A_m(\gamma^*_m(s^j_m), \omega_1) = A_m(\gamma^*_m(s^j_m), \omega_2) = A_m(\omega_1, \omega_2)$. To satisfy Theorem 2, the configuration with the smallest $\alpha(\omega)$, $\omega \in \Phi_m(s^j_m)$, becomes optimal for all $t \in (s^j_m, s^{j+1}_m]$. □

Theorem 3 indicates that, while solving for s^{j+1}_m in which $A_m(\gamma^*_m(s^j_m), \omega)$ has to be minimized over $L_m(s^j_m)$, the optimal configurations $\gamma^*_m(s^{j+1}_m)$ can be obtained simultaneously. Also, we can exclude the configurations $\omega \in G_m(s^j_m)$ during the determination of s^{j+1}_m and $\gamma^*_m(s^{j+1}_m)$. Note that even if $\Phi_m(s^j_m)$ is not empty, $\gamma^*_m(s^{j+1}_m)$ is not always needed since s^j_m may be greater than t_0. When there is more than one element in $\Phi_m(s^j_m)$ and $s^j_m < t_0$, the configuration with the smallest crash probability in $\Phi_m(s^j_m)$ is optimal for the interval $(s^j_m, s^{j+1}_m]$. Note, however, that all other configurations in $\Phi_m(s^j_m)$ are optimal only at $\lim_{\delta t \to 0^+}(s^j_m + \delta t)$. Theorem 3 also provides additional properties concerning the optimal reconfiguration strategy as shown in the following corollaries.

COROLLARY 1. *$\alpha(\gamma^*_m(s^i_m)) < \alpha(\gamma^*_m(s^j_m))$ and $\rho(\gamma^*_m(s^i_m)) < \rho(\gamma^*_m(s^j_m))$ for all $i > j$.*

PROOF. In Theorem 3, we proved $\alpha(\gamma^*_m(s^{j+1}_m)) < \alpha(\gamma^*_m(s^j_m))$. Since $A_m(\gamma^*_m(s^j_m), \gamma^*_m(s^{j+1}_m)) \geq E[W^*_{s^j_m,m-1}] > 0$, we have $\rho(\gamma^*_m(s^{j+1}_m)) < \rho(\gamma^*_m(s^j_m))$. □

Definition 1. A reconfiguration process is said to be *acyclic* if for $t_1 \neq t_2$, $\gamma_{M(t_1)}(t_1) = \gamma_{M(t_2)}(t_2)$ implies $\gamma_{M(t_1)}(t_1) = \gamma_{M(t)}(t) = \gamma_{M(t_2)}(t_2)$ for all $t \in [t_1, t_2]$.

From Corollary 1, $\gamma^*_m(s^i_m) \neq \gamma^*_m(s^j_m)$ for all $i \neq j$. Since no repair is allowed during the mission, the system degrades from the m module system to the $m-1$ module system in case of a module failure. Thus, we immediately get the following Corollary 2.

COROLLARY 2. *The reconfiguration process based on $RS^*_{t_0,m_0}$, denoted by $RF^*_{t_0,m_0}$, is acyclic.*

COROLLARY 3. *For all $m \in \{0, \ldots, m_0\}$ and $j \geq 0$, if $s^j_m < t_0$, then*

$$\frac{\rho(\gamma^*_m(s^{j+1}_m))}{\alpha(\gamma^*_m(s^{j+1}_m))} \geq \frac{\rho(\gamma^*_m(s^j_m))}{\alpha(\gamma^*_m(s^j_m))}.$$

PROOF. Consider a remaining mission lifetime $t \in (s^j_m, s^{j+1}_m]$. Since $E[W^*_{t,m}]$ is nondecreasing and continuous in t, $dE[W^*_{t,m}]/dt \geq 0$, and by rearranging eq. (7)

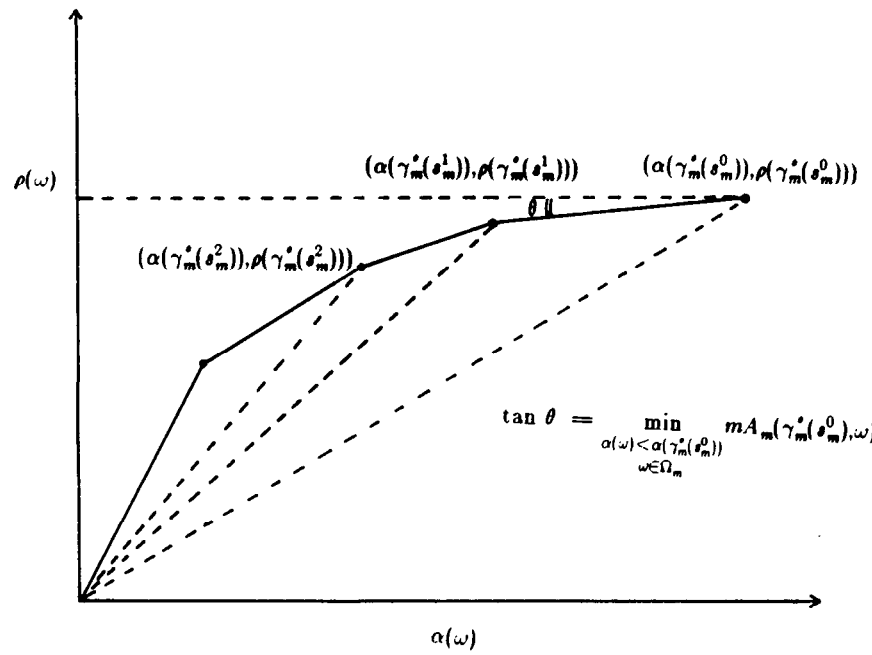

FIG. 3. The coordinates of optimal configurations.

we have

$$\frac{\rho(\gamma_m^*(s_m^{j+1}))}{m\alpha(\gamma_m^*(s_m^{j+1}))} \geq \lambda E[W_{t,m-1}^*].$$

From Theorem 3, we get

$$\lambda E[W_{t,m-1}^*] \geq A_m(\gamma_m^*(s_m^j),\ \gamma_m^*(s_m^{j+1})).$$

Thus,

$$\frac{\rho(\gamma_m^*(s_m^{j+1}))}{\alpha(\gamma_m^*(s_m^{j+1}))} \geq \frac{\rho(\gamma_m^*(s_m^j))}{\alpha(\gamma_m^*(s_m^j))}. \qquad \square$$

Definition 2. The *coordinates* of a configuration ω are defined as $(\alpha(\omega), \rho(\omega))$ in an x–y plane.

Using Figure 3, we can explain the relationships obtained from Corollary 3. Note that the slope of the line segment between $(\alpha(\gamma_m^*(s_m^j)), \rho(\gamma_m^*(s_m^j)))$ and $(\alpha(\gamma_m^*(s_m^{j+1})), \rho(\gamma_m^*(s_m^{j+1})))$ is equal to $mA_m(\gamma_m^*(s_m^j), \gamma_m^*(s_m^{j+1}))$. It is easy to see from Figure 3 or Theorem 3 that $A_m(\gamma_m^*(s_m^j), \gamma_m^*(s_m^{j+1}))$ is increasing in j. Figure 3 also indicates that the coordinates of the optimal configuration, that is, $(\alpha(\gamma_m^*(s_m^{j+1})), \rho(\gamma_m^*(s_m^{j+1})))$, are located within the triangle surrounded by (0, 0), $(0, \rho(\gamma_m^*(s_m^j)))$, and $(\alpha(\gamma_m^*(s_m^j)), \rho(\gamma_m^*(s_m^j)))$. When there is no configuration whose coordinates are within this area or when the inequality (6) is valid for all $t \in (s_m^{j+1}, t_0)$, we do not need to consider the next switch time s_m^j for the case with the mission lifetime t_0. Though the optimal configurations can be indicated from their coordinates, the switch times have to be computed on the basis of Theorem 2, in which no optimization problem is involved.

As a final solution step for the optimal reconfiguration strategy, we have developed the following algorithm A(RS), in which $\gamma^*_m(s^j_m)$ and $E[W^*_{t,m}]$ are calculated using $\gamma^*_{m-1}(s^j_m)$ and $E[W^*_{t,m-1}]$ for all $t \in [0, t_0]$. Complete algorithms to solve for $\gamma^*_m(s^j_m)$ will be presented in the next section. When t_0 is large, a different algorithm, where $\gamma^*_m(t + \delta t)$ and $E[W^*_{t+\delta t,m}]$ are computed based on $\gamma^*_m(t)$ and $E[W^*_{t,m}]$ for all $m \in \{0, \ldots, m_0\}$, is better than Algorithm A(RS) insofar as the storage requirement is concerned. However, the underlying principles are the same.

Algorithm A(RS)

Step 1. initialization
- 1a. For $m = 1, 2, \ldots, m_0$, find $\gamma^*_m(s^1_m)$ in which $\mathbf{n}$ maximizes $\rho(\omega)$, $\omega \in \Omega_m$.[5] When there is more than one configuration that maximizes $\rho(\omega)$, the configuration with the smallest crash probability is chosen.
- 1b. Find $E[W^*_{t,1}] = \rho(\gamma^*_1(s^1_1))(1/\lambda)(1 - e^{-\lambda t} - \lambda t e^{-\lambda t})$ for $t \in [0, t_0]$.
- 1c. Choose δt to digitize $t \in [0, t_0]$.
- 1d. Set $m := 1$ and start the following recursive steps.

Step 2. Set $m := m + 1$, $t := 0$, and $j := 2$. If $m > m_0$, stop the algorithm.
Step 3. Find $\gamma^*_m(s^j_m) \in \Phi_m(s^{j-1}_m)$ and $\alpha(\gamma^*_m(s^j_m)) \leq \alpha(\hat{\omega})$ for $\hat{\omega} \in \Phi_m(s^{j-1}_m)$.
Step 4. Using $\gamma^*_m(s^{j-1}_m)$ as the optimal configuration, calculate $E[W^*_{t+\delta t,m}]$ by eq. (5).
Step 5. Set $t := t + \delta t$. **If** $t \geq t_0$ **then** go to Step 2
else if $A_m(\gamma^*_m(s^{j-1}_m), \gamma^*_m(s^j_m)) \geq E[W^*_{t,m-1}]$ **then** go to Step 4
else $j := j + 1$ and go to Step 3.

4. *Determination of Optimal Configuration*

As defined in Section 2, a configuration $\omega \in \Omega_m$ can be specified by $(\mathbf{n}, \mathbf{r})$ where $\mathbf{n} = [n_1, n_2, \ldots, n_k]$, n_i represents the number of computing clusters for the task class i, $\mathbf{r} = [r_1, r_2, \ldots, r_k]$, and r_i is the degree of redundancy associated with the computing clusters in the task class i. It is assumed that the rewards gained from executing different task classes are independent. Thus, the system reward rate, $\rho(\mathbf{n})$, is equal to $\sum_{i=1}^{k} \rho_i(n_i)$ where ρ_i is the reward rate for the task class i. Furthermore, since all modules within the system are identical, the crash probability, $\alpha(\mathbf{n}, \mathbf{r})$, can be represented by $1/m \sum_{i=1}^{k} (n_i + r_i)\alpha_i(n_i, r_i)$ where $\alpha_i(n_i, r_i)$ is the crash probability when a module assigned to the task class i fails, given there are n_i computing clusters and r_i redundant modules in this task class.

Consider the execution of class i tasks. When there are n_i computing clusters, the task class can be treated as an n_i-server system. Let $f(x)$ denote the performance of an x-server system. Ideally, the performance of the system is additive in x, that is, $f(x_1 + x_2) = f(x_1) + f(x_2)$. However, owing to task communications, resource sharing, and other overheads, the performance of the system is subadditive, that is, $f(x_1) + f(x_2) \geq f(x_1 + x_2)$. In most cases, $f(x)$ is a nondecreasing concave function of x as shown in [3], [5], [12], and [34]. Following tradition, we consider here the cases in which the reward rate for each task class, $\rho_i(n_i)$, is a nondecreasing concave function of the number of computing clusters, n_i.

Let $h_i(r_i)$ be the crash probability of an r_i-module redundant computing cluster in the task class i, given that one of those modules fails. We assume that $h_i(r_i)$ is a decreasing convex function of r_i for all $i = 1, 2, \ldots, k$. Relaxation of this assumption may lead to a nonconvex programming problem and is discussed in Section 5. Thus, $r_i h_i(r_i)$ is also convex in r_i. The crash probability of the task class i then becomes

$$\alpha_i(n_i, r_i) = (r_i - n_i\hat{r}_i)\frac{\hat{r}_i + 2}{n_i + r_i} h_i(\hat{r}_i + 2) + (n_i - r_i + n_i\hat{r}_i)\frac{\hat{r}_i + 1}{n_i + r_i} h_i(\hat{r}_i + 1),$$

[5] It is easy to prove that $\gamma^*_m(s^1_m)$ maximizes $\rho(\omega)$ from Theorem 1 and $E[W^*_{\delta,m}] = 0$.

where $\hat{r}_i = \lfloor r_i/n_i \rfloor$ and $n_i > 0$. This implies that $(n_i + r_i)\alpha_i(n_i, r_i)$ is a piecewise linear and convex function of r_i. When $n_i\hat{r}_i \le r_i \le n_i(\hat{r}_i + 1)$, $\hat{r}_i \in \mathbf{I}^+$, the slope of the corresponding linear piece is $(\hat{r}_i + 2)h_i(\hat{r}_i + 2) - (\hat{r}_i + 1)h_i(\hat{r}_i + 1)$. When $h_i(r_i)$ is convex, it can be shown that an equal distribution of redundant modules over all computing clusters is optimal; this has been assumed in Section 2.

As indicated in Algorithm A(RS), there are two nonlinear integer programming problems to be solved for deriving the optimal reconfiguration strategy. The first, called $\mathbf{P}_0$, is a nonlinear *integer knapsack programming problem* which has to be solved for determining $\gamma^*_m(s^1_m)$.

$$\begin{aligned} \mathbf{P}_0\text{: maximize}_{n_i} \quad & \sum_{i=1}^{k} \rho_i(n_i) \\ \text{subject to} \quad & \sum_{i=1}^{k} n_i \le m, \qquad n_i \in \mathbf{I}^+ \quad \text{for} \quad i = 1, 2, \ldots, k. \end{aligned} \tag{8}$$

The second, called $\mathbf{P}_1$, is an *integer fractional programming problem* that is to determine $\gamma^*_m(s^{j+1}_m)$, given $\gamma^*_m(s^j_m)$ for $j \ge 1$. $\mathbf{P}_1$ can be expressed as

$$\begin{aligned} \mathbf{P}_1\text{: minimize}_{n_i, r_i} \quad & \frac{\rho(\omega_0) - \sum_{i=1}^{k} \rho_i(n_i)}{m\alpha(\omega_0) - \sum_{i=1}^{k} (n_i + r_i)\alpha_i(n_i, r_i)} \text{ given } \omega_0 \in \Omega_m \\ \text{subject to} \quad & \sum_{i=1}^{k} (n_i + r_i)\alpha_i(n_i, r_i) < m\alpha(\omega_0), \\ & n_i, r_i \in \mathbf{I}^+ \text{ for } i = 1, 2, \ldots, k, \\ & \sum_{i=1}^{k} (n_i + r_i) \le m, \qquad r_i = 0 \text{ when } n_i = 0. \\ & \sum_{i=1}^{k} (n_i + r_i) \le m, \end{aligned} \tag{9}$$

4.1 AN ALGORITHM FOR SOLVING $\mathbf{P}_0$. The nonlinear integer knapsack problem has been considered for various applications, such as resource allocation, portfolio selection, and capital budgeting. Several methods have been proposed for solving this problem, for example, dynamic programming approaches [6, 24], the shortest path algorithm [11], and ranking methods [20, 35]. Michaeli and Pollatschek investigated the problem and provided a necessary and sufficient condition for the optimal solution [23]. To solve the problem $\mathbf{P}_0$, since the coefficients of both the objective function and the linear constraints are all one, we can employ an algorithm similar to the one in [10], which is given below.[6]

Let $\Delta_i(n_i) = \rho_i(n_i + 1) - \rho_i(n_i)$. The principle of the algorithm is to allocate modules such that the system can have maximum return.

Algorithm A_0

Step 1. Set $n_i := 0$ for all $i = 1, 2, \ldots, k$.
Step 2. Select i^* such that $\Delta_{i^*}(n_{i^*}) = \max_{1 \le i \le k} \Delta_i(n_i)$.
If $\Delta_{i^*}(n_{i^*}) = 0$, then terminate the algorithm.
Step 3. $n_{i^*} = n_{i^*} + 1$ and $m := m - 1$. If $m = 0$, then terminate the algorithm.
Otherwise, go to Step 2.

To determine $\Delta_{i^*}(n^*_i) = \max_{1 \le i \le k} \Delta_i(n_i)$, we need to sort $\Delta_i(n_i)$. Clearly, it is not necessary to sort all $\Delta_i(n_i)$ for every m. $\Delta_i(0)$ must be sorted during the first

[6] Existence of the incremental algorithm in [10] was brought by one referee to the authors' attention. Because of the authors' unawareness of this reference, Algorithm A_0 had been developed and presented as a new algorithm in the earlier draft of this paper.

iteration. However, since n_{i^*} is changed only in later iterations, $\Delta_{i^*}(n_{i^*})$ has to be evaluated and inserted into the previous sorted sequence. Note that there are at most m iterations required for this algorithm to terminate. Another advantage of Algorithm A_0 is that all $\gamma_m^*(s_m^1)$ can be solved at once for all $m \in \{0, 1, \ldots, m_0\}$. By assuming $m = m_0$ at the beginning, $\gamma_m^*(s_m^1)$ is obtained at the end of the mth iteration.

4.2 AN ALGORITHM FOR SOLVING $\mathbf{P}_1$. The solution of $\mathbf{P}_1$ can be divided into two levels: the lower level is to determine r_i^*, $i = 1, 2, \ldots, k$, by minimizing the objective function (9) for a given $\mathbf{n}$; the higher level problem is to determine n_i^* by minimizing the objective function (9) with the calculated r_i^* from the lower level. Since the only place that r_i's appear is in the denominator of the objective function, the lower level problem can be stated as follows:

$$\begin{aligned} \mathbf{P}_2: \quad & \underset{r_i}{\text{minimize}} \quad \sum_{i=1}^{k} (n_i + r_i)\alpha_i(n_i, r_i) \\ & \text{subject to} \quad \sum_{i=1}^{k} r_i \le m - \sum_{i=1}^{k} n_i, \qquad r_i \in \mathbf{I}^+ \quad \text{for} \quad i = 1, 2, \ldots, k, \\ & \qquad\qquad\qquad\qquad\qquad\qquad\quad r_i = 0 \quad \text{when} \quad n_i = 0. \end{aligned} \tag{10}$$

By letting $r_i\alpha_i(0, r_i) = r_i$, the last constraint can be eliminated. Then, the problem $\mathbf{P}_2$ is an integer knapsack programming problem that is to minimize the sum of nonlinear functions. If $(n_i + r_i)\alpha_i(n_i, r_i)$ is convex with respect to r_i, we can apply an algorithm similar to A_0 in which we choose $\min_{1\le i\le k}\{(n_i + r_i + 1)\alpha_i(n_i, r_i + 1) - (n_i + r_i)\alpha_i(n_i, r_i)\}$ in place of $\max_{1\le i\le k}\{\rho_i(n_i + 1) - \rho_i(n_i)\}$ in Algorithm A_0. When $(n_i + r_i)\alpha_i(n_i, r_i)$ is not convex, the lower level problem becomes a nonconvex programming problem. There is no guarantee that the solution obtained by Algorithm A_0 is the global minimum.

Let $\beta(\mathbf{n}) = \min_{r_i} \sum_{i=1}^{k} (n_i + r_i)\alpha_i(n_i, r_i)$ obtained from solving $\mathbf{P}_2$, given $\mathbf{n}$. Thus, the problem $\mathbf{P}_1$ can be converted to the following form:

$$\begin{aligned} \mathbf{P}_3: \quad & \underset{n_i}{\text{minimize}} \quad \frac{\rho(\omega_0) - \rho(\mathbf{n})}{m\alpha(\omega_0) - \beta(\mathbf{n})} \\ & \text{subject to} \quad \beta(\mathbf{n}) < m\alpha(\omega_0) \quad \text{and} \quad \rho(\mathbf{n}) < \rho(\omega_0), \\ & \qquad\qquad \sum_{i=1}^{k} n_i \le m, \qquad n_i \in \mathbf{I}^+ \quad \text{for} \quad i = 1, 2, \ldots, k. \end{aligned} \tag{11}$$

By Corollary 3, the triangle area surrounded by $(0, 0)$, $(0, \rho(\mathbf{n}^0))$, and $((1/m)\beta(\mathbf{n}^0), \rho(\mathbf{n}^0))$ is a feasible region described in terms of the configuration coordinates. Since there does not exist an explicit form of mapping from the configuration coordinates to $\mathbf{n}$, an enumeration is the only means of finding whether or not a configuration is within this triangle region of the configuration coordinates.

Consider the use of an explicit enumeration (or brute force enumeration) for solving $\mathbf{P}_3$. For every possible combination of n_i satisfying the constant $\sum_{i=1}^{k} n_i \le m$, we map the combination into r_i^*, $\beta(\mathbf{n})$, and $(\rho(\omega_0) - \rho(\mathbf{n}))/(m\alpha(\omega_0) - \beta(\mathbf{n}))$. Then, the configuration with the minimum ratio is chosen as the optimal configuration. When there are k task classes and m modules available, the total number of combinations in $\mathbf{n}$ to satisfy the constraint $\sum_{i=1}^{k} n_i \le m$ is $\binom{m+k}{k}$. Thus, when the size of the system is moderate, explicit enumeration approach is reasonable. For example, when $k = 3$ and $m = 12$, the total number of enumerations is 455.

There are two important and advantageous properties associated with explicit enumeration: (i) The coordinates of *all* feasible configurations for m available

modules, that is, $((1/m)\beta(\mathbf{n}), \rho(\mathbf{n}))$, can be obtained from the enumeration. Thus, from Theorem 3, we can easily determine all optimal configurations, $\gamma_m^*(s_m^j)$ for $j = 1, 2, \ldots$. (ii) When we solve problem $\mathbf{P}_2$ for r_i^* with $m = m_0$, the other r_i^*'s are determined simultaneously for all $m \in \{\sum_{i=1}^{k} n_i, \ldots, m_0\}$. We obtain the coordinates, incorporated with $\rho(\mathbf{n})$, of all feasible configurations for $m \in \{1, \ldots, m_0\}$. Therefore, the optimal configurations $\gamma_m^*(s_m^j)$ for $m \in \{1, \ldots, m_0\}$ can be determined. These two properties lead to a situation in the determination of a reconfiguration strategy in which explicit enumeration has to be conducted only once.

Remarks on Fractional Programming Problems. For a general fractional programming problem, the objective function in eq. (11) is no longer convex or concave with respect to n_i even if $\beta(\mathbf{n})$ is convex or concave with respect to n_i [27]. For a continuous nonlinear fractional programming problem, some equivalent or dual problems have been proposed in [7] and [28]. With integer constraints, Chandra and Chandramohan [4] suggested applying a branch and bound method after solving the continuous problem. However, no example or analysis is provided to show the efficiency of their algorithm.

A recent survey on the methods of solving the fractional programming problem [14] has discussed three different state-of-the-art approaches. The first approach uses variable transformation and is probably the most efficient method for *linear* fractional programming problems. The second approach deals with the problem as a nonlinear programming problem and applies a suitable search algorithm to find the solution. The third approach uses an auxiliary parameterized problem suggested in [14], which is briefly discussed below. Let F_3 denote the feasible region for $\mathbf{P}_3$. Define the auxiliary problem $\mathbf{Q}(\eta)$ by

$$\mathbf{Q}(\eta): \quad \min_{\mathbf{n} \in F_3} \rho(\omega_0) - \rho(\mathbf{n}) - \eta(m\alpha(\omega_0) - \beta(\mathbf{n})).$$

Let $z(\eta)$ be the minimum value of $\mathbf{Q}(\eta)$ and $\mathbf{n}^*(\eta)$ be the optimal solution of $\mathbf{Q}(\eta)$. If there is an η^* such that $z(\eta^*) = 0$, we can have

$$\eta^* = \frac{\rho(\omega_0 - \rho(\mathbf{n}^*(\eta^*))}{m\alpha(\omega_0) - \beta(\mathbf{n}^*(\eta^*))} \leq \frac{\rho(\omega_0) - \rho(\mathbf{n})}{m\alpha(\omega_0) - \beta(\mathbf{n})}$$

for all $\mathbf{n} \in F_3$. The above equation implies that an optimal solution of $\mathbf{Q}(\eta')$ is also an optimal solution of $\mathbf{P}_3$. Hence, an algorithm is needed to search for η^* such that $z(\eta^*) \approx 0$ by solving $\mathbf{Q}(\eta)$ iteratively. The complexity of the algorithm is based on the efficiency of solving $\mathbf{Q}(\eta)$ and the search algorithm, for example, Newton's method or binary search. Meggido [21] proposed an algorithm that combines the search for η^* and the dynamic programming approach to $\mathbf{Q}(\eta)$. The resulting algorithm requires $O(km^2 \log m)$ evaluations of $\beta(\mathbf{n})$ and $\rho(\mathbf{n})$.

Note that the problem $\mathbf{Q}(\eta)$ is the same as the OR problem, that is, a nonlinear knapsack programming problem. It might be more efficient to solve the OR problem directly instead of searching for η^* and solving $\mathbf{Q}(\eta)$ iteratively. When $\beta(\mathbf{n})$ is convex with respect to n_i, the objective function becomes convex with respect to n_i, implying that Algorithm A_0 can be applied. The algorithm A_1, which includes Algorithm A_0, is provided below using an operator P_j and a function Δ_i defined by

$$P_j(\mathbf{n}) \equiv (n_1, n_2, \ldots, n_{j-1}, n_i + 1, n_{j+1}, \ldots, n_k)$$
$$\Delta_i(\mathbf{n}) \equiv \rho(P_i(\mathbf{n})) - \rho(\mathbf{n}) - \lambda \mathrm{E}[W_{t_h, m-1}](\beta(P_i(\mathbf{n})) - \beta(\mathbf{n})).$$

TABLE I. CRASH PROBABILITIES $h_i(n)$ AND REWARD RATES $\rho_i(n)$ USED IN THE EXAMPLE

n	1	2	3	4	5	6	7	8
$h_1(n)$	0.6	0.3	0.1	0.05	0.005	0.003	0.002	0.001
$\rho_1(n)$	1.0	1.8	2.4	2.9	3.3	3.6	3.8	3.9
$h_2(n)$	0.5	0.25	0.1	0.05	0.025	0.013	0.005	0.002
$\rho_2(n)$	0.8	1.5	2.2	2.8	3.4	4.0	4.6	5.1
$h_3(n)$	0.3	0.1	0.05	0.01	0.005	0.003	0.002	0.001
$\rho_3(n)$	0.6	1.2	1.7	2.2	2.7	3.2	3.6	4.0

Algorithm A_1

Step 1. Set $L := 0$ and $U := t_0$.
Step 2. a. Set $t_h := L + h(U - L)/N$ for $h = 1, 2, \ldots, N$.
b. For $h = 1, 2, \ldots, N$
b1. Set $dm := m$ and $n_i := 0$ for $i = 1, 2, \ldots, k$.
b2. Solve for $\beta(P_i(\mathbf{n}))$ as indicated in the solution of $\mathbf{P}_2$ and $\Delta_i(\mathbf{n})$ for $i = 1, 2, \ldots, k$.
b3. Select i^* such that $\Delta_{i^*} = \max_{1 \le i \le k} \Delta_i(\mathbf{n})$. If $\Delta_{i^*}(\mathbf{n}) \le 0$, go to (b5).
b4. $\mathbf{n} := P_{i^*}(\mathbf{n})$ and $dm := dm - 1$. If $dm \neq 0$, go to (b2).
b5. Set $\gamma^*_m(t_h) := \mathbf{n}$.

Algorithm A_1 determines the optimal configurations for a specific number of partitions of the mission lifetime, that is, N. If the optimal configurations at t_h and t_{h+1} are different, the solution can be refined by setting $L = t_h$, $U = t_{h+1}$ and then repeating Step 2 of A_1. This will lead to a more accurate switch time between t_h and t_{h+1}. Although Algorithm A_1 only provides one-level partitions, it is easy to extend more refined partitions.

4.3 AN EXAMPLE OF THE OPTIMAL RECONFIGURATION STRATEGY. Consider a system with 12 modules at the beginning of the mission. The module failure rate is assumed to be 0.0005. The tasks to be executed during the entire mission are grouped into three classes whose respective reward rates $\rho_i(n_i)$ and crash probabilities $h_i(n)$ for $i = 1, 2, 3$ are listed in Table I. In addition, we include the constraints $n_i \ge 1$ for $i = 1, 2, 3$, indicating that at least one computing cluster must be available for each task class throughout the entire mission.

To satisfy the constraint $n_i \ge 1$ for $i = 1, 2, 3$, we must begin with assigning one module to each task class. Applying the explicit enumeration given in Section 4.2, we can find all possible configurations and the respective switch time for a mission with infinite mission lifetime. The optimal configurations $\gamma^*_m(s^j_m)$ for $4 \le m \le 12$ are listed in Table II, whereas Table III gives the switch times for each optimal configuration. The optimal reconfiguration strategy is obtained from the combination of both of these two tables. Notice that certain optimal configurations will never be used. For instance, $\gamma^*_5(s^4_5)$ is one, since $s^3_5 = \infty$.

When the remaining mission lifetime is known, the optimal configuration can be found from Tables II and III. The optimal reconfiguration strategy is derived *off-line* in table form before the mission. For *on-line* use of the strategy, the system only needs to look up the tables of optimal configurations and switch times (e.g., Tables II and III). In this way, the system follows an optimal reconfiguration trajectory using (i) the switch times in the same row of the two tables for the case of no module failure or (ii) row changes in the tables and then the switch times in case of module failures. In Figure 4, given that the mission lifetime is 1000, the

TABLE II. OPTIMAL CONFIGURATIONS FOR THE EXAMPLE

m	$\gamma_m^*(s_m^1)$	$\gamma_m^*(s_m^2)$	$\gamma_m^*(s_m^3)$	$\gamma_m^*(s_m^4)$	$\gamma_m^*(s_m^5)$	$\gamma_m^*(s_m^6)$	$\gamma_m^*(s_m^7)$	$\gamma_m^*(s_m^8)$	$\gamma_m^*(s_m^9)$
4	$\binom{2\ 1\ 1}{0\ 0\ 0}$	$\binom{1\ 1\ 2}{0\ 0\ 0}$	$\binom{1\ 1\ 1}{1\ 0\ 0}$	—	—	—	—	—	—
5	$\binom{2\ 2\ 1}{0\ 0\ 0}$	$\binom{2\ 1\ 2}{0\ 0\ 0}$	$\binom{1\ 1\ 2}{1\ 0\ 0}$	$\binom{1\ 1\ 1}{1\ 1\ 0}$	—	—	—	—	—
6	$\binom{2\ 3\ 1}{0\ 0\ 0}$	$\binom{2\ 2\ 2}{0\ 0\ 0}$	$\binom{2\ 1\ 2}{1\ 0\ 0}$	$\binom{1\ 1\ 2}{1\ 1\ 0}$	$\binom{1\ 1\ 1}{2\ 1\ 0}$	—	—	—	—
7	$\binom{2\ 3\ 2}{0\ 0\ 0}$	$\binom{2\ 1\ 2}{2\ 0\ 0}$	$\binom{1\ 1\ 2}{2\ 1\ 0}$	$\binom{1\ 1\ 1}{2\ 1\ 1}$	—	—	—	—	—
8	$\binom{2\ 4\ 2}{0\ 0\ 0}$	$\binom{2\ 3\ 3}{0\ 0\ 0}$	$\binom{2\ 3\ 2}{1\ 0\ 0}$	$\binom{2\ 2\ 2}{2\ 0\ 0}$	$\binom{2\ 1\ 2}{2\ 1\ 0}$	$\binom{1\ 1\ 2}{2\ 1\ 1}$	$\binom{1\ 1\ 1}{2\ 2\ 1}$	—	—
9	$\binom{2\ 5\ 2}{0\ 0\ 0}$	$\binom{2\ 3\ 4}{0\ 0\ 0}$	$\binom{2\ 3\ 2}{2\ 0\ 0}$	$\binom{2\ 2\ 2}{2\ 1\ 0}$	$\binom{2\ 1\ 2}{3\ 1\ 0}$	$\binom{1\ 1\ 2}{2\ 1\ 2}$	$\binom{1\ 1\ 1}{3\ 2\ 1}$	—	—
10	$\binom{2\ 6\ 2}{0\ 0\ 0}$	$\binom{2\ 3\ 5}{0\ 0\ 0}$	$\binom{2\ 3\ 3}{2\ 0\ 0}$	$\binom{2\ 3\ 2}{2\ 1\ 0}$	$\binom{2\ 2\ 2}{2\ 2\ 0}$	$\binom{2\ 1\ 2}{4\ 1\ 0}$	$\binom{2\ 1\ 1}{4\ 1\ 1}$	$\binom{1\ 1\ 2}{2\ 2\ 2}$	$\binom{1\ 1\ 1}{3\ 3\ 1}$
11	$\binom{2\ 7\ 2}{0\ 0\ 0}$	$\binom{2\ 3\ 6}{0\ 0\ 0}$	$\binom{2\ 3\ 4}{2\ 0\ 0}$	$\binom{2\ 3\ 2}{2\ 2\ 0}$	$\binom{2\ 2\ 2}{3\ 2\ 0}$	$\binom{2\ 1\ 2}{4\ 1\ 1}$	$\binom{2\ 1\ 1}{4\ 2\ 1}$	$\binom{1\ 1\ 1}{3\ 3\ 2}$	—
12	$\binom{3\ 7\ 2}{0\ 0\ 0}$	$\binom{2\ 7\ 3}{0\ 0\ 0}$	$\binom{2\ 4\ 6}{0\ 0\ 0}$	$\binom{2\ 3\ 6}{1\ 0\ 0}$	$\binom{2\ 3\ 5}{2\ 0\ 0}$	$\binom{2\ 3\ 2}{2\ 3\ 0}$	$\binom{2\ 1\ 2}{4\ 1\ 2}$	$\binom{2\ 1\ 1}{5\ 2\ 1}$	$\binom{1\ 1\ 1}{4\ 3\ 2}$

Configuration: $\binom{n_1\ n_2\ n_3}{r_1\ r_2\ r_3}$

TABLE III. SWITCH TIMES FOR THE EXAMPLE

m	s_m^0	s_m^1	s_m^2	s_m^3	s_m^4	s_m^5	s_m^6	s_m^7	s_m^8
4	0	1194	∞	—	—	—	—	—	—
5	0	436	1391	∞	—	—	—	—	—
6	0	349	938	1150	∞	—	—	—	—
7	0	755	1364	2813	—	—	—	—	—
8	0	254	573	635	739	1379	∞	—	—
9	0	227	506	643	760	1108	∞	—	—
10	0	206	455	541	574	807	1098	1611	∞
11	0	188	413	490	602	716	1321	∞	—
12	0	104	173	314	378	447	646	∞	∞

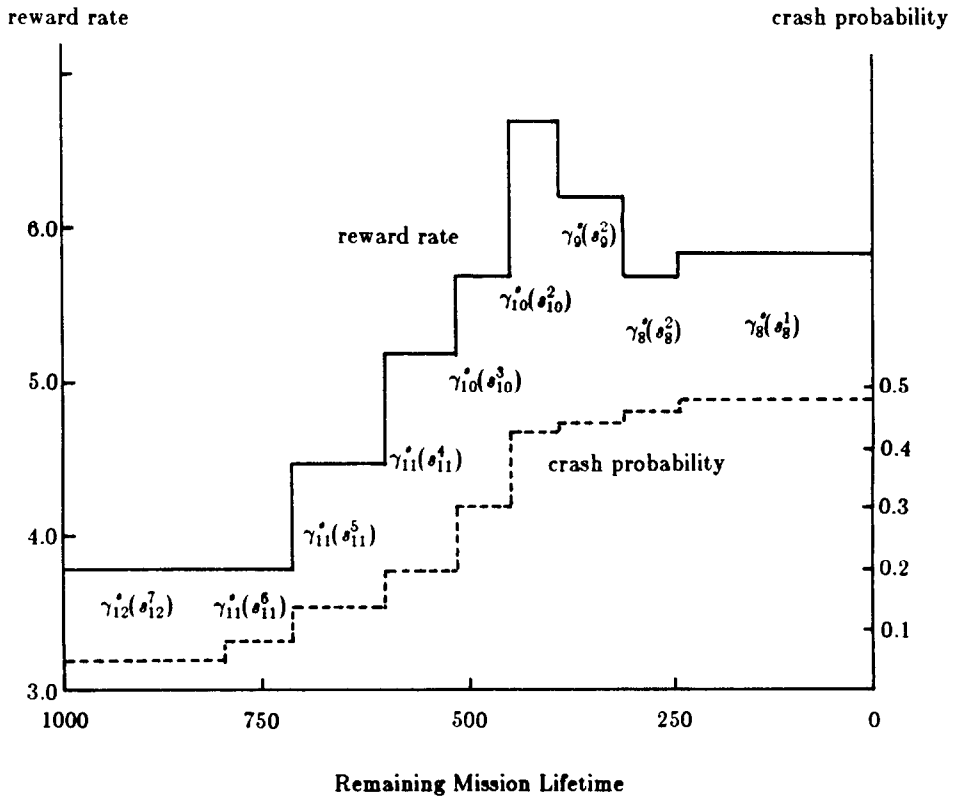

FIG. 4. Example of optimal reconfiguration trajectory. Module fails when the remaining lifetime is 800, 520, 400, 310.

configuration trajectories and the respective reward rates and crash probabilities are plotted if the module failures occur when RML are 800, 520, 400, and 310.[7]

5. *Discussion*

5.1 CONCLUDING REMARKS. In this paper we have addressed the problem of reconfiguring nonrepairable multimodule systems. Since we have treated the problem for general multimodule computing systems, both the problem formulation

[7] These are random and known only via failure detection mechanisms. For a demonstrative purpose, we used arbitrarily chosen values.

and the solution approach of this paper have high potential use for designing the growing number of fault-tolerant multiprocessors and computer networks.

Given multiple modules, computing clusters with appropriate redundancy for each task class are formed so that the resulting system may meet both requirements of performance and reliability in an optimal fashion. Because of the inherent trade-off between performance and reliability, we need to determine system configurations that specify the number of computing clusters and redundant modules for each task class. In addition to the conventional *passive* reconfiguration strategy, which is invoked only upon detection of a module failure, we have shown the need of an *active* reconfiguration strategy that allows the system to reconfigure itself as the mission progresses, regardless of the occurrence of module failure. Thus, the active reconfiguration strategy provides the optimal configurations by taking into account both the degradation of the system due to module failures and the progression of the mission.

Using the expected total reward as the criterion for determining the optimal configurations, we have explored the properties of the optimal configurations, which are useful for deriving solutions. A feasible region is described in terms of the *configuration coordinates.* Although it is easy to find the configuration coordinates, the inverse mapping from the coordinates to a configuration does not exist in closed form, thereby requiring less elegant enumerations.

In order to derive the optimal configurations, two nonlinear integer programming problems have to be solved. The first is a *knapsack problem,* which can be solved through a simple but elegant algorithm. The second is a *fractional programming problem,* which is basically a nonconvex programming problem. In addition to an explicit enumeration, we have discussed the other approaches known for solving the fractional programming problem. Since both the explicit enumeration and the fractional programming may become very complicated when the system size (i.e., m_0 and k) is large, some other approximations or heuristic algorithms need to be explored.

As shown in the example of Section 4.3, the optimal reconfiguration strategy can be represented by two tables: one is for optimal configurations and the other for switch times. Although the solution procedures for obtaining these two tables are complex, they can be computed off line. The real reconfiguration during the mission is performed just by looking up these two tables.

5.2 EXTENSIONS. Several assumptions that we have used can be relaxed by employing a more complex optimization procedure. For instance, the reward rate could be affected by the degree of redundancy incorporated in a task class. In such a case we need to change $\rho(\mathbf{n})$ to $\rho(\mathbf{n}, \mathbf{r})$. The optimization problem $\mathbf{P}_1$ can no longer be decomposed into two levels. When the concavity of $\rho_i(n_i)$ and the convexity of $\alpha(n_i, r_i)$ do not exist, the optimization problems become nonconvex and thus are more difficult to solve.

We can also remove the assumption that the reward rate and the crash probability must be stationary, that is, independent of the mission lifetime. For such a case, let $\rho(\omega, t)$ and $\alpha(\omega, t)$ be used in place of $\rho(\omega)$ and $\alpha(\omega)$, respectively. The problem OR, then, becomes to maximize

$$J_{t,m}(\omega) \equiv \rho(\omega, t) - \alpha(\omega, t)m\lambda \mathrm{E}[W^*_{t,m-1}] \qquad \text{for} \quad \omega \in \Omega_m.$$

Obviously, this time dependency does not increase any complexity if we solve the problem OR directly.

In place of the total expected reward, a generalized objective function for the optimal configurations could be defined as $E[R(W_{t_0,m_0})]$ where R is a function of W. Note that $R(W)$ should be nondecreasing but may not be continuous. It could be a step function as in the example in [13], or any other discontinuous function with finite jumps. When $R(W)$ is not additive, that is, $R(W_1 + W_2) \neq R(W_1) + R(W_2)$, the optimal configuration at the current moment is dependent on the total reward accumulated up to the current moment. Hence, the determination of an optimal configuration must consider both the past and future configuration trajectories, thereby making the optimization problem very complex and difficult. The difficulty can be foreseen by comparing the optimal configuration problem with the general stochastic optimal control problem. The optimal control problem usually uses the summation (or integration) of the functions of variables as an objective function to be optimized. However, the optimal configuration problem requires the use of a function of the summation of variables, making the decomposition of these variables impossible. This is a matter for further research.

Appendix. Glossary of Notation

Ω_m: Set of all feasible configurations when the system has m operational modules.

$\gamma_m(t)$: Reconfiguration function defined at time $t \in [0, t_0]$ and with a configuration in Ω_m.

$\Phi_m(s_m^j)$: Set of configurations in $L_m(s_m^j)$ that minimizes the function $A_m(\gamma_m^*(s_m^j), \omega)$. If γ_m^{j+1} exists, it must be a member of this set.

$\alpha(\omega)$: System crash probability when the system is with configuration ω, given a module failure.

$\alpha_i(n_i, r_i)$: System crash probability when a module assigned to the task class i fails.

$\beta(\mathbf{n})$: The solution of Problem $\mathbf{P}_2$, where $\sum_{i=1}^k (n_i + r_i)\alpha(n_i + r_i)$ for r_i, subject to $\sum_{i=1}^k r_i \leq m - \sum_{i=1}^k n_i$, etc., for a given $\mathbf{n} = (n_1, n_2, \ldots, n_k)$.

$\rho(\omega)$: Reward rate associated with the configuration ω.

$A_m(\omega_1, \omega_2)$: The ratio of $\rho(\omega_1) - \rho(\omega_2)$ to $m\alpha(w_1) - m\alpha(\omega_2)$.

$G_m(s_m^j)$: Set of configurations that belong to Ω_m and have crash probabilities greater than that of the optimal configuration at RML $= s_m^j$.

$L_m(S_m^j)$: Set of configurations that belong to Ω_m and have crash probabilities less than that of the optimal configuration at RML $= s_m^j$.

$M(t)$: The system state when the remaining mission lifetime is t. $M(t)$ is equal to the number of available modules if no crash occurs before t, or 0 otherwise.

$\mathrm{RF}_{t_0,m_0}(t)$: The stochastic process defined on the configuration used at RML $= t$ under a given reconfiguration strategy RS_{t_0,m_0}.

RML: Remaining mission lifetime.

RS_{t_0,m_0}: A reconfiguration strategy specified by the configuration functions $\gamma_m(t)$ where $0 \leq m \leq m_0$ and $0 \leq t \leq t_0$. Given a strategy, the system uses the configuration defined by $\gamma_m(t) \in \mathrm{RS}_{t_0,m_0}$, when RML $= t$ and there are m modules available.

$\mathrm{RS}^*_{t_0,m_0}$: The optimal reconfiguration strategy that maximizes the expected total reward for the system with mission lifetime t_0 and m_0 operational modules initially.

$W_{t,m}$: The total reward accumulated for the system with mission lifetime t and m operational modules initially. Since the failure occurs stochastically, $W_{t,m}$ is a random variable.

$h_i(r_i)$: Crash probability when a module, assigned to a computing cluster with r_i redundant modules for the task class i, fails.

$m(t)$, m: The number of total available modules in the system when RML = t.

$m_i(t)$, m_i: The number of available modules assigned to the task class i when RML = t.

$n_i(t)$, n_i: The number of functioning modules assigned to the task class i (or the number of computing clusters for the task class i) when RML = t.

$r_i(t)$, r_i: The number of redundant modules assigned to the task class i (or n_i computing clusters) when RML = t.

s_m^j: The jth switch time when the system is with m available modules. The system should use the configuration $\gamma_m^*(s_m^{j+1})$ continuously when $s_m^j < \text{RML} \leq s_m^{j+1}$ and change to $\gamma_m(s_m^j)$ when RML = s_m^j.

ACKNOWLEDGMENT. The authors wish to thank C. M. Krishna, Michael Woodbury, and anonymous referees for carefully reading this paper and making many useful comments. Also, they are indebted to Rick Butler and Milton Holt of NASA for their technical and financial support.

REFERENCES

1. ADAMS, G. B., AND SIEGEL, H. J. A fault-tolerant interconnection network for supersystems. *IEEE Trans. Comput. C-31* (May 1982), 443–454.
2. BARIGAZZI, G., CIUFFOLETTI, A., AND STRIGINI, L. Reconfiguration procedure in a distributed multiprocessor system. In *Proceedings of the 12th International Symposium on Fault-Tolerant Computing*. 1982, pp. 73–80.
3. BHANDARKAR, D. P. Some performance issues in multiprocessor systems. *IEEE Trans. Comput. C-26*, 5 (May 1977), 506–511.
4. CHANDRA, S., AND CHANDRAMOHAN, M. A branch and bound method for integer nonlinear fractional programs. *Z. Angew. Math. Mech. 60*, 12 (1980), 735–737.
5. CHU, W. W., HOLLOWAY, L. J., LAN, M.-T., AND EFE, K. Task allocation in distributed data processing. *Computer 13*, 11 (Nov. 1980), 57–70.
6. COOPER, M. W. The use of dynamic programming methodology for the solution of a class of nonlinear programming problem. *Naval Res. Log. Q. 27* (1980), 89–95.
7. CRAVEN, B. D., AND MOND, B. On fractional programming and equivalence. *Naval Res. Log. Q. 22* (1975), 405–410.
8. DONATIELLO, L., AND IYER, B. R. Analysis of a composite performance reliability measure for fault tolerant systems. *J. ACM 34*, 1 (Jan. 1987), 179–199.
9. FORTES, J. A. B., AND RAGHAVENDRA, C. S. Dynamically reconfigurable fault-tolerant array processors. In *Proceedings of the 14th International Symposium on Fault-Tolerant Computing*. pp. 386–392, 1984.
10. FOX, B. Discrete optimization via marginal analysis. *Management Sci. 13*, 3 (Nov. 1966), 210–216.
11. FRIEZE, A. M. Shortest path algorithms for knapsack type problem. *Math. Prog. 11* (1976), 150–157.
12. FULLER, S. H., OUSTERHOUT, J. K., RASKIN, L., RUBINFIELD, P. L., SINDHU, P. J., AND SWAN, R. J. Multi-microprocessors: An overview and working example. *Proc. IEEE 66*, 2 (Feb. 1978), 216–228.
13. FURCHTGOTT, D. G. Performability models and solutions. Comput. Res. Lab. Rep. CRL-TR-8-84. The University of Michigan, Ann Arbor, Mich., 1984.
14. IBARAKI, T. Solving mathematical programming problems with fractional objective functions. In Generalized Concavity in Optimization and Economics, S. Schaible and W. T. Ziemba, Eds. Academic Press, Orlando, Fla., 1981, pp. 441–472.
15. KAIN, R. Y., AND FRANTA, W. R. Interprocess communication schemes supporting system reconfiguration. In *Proceedings of the IEEE COMPSAC80*. IEEE, New York, 1980, pp. 365–371.
16. KARTASHEV, S. L., AND KARTASHEV, S. P. A multicomputer system with dynamic architectures. *IEEE Trans. Comput. C-28*, 10 (Oct. 1979), 704–721.

17. KIM, K. H. Error detection, reconfiguration and recovery in distributed processing system. In *Proceedings of the 1st International Conference on Distributed Computing Systems.* Oct. 1979, pp. 284–295.
18. KRISHNA, C. M., AND SHIN, K. G. Performance measures for multiprocessor controllers. In *Performance '83: Ninth International Symposium on Computer Performance, Measurement, and Evaluation.* 1983, pp. 229–250.
19. LEE, Y. H., AND SHIN, K. G. Optimal design and use of retry in fault tolerant real-time computer systems. Comput. Res. Lab. Rep. CRL-TR-28-84. Univ. of Michigan, Ann Arbor, Mich., May 1984.
20. LUSS, H., AND GUPTA, S. K. Allocation of effort resources among competing activities. *Oper. Res. 5* (1975), 613–629.
21. MEGIDDO, N. Combinatorial optimization with rational objective functions. *Math. Oper. Res. 4* (1979), 414–424.
22. MEYER, J. F. On evaluating the performability of degrading computer systems. *IEEE Trans. Comput. C-29*, 8 (Aug. 1980), 720–731.
23. MICHAELI, I., AND POLLATSCHEK, M. A. On some nonlinear knapsack problem. *Ann. Disc. Math. 1* (1977), 403–414.
24. MORIN, T. L., AND MARSTEN, R. E. An algorithm for non-linear knapsack problem. *Management Sci. 22*, 10 (June 1976), 1147–1158.
25. SAHEBAN, F., AND FRIEDMAN, A. D. Diagnostic and computational reconfiguration in multiprocessor system. In *Proceedings of the 1978 Annual ACM Conference* (Washington, D.C., Dec. 4–6). ACM, New York, pp. 68–78.
26. SAHEBAN, F., AND FRIEDMAN, A. D. A survey and methodology of reconfigurable multi-module system. In *Proceedings of the IEEE COMPSAC78.* IEEE, New York, 1978, pp. 790–796.
27. SCHAIBLE, S. Minimization of ratios. *J. Opt. Theory Applicat. 19*, 2 (June 1976), 347–352.
28. SCHAIBLE, S. A survey of fractional programming. In *Generalized Concavity in Optimization and Economics*, S. Schaible and W. T. Ziemba, Eds. Academic Press, Orlando, Fla., 1981, pp. 417–440.
29. SHIN, K. G., AND LEE, Y. H. Error detection process-model, design and its impact on computer performance. *IEEE Trans. Comput. C-33*, 6 (June 1984), 529–540.
30. SIEGEL, H. J., MCMILLEN, R. J., AND MUELLER, P. T., JR. A survey of interconnection methods for reconfigurable parallel processing system. In *Proceedings of the AFIPS 1979 National Computer Conference.* AFIPS Press, Reston, Va., 1979, pp. 529–542.
31. SMITH, T. B., III, AND LALA, J. H. Development and evaluation of a fault-tolerant multiprocessor (FTMP) computer. Volume IV: FTMP executive summary. NASA Contractor Rep. 172286. NASA Langley Research Center, Langley, Va., Feb. 1984.
32. STIFFLER, J. J., AND BRYANT, L. A. CARE III phase report—Mathematical description. NASA Contractor Rep. 3566. NASA Langley Research Center, Langley, Va., Nov. 1982.
33. TROY, R. Dynamic reconfiguration: An algorithm and its efficiency evaluation. In *Proceedings of the 9th International Symposium on Fault-Tolerant Computation.* 1979, pp. 44–49.
34. WULF, W. A., AND BELL, C. G. C.mmp—A multi-mini-processor. In *1972 Fall Joint Computer Conference. AFIPS Conference Proceedings*, vol. 41. AFIPS Press, Reston, Va., 1972, pp. 765–777.
35. ZIPKIN, P. H. Simple ranking methods for allocation of one resource. *Manage. Sci. 26*, 1 (Jan. 1980), 34–43.

RECEIVED SEPTEMBER 1984; REVISED FEBRUARY 1986; ACCEPTED JUNE 1986