PAPER

Evaluation of Two Load-Balancing Primary-Backup Process Allocation Schemes

Heejo LEE[†], **Jong KIM**[†], *and* **Sung Je HONG**[†], ***Nonmembers***

SUMMARY In this paper, we show two process allocation schemes to tolerate *multiple faults* when the primary-backup replication method is used. The first scheme, called *multiple backup scheme*, is running multiple backup processes for each process to tolerate multiple faults. The second scheme, called *regenerative backup scheme*, is running only one backup process for each process, but re-generates backup processes for processes that do not have a backup process after a fault occurrence to keep the primary-backup process pair available. In both schemes, we propose heuristic process allocation methods for balancing loads in spite of the occurrence of faults. Then we evaluate and compare the performance of the proposed heuristic process allocation methods using simulation. Next, we analyze the reliability of two schemes based on their fault-tolerance capability. For the analysis of fault-tolerance capability, we find the degree of fault tolerance for each scheme. Then we find the reliability of each scheme using Markov chains. The comparison results of two schemes indicate that the regenerative single backup process allocation scheme is more suitable than the multiple backup allocation scheme.

key words: *primary-backup replication, multiple faults, fault-tolerant multi-computer, load balancing process allocation, reliability analysis*

1. Introduction

Process allocation is important for multi-computer systems since it affects the performance of the system. In multi-computer systems, a process is no longer a single independent job. Either it has message dependencies with other processes or it blocks the execution of other processes by allocating useful system resources such as cpu, memory, and disks. A careful allocation of processes can minimize the overhead of inter-process interactions. In a system with replicated processes to tolerate a fault, process allocation affects the performance and reliability of the system [1]–[7]. Not only does the system's performance degrade because of the increment in the number of running processes, but the system's reliability also depends on the placement of replicated processes in the system.

Process allocation in fault-tolerant multi-computer systems has been studied by several researchers [1], [2], [4]. Nieuwenhuis [1] studied on the transformation rules which transform an allocation of non-replicated processes into an allocation of replicated processes. The transformed allocation is proven optimal in terms of reliability. Shatz and Wang [2] proposed a process allocation algorithm which maximizes the reliability of non-homogeneous systems. Bannister and Trivedi [4] proposed a process allocation algorithm which evenly distributes the load of the system to all nodes. A common assumption in all of the above research works is that each replicated process is not only a complete replica of the original process, but also has the same execution load as the original. This kind of fault-tolerant process is called an *active* process replica [8].

The fault-tolerant computing process model considered in this paper is the primary-backup process model [9], which is commonly used in distributed experimental and commercial systems such as Delta-4 and Tandem [10], [11]. In this model, there are backup copies for each process in the system. However, only one process in the primary-backup processes is running actively at any time. The active process is called the primary process and the non-active processes are called the backup (or secondary) processes. The active process regularly checkpoints its running state to the backup processes. During the normal operation, the non-active backup processes are either waiting for a checkpointing message or saving a received checkpointing message. When the node in which the primary process is running becomes faulty, the backup process takes over the role of the primary process. Thus, in order to be able to tolerate faults, any two processes from the same primary-backup processes should not be executed on the same node. When there are more than one backup processes, backup processes are sequentially numbered and the role of primary process is taken over by the backup processes in succession. This kind of fault-tolerant process is called a *passive* process replica [11].

In the passive replica process model, the load of a backup process is much less than that of its primary process before fault occurrence, i.e., 5–10% of the load of the primary process. But after a fault occurrence, one of the backup processes takes over the role of the primary process. The load of the backup process becomes the same as that of the original primary process. Therefore, the process allocation algorithm in passive replica process model should consider different load characteristics of the primary and backup processes for balancing the load before and after a fault occurrence.

The load-balancing process allocation problem in the passive process replica model has been dealt by Kim

Manuscript received September 10, 1998.

[†]The authors are with the Department of Computer Science and Engineering, Pohang University of Science and Technology, San 31 Hyoja Dong, Pohang 790-784, Korea.

and Lee [5]. They proposed a heuristic algorithm which balances the load among nodes before as well as after a fault occurrence. One drawback of this work is that only a single fault is considered. Hence, the number of backup process is one. In that model, processes have uneven primary-backup structure after the occurrence of a fault due to the broken primary-backup pair. Some processes have backup processes while others do not have backup processes.

In this paper, we show two schemes for tolerating *multiple faults* in the passive replica model. One is replicating multiple backup processes for each process. The other is re-generating a backup process after the occurrence of a fault for keeping the primary-backup process pair to tolerate the next consequent fault occurrence before the full-recovery of the system. Load-balancing is required to utilize system resources evenly, thereby enhancing the performance of the system. Heuristic process allocation methods are proposed for each scheme, which balance the load before as well as after each fault occurrence. The proposed algorithms consider distribution of primary and backup processes and the load increment to be added to each node in the event of a fault occurrence.

We evaluate the performance of the proposed two fault-tolerant allocation schemes to tolerate multiple faults and compare each other. Also, the reliability of two fault-tolerant process allocation schemes is also analyzed. First, we analyze the maximum number of tolerable faults for each scheme, which is called the *degree of fault tolerance*. Then we evaluate the reliability of each scheme using Markov chains.

This paper is organized as follows. The next section describes the fault-tolerant multi-computer system models considered in this paper and the formal definition of the load-balancing process allocation problem. Section 3 discusses heuristic load-balancing process allocation algorithms for two fault-tolerance schemes in the passive replica model. In Sect. 4, we discuss the performance of two heuristic algorithms proposed in Sect. 3. The reliability of each scheme is evaluated in Sect. 5. In Sect. 6, we summarize and conclude the paper.

2. Fault-Tolerant Process Allocation

2.1 Notation

- n : number of nodes.
- m : number of processes before replication.
- p_i : load of primary process i.
- b_i : load of backup process i.
- ρ : maximum ratio of a backup process load to its primary.
- ω: average load of nodes by primary processes before the fault occurrence.
- x_{ij}^k : 1 if primary process i is allocated to node j before k-th fault occurrence, 0 otherwise.
- y_{ij}^k : 1 if backup process i which will become active by the next fault on its primary is allocated to node j before k-th fault occurrence, 0 otherwise.
- α_j^k : 0 if k-th fault occurs on node j, 1 otherwise.
- τ_j^k : load increment to be added to node j when the k-th fault occurs.
- R : degree of fault tolerance.
- $P^k(j)$: total load of node j after the k-th fault occurrence.
- P_{non}: set of processor's loads with no faulty nodes.
- P_{f_k}: set of processor's loads after k-th fault.
- $\Phi(P)$: difference between the maximum and the minimum load of load set P.
- Ψ : objective cost function to be used.

2.2 Fault-Tolerant System Model

The system model for tolerating multiple faults is an extension of the system model for tolerating single fault with the passive replica process model. Hence, to make it easy to understand the system model for tolerating multiple faults, we first describe the system model for tolerating a single fault considered by Kim and Lee [5].

The fault-tolerant multi-computer system considered for tolerating a single fault consists of n nodes (processors). To tolerate a fault, each process is replicated and executed as a pair, referred to as a primary-backup process pair. Primary processes can be allocated to any node. However, there is one restriction on the placement of backup processes. That is, a primary process and its backup process cannot be allocated to the same node. It is assumed that there are m primary processes running in the system and that the cpu loads of the primary and backup processes are known in advance. This assumption about the load requirement is not unrealistic since many on-line transaction systems run the same processes continuously [10], [12]. Note that we assume the number of processes is a lot larger than the number of processors, i.e., $m \gg n$.

The primary-backup process model is actually used in experimental and commercial systems such as the Tandem Nonstop system [10], [13]. In the Nonstop system, every process has an identical backup process which is allocated to a different node. The backup process is not executed concurrently with the primary process, but is in an inactive mode, prepared to assume the function of the primary process in the event of a primary process failure. To guarantee that the backup process has the information necessary to execute the required function, the primary process sends periodic checkpoint messages to its backup process.

In the fault-free situation, the cpu load of the backup processes is much less than that of their respective primaries. The actual load of a backup process is determined by the interval and number of messages checkpointed by the primary process. Each backup process has a different percentage of the primary process' cpu load. When a fault occurs, the backup processes of the primary processes which were running on the faulty node take over the role of the primary processes. The backup process is executed continuously starting from the last point at which it received a valid checkpointing message from its primary process. Therefore, the cpu load of the backup process becomes the same as that of its primary.

A modification of the system model based on the primary-backup process pair is required to tolerate multiple faults. We can consider two approaches for continuous operation with the passive replica. The first approach is *the multiple backup allocation scheme* (MUL) that makes R ($< n$) backup processes for each primary process to tolerate at maximum R faults such as shown in Fig. 1 [9], [11]. In MUL, the R backup processes are executed on different R nodes except the node on which its primary is running. Each primary process sends checkpointing messages to all R backup processes. When a fault occurs, the role of the primary process is taken over by the designated backup process until no more backup process is available.

The second approach is *the regenerative single backup allocation scheme* (REG) having only one backup process for each primary process. After a fault occurrence, the backup processes whose primaries were running on the faulty node become the primaries. However, these processes do not have backup processes. The primary processes which have their backups on the faulty node also do not have backup processes. For those processes not having backup processes after each fault occurrence, backup processes are re-generated to keep one backup process for each primary process. Figure 2 illustrates the process of REG for tolerating two faults in a three node system.

2.3 Load Balancing Process Allocation Problem

In this subsection, we formally describe the load-balancing process allocation problem. The load-balancing process allocation problem is represented as a constrained optimization problem. Let us assume that there are n nodes and m processes in the system. One prominent constraint is on the allocation of primary and backup processes. To be able to tolerate a fault in a node, a primary process and its backups should not be allocated to the same node.

$$\sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij}^k = \sum_{i=1}^{m}\sum_{j=1}^{n} y_{ij}^k = m, \sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij}^k y_{ij}^k = 0. \tag{1}$$

It should be noted that $y^i \cdot y^j = 1$ if $i = j$ and 0 otherwise.

When the k-th fault occurred in node s, the load increment to be added to node j by the fault is $\sum_{i=1}^{m} x_{is}^k y_{ij}^k (p_i - b_i)$. Thus, the load increment (τ_{jk}) for node j by the k-th fault is

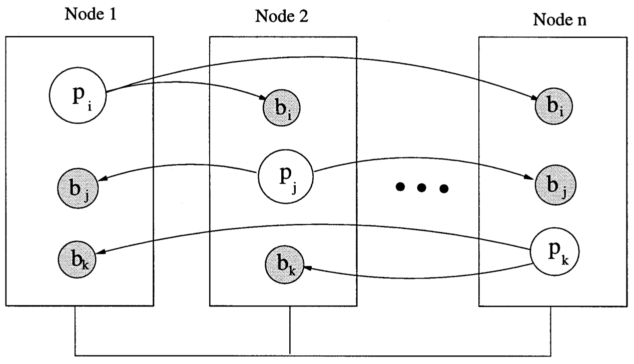
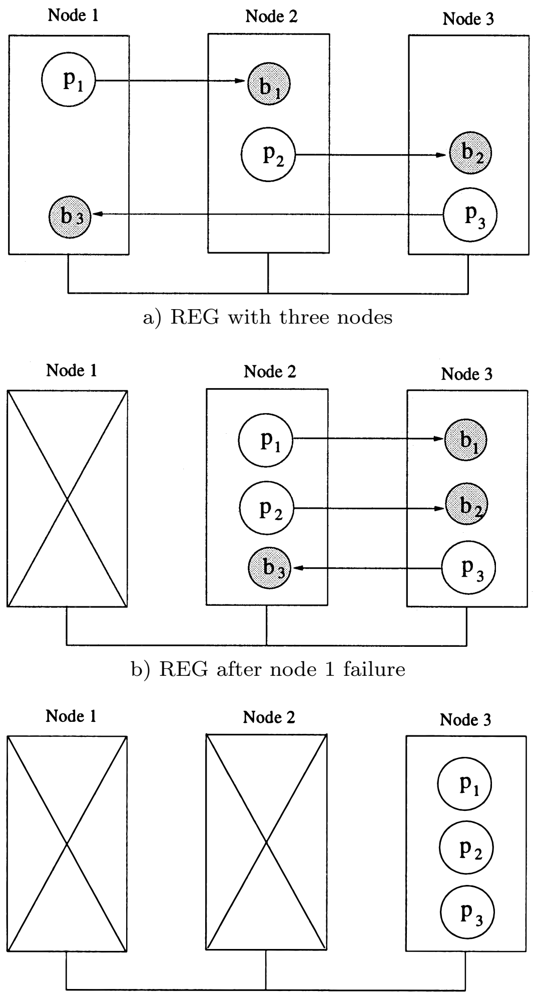


Fig. 1 MUL for tolerating multiple faults.



c) REG after node 2 failure

Fig. 2 A running example of REG.

$$\tau_j^k = \sum_{s=1}^{n} \left[\left(1-\alpha_s^k\right) \sum_{i=1}^{m} x_{is}^k y_{ij}^k \left(p_i - b_i\right) \right]. \qquad (2)$$

The load of node j after the k-th fault, denoted as $P^k(j)$, is the sum of the load before the fault occurrence and the load increment incurred before the k-th fault. Therefore, the load of node j in MUL with R replication is

$$\begin{aligned} P_{MUL}^k(j) &= P_{MUL}^{k-1}(j) + \tau_j^k \\ &= \sum_{f=1}^{k} \tau_j^f + \sum_{i=1}^{m} \left(x_{ij}^1 p_i + \sum_{l=1}^{R} y_{ij}^l b_i \right). \end{aligned}$$

The load of node j in REG is

$$P_{REG}^k(j) = \tau_j^k + \sum_{i=1}^{m} \left(x_{ij}^k p_i + y_{ij}^k b_i \right).$$

In REG, since the value y_{ij}^k is decided after the $(k-1)$-th fault occurrence, the load imbalance can be reduced by adjusting $P_{REG}^k(j)$, which is done by choosing y_{ij}^k. This dynamic allocation capability in REG is used to improve the performance when we allocate newly generated backup processes.

The commonly used metric for evaluating load-balance between nodes is the standard deviation of the processor loads [4]. Another metric that can be used in the load-balancing process allocation problem is the load difference between the node with the heaviest load and the node with the lightest load [5]. The load difference, denoted as Φ, for the given set of process loads P can be represented as follows:

$$\Phi(P) = \max(P) - \min(P).$$

The set of process loads before any node failure is represented as P_{non} and the set of process loads after any one node failure is represented as P_{f1}. After the k-th fault, the set of loads is represented as P_{fk}.

The load-balancing process allocation problem is the problem of finding values of x_{ij}^k and y_{ij}^k for all possible i, j, and k, that minimize the multiple cost functions $\Phi(P_{non})$, $\Phi(P_{f_1})$, $\Phi(P_{f_2})$, ..., $\Phi(P_{f_R})$ in R fault tolerant systems with the constraint given in Eq.(1). There are two main approaches to solve an optimization problem that involves multiple objective functions [14]. One approach is to solve the problem a number of times with each objective in turn. When solving the problem using one of the objective functions, the other objective functions are considered as constraints. The other approach is to build a suitable linear combination of all the objective functions and optimize the combination function. In this case, it is necessary to attach a weight to each objective function depending on its relative importance. In this paper, the second approach is used to formulate the load-balancing process allocation problem.

We define an objective function Ψ as

$$\Psi = w_0 \cdot \Phi(P_{non}) + \sum_{k=1}^{R} w_k \cdot \Phi(P_{f_k}), \qquad (3)$$

where w_k $(0 \le k \le R)$ is the relative weight of importance before and after a fault occurrence. If we assume that the weights have the same value in two fault-tolerant systems, i.e. $w_0 = w_1 = w_2 = 1$, the objective function is

$$\Psi = \Phi(P_{non}) + \Phi(P_{f1}) + \Phi(P_{f2}). \qquad (4)$$

The objective function of the load balancing process allocation problem, Ψ, is defined as the sum of the load differences before and after the occurrence of faults. An optimal allocation is an assignment of processes that minimizes the objective cost function Ψ. The load-balancing process allocation problem in fault-tolerant systems was proved as NP-Hard in [5]. Hence, we propose heuristic approximation algorithms which are cost-effective and result in well balanced processor loads before and after the occurrence of multiple faults.

3. Heuristic Process Allocation Algorithms

3.1 Basic Primitives for Load Balancing Process Allocation

In this section, it is assumed that processes are allocated one by one, and, once allocated, processes are not reallocated to other nodes. The summary of basic primitives of load balancing process allocation which were discussed in [5] is as follows.

1. A process should be allocated to the node with the minimum load first in order to minimize Φ.
2. Processes with more load should be allocated prior to processes with less load in order to minimize Φ.
3. When two ordered sets with equal cardinality are merged to one set with equal cardinality such a way that the maximum from one set is combining with the minimum from the other set, the Φ of the merged set is less than or at least equal to the minimum Φ of two ordered sets.

The proposed heuristic algorithm, which satisfies the above allocation primitives, works in two stages. In the first stage, primary processes are allocated using a standard load balancing algorithm. We use a greedy method which allocates the process with the heaviest load to the node with the lightest load. As shown in the above summaries 1 and 2, this greedy method minimizes Φ. In the second stage, backup processes are allocated considering the load increment to be added to each node in the event of a fault. Let us assume that the algorithm is currently working on node j. The backup processes whose primary processes are assigned to node j are divided into $(n-1)$ groups having approximately equal incremental load in the event of a fault in node j. These $(n-1)$ groups of backup processes are allocated to $(n-1)$ nodes, excluding node j. The detail of the process allocation algorithms for each scheme is described in the next two subsections.

3.2 Process Allocation Algorithm for MUL

The process allocation algorithm for MUL is formally described below.

Process Allocation Algorithm for MUL

Stage 1: *Allocate primary processes*

1. Sort primary processes in descending order of cpu load.
2. Allocate each primary process to the node with the minimum load from the heaviest load to the lightest.

Stage 2: *Allocate backup processes*
Do the following steps for each node.

1. Compute the load difference between each primary process and its backup process for all primary processes assigned to the node.
2. Sort, in descending order, the backup processes using the load difference.
3. Divide the backup processes into $(n-1)$ groups having an approximately equal incremental load by assigning *each R backup processes to the R groups with the smallest load*, in the order of the sorted list in the previous step.
4. Compute the actual backup process load of each group.

Do the following for the $n(n-1)$ backup groups which are generated by the above steps.

1. Sort all $n(n-1)$ backup groups using the actual loads in descending order.
2. Sort the n nodes using their current loads in ascending order.
3. Allocate each backup group to the node with the minimum load. However, if the backup group has a corresponding primary process in this node or if one of the backup groups which are already allocated to this node comes from the same node as the backup group to be allocated, choose the node with the next-to-the-minimum load.

The allocation of backup processes is the most important part of this algorithm. For each node, the backup processes are first divided into $(n-1)$ groups using the load difference between each primary process and its backup. This load difference is the amount of load increment to be incurred upon the occurrence of a fault. Next, the algorithm computes the actual load of each group before a failure using the actual load of the backup processes. The total number of backup groups is $n(n-1)$. Each group is assigned to a node depending on its computed actual load. When we allocate each backup group, we check whether there is a pre-allocated backup group which comes from the same node as the to-be allocated backup group. In such a case, we select the node with the next-to-the-minimum load.

The purpose of dividing the backup processes into $(n-1)$ groups for each node is to guarantee that each node has an approximately equal amount of load increment. Hence, the system will have a balanced load when a fault occurs. The purpose of computing the actual load of each group and assigning groups based on their actual loads is to guarantee that each processor's load is balanced before the occurrence of a fault. Therefore, we can balance the processor load before as well as after the occurrence of faults.

3.3 Process Allocation Algorithm for REG

The allocation algorithm for REG is as follows.

Process Allocation Algorithm for REG

The algorithm is exactly the same as the algorithm for MUL except the third step at the stage 2.

3. Divide the backup processes into $(n-1)$ groups having an approximately equal incremental load by assigning each backup process to the group with the smallest load, in the order of the sorted list in the previous step.

The key idea of the two-stage allocation is the allocation of backup processes for balancing the loads by an equal load increment in non-faulty nodes. After a fault occurrence, the backup processes whose primary processes were running on the faulty node become primary processes. These converted primary processes do not have backup processes. Also, the primary processes which were running their backup processes on the faulty node lose their backup processes. Backup processes for these primary processes should be re-generated to tolerate the next fault.

Regenerated backup processes are distributed to balance the load dynamically with consideration of current load states of non-faulty nodes. The idea of two-stage allocation is to distribute newly generated backup processes. The allocation of regenerated backup processes after the k-th fault occurrence is a decision procedure for the value y_{ij}^{k+1} which is the same as x_{ij}^{k+2}. Therefore, the following equation should hold.

$$x_{ij}^{k+1} = y_{ij}^{k}, \quad \text{for any i,j,k}$$

The proposed dynamic allocation algorithm for allocating new backup processes after the k-th fault occurrence works as follows.

Dynamic Allocation of Regenerated Backup Processes

1. Find backup processes whose primary has no backup process.
2. Sort the backup processes in descending order using the load difference, i.e., $(p_i - b_i)$.
3. Allocate each backup process i whose primary is running in node j $(x_{ij}^{k+1} = 1)$ to the node s $(\neq j)$ with minimum $P^k(s) + \sum_{i=1}^{m} x_{ij}^{k+1} y_{is}^{k+1} (p_i - b_i)$ among $(n-k-1)$ nodes for minimizing $\Phi(P_{f_{k+1}})$.

Backup processes are allocated to balance loads after the occurrence of next $(k+1)$-th fault. The allocation of backup processes does not break the currently balanced load since the allocated backup processes have small loads, but also guarantees the load balance for the next fault.

4. Performance Evaluation

In this section, we compare the two proposed schemes in order to find which scheme is more suitable for tolerating multiple faults with good performance.

We compare the performance of both heuristic algorithms using simulation. For the convenience of simulation, we let the number of faults that can be tolerated be two, i.e., $R = 2$. Other parameters used in the simulation are as follows. To keep the total load of each node below 100%, the load of the primary processes is chosen randomly in the range of 0.5 to 2.5 times $100 \cdot (n-2)/m$ based on a uniform distribution. The load of the backup processes is also chosen randomly between 5–10% of the load of their primaries, also based on a uniform distribution.

Figure 3 shows the load difference Φ between the minimum and the maximum load before and after the occurrence of one and two faults. The simulation result is obtained for $n = 16$. When there is no fault, the two schemes show almost identical results. The load difference becomes larger when a fault occurs. However, the results of two schemes are in a very close agreement. The load difference becomes much larger for MUL due to the uneven distribution of primary processes. From the figure, we can conclude that the load difference of MUL becomes larger than REG as the number of faults increases.

Figure 4 shows the objective cost function Ψ of both schemes when the number of processes varies from 100 to 500. The simulation results are obtained for $n = 8$ and 16. The Ψ in Fig. 4 is relatively high since we merely added the load differences of no-fault, one-fault, and two-fault cases $(w_o = w_1 = w_2 = 1)$. Regardless of the number of processes before replication, the Ψ of

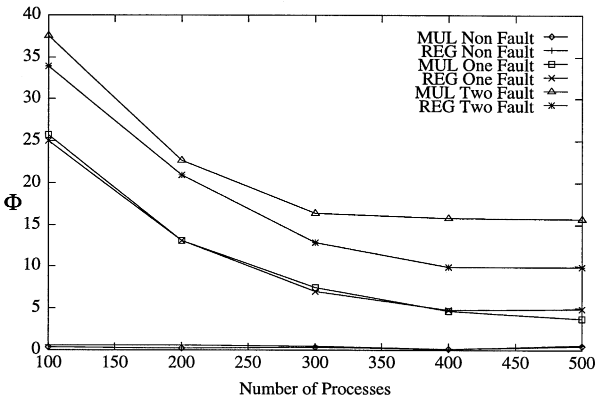


Fig. 3 Load difference Φ in both model.

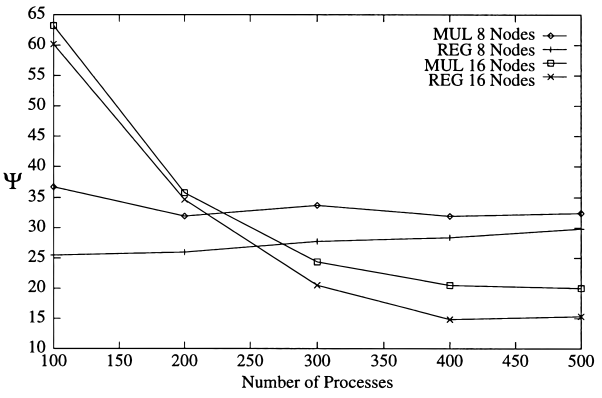


Fig. 4 Ψ vs. the number of processes.

REG is less than the Ψ of MUL. From this figure, we can also know that the performance of REG is superior to that of MUL.

In Fig. 5, the effect of the number of nodes to the objective cost function is shown. The number of processes is fixed to 200 or 400. The best balanced state in both schemes with respect to the specified number of processes is obtained for different number of nodes. When the number of processes is 200, a system with 10 nodes can achieve the best balanced state in both models by the proposed algorithms. However, when the number of processes is 400, a system with 16 nodes shows the best balanced state in both models. In this experiment, REG also outperforms MUL.

In previous simulations, we assumed that backup processes have 5–10% of the load of their primary processes. In this simulation, we compare the performance of both models by varying the load ratio ρ of backup processes from 10–100% of their primaries. As shown in Fig. 6, when the load ratio ρ is small, the performance of MUL is slightly less than REG. However, when the load ratio increases, the performance of MUL degrades linearly. On the contrary, REG performs well irrespectively to the load ratio ρ. As the load ratio ρ increases, the performance of REG improves.

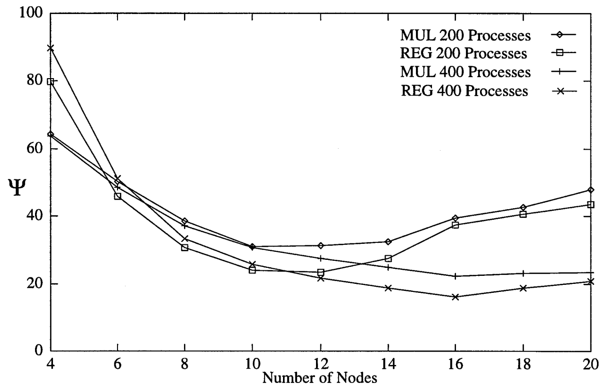


Fig. 5 Ψ vs. the number of nodes.

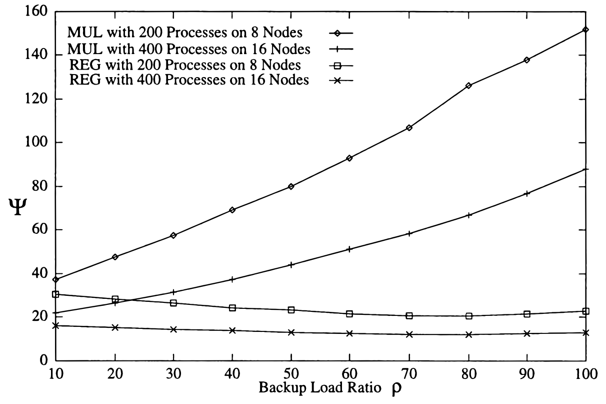


Fig. 6 Ψ vs. the load ratio ρ.

5. Reliability Evaluation

In this section, we evaluate and compare the reliability of two different fault-tolerant process allocation schemes. The most popular reliability evaluation technique is using Markov chains, which is used in this paper. The evaluation is conducted in two steps. First, we find the maximum number of tolerable faults, which is called the *degree of fault tolerance*, for each scheme. Next, we model two schemes using Markov chains and evaluate the models with the Hybrid Automated Reliability Predictor (HARP) [15].

5.1 Degree of Fault Tolerance

To tolerate a fault occurred, there are two conditions to be satisfied. They are the availability of backup processes and the system capacity. A fault occurred can be tolerated if backup processes are available for all the primary processes on a faulty node and the capacity of the system is enough to handle the transformation of backup processes to primary processes. From the condition of backup process availability, we know that backup processes are available until the R-th fault in MUL with R replicated process allocation and until the $(n-1)$-th fault in REG. From the condition of system capacity, the load of any node should not exceed 100% after a fault occurrence[†]. From these conditions, we can estimate the maximum number of tolerable faults for each scheme.

The total load of processes in MUL varies as a fault occurs, while the total load of processes in REG remains constant. As the number of faults increases, the total load of processes in MUL decreases due to the lost backup processes. To find the degree of fault tolerance in MUL, we have to know the average number of backup processes remained after a fault occurrence.

Lemma 1: In MUL with R replication, the average number of backup processes remained among R backups after the k-th fault occurrence ($1 \le k \le R$) is $((n-k)R-k)/n$.

Proof: Initially, there are m primary processes and mR backup processes. When a fault occurs, all primary and backup processes on a faulty node are vanished. However, the role of the primary processes on the faulty node is taken by the corresponding backup processes on non-faulty nodes. Therefore, after the k-th fault occurrence, still m primary processes are running on $(n-k)$ non-faulty nodes. But, the total number of backup processes in the system becomes less than mR since some of backup processes are vanished and some are transformed to primaries. The average number of backup processes vanished until the k-th fault occurrence is $\frac{k}{n}mR$ and the average number of backup processes that become primary processes on non-faulty $(n-k)$ nodes is $\frac{k}{n}m$. Therefore, the average number of backup processes after the k-th fault occurrence is

$$mR - \left(\frac{k}{n}\right)mR - \left(\frac{k}{n}\right)m = \left(\frac{n-k}{n}R - \frac{k}{n}\right)m.$$

Thus the average number of backup processes remained for each primary is $((n-k)R-k)/n$ among R replicated backup processes. □

Theorem 1: When $\sum_{i=1}^{m} p_i = \omega n$ and $\sum_{i=1}^{m} b_i = \rho\omega n$, the degree of fault tolerance of MUL is bounded by $\lfloor(\sqrt{(100+(n+1)\rho\omega)^2 + 4(100-\omega)\rho\omega n} - (100+(n+1)\rho\omega))/2\rho\omega\rfloor$.

Proof: Let R be the number of replicated backup processes for each primary. To tolerate the k-th fault ($1 \le k \le R$), the maximum load on non-faulty nodes should be less than 100%.

$$\max(P_{f_k}) \le 100 \tag{5}$$

Let R_k be the average number of backup processes for each primary after the k-th fault occurrence. Then, the following should be satisfied.

$$\frac{1}{n-k}\sum_{i=1}^{m}(p_i + R_k b_i) \le 100 - \alpha \tag{6}$$

[†]We consider that nodes have a severely degraded performance if their load exceeds 100%.

where α is defined as $\max(P_{f_k}) - \text{avg}(P_{f_k})$ and is approximated to $\frac{1}{2}\Phi(P_{f_k})$. α has a non-negative value (i.e., $\alpha \geq 0$) and is dependent on the process allocation algorithm and the load distribution of primary and backup processes. The better the allocation algorithm is used, the closer α is to zero.

Since R_k is $((n-k)R-k)/n$ from Lemma 1, Eq.(6) becomes as follows.

$$\frac{\omega n}{n-k} + (\frac{R}{n} - \frac{k}{n(n-k)})\rho\omega n \leq 100 - \alpha \qquad (7)$$

From Eq.(7), k is bounded by

$$k \leq \frac{(100 - \alpha - \omega - R\rho\omega)n}{100 - \alpha + (R+1)\rho\omega}. \qquad (8)$$

As R increases, the upper bound of k in Eq.(8) decreases. Since it is meaningless to have R larger than k, we have to find the maximum of k by letting $R = k$. Since k has the maximum when $\alpha = 0$, Eq.(8) becomes as follows.

$$k \leq \frac{(100 - \omega - k\rho\omega)n}{100 + (k+1)\rho\omega} \qquad (9)$$

From Eq.(9), the range of k is determined by $0 \leq k \leq (\sqrt{(100 + (n+1)\rho\omega)^2 + 4(100-\omega)\rho\omega n} - (100 + (n+1)\rho\omega))/2\rho\omega$. □

Theorem 2: When $\sum_{i=1}^{m} p_i = \omega n, \sum_{i=1}^{m} b_i = \rho\omega n$, the degree of fault tolerance of REG is bounded by $\lfloor(1 - (1+\rho)\frac{\omega}{100})n\rfloor$.

Proof: After the k-th fault occurrence ($1 \leq k \leq n-1$), the maximum load on each of the $(n-k)$ non-faulty nodes should be less than 100%. Since the total load of primary and backup processes is $\sum_{i=1}^{m}(p_i + b_i)$, the following should be satisfied.

$$\frac{1}{n-k}\sum_{i=1}^{m}(p_i + b_i) \leq 100 - \alpha \qquad (10)$$

where α has the same meaning as in the proof of Theorem 1. From the load condition of primary and backup processes, Eq.(10) becomes as follows.

$$(1+\rho)\omega n \leq (n-k)(100-\alpha) \qquad (11)$$

From Eq.(11), k is bounded by

$$k \leq \frac{(100 - \alpha - (1+\rho)\omega)n}{100 - \alpha}. \qquad (12)$$

In Eq.(12), as α increases, the maximum value of k decreases. Since $\alpha \geq 0$, the maximum value of k is bounded by $\lfloor(1 - (1+\rho)\frac{\omega}{100})n\rfloor$. □

Let R_{MUL} represent the degree of fault tolerance of MUL and R_{REG} the degree of fault tolerance of REG. For example, $R_{MUL} = 2$, but $R_{REG} = 3$ when $\omega = 50, \rho = 0.1$, and $n = 8$. Also, when $\omega = 50$, $\rho = 0.1$, and $n = 16$, then $R_{MUL} = 3$ and $R_{REG} = 7$. Usually, the degree of fault tolerance of REG is bigger than that of MUL in the same workload condition.

5.2 Reliability Modeling

The system with MUL or REG replication method is modeled using a continuous time Markov chain. The processing nodes have identical and exponential distribution in failure time with mean $\frac{1}{\lambda}$. We assume that there is no repair. When a node failure is detected, backup processes whose primary processes were running on a faulty node take over the role of their primaries within $\frac{1}{\mu_1}$ time on average. In REG, lost backup processes are regenerated within $\frac{1}{\mu_2}$ time on average. Usually, the time to generate processes takes longer than the time to switch backup processes to primaries. Thus $\frac{1}{\mu_1} \leq \frac{1}{\mu_2}$. Hence, the average fault-tolerance latency of MUL and REG are $\frac{1}{\mu_1}$ and $\frac{1}{\mu_1} + \frac{1}{\mu_2}$, respectively. It is also assumed that if another fault is occurred during the period of tolerating previous fault, it causes a system failure. This kind of faults are called as *"near-coincident" faults.* We assume a perfect fault detection and reconfiguration.

The R-resilient system with MUL is modeled such as shown in Fig. 7 and the R-resilient system with REG is modeled such as shown in Fig. 8. The degree of fault tolerance, R, is measured on previous subsection. State S_i represents the state that has $(n-i)$ operational nodes in the system. State FB_i represents the fault benign state after the i-th fault occurrence and state RG_i represents the state of regenerating lost backup processes.

5.3 Reliability Evaluation Using HARP

The reliability of the system modeled in the previous subsection is evaluated using HARP [15]. For a system with $n = 8$, $\omega = 50$, and $\rho = 0.1$, the degree of fault tolerance of MUL is $R_{MUL} = 2$ and that of REG is $R_{REG} = 3$. The reliabilities for each allocation model for different μ_2 values are shown in Fig. 9

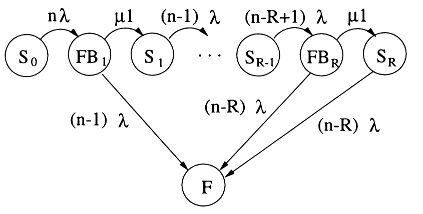


Fig. 7 State transition diagram of MUL.

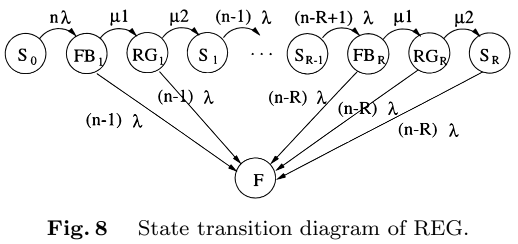


Fig. 8 State transition diagram of REG.

for 10000 hours (about one year) when $\lambda = 10^{-4}$ and $\mu_1 = 100$. The lower solid line represents the reliability of MUL. The upper dotted line is the reliability of REG when $\mu_2 = 100$, which is overlapped with the lines when $\mu_2 = 10$ and 1. In a system with 10^{-4} node failure rate and one hour fault-tolerance latency, the effect of near-coincident faults is negligible.

For a system with $n = 16$, $\omega = 50$, and $\rho = 0.1$, the degree of fault tolerance of MUL is $R_{MUL} = 3$ and that of REG is $R_{REG} = 7$. The reliabilities when $\lambda = 10^{-4}$ and $\mu_1 = 100$ are shown in Fig. 10. From Figs. 9 and 10, we can see that the system with REG can maintain much higher reliability than the system with MUL. We also experimented when node failure rate is 10^{-5}. The reliability distributions for each scheme show similar behaviors like previous figures with 10^{-4} node failure rate.

From the reliability evaluation, we get the following results.

- In the situation that the life time of nodes is several years ($\lambda = 10^{-4}$–10^{-5}) and the fault-tolerance latency is less than hours, the possibility of system failure due to a near-coincident fault is negligible.
- The reliability of a system is highly dependent on the degree of fault tolerance.
- The degree of fault tolerance of REG is much larger than that of MUL, especially when the number of nodes in a system increases, or when the total load of processes decreases.
- The system with REG is more reliable than the system with MUL at the same workload condition.

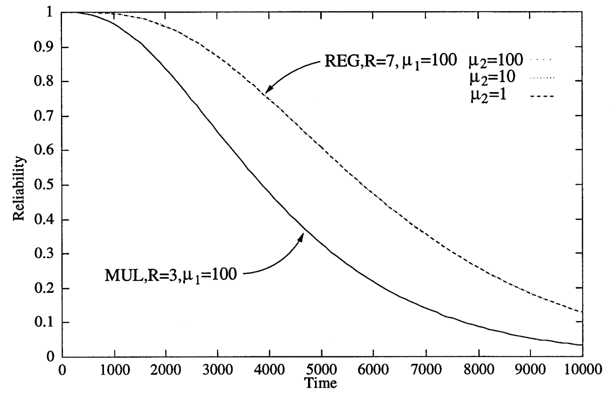


Fig. 9 Reliability of a 8-node system with $\lambda = 0.0001$ and $\mu_1 = 100$.

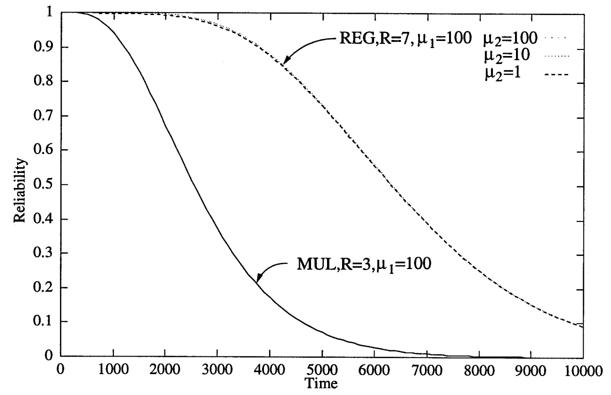


Fig. 10 Reliability of a 16-node system with $\lambda = 0.0001$ and $\mu_1 = 100$.

6. Conclusion

In this paper, we considered a load-balancing process allocation for fault-tolerant systems that balances the load before as well as after the occurrence of faults. We showed two schemes which tolerate multiple faults in the passive replica model. The first scheme is called the *multiple backup process model* (MUL) and the other scheme is called the *regenerative single backup process model* (REG). Heuristic allocation algorithms are proposed for both schemes, and the performance of each scheme is evaluated using simulation in terms of load imbalance after process allocation. The regenerative single backup process model has less overhead and better performance than the multiple backup allocation method.

Since the reliability enhancement is the main concern of fault-tolerant system design, the reliability of the systems with each scheme is evaluated under various environments. From the reliability evaluation, the system with REG maintains much higher reliability than the system with MUL. Especially, as the number of nodes in a system increases or as the total load of processes decreases, the system with REG is much reliable than the system with MUL. It is also shown that the effect of near-coincident faults is negligible.

From the extensive comparison of the system with each scheme in terms of performance and reliability, the regenerative single backup process allocation scheme is more suitable for the fault-tolerant system with passive replica than the multiple backup allocation scheme.

Acknowledgement

A part of this paper was presented at the Pacific Rim Int'l Symp. on Fault-Tolerant Systems (PRFTS'95). This research was supported in part by NON DIRECTED RESEARCH FUND of Korea Research Foundation, and KOSEF under Grant 941-0900-055-2.

References

[1] L.J.M. Nieuwenhuis, "Static allocation of process replicas in fault-tolerant computing systems," Proc. FTCS-20, pp.298–306, June 1990.

[2] S.M. Shatz, J.P. Wang, and M.Goto, "Task allocation for maximizing reliability of distributed computer systems," IEEE Trans. Comput., vol.41, pp.1156–1168, Sept. 1992.

[3] M. Livingston and Q.F. Stout, "Fault tolerance of allocation schemes in massively parallel computers," Proc. 2nd Massively Parallel Computers Conference, pp.491–494, 1989.

[4] J.A. Bannister and K.S. Trivedi, "Task allocation in fault-tolerant distributed systems," Acta Informatica, vol.20, pp.261–281, 1983.

[5] J. Kim, H. Lee, and S. Lee, "Replicated process allocation for load distribution in fault-tolerant multicomputers," IEEE Trans. Comput., pp.499–505, April 1997.

[6] C. H. Chen and V. Cherkassky, "Task allocation and reallocation for fault tolerance in multicomputer systems," IEEE Trans. Aerosp. & Electron. Syst., vol.30, pp.1094–1104, 1994.

[7] R. Davoli, L.A. Giachini, Ö. Babaoglu, A. Amoroso, and L. Alvisi, "Parallel computing in networks of workstations with Paralex," IEEE Trans. Parallel & Distributed Systems, pp.371–384, April 1996.

[8] M. Chereque, D. Powell, P. Reynier, J.L. Richier, and J. Voiron, "Active replication in delta-4," Proc. FTCS-22, pp.28–37, June 1992.

[9] R. Guerraoui and A. Schiper, "Software-based replication for fault tolerance," IEEE Comput., pp.68–74, April 1997.

[10] D.P. Siewiorek and R.S. Swartz, Reliable System Design: The theory and practice, Digital Press, New York, 1992.

[11] N. Speirs and P. Barrett, "Using passive replicates in delta-4 to provide dependable distributed computing," Proc. FTCS-19, pp.184–190, June 1989.

[12] A. Nangia and D. Finkel, "Transaction-based fault-tolerant computing in distributed systems," 1992 IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems, pp.92–97, July 1992.

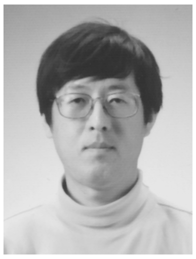
[13] D.K. Pradhan, Fault-Tolerant Computer System Design, Prentice-Hall, Inc., 1995.

[14] H.P. Williams, Model Building in Mathematical Programming, 2nd ed., John Wiley & Sons Ltd., 1985.

[15] J.B. Dugan, R. Geist, and K.S. Trivedi, "The hybrid automated reliability predictor," AIAA J. Guidance, Control, and Dynamics, vol.9, no.3, pp.319–331, 1986.

Heejo Lee received his BS and MS in computer science and engineering from Pohang University of Science and Technology, Korea in 1993 and 1995, respectively. He is currently working toward the Ph.D. at the same universtiy. His current research interest includes fault-tolerant computing, parallel scientific computing, and computer security.

Jong Kim received the BS degree in electronic engineering from Hanyang University in 1981, the MS degree in computer science from the Korea Advanced Institute of Science and Technology in 1983, and the Ph.D. degree in computer engineering from Pennsylvania State University in 1991. He is currently an Associate Professor in the Department of Computer Science and Engineering, Pohang University of Science and Technology, Pohang, Korea. Prior to this appointment, he was a research fellow in the Real-Time Computing Laboratory of the Department of Electrical Engineering and Computer Science at the University of Michigan from 1991 to 1992. From 1983 to 1986, he was a system engineer in the Korea Securities Computer Corporation, Seoul, Korea. His major areas of interest are fault-tolerant computing, performance evaluation, and parallel and distributed computing.



Sung Je Hong received the BS degree in electronics engineering from Seoul National University in 1973, the MS degree in computer science from Iowa State University in 1979, and the Ph.D. degree in computer science from the University of Illinois, Urbana-Champaign in 1983. He is currently a Professor in the Department of Computer Science and Engineering, Pohang University of Science and Technology, Pohang, Korea. From 1983 to 1989, he was a staff member of Corporate Research and Development, General Electric Company, Schenectady, NY, U.S.A. From 1975 to 1976, he was with Oriental Computer Engineering, Korea, as a logic design engineer. From 1973 to 1975, he served the army as a system analyst at the Army Finance Center, Korea. His current research interest includes VLSI design, CAD algorithms, testing, and parallel processing.