THE UNIVERSITY OF MICHIGAN

COMPUTING RESEARCH LABORATORY[1]

OPTIMAL RECONFIGURATION STRATEGY FOR A DEGRADABLE MULTI-MODULE COMPUTING SYSTEM

Yann-Hang Lee and Kang G. Shin

CRL-TR-41-84

September 1984

Room 1079, East Engineering Building
Ann Arbor, Michigan 48109
USA
Tel: (313) 763-8000

[1]Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the funding agency.

OPTIMAL RECONFIGURATION STRATEGY FOR A DEGRADABLE MULTI-MODULE COMPUTING SYSTEM [1]

Yann-Hang Lee and Kang G. Shin

Division of Computer Science and Engineering
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109.

ABSTRACT

In this report, we present a new quantitative approach to the problem of reconfiguring a degradable multi-module system. 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 tradeoff between the computation capacity and fault-tolerance of a multi-module computing system, the conventional approach is a passive action and does not yield a configuration which provides an optimal compromise for the tradeoff. Using the expected total reward as the optimal criterion, we show the need and existence of an active reconfiguration strategy in which the system reconfigures itself on the basis of not only the *progression of the mission* but also the *occurrence of a failure*.

Following the problem formulation, we investigate some important properties of an optimal reconfiguration strategy which specify (i) the times at which the system should undergo reconfiguration, and (ii) the configurations that the system should change to. 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. Additional extensions of the optimal reconfiguration problem are also discussed.

---

[1]This work was supported in part by NASA under both Grant NAG 1-296 and Grant NAG 1-492. Any opinions, findings, and conclusions or recommendations expressed in this report are those of the authors and do not necessarily reflect the views of NASA. All correspondence should be addressed to Prof. Kang G. Shin at the above address.

1. INTRODUCTION

Reconfiguration of a system is the process of changing an already-existing system organization or 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 tasks to be executed. Reconfiguration of a system could have many different ways such as the change of partition [1], word size [2], etc. 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 [3], the reconfiguration algorithm to maintain the system in a safe state [4], and the like.

For the purpose of fault-tolerance, several authors have proposed the procedures and principles of reconfiguration of computer systems [5,6,7]. These procedures are all intended to make the system operational in the face of subsystem failures. Saheban and Friedman have investigated the degradation of computation capability and diagnosability in terms of the number of switches to connect modules [8,9]. They have also proposed a methodology for the design of reconfigurable multi-module systems. Fortes and Raghavendra have examined the design of reconfigurable array processors with VLSI chips and analyzed the improved reliability, performability, computation capability, and the additional hardware cost [10].

We classify the conventional system reconfigurations discussed above as a passive action, since it is performed only when a fault is detected. 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 construct.

In this report, we are concerned with developing a quantitative method for design and analysis of the reconfiguration of a multi-module system. Particularly, we will derive an optimal reconfiguration strategy under which the system is able to be optimally reconfigured to have high performance and survive the occurrences of subsystem failure during the entire mission lifetime.

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 is capable of assigning a module to be a *redundant* unit, a *functioning* unit, or a *spare* unit. Functioning modules are executing 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 multi-module 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 multi-module 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, while the latter providing higher performance. Some criterion is needed to judge the goodness of different configurations. Based on the criterion, it is possible that the best configuration at a particular moment no longer becomes the best one at another moment. In such a case, reconfiguration is needed even if there is no occurrence of failure; we term this *active reconfiguration*. Thus, we need to have an optimal reconfiguration strategy which specifies the optimal configurations for the whole mission lifetime. Since it is invoked for both the cases of failure occurrence as well as no occurrence of failure, the active reconfiguration subsumes the conventional passive reconfiguration.

This report is organized as follows. In Section 2, we introduce several convenient concepts, notations, and terminologies for reconfiguration of multi-module systems. Then, we develop a criterion that will be used for judging the goodness of configurations. The need of active reconfiguration is also justified in this section. Section 3 examines the properties of an optimal reconfiguration strategy during the mission lifetime. Also,

presented is an algorithm for determining 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. The report concludes with Section 5.

2. RECONFIGURATION STRATEGY

Consider a multi-module system which begins its mission with m_0 identical modules. Let the mission lifetime be t_0 during which no system repair is allowed, i.e. a non-repairable system. We consider here only the failures which are caused by hardware faults and will -- along with the progression of the mission -- trigger system reconfiguration. Transient and intermittent faults from which the system can be recovered through retry [11,12] are not considered, since they do not cause reconfiguration.

As we shall see, the optimal configurations are generated *off-line* in table form and, therefore, the (on-line) overhead of reconfiguration is simply task switching times. These do not generate any significant impacts 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 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. Due to these reasons, a reliability analysis including the impact of failure handling on executing tasks is more general and powerful than the classical methods [13]. In addition, system performance should depend upon 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 it is completed successfully, and will suffer the same penalty if its execution is unsuccessful. Although the pattern of the incoming tasks could change in reality, we assume for simplicity that the combined workload in each task class is stationary throughout the entire mission, i.e. the task arrival rate and the required computations in each task class are constant.[2]

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 $m(t) \in \{0,...,m_0\}$ modules. Let $m_i(t)$ be the number of modules assigned to the class i tasks. With these $m_i(t)$ modules, $n_i(t)$ *computing clusters* are to be constructed. Each computing cluster is a computation unit and could have 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 simplexes (without redundancy), dyads (with one redundant module each), triads (with two redundant modules each), etc. For notational simplicity, we will leave out the time

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

dependency of m_i, n_i, and r_i in the rest of this report as long as it does not cause any ambiguity. Obviously, $n_i + r_i = m_i$ and $\sum_{i=1}^{k} m_i \leq m$.[3] Since all computing clusters assigned to the same task class are homogeneous insofar as their capabilities of computation and fault-tolerance are concerned, the r_i redundant modules should be equally distributed over n_i computing clusters. Thus, for the task class i there are $r_i - n_i \lfloor \frac{r_i}{n_i} \rfloor$ computing clusters with the module redundancy $\lfloor \frac{r_i}{n_i} \rfloor + 2$, and $n_i(1+\lfloor \frac{r_i}{n_i} \rfloor) - r_i$ computing clusters with the module redundancy $\lfloor \frac{r_i}{n_i} \rfloor + 1$, where $\lfloor x \rfloor$ is the maximum integer that is less than or equal to x.

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

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

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

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

$$RS_{t,m} \equiv \left\{ \gamma_{\hat{m}}(\hat{t}) \mid \hat{t} \in [0,t],\ \hat{m} \in \{0,...,m\} \right\}$$

Hence, given the initial reconfiguration strategy RS_{t_0,m_0}, the system uses the configuration $\gamma_m(t) \in RS_{t_0,m_0}$ when $RML=t$ and there are m healthy modules available.

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

2.2. Reconfiguration Model

During the mission lifetime, system degradation is unavoidable due to module failures. A simple model for system degradation is presented in Figure 1 where state S_m represents the availability of m healthy modules. The transition from S_m to S_{m-1}, $m>1$, implies failure of one module and the subsequent recovery of system operation. However, failure of one module could be fatal to the system, for example, coverage failure [14,15] or dynamic failure [13], which results in loss of the whole mission. In such a case, the system will transfer directly to the total system failure state S_0. When the system state is S_m and $RML=t$, the system has the configuration $\gamma_m(t)$.

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

Within a reconfiguration strategy RS_{t_0,m_0}, system configurations are another stochastic process, called the *reconfiguration process* and denoted by $RF_{t_0,m_0}(t)$, $t\in[0,t_0]$. $RF_{t_0,m_0}(t)$ includes a configuration $\gamma_{M(t)}(t)\in RS_{t_0,m_0}$ to be used at $RML=t$. The transitions between configurations within $RF_{t_0,m_0}(t)$ 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 does not have to change its configuration for the period δt.

For the system which is capable of graceful degradation and reconfiguration, Meyer's *performability* [16,17,18] is a useful measure of the system capability. Performability is a composite measure of performance and reliability which automatically takes into account the performance degradation due to component failures. To incorporate the concept of performability, two functions associated with the configuration $\omega\in\Omega$ are introduced here. The first is a non-negative 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 is analogous to the *reward structure* defined by Furchtgott [18], and the *reward function* defined by Donatiello and Iyer [19]. Clearly, $\rho(\omega)=0$ when $\omega\in\Omega_0$, i.e. zero reward rate in case of no module available or a system crash. Thus, the total reward accumulated during the mission lifetime t_0 is given as

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

Note that the definition of W_{t_0,m_0} is the same as Meyer's performability [16,17,18].

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 [14,15], or the probability of dynamic failure [13] 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, $\alpha(\omega)$ will depend on the number of redundant modules associated with each computing cluster. Since a configuration is 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 $\alpha(\omega)$ will be used interchangeably with $\rho(\mathbf{n})$ and $\alpha(\mathbf{n},\mathbf{r})$, respectively in the rest of this report.

Several authors have derived the distribution and the moments of performability for gracefully degradable systems under the restriction that system reconfiguration is allowed only upon failure [16,17,18,19]. 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 a new configuration. This fact can be seen easily with the following two simple cases. One is the case when the remaining mission lifetime is very short in which the probability of having failures is very small. Thus, the computation capability is more important than reliability for higher rewards. The other case is when 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.

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

For these reasons, we must first examine the effects of the remaining mission lifetime on system reconfiguration, i.e. the progression of the mission, and then choose a new configuration. Specifically, it is necessary to determine 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 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

Denote the optimal reconfiguration strategy by

$RS^*_{t_0,m_0} \equiv \left\{ \gamma^*_m(t) \mid t \in [0,t_0],\ m \in \{0,...,m_0\} \right\}$ under which the total expected reward $E[W_{t_0,m_0}]$ is maximized. 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 $r_i^*(t)$[5] for all $t \in [0, t_0]$ and $m \in \{0,...,m_0\}$ that maximize $E[W_{t_0,m_0}]$.

3.1. Problem Formulation

Based on 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 $RML = t+\delta t$, then the reward rate at that time, $W_{t+\delta t,m}$, can be expressed as:

[5]Although dependence of n_i and r_i on t is re-introduced here for clarity, it will be omitted again throughout the report.

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

Thus, a recursive expression for the expected reward is derived as follows.

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

Assume that at any moment there is at most one occurrence of failure implying that the maximum jump in $M(t)$ is one. Notice that when the system reconfigures itself into a new configuration $\gamma_m(t)$ from $\gamma_m(t+\delta t)$ or from $\gamma_{m+1}(t+\delta t)$, the system 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 $RS^*_{t^+,m} \equiv \lim_{\delta t\to 0} RS^*_{t+\delta t,m}$ 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. $RS^*_{t^+,m} = \{\gamma^*_m(t^+)\} \bigcup RS^*_{t,m} \bigcup RS^*_{t,m-1}$.

Proof. Suppose that the configuration chosen at $RML=t+\delta t$ by a reconfiguration strategy will be 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, $RS^*_{t+\delta t,m} \supset \{\gamma^*_m(t+\delta t)\} \bigcup RS^*_{t,m}$. Similarly, $RS^*_{t+\delta t,m} \supset \{\gamma^*_m(t+\delta t)\} \bigcup RS^*_{t,m-1}$ when we consider the case of one failure occurrence. On the other hand, by definition, the system does not transform into any configuration other than (i) $\gamma^*_m(t+\delta t)$ and (ii) the

configurations belonging to $RS^*_{t,m}$ and $RS^*_{t,m-1}$. Thus, Lemma 1 follows immediately, since the assumption at the beginning of this proof becomes true as $\delta t \to 0$.

Q.E.D.

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 $\mathrm{E}[W^*_{t,m}]$ and $\mathrm{E}[W^*_{t,m-1}]$, the optimal configuration $\gamma^*_m(t^+)$ can be determined by Theorem 1 below.

Theorem 1. $\gamma^*_m(t^+)$ maximizes $J_{t,m}(\omega)$ for $\omega \in \Omega_m$ where

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

Proof: >From Eq. (3), the variation of $\mathrm{E}[W_{t,m}]$ in δt is

$$\mathrm{E}[W_{t+\delta t,m}] - \mathrm{E}[W_{t,m}] = \rho(\gamma_m(t+\delta t))\,\delta t - \alpha(\gamma_m(t+\delta t))\, m\lambda\, \delta t\, \mathrm{E}[W_{t,m-1}] + m\lambda\, \delta t\, \mathrm{E}[W_{t,m-1} - W_{t,m}] \tag{5}$$

Dividing both sides of Eq. (5) by δt and taking limits as $\delta t \to 0$, we get the first derivative of $\mathrm{E}[W_{t,m}]$ with respect to t:

$$\frac{d\mathrm{E}[W_{t,m}]}{dt} = \rho(\gamma_m(t^+)) - \alpha(\gamma_m(t^+))\, m\lambda\, \mathrm{E}[W_{t,m-1}] - m\lambda\, \mathrm{E}[W_{t,m} - W_{t,m-1}] \tag{6}$$

The last term in the right-hand side of Eq. (6) is independent of the configuration $\gamma_m(t^+)$. By Lemma 1, $\gamma^*_m(t^+)$ maximizes the first two terms of the right-hand side of Eq. (6) over all $\omega \in \Omega_m$. Since this is true for all $\mathrm{E}[W_{t,m-1}]$ including $\mathrm{E}[W^*_{t,m-1}]$, the theorem is proved. **Q.E.D**

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

Problem OR: $\underset{\mathbf{n},\,\mathbf{r}}{\text{maximize}} \quad J_{t,m}(\mathbf{n},\mathbf{r}) = \rho(\mathbf{n}) - \alpha(\mathbf{n},\mathbf{r})\, m\lambda\, \mathrm{E}[W^*_{t,m-1}]$

$$\text{subject to} \quad \sum_{i=1}^{k}(n_i+r_i) \leq m, \text{ and } n_i, r_i \in I^+ \text{ for } i=1,2,..,k.$$

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 to this difficulty, we will in the next section 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 interval. Given m, it is therefore natural to look for the *switch times* at which the optimal configuration has to be changed to maximize the expected total 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 $s^j_m\in[0,\infty]$ $j=0,1,2,...$ and $m=1,2,...,m_0$ be the switch times for $RS^*_{t_0 m_0}$ where $s^0_m=0$, and 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 $j\geq 1$. Thus, 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 $RML=s^{j-1}_m$.

For two configurations ω_1 and ω_2, one can say that ω_1 is *better* than ω_2 if $\alpha(\omega_1)=\alpha(\omega_2)$ and $\rho(\omega_1)>\rho(\omega_2)$. Thus, in the following discussions we are concerned only with configurations with different crash probabilities, i.e. $\alpha(\omega_i)\neq\alpha(\omega_j)$, for all $i\neq j$. For convenience define a function $A_m:\Omega_m\times\Omega_m\to\mathbf{R}$ as follows.

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

where $\alpha(\omega_1)\neq\alpha(\omega_2)$ as mentioned above. Also, define two convenient sets

$$L_m(s_m^j)\equiv\left\{\hat{\omega}\in\Omega_m \mid \alpha(\hat{\omega})<\alpha(\gamma_m^*(s_m^j))\right\} \quad \text{and} \quad G_m(s_m^j)\equiv\left\{\hat{\omega}\in\Omega_m \mid \alpha(\hat{\omega})>\alpha(\gamma_m^*(s_m^j))\right\}.$$

Then, the following theorem elucidates a useful property of the switch times s_m^j.

Theorem 2. For all $t\in(s_m^{j-1},s_m^j]$

$$\min_{\omega\in L_m(s_m^j)} A_m(\gamma_m^*(s_m^j),\omega) \geq \lambda\, E[W_{t,m-1}^*] \geq \max_{\omega\in G_m(s_m^j)} A_m(\gamma_m^*(s_m^j),\omega) \tag{7}$$

Proof: Since $\gamma_m^*(t)=\gamma_m^*(s_m^j)$ for all $t\in(s_m^{j-1},s_m^j]$, we have the following relationship from Theorem 1:

$$\rho(\gamma_m^*(s_m^j)) - \alpha(\gamma_m^*(s_m^j))\, m\lambda\, E[W_{t,m-1}^*] \geq \rho(\omega) - \alpha(\omega)\, m\lambda\, E[W_{t,m-1}^*] \quad \text{for all } \omega\in\Omega_m.$$

Therefore, $\dfrac{\rho(\gamma_m^*(s_m^j)) - \rho(\omega)}{m\alpha(\gamma_m^*(s_m^j))-m\alpha(\omega)} \geq \lambda E[W_{t,m-1}^*]$ for all ω satisfying $\alpha(\omega)<\alpha(\gamma_m^*(t))$, and

$\dfrac{\rho(\gamma_m^*(s_m^j))-\rho(\omega)}{m\alpha(\gamma_m^*(s_m^j))-m\alpha(\omega)} \leq \lambda E[W_{t,m-1}^*]$ for all ω satisfying $\alpha(\omega)>\alpha(\gamma_m^*(t))$.

Q.E.D.

Note that $E[W_{t,m-1}^*]$ is a function of t. However, the function $A_m(\gamma_m^*(s_m^j),\omega)$ is dependent upon configurations only. Once the optimal configuration $\gamma_m^*(s_m^j)$ is known, $\min_{\omega\in L_m(s_m^j)} A_m(\gamma_m^*(s_m^j),\omega)$ and $\max_{\omega\in G_m(s_m^j)} A_m(\gamma_m^*(s_m^j),\omega)$ can be determined. The only computation needed to determine s_m^{j+1} is to calculate $E[W_{t,m-1}^*]$ recursively by Eq. (5) until Eq. (7) becomes invalid. Thus, Theorem 2 provides the solution for s_m^{j+1}, which is the infimum of $\{t \mid t>s_m^j$ and t that does not satisfy Eq. (7) $\}$. If Eq. (7) is valid for all $t>s_m^j$, then $s_m^{j+1}=\infty$. Theorem 2 also presents a special property of an optimal

configuration. With an optimal configuration $\gamma_m^*(s_m^j)$, $A_m(\gamma_m^*(s_m^j),\omega)$ *partitions* Ω_m into two sets, $L_m(s_m^j)$ and $G_m(s_m^j)$, as shown above. An arbitrary configuration without this property can never be an optimal configuration regardless of the remaining mission life-time. It is important to notice that s_m^j represents the reconfiguration time independent of the presence of a module failure or not. Moreover, the solution to the **OR** problem is embedded in the process of solving for 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 non-decreasing in t.

Proof: >From Eq. (6), we can write a first order differential equation of $E[W_{t,m}]$ with respect to t as follows.

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

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

To prove that $E[W_{t,m}^*]$ is non-decreasing in t, we must show that $E[W_{t+\Delta t,m}^*] \ge E[W_{t,m}^*]$ for all $\Delta t\ge 0$. Let $RS_{t+\Delta t,m} \equiv \{\gamma_m(\hat{t}) \mid 0\le\hat{t}\le t+\Delta t,\ \gamma_m(\hat{t})=\gamma_m^*(\hat{t}-\Delta t)$ if $\hat{t}\ge\Delta t$, or $\gamma_m^*(0^+)$ if $\hat{t}<\Delta t\}$. Then, it is easy to see that the expected reward based on $RS_{t+\Delta t,m}$ is larger than $E[W_{t,m}^*]$. Thus, $E[W_{t+\Delta t,m}^*]\ge E[W_{t,m}^*]$. **Q.E.D.**

Lemma 3. For real numbers a_i and b_i, i=1,2,3, where $b_1>b_2>b_3$, $\frac{a_1-a_2}{b_1-b_2} \le \frac{a_1-a_3}{b_1-b_3}$ if and only if $\frac{a_1-a_2}{b_1-b_2} \le \frac{a_2-a_3}{b_2-b_3}$. Also, $\frac{a_1-a_2}{b_1-b_2} \ge \frac{a_1-a_3}{b_1-b_3}$ if

and only if $\frac{a_1-a_3}{b_1-b_3} \geq \frac{a_2-a_3}{b_2-b_3}$.

Proof: The proof is trivial and thus omitted.

Let $\Phi_m(s_m^j) \equiv \left\{ \hat{\omega} \mid \hat{\omega} \in \Omega_m,\ A_m(\gamma_m^*(s_m^j),\hat{\omega}) = \min_{\omega \in L_m(s_m^j)} A_m(\gamma_m^*(s_m^j),\omega) \right\}$. Then, $\Phi_m(s_m^j) \neq \phi$ if $L_m(s_m^j) \neq \phi$. 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 $s_m^j < \infty$, then $\gamma_m^*(s_m^{j+1}) \in \Phi_m(s_m^j)$, and $\alpha(\gamma_m^*(s_m^{j+1})) \leq \alpha(\hat{\omega})$ for all $\hat{\omega} \in \Phi_m(s_m^j)$.

Proof: >From the definition of switch time, $\gamma_m^*(s_m^{j+1})$ exists if $s_m^j < \infty$. First, we prove that $\alpha(\gamma_m^*(s_m^{j+1})) < \alpha(\gamma_m^*(s_m^j))$. Since $\mathrm{E}[W_{t,m}^*]$ is non-decreasing, we have

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

for $t \in (s_m^j, s_m^{j+1}]$. Thus, $J_{t,m}(\gamma_m^*(s_m^j)) \geq J_{t,m}(\omega)$ 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 that $\Phi_m(s_m^j) \neq \phi$.

Next, we prove $\gamma_m^*(s_m^{j+1}) \in \Phi_m(s_j)$. When there is one and only one $\omega \in L_m(s_m^j)$, the theorem automatically holds in case of $s_m^j < \infty$, since $\gamma_m^*(s_m^{j+1})$ exists and $\alpha(\gamma_m^*(s_m^{j+1}))$ cannot be larger than $\alpha(\gamma_m^*(s_m^j))$. Consider a configuration $\omega \in \{\hat{\omega} \mid \alpha(\hat{\omega}) < \alpha(\gamma_m^*(s_m^{j+1})) < \alpha(\gamma_m^*(s_m^j)),\ \hat{\omega} \in \Omega_m\}$. Since $\gamma_m^*(s_m^{j+1})$ is optimal and $\mathrm{E}[W_{t,m}^*]$ is continuous (and so is $\mathrm{E}[W_{t,m-1}^*]$), the following order can be obtained from Theorem 2:

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

where $t \in (s_m^j, s_m^{j+1}]$. Appling Lemma 3, we have $A_m(\gamma_m^*(s_m^j),\omega) \geq A_m(\gamma_m^*(s_m^{j+1}),\gamma_m^*(s_m^j))$.

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

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

Q.E.D.

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

Corollary 1. $\alpha(\gamma_m^*(s_m^i)) < \alpha(\gamma_m^*(s_m^j))$ and $\rho(\gamma_m^*(s_m^i)) < \rho(\gamma_m^*(s_m^j))$ for all $i > j$.

Proof: In Theorem 3, we proved $\alpha(\gamma_m^*(s_m^{j+1}))<\alpha(\gamma_m^*(s_m^j))$. Since $A_m(\gamma_m^*(s_m^j),\gamma_m^*(s_m^{j+1})) \geq E[W^*_{s_m^j,m-1}]>0$, we have $\rho(\gamma_m^*(s_m^{j+1}))<\rho(\gamma_m^*(s_m^j))$. **Q.E.D.**

Definition: 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, $\gamma_m^*(s_m^i)\neq\gamma_m^*(s_m^j)$ for all $i\neq j$. Also, if 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 $m\in\{0,...,m_0\}$ and $j\geq 0$, if $s_m^j<\infty$, then $\dfrac{\rho(\gamma_m^*(s_m^{j+1}))}{\alpha(\gamma_m^*(s_m^{j+1}))} \geq \dfrac{\rho(\gamma_m^*(s_m^j))}{\alpha(\gamma_m^*(s_m^j))}$.

Proof: Consider a remaining mission lifetime $t\in(s_m^j,s_m^{j+1}]$. Since $E[W^*_{t,m}]$ is non-decreasing and continuous in t, $\dfrac{dE[W^*_{t,m}]}{dt} \geq 0$ and, therefore, we have $\dfrac{\rho(\gamma_m^*(s_m^{j+1}))}{m\alpha(\gamma_m^*(s_m^{j+1}))} \geq \lambda E[W^*_{t,m-1}]$ from Eq. (6). >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, $\dfrac{\rho(\gamma_m^*(s_m^{j+1}))}{\alpha(\gamma_m^*(s_m^{j+1}))} \geq \dfrac{\rho(\gamma_m^*(s_m^j))}{\alpha(\gamma_m^*(s_m^j))}$.

Q.E.D.

Definition: 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, i.e. $(\alpha(\gamma_m^*(s_m^{j+1})),\rho(\gamma_m^*(s_m^{j+1})))$, is 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 Eq. (7) is valid for all $t>s_m^{j-1}$, s_m^j becomes infinite. Though the optimal configurations can be indicated from their coordinates, the switch times have to be computed based on 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 $\gamma_m^*(s_m^j)$ and $E[W_{t,m}^*]$ are calculated using $\gamma_{m-1}^*(s_m^j)$ and $E[W_{t,m-1}^*]$ for all $t\in[0,t_0]$. Complete algorithms to solve for $\gamma_m^*(s_m^j)$ 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 $m\in\{0,...,m_0\}$, 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

1.a For $m=1,2,..m_0$, find $\gamma_m^*(s_m^1)$ in which n maximizes $\rho(\omega)$, $\omega\in\Omega_m$.[6] When there is more than one configuration which maximizes $\rho(\omega)$, the configuration with the smallest crash probability is chosen.

[6]It is easy to prove that $\gamma_m^*(s_m^1)$ maximizes $\rho(\omega)$ from Theorem 1 and $E[W_{0,m}^*]=0$.

1.b Find $E[W^*_{t,1}]=\rho(\gamma_1^*(s_1^1))\frac{1}{\lambda}(1-e^{-\lambda t}-\lambda t e^{-\lambda t})$ for $t\in[0,t_0]$. (Note that reconfiguration is not needed when $m=1$).

1.c Choose δt to digitize $t\in[0,t_0]$.

1.d 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 $\gamma_m^*(s_m^j)\in\Phi_m(s_m^{j-1})$ and $\alpha(\gamma_m^*(s_m^j))\leq\alpha(\hat{\omega})$ for $\hat{\omega}\in\Phi_m(s_m^{j-1})$.

Step 4. Using $\gamma_m^*(s_m^{j-1})$ as the optimal configuration, calculate $E[W^*_{t+\delta t,m}]$ by Eq. (5).

Step 5. Set $t:=t+\delta t$. **If** $t\geq t_0$, go to Step 2

else if $A_m(\gamma_m^*(s_m^{j-1}),\gamma_m^*(s_m^j))\geq E[W^*_{t,m-1}]$, go to Step 4

else $j:=j+1$ and go to Step 3.

4. DETERMINATION OF OPTIMAL CONFIGURATION

As we defined in Section 2, a configuration $\omega\in\Omega_m$ can be specified by $(\mathbf{n},\mathbf{r})$ where $\mathbf{n}=[n_1,n_2,..,n_k]$ represents the number of computing clusters for the task class i, and $\mathbf{r}=[r_1,r_2,..,r_k]$ 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 $\frac{1}{m}\sum_{i=1}^{k}(n_i+r_i)\,\alpha_i(n_i,r_i)$ 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-sever system. Ideally, the performance of the system is additive in x, i.e. $f(x_1+x_2) = f(x_1)+f(x_2)$. However, due to task communications, resource sharing, and other overheads, the performance of the system is sub-additive, i.e. $f(x_1)+f(x_2) \geq f(x_1+x_2)$. In most cases, $f(x)$ is a non-decreasing concave function of x as evidenced in [20,21,22,23]. Following the evidenced tradition, we will consider here the cases in which the reward rate for each task class, $\rho_i(n_i)$, is a non-decreasing concave function of the number of computing clusters, n_i.

Let $h_i(n)$ be the crash probability of an n-module redundant computing cluster in the task class i. We assume that $h_i(n)$ is a decreasing convex function of n for all $i=1,2,...,k$. For simplicity, we assume that $n\,h_i(n)$ is also convex in n.[7] Thus, the crash probability of the task class i is given by:

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

where $\hat{r}_i = \lfloor \frac{r_i}{n_i} \rfloor$. This implies that $(n_i+r_i)\,\alpha_i(n_i,r_i)$ is a piecewise linear and convex function of r_i. When $n_i\hat{r}_i \leq r_i \leq n_i(\hat{r}_i+1)$, $\hat{r}_i \in I^+$, the slope of the corresponding linear piece is $(\hat{r}_i+2)\,h_i(\hat{r}_i+2) - (\hat{r}_i+1)\,h_i(\hat{r}_i+1)$.

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 P_0, is a nonlinear *integer knapsack programming problem* which has to be solved for determining $\gamma_m^*(s_m^1)$.

---

[7]Relaxation of these two assumptions will be discussed in Section 5.

$$\mathbf{P}_0: \quad \max_{n_i} \sum_{i=1}^{k} \rho_i(n_i) \tag{8}$$

$$\text{subject to} \quad \sum_{i=1}^{k} n_i \leq m \quad \text{and} \quad n_i \in I^+ \text{ for } i=1,2,...,k.$$

The second, called $\mathbf{P}_1$, is an ***integer fractional programming problem*** which is to determine $\gamma_m^*(s_m^{j+1})$, given $\gamma_m^*(s_m^j)$. $\mathbf{P}_1$ can be expressed as:

$$\mathbf{P}_1: \quad \min_{n_i, r_i} \frac{\rho(\omega_0) - \sum_{i=1}^{k} \rho_i(n_i)}{m\alpha(\omega_0) - \sum_{i=1}^{k} (n_i + r_i)\, \alpha_i(n_i, r_i)} \quad \text{given} \quad \omega_0 \in \Omega_m \tag{9}$$

$$\text{subject to} \quad \sum_{i=1}^{k} (n_i + r_i)\, \alpha_i(n_i, r_i) < m\alpha(\omega_0), \quad \sum_{i=1}^{k} \rho_i(n_i) < \rho(\omega_0) \quad \text{and}$$

$$\sum_{i=1}^{k} (n_i + r_i) \leq m, \quad n_i, r_i \in I^+ \text{ for } i=1,2,...,k.$$

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, capital budgeting, etc. Several methods have been proposed for solving this problem; for example, dynamic programming approaches [24,25], the shortest path algorithm [26], and ranking methods [27,28]. Michaeli and Pollatschek investigated the problem and provided a necessary and sufficient condition for the optimal solution [29]. In this report, we develop a simple but elegant algorithm for problem $\mathbf{P}_0$, under the assumption that $\rho_i(n_i)$ is a concave function of n_i.[8]

Let $\Delta_i(n_i) = \rho_i(n_i + 1) - \rho_i(n_i)$. The principle of the algorithm is to allocate

[8]Our algorithm is extremely efficient when m is not too large. According to [29], a general solution for the problem $\mathbf{P}_0$ is known to be intractable. However, the algorithm we developed is very simple as will be shown below.

modules such that the system can have maximum return.[9]

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.

Clearly, it is not necessary to sort all $\Delta_i(n_i)$ for every m. $\Delta_i(0)$ must be sorted during the first 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.

Theorem 4. The result obtained from Algorithm A_0, denoted by $\mathbf{n}^* = [n_1^*, n_2^*, \ldots, n_k^*]$, is a solution to $\mathbf{P}_0$.

Proof: First, we introduce an additional task class $k+1$ where $\rho_{k+1}(j)=0$ for all $j \in I^+$.[10] Thus, the problem is converted to:

$$\max_{n_i} \sum_{i=1}^{k+1} \rho_i(n_i)$$

$$\text{subject to } \sum_{i=1}^{k+1} n_i = m \text{ and } n_i \in I^+ \text{ for } i=1,2,\ldots,k+1$$

[9] The problem $\mathbf{P}_0$ is not in the general form of the nonlinear integer knapsack programming problem, since the coefficients of the linear constraint are all equal as indicated in Eq. (8).

[10] The modules in this task class can be regarded as consisting of spare modules. So, the reconfiguration problems for a system with and without spare modules can be treated as the same.

As shown in [29], a necessary and sufficient condition[11] for the above problem is $\rho_i(n_i)+\rho_j(n_j) \geq \rho_i(n_i+x)+\rho_j(n_j-x)$ for each pair i, j (i=1,2,...,k+1; j=1,2,...,k+1; and $i \neq j$) and integer x where $\max(-1,-n_i) \leq x \leq \min(n_j,1)$. Thus, to prove the theorem, we only need to show that the above equation is valid for x=1 when $n_j > 0$. If n_i^* and n_j^* are obtained from Algorithm $\mathbf{A}_0$, then $\Delta_j(n_j^*-1) \geq \Delta_i(n_i^*)$. Thus, the theorem follows. **Q.E.D.**

Another advantage of Algorithm $\mathbf{A}_0$ is that all $\gamma_m^*(s_m^1)$ can be solved at once. By assuming m_0 at the beginning, $\gamma_m^*(s_m^1)$, $m \in \{0,...,m_0\}$ is obtained at the end of the m-th 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,..,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:

$$\mathbf{P}_2: \quad \min_{r_i} \sum_{i=1}^{k} (n_i+r_i)\,\alpha_i(n_i,r_i) \tag{10}$$

$$\text{subject to } \sum_{i=1}^{k} r_i \leq m-\sum_{i=1}^{k} n_i \text{ and } r_i \in I^+ \text{ for } i=1,2,...,k$$

The problem $\mathbf{P}_2$ is an integer knapsack programming problem which 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

[11] The necessary and sufficient condition given in [29] is used for minimizing the sum of convex functions. Here, we consider the maximization of the sum of concave functions.

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 non-convex 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:

$$\mathbf{P}_3: \quad \min_{n_i} \frac{\rho(\omega_0)-\rho(\mathbf{n})}{m\alpha(\omega_0)-\beta(\mathbf{n})} \tag{11}$$

$$\text{subject to } \beta(\mathbf{n})<m\alpha(\omega_0) \text{ and } \rho(\mathbf{n})<\rho(\omega_0)$$

$$\sum_{i=1}^{k} n_i \le m \text{ and } n_i \in I^+ \text{ for } i=1,2,\ldots,k.$$

>From Corollary 3, the triangle area surrounded by (0,0), $(0,\rho(\mathbf{n}^0))$, and $(\frac{1}{m}\beta(\mathbf{n}^0),\rho(\mathbf{n}^0))$ 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 to find whether 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 constraint $\sum_{i=1}^{k} n_i \le m$, we map the combination into r_i^*, $\beta(\mathbf{n})$, and $\frac{\rho(\omega_0)-\rho(\mathbf{n})}{m\alpha(\omega_0)-\beta(\mathbf{n})}$. 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

$\sum_{i=1}^{k} n_i \leq m$ is $\binom{m+k}{k}$. Thus, when the size of the system is moderate, the explicit enumeration approach is reasonable. For example, when k=3 and m=12, the total number of enumerations is 455.

Moreover, there are two important and advantageous properties associated with the explicit enumeration: (i) The coordinates of *all* feasible configurations for m available modules, that is $(\frac{1}{m}\ \beta(\mathbf{n}),\rho(\mathbf{n}))$, can be obtained from the enumeration. Thus, from Theorem 3, we can easily determine all optimal configurations, $\gamma_m^*(s_m^j)$; (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 $m\in\{1,...,m_0\}$. Therefore, the optimal configurations $\gamma_m^*(s_m^j)$ for $m\in\{1,...,m_0\}$ can be determined. These two properties lead to a situation in the determination of reconfiguration strategy that the explicit enumeration has to be conducted only once.

When both k and m are large, the explicit enumeration becomes intractable. An immediate candidate for approximate solutions is to aggregate task classes into a small number of groups and then perform the explicit enumerations for the system and for each group.

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 [30]. For a continuous nonlinear fractional programming problem, some equivalent or dual problems have been proposed in [31,32]. With integer constraints,

Chandra and Chandramohan [33] suggested to apply 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 [34] 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 which is briefly discussed below. Let F_3 denote the feasible region for $\mathbf{P}_3$. Define the auxiliary problem $\mathbf{Q}(\eta)$ by

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

$$\mathbf{n} \in F_3$$

Let $z(\eta)$ be the minimum value of $\mathbf{Q}(\eta)$. It is shown that if $z(\eta)=0$, then an optimal solution of $\mathbf{Q}(\eta)$ is also an optimal solution of $\mathbf{P}_3$ [34]. Hence, an algorithm is needed to search for η^* such that $z(\eta^*)\approx 0$ by solving $\mathbf{Q}(\eta)$ iteratively. Thus, the complexity of this algorithm is based on the efficiency of solving $\mathbf{Q}(\eta)$ and the search algorithm, e.g. Newton's method, binary search, etc. Meggido [35] proposed an algorithm which 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 problem **OR**, i.e. a nonlinear knapsack programming problem. It might be more efficient to solve the **OR** problem directly instead of searching for η^*. When $\beta(\mathbf{n})$ is convex with respect to n_i, the objective function becomes concave 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:

$$P_j(\mathbf{n}) \equiv (n_1, n_2, .., n_{j-1}, n_j + 1, n_{j+1}, \ldots, n_k)$$

$$\Delta_i(\mathbf{n}) \equiv \rho(P_i(\mathbf{n})) - \rho(\mathbf{n}) - \lambda E[W_{t_h, m-1}](\beta(P_i(\mathbf{n})) - \beta(\mathbf{n}))$$

Algorithm A_1

Step 1. Set $L:=0$ and $U:=t_0$.

Step 2. (a). Set $t_h := L + \frac{h(U-L)}{N}$ for $h=1,2,...,N$.

(b). For $h=1,2,...,N$

(b1). Set $dm:=m$ and $n_i:=0$ for $i=1,2,...,k$.

(b2). Solve for $\beta(P_i(\mathbf{n}))$ as indicated in the solution of $\mathbf{P}_2$ and $\Delta_i(\mathbf{n})$ for $i=1,2,...,k$.

(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 the small number of partitions of the mission lifetime. 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 to 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 and the mission lifetime is 1000. 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 1. In addition, we include the constraints $n_i \geq 1$ for i=1,2,3, indicating that at least one computing cluster must be available for each task class throughout the entire mission.

Applying the explicit enumeration given in Section 4.2, we can easily find all $\gamma_m^*(s_m^j)$ for $4 \leq m \leq 12$ which are listed in Table 2. Table 3 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, $\gamma_5^*(s_5^3)$ is one of them since $s_5^3 = \infty$.

As shown in this example, 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 2 and 3). In this way, the system will follow 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, the configuration trajectories, reward rates, and crash probabilities are plotted if the module failures occur when *RML* are 800, 520, 400, and 310.[12]

5. DISCUSSION

5.1. Concluding Remarks

In this report, we have addressed the problem of reconfiguring non-repairable multi-module systems. Since we treated the problem for general multi-module computing systems, both the problem formulation and the solution approach of this report have

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

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 which 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 which 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 progression of the mission and the degradation of the system due to module failures.

Using the expected total reward or Meyer's performability 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* for which we have developed a simple but elegant algorithm. The second is a *fractional programming problem*, which is basically a non-convex programming problem. For the fractional programming problem we have used an explicit enumeration which is effective for small m and k.

We have also discussed the other approaches known for solving the fractional programming problem. They do not appear any better than the explicit enumeration or solving the optimization problem, **OR**, directly. Also, note that the elimination of this assumption is equivalent to dynamic classification of incoming tasks.

As shown in the example of Section 4.3, the optimal reconfiguration strategy can be represented by two tables: one is for optimal configurations; the other for switch times. Although the solution procedures to obtain 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 used can be relaxed at the expense of 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 non-convex and thus, are more difficult to solve.

We can also remove the assumption that the reward rate and the crash probability must be stationary, i.e. 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

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

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 $E[R(W_{t_0,m_0})]$. Note that $R(W)$ should be non-decreasing but may not be continuous. It could be a step function as in the example in [19], or any other discontinuous function with finite jumps. When $R(W)$ is not additive, i.e. $R(W_1+W_2)\neq R(W_1)+R(W_2)$, 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 of further research.

ACKNOWLEDGMENT

The authors wish to thank Michael Woodbury for carefully reading this report 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] H. J. Siegel, R. J. McMillen, and P. T. Mueller, Jr., "A Survey of Interconnection Methods for Reconfigurable Parallel Processing System," *Proc. AFIPS 1979 Nat. Comp. Conf.*, pp. 529-542, June 1979.

[2] S. L. Kartashev and S. P. Kartashev, "A Multicomputer System with Dynamic Architectures" *IEEE Trans. Comput.*, Vol. C-28, No. 10, pp. 704-721, Oct. 1979.

[3] G. B. Adams and H. J. Siegel, "A Fault-tolerant Interconnection Network for Supersystems," *IEEE Trans. Comput.*, Vol. C-31, pp. 443-454, May 1982.

[4] R. Troy, "Dynamic Reconfiguration: An Algorithm and its Efficiency Evaluation," *Proc. 9th Int. Symp. Fault-tolerant Comput.*, pp. 44-49, 1979.

[5] G. Barigazzi, A. Ciuffoletti, and L. Strigini, "Reconfiguration Procedure in a Distributed Multiprocessor System," *Proc. 12th Int. Symp. Fault-tolerant Comput.*, pp. 73-80, 1982.

[6] K. H. Kim, " Error Detection, Reconfiguration and Recovery in Distributed Processing System," *Proc. 1st Int'l. Conf. on Distributed Computing Systems*, pp. 284-295, Oct. 1979.

[7] R. Y. Kain and W. R. Franta, "Interprocess Communication Schemes Supporting System Reconfiguration," *IEEE COMPSAC*, pp. 365-371, 1980.

[8] F. Saheban and A. D. Friedman, "Diagnostic and Computational Reconfiguration in Multiprocessor System," *ACM Proc. of 1978 Annual Con.*, pp. 68-78, 1978.

[9] F. Saheban and A. D. Friedman, "A Survey and Methodology of Reconfigurable Multi-Module System," *IEEE COMPSAC*, pp. 790-796, 1978.

[10] J. A. B. Fortes and C. S. Raghavendra, "Dynamically Reconfigurable Fault-Tolerant Array Processors," *Proc. 14th Int. Symp. Fault-tolerant Comput.*, pp. 386-392, 1984.

[11] K. G. Shin and Y. H. Lee, "Error Detection Process-Model, Design and Its Impact on Computer Performance," *IEEE Trans. Comput.*, Vol. C-33, No. 6, pp. 529-540, June 1984.

[12] Y. H. Lee and K. G. Shin, "Optimal Design and Use of Retry in Fault Tolerant Real-time Computer Systems," *CRL-TR-28-84*, Computing Research Lab, The University of Michigan, Ann Arbor, MI, May 1984.

[13] C. M. Krishna and K. G. Shin, "Performance Measures for Multiprocessor Controllers," *Performance '83: Ninth Int'l Symp. Comp. Perf., Meas., and Eval.*, pp. 229-250, 1983.

[14] J. J. Stiffler and L. A. Bryant, "CARE III Phase Report - Mathematical Description," *NASA Rep. 3566,* Nov. 1982.

[15] K. Trivedi, J. B. Dugan, R. Geist, and M. Smotherman, "Modeling Imperfect Coverage in Fault-Tolerant System," *Proc. 14th Int. Symp. Fault-tolerant Comput.*, pp.77-82, 1984.

[16] J. F. Meyer, "On Evaluating the Performability of Degrading Computer Systems," *IEEE Trans. Comput.*, Vol. C-29, No. 8, pp. 720-731, August 1980.

[17] J. F. Meyer, "Closed-form Solutions of Performability," *IEEE Trans. Comput.*, Vol. C-31, No. 7, pp. 648-657, July 1982.

[18] D. G. Furchtgott, "Performability Models and Solutions," *CRL-TR-8-84,* Computing Research Laboratory, The University of Michigan, Ann Arbor, Michigan, 1984.

[19] L. Donatiello and B. R. Iyer, "Analysis of a Composite Performance Reliability Measure for Fault Tolerant Systems," *RC-10325,* IBM Thomas J. Watson Research Center, Yorktown Heights, NY, Jan. 1984.

[20] W. W. Chu, L. J. Holloway, M.-T. Lan, and K. Efe, "Task Allocation in Distributed Data Processing," *Computer,* Vol. 13, No. 11, pp. 57-70, Nov. 1980.

[21] D. P. Bhandarkar, "Some Performance Issues in Multiprocessor System," *IEEE Trans. Comput.*, Vol. C-26, No. 5, pp. 506-511, May 1977.

[22] S. H. Fuller, J. K. Ousterhout, L. Raskin, P. L. Rubinfield, P. J. Sindhu and R. J. Swan, "Multi-microprocessors: An Overview and Working Example," *Proc. IEEE,* Vol. 66, No. 2, pp. 216-228, Feb. 1978.

[23] W. A. Wulf and C. G. Bell, "C.mmp - A Multi-mini-Processor," *1972 Fall Joint Computer Conf., AFIPS Cof. Proc.*, Vol. 41, pp. 765-777, 1972

[24] M. W. Cooper, "The Use of Dynamic Programming Methodology for the Solution of a Class of Nonlinear Programming Problem," *Naval Research Logistics Quarterly,* Vol. 27, pp.89-95, 1980

[25] T. L. Morin and R. E. Marsten, "An Algorithm for Non-linear Knapsack Problem," *Management Science,* Vol. 22, No. 10, pp. 1147-1158, June 1976.

[26] A. M. Frieze, "Shortest Path Algorithms for Knapsack Type Problem," *Mathematical Programming,* Vol. 11, pp.150-157, 1976.

[27] H. Luss and S. K. Gupta, "Allocation of Effort Resources among Competing Activities," *Operations Research,* Vol. 5, pp. 613-629, 1975.

[28] P. H. Zipkin, "Simple Ranking Methods for Allocation of one Resource," *Management Science,* Vol. 26, No. 1, pp. 34-43, Jan. 1980.

[29] I. Michaeli and M. A. Pollatschek, "On Some Nonlinear Knapsack Problem," *Annals, of Discrete Math.,* Vol. 1, pp. 403-414, 1977.

[30] S. Schaible, "Minimization of Ratios," *Journal of Optimization Theory and Application,* Vol. 19, No. 2, pp. 347-352, June 1976.

[31] B. D. Craven and B. Mond, "On Fractional Programming and Equivalence," *Naval Research Logistics Quarterly,* Vol. 22, pp. 405-410, 1975.

[32] S. Schaible, "A Survey of Fractional Programming," *Generalized Concavity in Optimization and Economics,* edited by S. Schaible and W. T. Ziemba, Academic Press, pp. 417-440, 1981.

[33] S. Chandra and M. Chandramohan, "A Branch and Bound Method for Integer Nonlinear Fractional Programs'," *Z. Angew. Math. and Mech.,* Vol. 60, No. 12, pp. 735-737, 1980.

[34] T. Ibaraki, "Solving Mathematical Programming Problems with Fractional Objective Functions," *Generalized Concavity in Optimization and Economics,* edited by S. Schaible and W. T. Ziemba, Academic Press, pp. 441-472, 1981.

[35] N. Megiddo, "Combinatorial Optimization with Rational Objective Functions," *Mathematics of Operations Research,* Vol. 4, pp. 414-424, 1979.

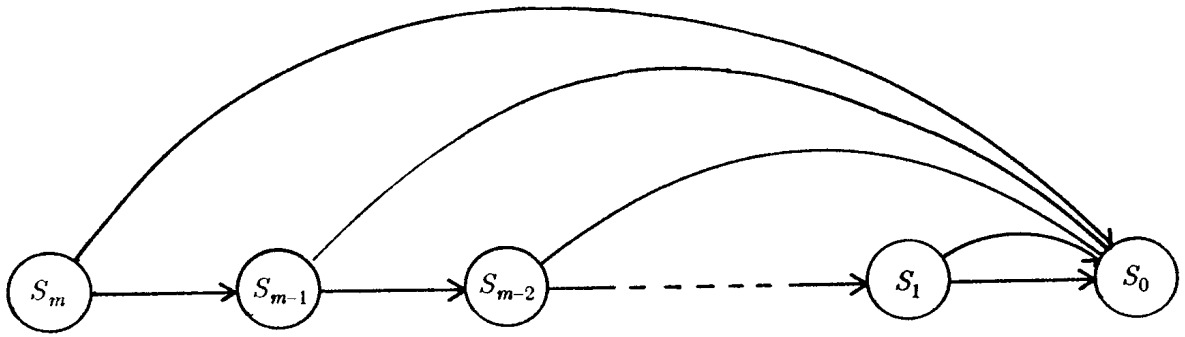


Figure 1. A Model of System Degradation.

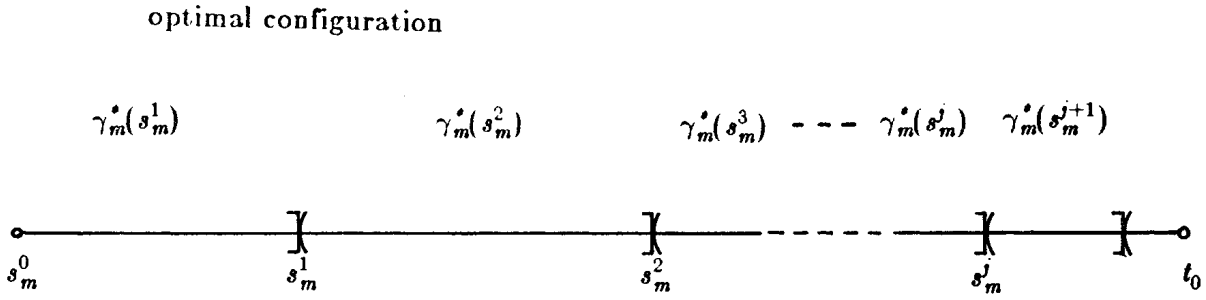


Figure 2. The Switch Times and Optimal Configurations.

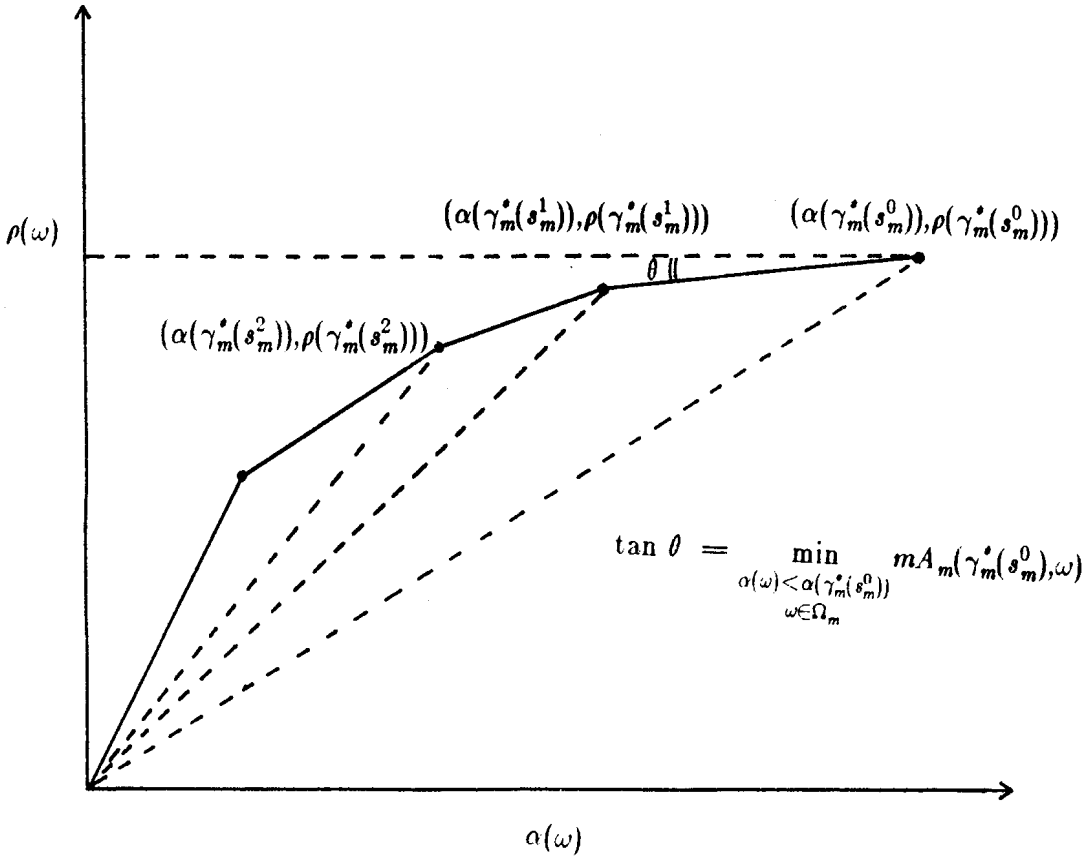
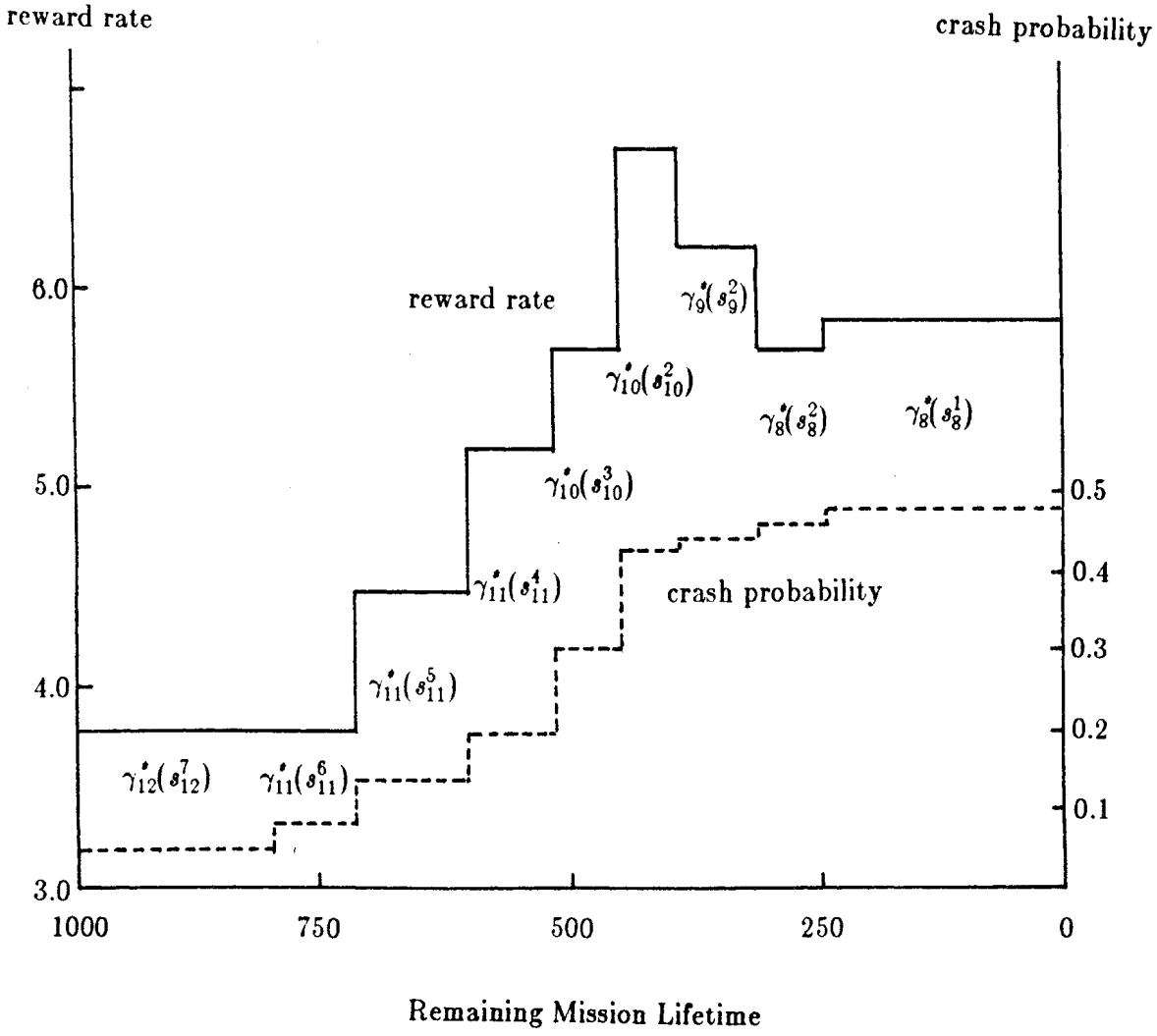


Figure 3. The Coordinates of Optimal Configurations.



Module fails when the remaining lifetime is 800, 520, 400, 310.

Figure 4. Example of Optimal Reconfiguration Trajectory.

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	0.05	0.03	0.02	0.01
$\rho_3(n)$	0.6	1.2	1.7	2.2	2.7	3.2	3.6	4.0

Table 1. Crash Probabilities $h_i(n)$ and Reward Rates $\rho_i(n)$ Used in the Example.

	$\gamma_m^*(s_m^1)$	$\gamma_m^*(s_m^2)$	$\gamma_m^*(s_m^3)$	$\gamma_m^*(s_m^4)$	$\gamma_m^*(s_m^5)$	$\gamma_m^*(s_m^6)$	$\gamma_m^*(s_m^7)$	$\gamma_m^*(s_m^8)$	$\gamma_m^*(s_m^9)$
m=4	$\begin{pmatrix}2&1&1\\0&0&0\end{pmatrix}$	$\begin{pmatrix}1&1&2\\0&0&0\end{pmatrix}$	$\begin{pmatrix}1&1&1\\1&0&0\end{pmatrix}$	-	-	-	-	-	-
m=5	$\begin{pmatrix}2&2&1\\0&0&0\end{pmatrix}$	$\begin{pmatrix}2&1&2\\0&0&0\end{pmatrix}$	$\begin{pmatrix}1&1&2\\1&0&0\end{pmatrix}$	$\begin{pmatrix}1&1&1\\1&1&0\end{pmatrix}$	-	-	-	-	-
m=6	$\begin{pmatrix}2&3&1\\0&0&0\end{pmatrix}$	$\begin{pmatrix}2&2&2\\0&0&0\end{pmatrix}$	$\begin{pmatrix}2&1&2\\1&0&0\end{pmatrix}$	$\begin{pmatrix}1&1&2\\1&1&0\end{pmatrix}$	$\begin{pmatrix}1&1&1\\2&1&0\end{pmatrix}$	-	-	-	-
m=7	$\begin{pmatrix}2&3&2\\0&0&0\end{pmatrix}$	$\begin{pmatrix}2&1&2\\2&0&0\end{pmatrix}$	$\begin{pmatrix}1&1&2\\2&1&0\end{pmatrix}$	$\begin{pmatrix}1&1&1\\2&1&1\end{pmatrix}$	-	-	-	-	-
m=8	$\begin{pmatrix}2&4&2\\0&0&0\end{pmatrix}$	$\begin{pmatrix}2&3&3\\0&0&0\end{pmatrix}$	$\begin{pmatrix}2&3&2\\1&0&0\end{pmatrix}$	$\begin{pmatrix}2&2&2\\2&0&0\end{pmatrix}$	$\begin{pmatrix}2&1&2\\2&1&0\end{pmatrix}$	$\begin{pmatrix}1&1&2\\2&1&1\end{pmatrix}$	$\begin{pmatrix}1&1&1\\2&2&1\end{pmatrix}$	-	-
m=9	$\begin{pmatrix}2&5&2\\0&0&0\end{pmatrix}$	$\begin{pmatrix}2&3&4\\0&0&0\end{pmatrix}$	$\begin{pmatrix}2&3&2\\2&0&0\end{pmatrix}$	$\begin{pmatrix}2&2&2\\2&1&0\end{pmatrix}$	$\begin{pmatrix}2&1&2\\3&1&0\end{pmatrix}$	$\begin{pmatrix}1&1&2\\2&1&2\end{pmatrix}$	$\begin{pmatrix}1&1&1\\3&2&1\end{pmatrix}$	-	-
m=10	$\begin{pmatrix}2&6&2\\0&0&0\end{pmatrix}$	$\begin{pmatrix}2&3&5\\0&0&0\end{pmatrix}$	$\begin{pmatrix}2&3&3\\2&0&0\end{pmatrix}$	$\begin{pmatrix}2&3&2\\2&1&0\end{pmatrix}$	$\begin{pmatrix}2&2&2\\2&2&0\end{pmatrix}$	$\begin{pmatrix}2&1&2\\4&1&0\end{pmatrix}$	$\begin{pmatrix}2&1&1\\4&1&1\end{pmatrix}$	$\begin{pmatrix}1&1&2\\2&2&2\end{pmatrix}$	$\begin{pmatrix}1&1&1\\3&3&1\end{pmatrix}$
m=11	$\begin{pmatrix}2&7&2\\0&0&0\end{pmatrix}$	$\begin{pmatrix}2&3&6\\0&0&0\end{pmatrix}$	$\begin{pmatrix}2&3&4\\2&0&0\end{pmatrix}$	$\begin{pmatrix}2&3&2\\2&2&0\end{pmatrix}$	$\begin{pmatrix}2&2&2\\3&2&0\end{pmatrix}$	$\begin{pmatrix}2&1&2\\4&1&1\end{pmatrix}$	$\begin{pmatrix}2&1&1\\4&2&1\end{pmatrix}$	$\begin{pmatrix}1&1&1\\3&3&2\end{pmatrix}$	-
m=12	$\begin{pmatrix}3&7&2\\0&0&0\end{pmatrix}$	$\begin{pmatrix}2&7&3\\0&0&0\end{pmatrix}$	$\begin{pmatrix}2&4&6\\0&0&0\end{pmatrix}$	$\begin{pmatrix}2&3&6\\1&0&0\end{pmatrix}$	$\begin{pmatrix}2&3&5\\2&0&0\end{pmatrix}$	$\begin{pmatrix}2&3&2\\2&3&0\end{pmatrix}$	$\begin{pmatrix}2&1&2\\4&1&2\end{pmatrix}$	$\begin{pmatrix}2&1&1\\5&2&1\end{pmatrix}$	$\begin{pmatrix}1&1&1\\4&3&2\end{pmatrix}$

configuration: $\begin{pmatrix}n_1&n_2&n_3\\r_1&r_2&r_3\end{pmatrix}$

Table 2. Optimal Configurations for the Example

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

Table 3. Switch Times for the Example