Recovery Modeling in Performability Analysis for Multi-Computer Systems

Meng-Lai Yin, Douglas M. Blough, and Lubomir Bic

Technial Report # 92-99
October, 1992

Department of Information and Computer Science
University of California, Irvine

Abstract

Performability is an attribute of a system which combines reliability and performance. Recovery procedures in gracefully-degradable multi-computer systems can have significant influence on performability. A methodology is proposed in this paper to incorporate recovery modeling into performability analysis. This allows the derivation of probabilities for each performance level under different recovery schemes. A general model for n-primary, m-spare multi-computer systems is developed. Also, a general model for pair and spare multi-computer systems is formed in the paper. Several examples are given to demonstrate the methodology. These examples show that the methodology provides more accurate information about the performability of the system being investigated. The use of this methodology, when combined with existing performance evaluation techniques, can help system designers to obtain an accurate evaluation of the performability of systems with recovery and to compare the effect of varying recovery procedures.

Keywords: Fault tolerance, pair and spare, performability, recovery modeling, standby spares, multi-computer systems.

1 Introduction

Performability analysis provides the capability to do performance assessment in the presence of failures. This ability is especially important for multi-computer systems which allow performance degradation. For performance-degradable multi-computer systems, the failure of a single computer will not cause the system to fail. Instead, the system will allow performance degradation and will continue to provide service after one or more failures. Thus, several performance levels can be formed and one can evaluate the performance of the system in a probabilistic sense by determining the probabilities that the system operates at each of these levels. This is referred to as performability evaluation since it combines the concepts of performance and reliability [Meyer 80A]. Such a multi-computer system will experience several different levels of performance before it fails. In particular, a multi-computer system might have L performance levels. The system performs in level L initially; if one of the computers fails, then the performance of the system drops to level $L-1$. A *recovery process* is needed when performance degradation occurs in order to restore service.

Recovery is the process of maintaining or regaining operational status in the presence of faults. A fault might be detected, or it might go undetected; moreover, a fault could be permanent, transient, or intermittent. For detected permanent faults, the failed computer must be removed. If the removal of a computer causes performance degradation, then other computers in the system must take over the jobs left from the failed one. On the other hand, if the fault is transient or intermittent, then *retry* may be sufficient and, if so, the system provides service without any performance degradation. For undetected faults, the faulty computer continues to provide service. Incorrect results could be used in other computers and hence, widespread damage can result. In the worst case, the system crashes due to the undetected faults.

It is common for a multi-computer system to have error propagation. This phenomenon will cause other computers (not only the failed one) to require recovery. Hence, the recovery process caused by one failed computer might affect the performance of many computers in the system. It is even possible that the recovery process will have degraded the system's performance down to the level 0. Thus, the recovery process can have significant influence on performability. Most research on performance or performability ignores the effects of recovery. However, without considering recovery, it is not possible to provide accurate information about the performability of a system. Section 2 briefly reviews previous work on performability and recovery as well as providing background.

Since the design and implementation of a multi-computer system is a formidable task, it is desirable to provide design assessment before the system is built. Performability analysis is a tool to support this goal. Moreover, in order to do design assessment accurately, recovery modeling should be included in performability analysis. It is the purpose of this paper to provide a methodology to do recovery modeling in performability analysis. Section 3 will give the details of this methodology.

Different recovery policies and different system configurations will influence the results of performability analysis. The design trade-off problem is an interesting area of research. For example, if the system uses a fault-masking technique such as the pair and spare configuration, then the redundancy will increase the cost of the system. However, the

probability that the system will stay in the highest performance level will be higher. The cost-efficiency is an important issue, which determines whether the design is worthwhile. This paper explores the performability of different designs based on the methodology provided. In order to investigate large systems, general models for multi-computer systems, in particular, the n-primary, m-spare multi-computer system, and the pair and spare multi-computer systems are developed. Section 4 examines the n-primary, m-spare systems. Section 5 explores the pair and spare multi-computer systems.

2 Background and Previous Work

Performability is an attribute of a system which combines reliability and performance characteristics by considering a system's performance in the presence of failures [Beaudry 78], [Meyer 80A], [Muppala 91]. As performability has become widely used, several software tools have been developed for assisting performability analysis. Some examples of such tools are MetaSAN, UltraSAN, and SHARPE [Sanders 86], [Couvillion 91], [Johnson 88], [Sahner 86],[Sahner 87].

One way to analyze systems' performability is to model systems' reliability and performance separately, and then to combine the results of the two models [Reibman 90]. The reliability model gives the possible states of a system. The performance model (also called the reward model) assigns a performance value to each of the states. For multi-computer systems, a state represents the status (failed/operational/recovering) of all computers in the system. Transitions between states are the activities (such as failures, repairs, and recoveries) that move a system from one state to another. For multi-computer systems, we use the performance level as an indication of performance, as we described in the previous section. As a simple example without considering recovery, consider the two-computer system shown in Figure 1. This system has three states. State 2 represents the situation where both computers are working, state 1 represents the situation where only one computer is working, and state 0 represents the situation where both computers are down. With each of these states, there is an associated performance level. In general, a system could have more than one state at the same performance level. In the above example, state 1 could be split into two states by distinguishing computer 1 from computer 2. These two states have the same performance level, since they both correspond to the situation where only one computer is operational. To calculate performability measures, one can solve the reliability sub-model and combine this with performance information.

Note that no recovery process is represented in this example. Recovery is the process that keeps the system operational or regains operational status after a failure occurs. For transient or intermittent faults, reexecuting tasks can bring the system to the operational status. Reexecuting can be either instruction retry or rollback to a previous state followed by reexecution. If the fault is detected before system status changes, then retry may succeed. If system status has been changed between error occurrence and error detection, then checkpointing and rollback recovery will be needed. For permanent faults, reexecution will not succeed and replacing and/or removing failed modules is necessary. For replacement, several standby strategies can be applied, *i.e.*, hot standby, cold standby, or warm standby. Removing computers from a multi-computer system might not cause the whole system to fail, as long as a proper recovery procedure is provided.

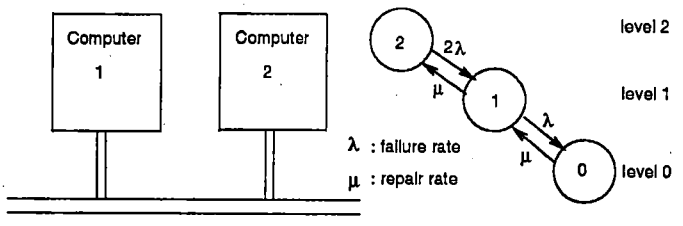

Figure 1: Two-Computer-System Model Without Recovery Consideration

Since the exact nature of faults and their effects are unknown upon occurrence, different systems use different recovery procedures. For example, the Advanced Architecture On-board Processor (AAOP) system applies the retry process first. If retry fails, then the system removes the failed node and applies a reconfiguration process (replace) [Iacoponi 91]. Berg and Koren [Berg 87] proposed a dynamic recovery policy for single computer systems. This dynamic policy provides a method to determine the optimal order of recovery procedures. Only retry and replacement are considered for recovery processes in their paper, because only single computer systems are studied. They also suggest that if a component attempts retry too many times, then it should be replaced, because the overall cost for retry procedures is too expensive.

The computers which are involved in recovery procedures do not provide useful work during the recovery. In particular, if the failed computer is processing the retry procedure, then this failed computer does not provide useful work during the retry process. If the faults are permanent and replacement or removal has to be done, then other computers in the system will be involved in the recovery. This is because other computers have to take over tasks from the failed computer. Furthermore, if there is error propagation, then more computers and more time for recovery is to be expected. As recovery might have significant influence on performability, including it in performability analysis is important.

Previous work on performability for fault-tolerant multi-computer systems can be found in [Smith 87], [Meyer 80B], [Islam 89]. In [Smith 87], C.mmp and a shared-storage multiprocessor system are analyzed. [Meyer 80B] explores the performability of the SIFT (Software Implemented Fault Tolerance) computer. [Islam 89] analyzes the performability of the hypercube architecture. However, no study on recovery for performability analysis of multi-computer systems has been found. It is the purpose of this paper to provide a methodology to model recovery processes in performability analysis. We introduce this methodology in the next section. Examples which demonstrate the application of this methodology will be given later in the paper.

3 Model Construction

The recovery process has influence on both the reliability and performance sub-models within a performability model. This section explores the effects of recovery on these sub-models. Understanding the relationships between recovery and reliability, and between recovery and performance will make the overall performability analysis clear.

As we mentioned above, when parts of a multi-computer system are performing recovery tasks, the system is no longer providing the same service that it was before the fault occurred. Therefore, the system forms other states which are distinct from the operational states which we refer to as *recovery states*. Since the reliability model now has been augmented to incorporate the recovery processes, the effect of recovery can be determined.

The recovery states mentioned above provide different performance levels. For instance, if the failed computer is retrying and no error propagation occurred, then the performance degrades by one level only. However, error propagation might cause recovery within multiple computers, and so the performance degradation might be several levels deep. Or, the fault may be permanent, requiring replacement or removal, while other computers must take over the remaining tasks. This will degrade the performance level even further. The performance degradation for these recovery states is so diverse that different performance tools like simulation or queuing models have to be used to determine its extent. This paper will not address the problem of performance analysis within recovery states. Instead, we assume some performance degradation behaviors for the recovery states.

3.1 Reliability-Related Recovery Modeling

The nature of the recovery states in the reliability model depends on the recovery policy which the system employs. As mentioned before, different systems apply different procedures to recover from faults. In this study, we examine a static recovery procedure, and explore its influence on different systems. This recovery procedure ensures that the system regains the operational state with the highest possible performance level. This is accomplished by first retrying the failed computer, then replacing it if the retry has failed and a spare is available, and finally removing it if all other recovery attempts have failed. The recovery procedure is stated below:

If a fault is detected
  retry
  If retry succeeds
    resume original state
  Else (retry does not succeed)
    If spares available
      replace
      resume original performance level, but with one less spare
    Else (no spare available)

remove
degrade performance

Figure 2 shows the state diagram for an $n+1$ computer system which follows this recovery procedure from a configuration containing n primary computers with one spare to the configuration which has $n-1$ computers without any spare. Five types of states exist in this model. The first one is the operational state, *e.g.*, $(n,1)$. The second type is the retry state, *e.g.*, Retry (n,1). Others are the replace state, *e.g.*, Replace (n,1), the remove state, *e.g.*, Remove (n,1), and the undetected state, *e.g.*, UD(n,1).

The undetected state represents the situation where the fault is not detected. Since the fault is not detected, the faulty computer keeps on performing, and eventually the effects of the fault become significant enough to cause the system to crash. Consequently, an undetected fault always does severe damage to the system, and needs to be considered in the model. Once the undetected fault causes a crash in the system, then the system is brought to the retry state. In detail, the $UD(n,1)$ state will be brought to the $Retry(n,1)$, and the $UD(n,0)$ will be brought to the $Retry(n,0)$ state, as shown in Figure 2. Note that there is a chance that additional failures will occur while the system is in the undetected state. We make the simplifying assumption that only one undetected fault can exist in the system at one time implying that subsequent faults are always detected. In particular, the probability of a second failure during this period is quite small so this assumption should have little impact on the results. The symbol C represents the coverage of the fault detection mechanism in the system. Hence, faults are detected with probability C and so the transition rate from a state with n operaional computers to the retry state is $Cn\lambda$ and the transition rate to the fault undetected state is $(1-C)n\lambda$.

Two numbers are used for each state (except the failed state) to identify the status of the system. The first number indicates the number of operational computers in the system, the second number indicates the number of available spares in the system. Therefore, (n,m) means there are n computers working on different tasks (in parallel) and hence the performance level is n. The m computers in the system are used as spares (cold standby is assumed here). The failure rates are represented as λ, and the repair rates are denoted as μ. The single-person repair model is used here which indicates that only one repair can be taken at any time.

If one of the computers fails, our recovery procedure dictates that we first retry the failed computer. Hence, the system goes to the retry state, *i.e.*, Retry (n,m). This state represents the situation that, although there are still n operational computers, one of the operational computers is doing the retry task. If retry succeeds, then the system returns to the original state (state (n,m)). If retry fails, then the system goes to the replace state , *i.e*, Replace (n,m). The replace state only exists if there is at least one spare available. Otherwise, the system has to remove the failed computer, and it goes to the Remove (n,m) state. Replace (n,m) represents the situation that one of the n computers is determined to be permanently faulty, and one of the m computers is going to replace the failed one. Remove (n,0) means that no spare exists, therefore, the system has to remove the failed computer. Note that the second number for remove states is always 0, since it only occurs if no spare is available.

It is assumed that no other failure occurs during the recovery process (except for the case of undetected faults). This is justifiable since the mean recovery times are many orders of magnitude shorter than the mean time to failure. The system described in Figure 2 does not have fault-masking capability. If a fault is detected, then the system goes into the retry state. All the standby spares are cold-standby in this model, that is why both transitions from state $(n, 1)$ to state Retry (n,1) and from state $(n, 0)$ to state Retry (n,0) are $Cn\lambda$. The single-person-repair model is used here, which means that only one repair can happen at a time. Therefore, all the repair transitions are represented by μ. The retry process may succeed if the fault is not a permanent fault; the probability that the retry process succeeds is denoted as p. R_{retry} is the retry rate, in other words, the rate for the retry process to be completed. For systems where the faults are detected instantaneously, the retry rate can be large (the time for retry is very small). Once the retry succeeds, the system resumes its original state.

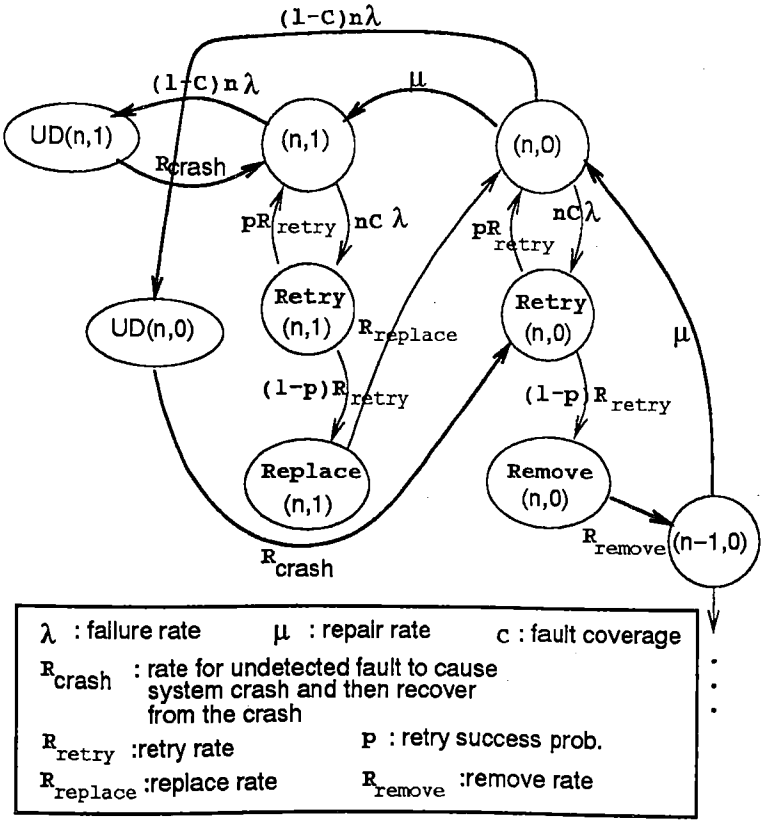

Figure 2: Reliability-Related Recovery Modeling

If the retry does not succeed (this has probability $1 - p$), then the system has to check whether there is a spare available or not. If there is a spare, then the system can proceed with replacement. $R_{replace}$ represents the rate for the replacement. The replacement will bring the system into a state where n computers are still working, however, the number of spares is one less than the original state. In general, $R_{replace}$ is smaller than R_{retry}. This is because

replacement with a cold standby involves complete initialization which usually requires a minimum of several minutes. If the spares are hot-standby spares, then $R_{replace}$ would be larger than it is for cold-standby spares.

If the system does not have any spares, then removal will be attempted. R_{remove} is the rate for the removal process. Note that other computers within this multi-computer system have to take over the tasks left from the failed computer. This process may involve only one computer if the recovery policy is such that one computer takes over all the tasks from the failed computer. Or, more computers can be involved if the policy says all the operational computers should participate in the recovery process. Thus, different policies will give different values for R_{remove}.

R_{crash} is the rate at which computers crash while in an undetected state. This is the transition rate from an undetected state to an retry state, for example, from UD (n,1) to Retry (n,0) in Figure 2.

For the two computer system example in Figure 1, the model is augmented by this recovery modeling as shown in Figure 3. Since the system does not have any spares, there is no replace state in the model. If retry does not succeed, the system proceeds directly to removal. When the system degrades to performance level 1 where only one computer is operational, then retry failure causes the system to fail. The UD (2,0) state may lead to the UD (1,0) state, since a second failure might occur during the period that the undetected fault is in the system. This has the transition rate λ. In general, the transition rate from the UD (n,0) state to the $UD(n-1,0)$ state is $(n-1)\lambda$.

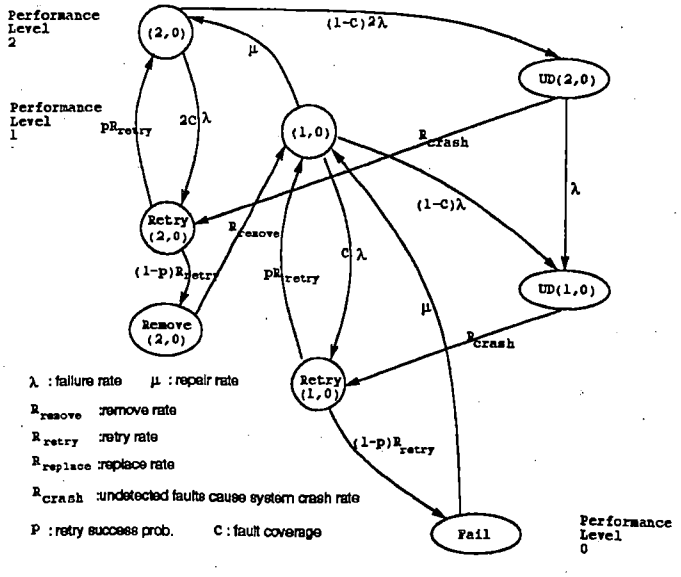

Figure 3: Performability Model for Two-Computer System

3.2 Performance Levels for Recovery States

The performance levels for the recovery states are so varied that performance tools such as simulation or queueing models are required in order to provide more accurate information. However, we discuss briefly the influence of recovery on performance level here.

In Figure 2, if the fault does not result in any error propagation, then the retry state Retry (k,i) will have performance level $k-1$. This is due to the fact that only the failed computer is retrying (not doing useful work) while all the other computers are still providing useful work. Thus, the performance only degrades one level during the retry process. However, if error propagation does happen and x additional computers have been affected, then the performance level will be degraded $x+1$ levels, because $x+1$ computers are performing the recovery process.

The performance level for the remove state Remove (n,i) depends on the recovery policy that the system takes. If the system allows only one operational computer to take over the failed computer's tasks, then the remove state will have performance level $n-2$, assuming no error propagation happens. This is because one computer is failed, and another one is initializing and reexecuting the failed computer's tasks. If the system's policy says all the other computers should help in recovering from the fault, then the performance will be degraded to zero. However, the time to recover from the fault will be shorter (the recovery rate is larger). The undetected state effectively has performance level 0 since the system will be producing incorrect output while in this state.

The performance level for the replace state Replace (n,i) depends on the recovery policy and the type of spares. If the spares are cold standby, then another operational computer need to involve in the recovery process. In particular, the cold standby spare needs to be initialized and reloaded the status of the failed computer. This task needs some operational computer's help. If the spares are hot standby, then the spare can be replaced immediately without other computers assistance. In our example, cold standby spares are used, and so the performance for Replace state is degraded by two levels.

For the example in Figure 3, the Retry (2,0) state could have performance level 1, if no error propagation is assumed. The Remove (2,0) state has performance level 0, this is the lowest performance level that the system can provide. The Retry (1,0) state has performance level 0. All the undetected states have performance level 0.

Thus, our methodology extends the reliability modeling to incorporate the recovery process. In the next two sections, we give examples to demonstrate the use of this methodology. Since performance modeling for multi-computer systems is by itself a difficult research topic, we do not address this problem in the paper. Instead, we assume the performance for the performance levels are given, and study only the probability of being in each performance level. To evaluate systems' overall performability, performance tools are needed to provide information about performance.

4 Multi-Computer Systems without Fault Masking

This section and the next section give examples which follow the methodology that we have introduced. This section explores three issues related to degradable multi-computer systems without fault masking. The first issue involves

the effect of spares on performability. The second issue addressed here is the influence of increased failure rates due to the recovery processes. The third issue involves a comparison between two different recovery policies, one recovery policy requires all the remaining computers to help with the tasks left from the failed computer, the other requires only one computer to take over these tasks.

4.1 The Effect of Spares on Performability

Assume we are given a system with $n+m$ computers, configured with n primary computers and m spares. The n computers provide service until one of them fails, then a spare replaces the failed computer. Until all of the spares have been used up, the system can maintain the highest performance level (performance level n). The question is how many of the computers should be used as primary computers, and how many of them should be used as spares ? The answer to this question partly depends on the power requirements of the system. The main reason for utilizing cold standby spares is that they provide fault tolerance with no increase in power consumption. While performability does not measure power consumption, the designer of such a system will have to balance these two quantities when choosing a system configuration. If the configuration does not include any spares, then the probability of being in the highest performance level will be lower than the one that has spares. However, the highest performance level that the system can provide will be lower for the configuration with spares than for the one without spares. We now introduce a general model for n-primary and m-spare systems.

Figure 4 gives the general Markov model for this type of configuration (n primary, m spare, non-fault-masking multi-computer systems). Five types of states exist in this model, as explained in Section 3.1, *i.e.*, the operational state, *e.g.*, (n,m), the retry state, *e.g.*, Retry (n,m), the replace state, *e.g.*, Replace (n,m), the remove state, *e.g.*, Remove (n,m), and the failed state. For simplicity, the undetected states are not shown in this figure. However, this model can be expanded as described in Section 3 to include the undetected states.

Thus, n primary computers are working in parallel and m computers in the system are used as spares (cold standby is assumed here). The failure rate of a single computer is given by λ, and the repair rate is denoted by μ. The single-person repair model is used here which implies that only one repair can be taken at any time. The retry process time is estimated by a rate R_{retry}, the replace rate is represented as $R_{replace}$, and the remove rate is denoted by R_{remove}. If one of the computers fails, since we don't know whether it is a permanent fault or not, we would like to retry first (according to the recovery procedure in Section 3.1). Hence, the system goes to the retry state. If retry succeeds (with probability p), then the system returns to the original state. Otherwise, the system goes to the replace state or to the remove state, depending on whether there is a spare or not.

In more detail, the operational states can be classified into three types: the initial operational state, *i.e.*, state (n,m), the states with spares, *i.e.*, state (n,x) where $0 < x < m$, and the states without spares, *i.e.*, state $(k,0)$ where $1 \leq k \leq n$. Figure 5 represents the transition-related diagram for each of these state types. The diagrams for the retry state, replace state, and remove state are also shown. From this diagram, we can form the balance equation for each state in the model. Note that the generation of these equations can be automated.

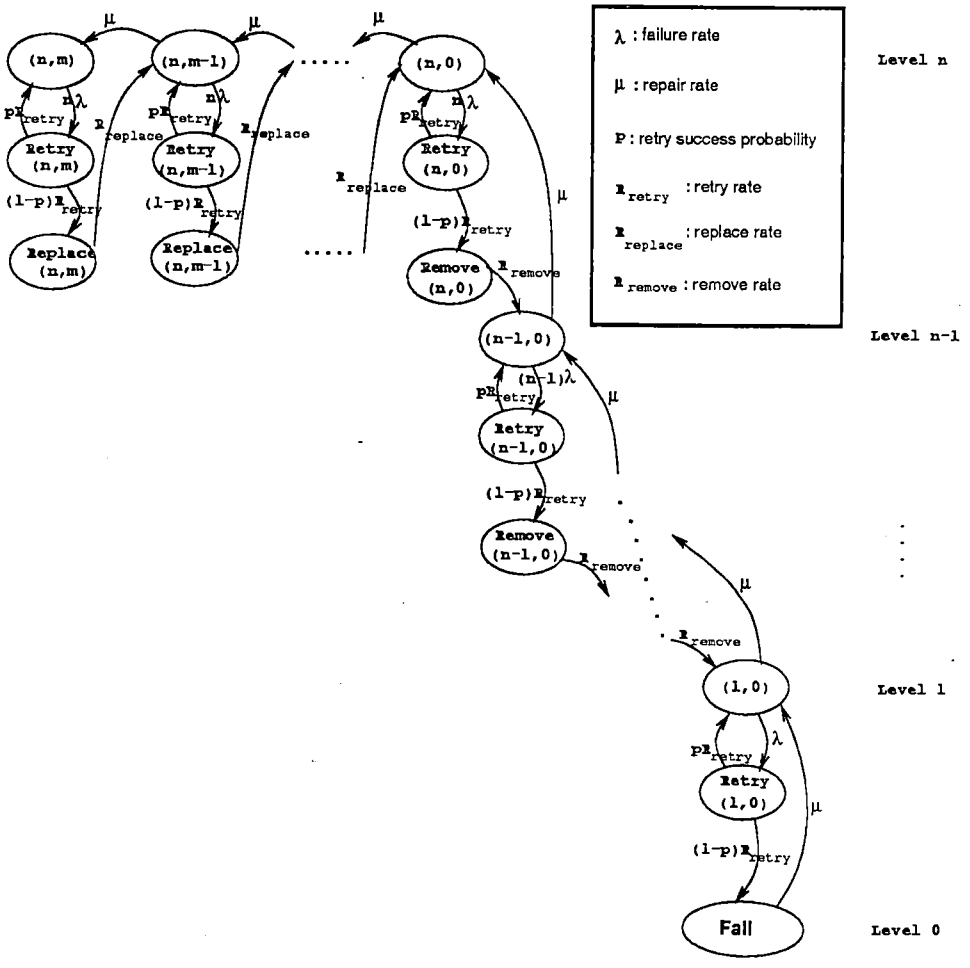

Figure 4: General Model for N-Primary, M-Spare Multi-Computer Systems (C=1)

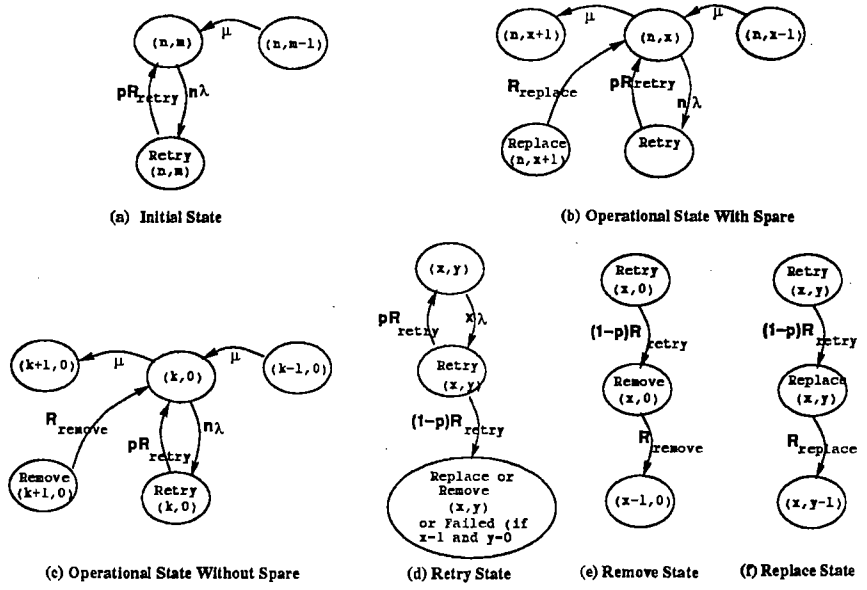

Figure 5: Transition Diagrams for States in N-Primary, M-Spare General Model

Although the model can be applied to any size system, we examine an 8-computer system in this section. The cases shown below are with 1, 2, 3, and 4 spares, and the total number of computers in the system fixed. For the system without spares, eight performance levels can be formed. With one spare, the system can only form 7 performance levels, and so on. The more spares are used, the fewer performance levels can be formed. The distribution of the probability on different performance levels for different configurations is shown in Figure 6. The parameters used for this study are as follows: the fault detection coverage is assumed to be 1; about one failure is detected every year; the repair takes about 2 days; the retry process takes an average of 10 milli-seconds and 60% of the retries succeed; the replace takes an average of 5 minutes, and the remove process takes approaximately 1 minute. The average time for undetected state to cause system crash is approximately 1 hour. The number shown in Figure 6 are not particularly significant. The data simply demonstrates the feasibility of incorporating recovery into the performability modeling process.

Furthermore, a comparison which compares the results between the model with recovery consideration and the one without recovery consideration is shown in Figure 7. The configuration under investigation has 8 primary computers without spares. As shown in the figure, the probability distribution with recovery is significantly different from the one which does not account for recovery. This shows that neglecting the effect of recovery (as is done in all previous work on multi-computer system performability) can produce misleading results. This can be quite costly and even dangerous in system which must meet specified performability goals.

Performance Level	Probability				
8	0.9872162954				
7	0.0126363704	0.9998351164			
6	0.0001459150	0.0001415267	0.9999764446		
5	0.0000014078	0.0000233455	0.0000013605	0.9999778014	
4	1.134 x 10^-8	1.134 x 10^-8	0.0000221948	1.272 x 10^-8	0.9999778122
3	7.334 x 10^-11	7.334 x 10^-11	7.334 x 10^-11	2.481 x 10^-7	1.918 x 10^-9
2	3.581 x 10^-13	3.581 x 10^-13	3.581 x 10^-13	3.581 x 10^-13	0.0000221858
1	1.184 x 10^-15	1.184 x 10^-15	1.184 x 10^-15	1.184 x 10^-15	1.184 x 10^-15
0	2.081 x 10^-18	2.081 x 10^-18	2.081 x 10^-18	2.081 x 10^-18	2.081 x 10^-18
	n=8, m=0	n=7, m=1	n=6, m=2	n=5, m=3	n=4, m=4

Figure 6: Probability Distributions for Different Configurations (C=1)

Performance Level	Probability Distribution	
8	0.9872162954	0.9681311169
7	0.0126363704	0.0309801957
6	0.0001459150	0.0008674455
5	0.0000014078	0.0000208187
4	1.134 x 10^-8	4.164 x 10^-7
3	7.334 x 10^-11	6.662 x 10^-9
2	3.581 x 10^-13	7.994 x 10^-11
1	1.184 x 10^-15	2.558 x 10^-15
0	2.081 x 10^-18	6.396 x 10^-18
	with recovery modeling	without recovery modeling

total number of computers : 8

Figure 7: The Effect of Recovery Modeling (n=8, m=0, C=1)

4.2 The Effects of Recovery Failure Rates

The failure rate of a single computer in systems that provide recovery capability is usually higher than that for systems without recovery. This is due to the fact that more hardware and software are needed to support the recovery process. This factor will certainly affect the performability results. The influence of recovery capability on performability due to different failure rates is the topic of this section. In other words, how should the designer decide whether to support recovery or not ?

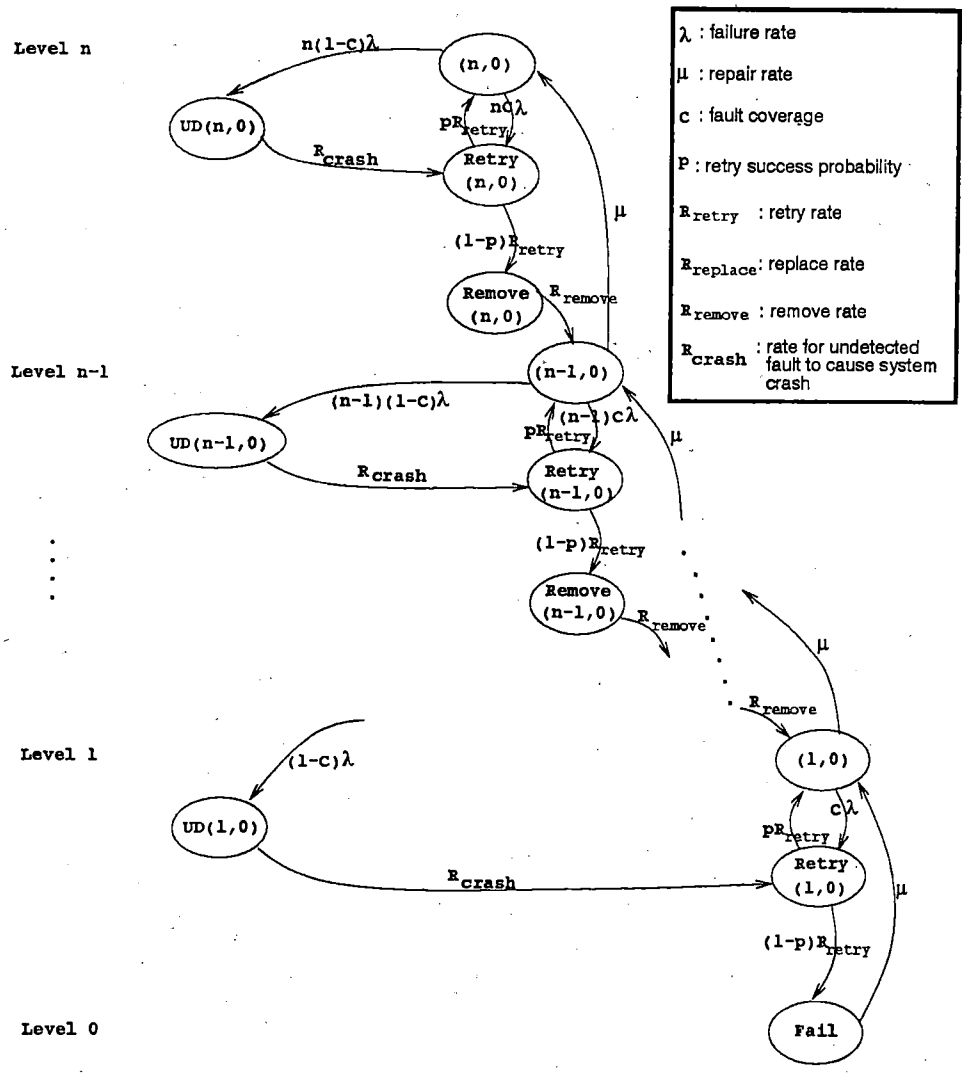

Figure 8: General Model for N-primary, No-spare Multi-computer Systems (with Undetected States)

We compare the system that provides recovery capability to the system which does not support recovery. No spare is

used in this example. The general model for n-primary, no-spare systems without fault masking technique is shown in Figure 8. The model for an n-computer system without recovery includes only two states, *i.e.*, the operational state and the failed state, assuming that the fault detection coverage is one. Because the failure rate for systems with recovery is larger than the failure rate for systems without recovery, we set up a ratio which represents the difference of these two failure rates. Thus,

$$ratio = \frac{\text{failure rate with recovery}}{\text{failure rate without recovery}}$$

We investigate these two types of systems with different values of *ratio*, and the results are shown in Figure 9. This shows the probability for the systems being in the highest performance level (performance level 8 in this case). We refer to the value of the ratio at the intersection point of the two curves as the *crosspoint ratio*. In this example, the crosspoint ratio is approximately 2.471. If the increase of failure rate due to the recovery process is less than the crosspoint ratio, then providing recovery capability lead to a higher probability of performing at the highest level. However, if the increase of failure rate is larger than the crosspoint ratio, then not providing recovery capability is better.

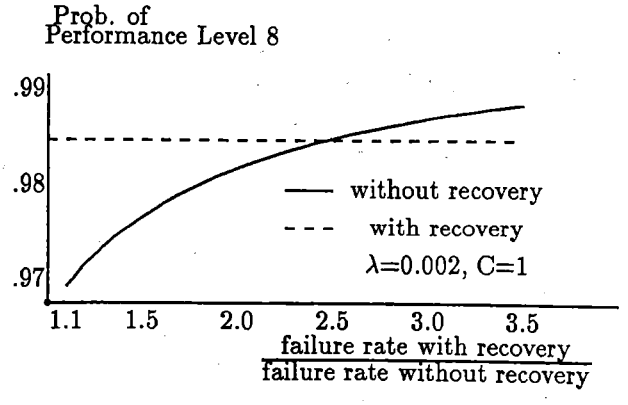

Figure 9: Influence of Increased Failure Rate Due To Recovery

The crosspoint ratio varies when the parameters of the system change. For example, the larger the system failure rate is, the smaller the crosspoint ratio becomes. Figure 10 shows the crosspoint ratios for systems containing from 2 to 16 computers. The *crosspoint ratio* for these two configurations decreases when the number of computers in the system increases. This implies that for large parallel computer systems, providing recovery capability must produce only a small increase in failure rate in order for the recovery to be worthwhile.

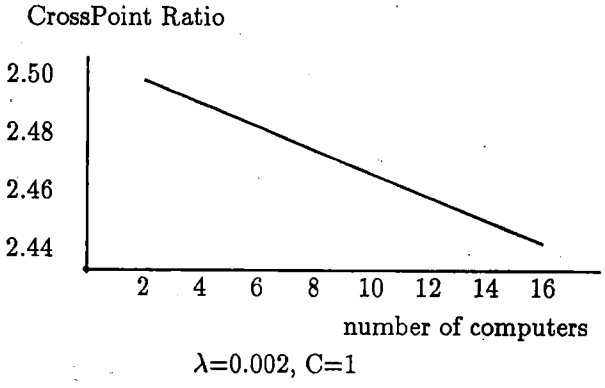

Figure 10: Crosspoint Ratios for N-Computer Systems

4.3 Recovery Strategies

Multi-computer systems can have different policies for recovery process participation. One policy is to let only one operational computer take care of the tasks left from the failed computer. A different approach is to let all the working computers be involved in the recovery process for the failed computer. In this case, all operational computers combine to take over the tasks that are left from the failed computer. The duration of recovery for one-participation systems is longer than the recovery time for all-participation systems. Therefore, the recovery rate for one-participation systems will be smaller than the one for all-participation systems. Note that the previous examples assume the one-participation policy.

Assuming the recovery rate is in proportion to the number of computers participating, we compare these two policies on the system without any spares. The model for all-participation systems is similar to the one for one-participation systems, except that the remove states now have performance level 0. This is due to the fact that all the computers are doing recovery jobs, but not normal tasks. Also, the remove rates are multiplied by the number of computers remaining in the system.

The comparison for 8-computer and 16-computer systems with these two recovery policies is illustrated in Figure 11. Note that the probability of performance level 0 for systems with all-participation is much higher than that for systems with one-participation. This is due to the fact that all the remove states are considered to be in level 0. In addition, the highest two performance levels for all-participation systems have probabilities higher than that for one-participation systems. Hence, while the systems with all-participation spend more time at performance level 0, they also spend more time at the highest performance levels. For systems that can tolerate short periods of low performance, the all-participation scheme would therefore appear to be preferable.

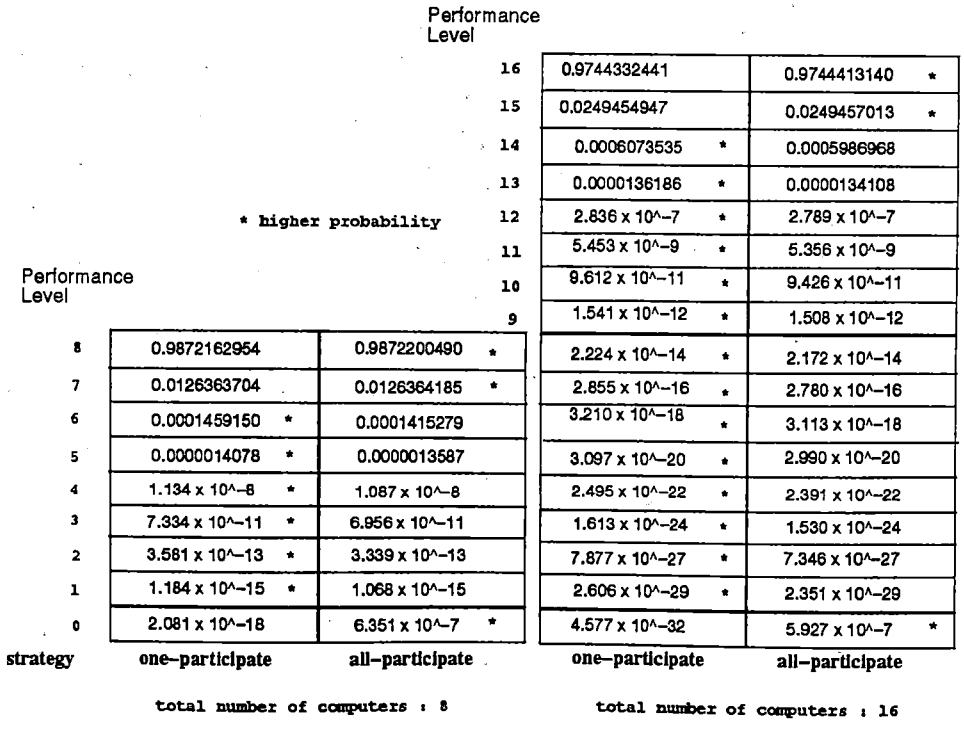

Figure 11: Probability Comparison for Different Participation for Recovery Processes (C=1)

5 Multi-Computer Systems with Pair and Spare Architecture

The pair and spare architecture discussed here utilizes two replaceable units to form a single computer element for a multi-computer system. Each replaceable unit contains a pair of processors. The two processors within the replaceable unit operate in lock step and a cycle-by-cycle comparison of their output is performed. The second pair performs as a hot standby spare for the primary pair. This pair and spare architecture is used in the Stratus multicomputer system [Webber 91]. In this work however, we are concerned only with the processor and local memory portion of the system. Since each computer element in the Stratus system is a complete computer system containing I/O devices such as disk and tape drives in addition to processors and their local memories, our results cannot be directly applied to the Stratus system.

5.1 Two-Computer System with Pair and Spare Architecture

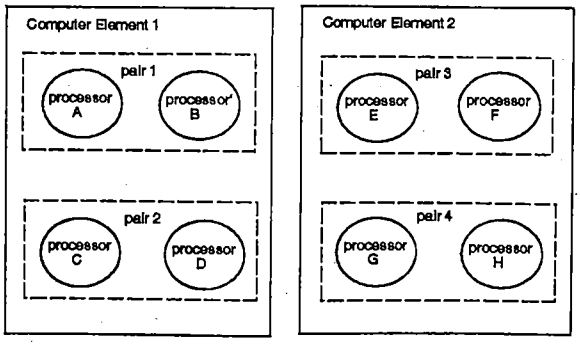

Figure 12: Two-Computer System with Pair and Spare Architecture

A two-computer-system with the pair and spare architecture is shown in Figure 12. Eight processors are included in this system; four processors constitute one computer element. Two processors are grouped together (dashed box) to indicate that they are on one replaceable unit (a pair). Each pair contains a comparator circuit which compares the outputs from the two processors on a cycle-by-cycle basis. Computer element 1 has two pairs, i.e., pair 1 and pair 2. Pair 1 consists of the two processors A and B, while pair 2 includes processor C and D. Computer element 2 is configured in exactly the same manner.

The system begins at performance level 2 (indicating that two computer elements are functioning) with all processors operational. If processor A or processor B fails, this is detected by the comparator in pair 1. Under this situation, pair 2 becomes the primary one for computer element 1, in the mean time, pair 1 will report the failure and wait for its replacement. Note that the system remains in performance level 2 without requiring recovery at this point. A second failure could occur before pair 1 is replaced. This second failure might occur in pair 2, in pair 3, or in pair 4.

If the second failure happens within pair 3 or pair 4, then the system can still maintain performance level 2. If the second failure is within pair 2, then the system has to proceed with the recovery process.

No recovery process is taken if no performance degradation occurs. This is because providing continuous service is the main purpose for the design of pair and spare architecture. If recovery is undertaken, then the computer element which has only one failed processor will have to stop providing normal service, and start working on the recovery process. Therefore, the computer element will not be able to provide continuous service even if only one pair fails. This is against the design philosophy of fault-masking techniques. Consequently, the recovery process only happens when performance degradation occurs.

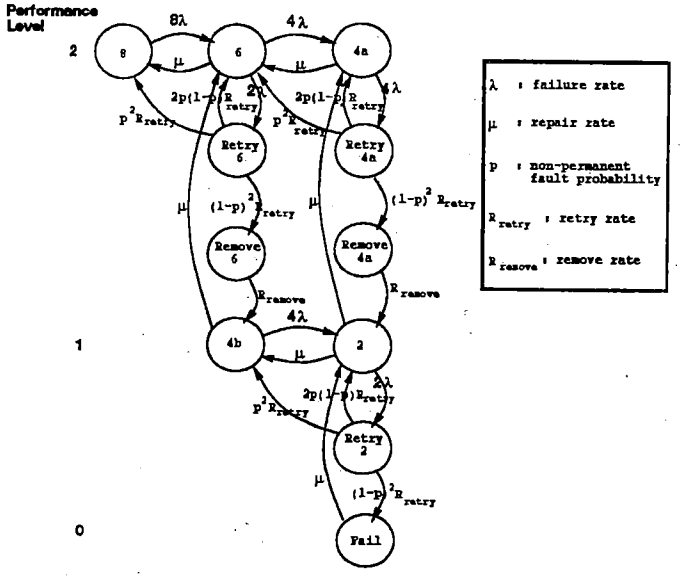

Figure 13: Markov Model for Pair and Spare Architecture (Two-Processing-Element System)

The Markov model for this system is shown in Figure 13. It is assumed that processor failures occur independently, at rate λ, and that repairs complete at rate μ. As in the non-fault-masking model, we utilize the single-repair-person model which assumes that failures are repaired one at a time. Because fault detection in the pair and spare architecture is done using duplication and comparison, the only manner in which an undetected fault can occur is if the idential fault occurs in two processors of the same pair at exactly the same time. Since this situation is extremely rare in practice, we do not account for undetected faults in our model.

Four types of states are included in this model, *i.e.*, the operational states, the retry states, the remove states, and the failed state. Unlike the non-fault-masking model, the replace state does not exist, because the replacement is done instantaneously when one pair of the computer element fails. The number indicated within each operational state represents the number of operational physical processors for that state. For example, state 8 means that eight

processors are working. Because a pair of processors will be out of service if one of the processors fails, there is no state with an odd number.

Note that there are two operational states with four processors. State 4a represents the case that both processing elements are operational while state 4b represents the situation that only one processing element is working. In other words, within state 4a, both processing elements have one pair working, whereas in state 4b, both working pairs belong to the same processing element. Therefore, state 4a has performance level 2; while state 4b has performance level 1.

As we mentioned before, the recovery process is taken whenever performance degradation occurs. For example, state 6 will go into state Retry 6 if the second failure happens within the same computer element that already had one faulty pair. Note that the retry process will retry both pairs within the computer element, because either pair may have a non-permanent fault. Therefore, there is a transition from the Retry 6 state to state 8. Assuming the probability that a pair has a non-permanent fault is p, and the faults are independent, then the probability that both pairs have non-permanent faults is p^2. The probability that only one pair has a non-permanent fault is $2p(1-p)$. Thus, we multiply these factors by the Retry transition rates in the model. The probability that both pairs have permanent faults is $(1-p)^2$. If both pairs have permanent faults, then the remove process has to be taken, with the rate R_{remove}.

The state diagram is organized so that the states which belong to the same performance level are drawn on the same horizontal level. Hence, state 8, state 6 and state 4a belong to performance level 2, and so on.

5.2 General Pair and Spare Model

The analysis process described above can be applied to any n-computer system having the pair and spare architecture. As with the multi-computer systems without fault masking capability, we can construct a general model for pair and spare multi-computer systems. This general model is shown in Figure 14. Each operational state is represented as state Kx, where K is an even number (the number of operational processors), and x is a symbol which distinguishes the states that have the same number of processors, but have different performance levels. The corresponding retry state and remove state are denoted by as Retry Kx and Remove Kx.

Considering a pair and spare multiprocessor system with N processors, we can form $L = \frac{N}{4}$ different performance levels for operational states. At any performance level i in the system, $0 \leq i \leq L$, there are $i+1$ states. Therefore, the total number of operational states in the system is $\sum_{i=0}^{L}(i+1)$. For those states that have K operational processors, there are X different performance levels over which these states will be distributed, where

$$X = \begin{cases} \lfloor \frac{K}{4} \rfloor + 1 & \text{if } K \leq \frac{N}{2} \\ \lfloor \frac{N-K}{4} \rfloor + 1 & \text{if } K \geq \frac{N}{2} \end{cases}$$

Moreover, the lowest performance level for states that have K operational processors is $\lceil \frac{K}{4} \rceil$.

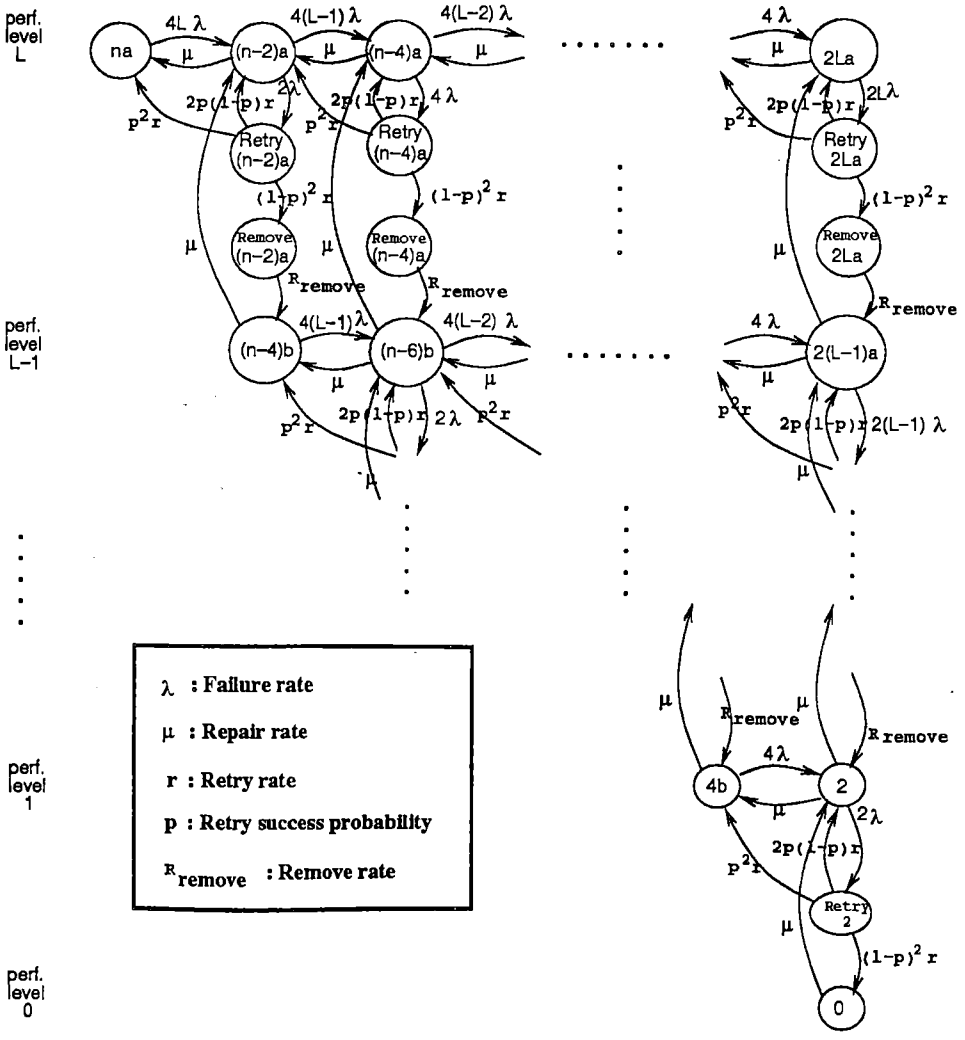

Figure 14: General Model for Pair and Spare Architecture

For any performance level i in the model, there are i Retry states and i Remove states corresponding to the operational states on this level. Note that there is no Remove 1 state in the model. Thus, the total number of Retry states in the system is $\sum_{i=1}^{L} i \ (= \frac{L(L+1)}{2})$ and the total number of Remove states is $\sum_{i=2}^{L} i \ (= \frac{L(L+1)}{2} - 1)$.

Note that for each state of the same type in the model, the balance equation has the same form. Figure 15 shows the transition diagram for an operational state in the model (not the boundary state). The transitions are only related to the number of working processors of that operational state, and the performance level corresponding to that state. Realizing that all these states have regularities, the balance equation for each state can be formed automatically.

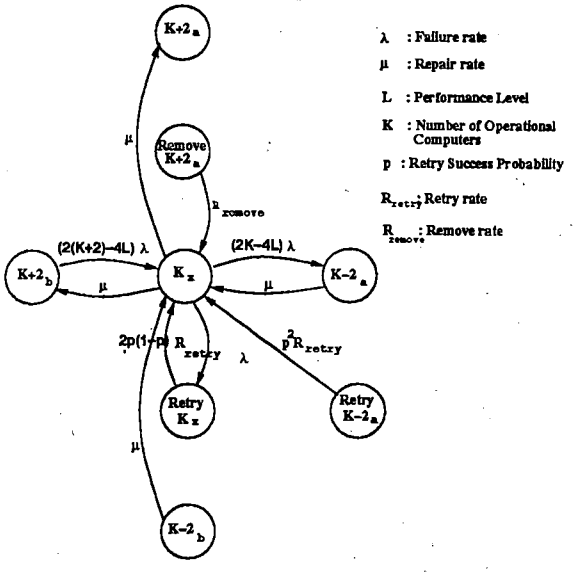

Figure 15: Transition Diagram for an Operational State in the General Pair and Spare Model

5.3 The Effect of the Pair and Spare Technique

This section examines the effects of fault masking capability, in particular, the pair and spare technique. We compare the system with pair and spare architecture to the system without any fault-masking technique (the one shown in Figure 8). Eight processors are used for the comparison. Note that eight processors can form a two-computer system for pair and spare architecture, while a non-fault-masking 8-computer system can be developed out of these 8 processors. Therefore, we are comparing two systems with different numbers of performance levels. The parameters are the same as we used before, however, the fault coverage for the system without any fault masking technique is varied from 0.9 to 0.95. The fault detection coverage for the pair and spare architecture is assumed to be one, due to the assumption that the comparator within each pair can detect almost all the faults.

The pair and spare two-computer system has the probability 0.999959(approximately) of being in the highest performance level (performance level 2), shown as the dashed line in Figure 16. For the non-fault-masking system, the probability of being in the highest performance level (performance level 8) increases as the fault detection coverage increase. Since the highest performance levels that are compared here are different, we summarize the probability of being in the states above performance level 2 (level 2 to level 8), and show it in the figure. When the fault coverage is higher than 0.94, the probability of being in the states higher than level 2 for the non-fault-masking 8-computer system is higher than that for the pair-and-spare, 2-computer (8 processors) system.

6 Conclusion

We have given a methodology for modeling recovery processes in performability analysis for mutli-computer systems. The application of this methodology is demonstrated by several examples. These examples show that neglecting the effects of recovery on performability can produce results that are at best misleading and at worst dangerously incorrect. The examples have also shown how these models can be used to evaluate and make design decisions involving competing multi-computer configurations. Specific design decisions that we have shown how to make are: 1) whether to support recovery at all, 2) which recovery policy to choose, and 3) which system configuration to choose.

Future research direction are twofold. Since out methodology can be applied to massively parallel computer systems, more research about fault-tolerant architectures is motivated by this study. In particular, Stratus and Tandem systems are now being investigated. The second promising area of research concerns dynamic recovery strategies. One such dynamic strategy has been shown to be efficient for single computer systems in [Berg 87]. However, for multi-computer systems, dynamic strategies have yet to be developed and evaluated.

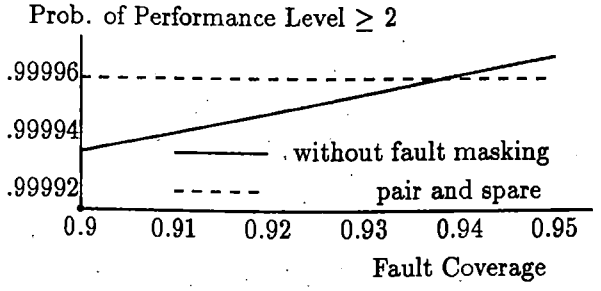

Figure 16: The Effect of the Pair and Spare Technique (number of processors = 8)

References

[Beaudry 78] M. Beaudry. "Performance-Related Reliability Measures for Computing Systems." *IEEE Trans. on Computers,* Vol.C-27, No.6, June 1978, pp.540-547.

[Berg 87] Menachem Berg, Israel Koren. "On Switching Policies for Modular Redundancy Fault-Tolerant Computing Systems." *IEEE Transactions on Computers,* Vol. C-36, No.9, Sep. 1987, pp.1052-1062.

[Couvillion 91] J.A. Couvillion, et al. "Performability Modeling with UltraSAN." *IEEE SOFTWARE,* Sep. 1991, pp.69-80.

[Iacoponi 91] Michael J. Iacoponi, S. Fenton McDonald. "Distributed Reconfiguration and Recovery in the Advanced Architecture On-board Processor." *FTCS 1991,* pp.436-443.

[Islam 89] S.M.R. Islam, H.H. Ammar. "Performability of the Hypercube." *IEEE Trans. on Reliability,* Vol.38, No.5, Dec. 1989, pp.518-526.

[Johnson 88] A.M. Johnson, M. Malek. "Survey of Software Tools for Evaluating Reliability, Availability, and Serviceability." *ACM Computing Surveys,* Vol. 20, No.4, Dec. 1988, pp.227-269.

[Meyer 80A] J.F. Meyer. "On Evaluating the Performability of Degradable Computing Systems." *IEEE Trans. on Computers,* Vol. C-29, No.8, August 1980, pp.720-731.

[Meyer 80B] J.F. Meyer, D.G. Furchtgott, L.T. Wu. "Performability Evaluation of the SIFT Computer." *IEEE Trans. on Computers,* Vol.C-29, No.6, June 1980, pp.501-509.

[Movaghar 84] A. Movaghar and J.F. Meyer. "Performability Modeling with Stochastic Activity Networks." *Proceedings of 1984 Real-Time Systems Symposium,* 1984, pp.215-224.

[Muppala 91] J.K. Muppala, S.P. Woolet, and K.S. Trivedi. "Real-Time-Systems Performance in the Presence of Failures." *IEEE Computer,* May 1991, pp.37-47.

[Reibman 90] A.L. Reibman. "Modeling the Effect of Reliability of Performance." *IEEE Trans. on Reliability,* Vol. 39, No.3, August 1990, pp.314-320.

[Sahner 86] R.A. Sahner, K.S. Trivedi. "A Hierarchical, Combinatorial-Markov Method of Solving Complex Reliability Models." *Proceedings of the 1986 Fall Joint Computer Conference, AFIPS,* pp.817-825.

[Sahner 87] R.A. Sahner and K.S. Trivedi. "Reliability Modeling Using SHARPE." *IEEE Trans. on Reliability,* R-36, 2, June 1987, pp.186-193.

[Sanders 86] W.H. Sanders and J.F. Meyer. "METASAN: A performability Evaluation Tool Based on Stochastic Activity Networks." *Proceedings of the 1986 Fall Joint Computer Conference, AFIPS,* pp.807-816.

[Serlin 85] O. Serlin. "Fault-Tolerant Systems in Commercial Applications." *IEEE Computer,* August 1984, pp.19-30.

[Siewiorek 90] D. Siewiorek. "Fault Tolerance in Commercial Computers." *IEEE Computer*, July 1990, pp.26-37.

[Smith 87] R.M. Smith and K.S. Trivedi. "A performability Analysis of Two Multi-Processor Systems." *FTCS Proceedings, 1987*, pp.224-229.

[Webber 91] S. Webber and J. Beirne. "The Stratus Architecture." *FTCS Proceedings, 1991*, pp.79-85.