Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

A Module-Sharing Scheme to Fault-Tolerant Multiprocessors

Shun-Yue Wu[1] and Chu-Sing Yang[2]
[1] Department of Computer Science and Information Management
Chinese Military Academy, Kaohsiung, Taiwan, ROC.
Tel: 886-7-7438179 ext 10
Email: sywu@cc.cma.edu.tw.
[2] Institute of Computer and Information Engineering,
National Sun Yat-Sen University, Kaohsiung, Taiwan, ROC.
Tel: 886-7-5254300
Fax: 886-7-5254301
Email: csyang@cie.nsysu.edu.tw.

Abstract

This paper presents a module-sharing scheme for constructing global fault-tolerant multiprocessors which, not limited to special topologies, can apply to any general multiprocessor systems. The system is constructed from modules in which one spare node in any module can serve to replace any faulty node in the same module or in any other remote module. Therefore, the proposed scheme possesses global sharing capability and achieves maximum utilization of spares. To meet such requirements, a reconfiguration strategy is developed to replace faulty nodes by allocating available spares. The findings demonstrate that in most cases the strategy successfully reconfigures the architecture and requires one additional replacement step in a few cases. Compared to certain other global schemes, the proposed scheme achieves the same measure of reliability at less hardware cost. Employing hardware of the same cost, the proposed scheme achieves higher reliability than certain other modular schemes. Our findings conform that the proposed scheme is superior to other analogous systems.

Keywords: Fault-tolerance, Module-sharing, Reconfiguration, Reliability.

1. Introduction

Multiprocessor architectures for high-performance computing may require a large number of processors linked into interconnection networks. Interconnection topologies such as Boolean n-cubes [2, 9], tree [7], or cube-connected cycles [10] are widely preferred techniques. When the number of nodes (processors) in multiprocessors increases rapidly, the probability of node failures is quite considerable. Systems may be unreliable if no fault-tolerant features are embedded.

Fault-tolerant schemes generally encompass three approaches: global, modular, and hybrid. In global schemes [1, 3, 6], spares can replace any faulty nodes in the whole system, in modular schemes, such as proposed in [7, 11], spares can only replace faulty nodes in a local module. In hybrid schemes [4, 12], spares can replace faulty nodes in the same module and also be partially shared by a number of other modules. The following reviews various schemes mentioned in the literature.

Rennels [11] proposed two fault-tolerant architectures based on Boolean n-cubes, the first of which, failing to achieve high reliability, divides an n-cube of 2^n nodes into 2^{n-m} subcubes of 2^m nodes each where $n \geq m$. A spare node is added to each subcube. A subcube of 2^m nodes with a spare node as well as the interconnection elements among

the nodes, makes a fault-tolerant module. In any module, a spare node can replace any faulty node. Therefore, the spare node requires a connection to every node in the subcube and to each of their neighboring nodes in the other subcubes. Thus, the modular scheme achieves only low reliability because the presence of two faulty nodes in any module renders the system unreliable. For long-life unmaintained systems, where the highest reliability is required, Rennels' second of the above mentioned is the level redundancy approach. In this scheme, one spare node is attached to each subcube of four nodes via a high speed bus. The four nodes with a spare node constitute a level-one group. This approach is applied recursively, i.e., four level-one groups with a spare group defines a level-two group. The hybrid scheme may provide high reliability, but the high hardware overhead and low spare utilization may limit its applications.

Chau & Liestman [3] presented a global fault-tolerant Boolean n-cube architecture, employing k levels of decoupling networks to hook up k spare nodes into a system where the status of a level of decoupling networks is changed for joining a spare node into the system. Once a failed node is identified, a spare node is employed by setting the direction of the lowest unused level of decoupling networks to right. Note that the spare node does not replace the faulty node directly. The authors have verified that the scheme is far more reliable than either of Rennels' scmeme, while involving less hardware overhead. Note that although the proposed scheme can be generalized to other architectures, i.e., it is not limited to n-cube, it requires about N/2 task reassignments per reconfiguration where N is the number of the original nodes.

In 1991, Alam & Melhem [1] proposed an efficient modular spare allocation scheme, applied to fault-tolerant Boolean n-cubes. They demonstrated that for a given hardware overhead, by setting bigger fault-tolerant modules without 100% spare utilization more reliable systems can be designed than by smaller fault-tolerant modules with 100% spare utilization. The findings clearly reveal that global fault-tolerant designs achieve higher reliability than modular designs, but entail complicated reconfiguration. They straightforward employ multiplexers and demultiplexers to attach the spare nodes into the system. Thus, the scheme [1], involves less hardware overhead and no task reassignment, achievesw higher reliability than the schemes proposed in [3].

Dutt & Hayes [6] presented another general approach to fault-tolerant multiprocessor design termed the node-covering method, which is an extension of the covering methodology proposed in [5] for designing near-optimal fault-tolerant tree architectures. Their work describes some important practical design features of fault-tolerant multiprocessors, including the ability both to perform rapid reconfiguration without a centralized controller, and to reconfigure around new faults without undoing any reconfiguration around earlier faults.

In [4], Chen and Upadhyaya presented a multi-level redundancy fault-tolerant tree which belongs to a hybrid approach. The scheme defines a module by three original nodes with a spare node. In such a module, the spare node can replace any faulty node. In addition, several spare modules are allocated at every level for replacing faulty modules. A module becomes faulty if the number of faulty nodes in that module is more than one. By switch implementation, this scheme is modeled by decoupling networks [4]. To improve spares utilization, the optimized allocation of spare modules is also presented. That is, how should a given number of spare modules be allocated to every level to ensure the maximum reliability.

In this paper, a module-sharing approach is proposed. The spares can not only replace any faulty nodes in the same module, but also those in remote modules. Therefore, full utilization of spares can be achieved, ensuring that and the system is as reliable as the global

scheme. However, the proposed scheme entails less extra hardware cost. Compared to modular schemes, the proposed scheme achieves higher reliability for the same level of hardware cost. The organization of this paper is as follows. Section 2 presents the system architecture and the reconfiguration strategy. Section 3 analyzes the reliability and hardware cost. Concluding remarks are made in section 4.

2. A Module-Sharing Fault-Tolerant Architecture

This section exmines a global fault-tolerant architecture. By sharing spares with other remote modules, the scheme achieves full utilization of spares. It achieves global sharing. Furthermore, it may be applied to any interconnection topology.

2.1 The System Architecture

Figure 1 displats the generic global fault-tolerant architecture. It consists of N original nodes, K spare nodes, a topology block and a switching block. The topology block is the interconnection network of the desired multiprocessor system. This may be a hypercube [2, 9], a cube-connected cycle [10] or a tree [7], etc. The switching block connects the N available nodes of the N original and K spare nodes to the topology block to form the multiprocessor system. If the system identifies one or more faulty nodes, the switching block selects the suitable spare nodes to replace the faulty ones and maintains the solid topology. Each spare node can be employed to replace any of the faulty original N nodes. With the global approach, the switching block must realize all pairs between N and (N+K) entries, rendering cost of such implementation too high to be practical.

To reduce the cost, we propose a module-sharing scheme as follows. Let N, K and ρ be the number of original nodes, spare nodes and modules in a system, respectively. A module M_i, $1 \le i \le \rho$, contains N_i original nodes, K_i spare nodes, a selector, and a connector, where $N = \sum_{i=1}^{\rho} N_i$, $K = \sum_{i=1}^{\rho} K_i$. The selector and connector (constructed in the next subsection 2.2) are utilized to select active nodes (see Fig. 2). Each module may be of equal or different size. If every node has the same failure probability, we recommend constructing every module of equal size, thus rendering the system more reliable.

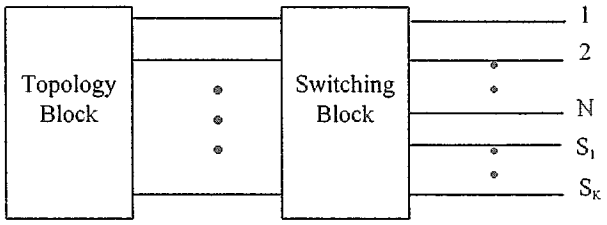


Fig. 1- The generic global fault-tolerant architecture.

The selector in module i connects the N_i available nodes of the N_i original and K_i local spare nodes in the same module and the $(K-K_i)$ remote spare nodes in the remote modules to the topology block. The selector first selects the required number of available spare nodes among what such nodes are available in the same module, or from among any remote spare nodes in any remote modules. The connector in module i connects itself to all other connectors in different modules. Thus, either an available local spare node can be located to replace the faulty node in the same or in a remote module, or an available remote spare node can be found to replace the faulty node in the local module. In our design, an available spare node in the same module has higher a priority than one in a remote module as a replacement candidates.

2.2 The Selector and Connector

In this subsection the module's selector and connector are constructed. Figure 3 depicts the configuration of a module, containing N_i original nodes with K_i spare nodes, a selector and a connector. The T-switch and B-switch of Figure 3 are separately illustrated in Figure 4. The selector has N_i T-switches. Each T-switch j has $(2K_i+2)$ entries one of which is connected to

the topology block, one is connected from the jth original node, K_i are connected from K_i spare nodes, and the remaining K_i are connected to K_i B-switches where K_i is the number of spare nodes in module M_i. Thus, the selector has N_i ports connecting to the topology block. In Figures 3 and 4, we simply depict the configuration if K_i=1. The connector consists of K_i (N_i-to-1) switches and K_i B-switches where N_i is the number of original nodes in module M_i. In the event of a node failing, some available local spare is directly requistioned, but where no local spare is available, a spare in a remote module will be accessed.

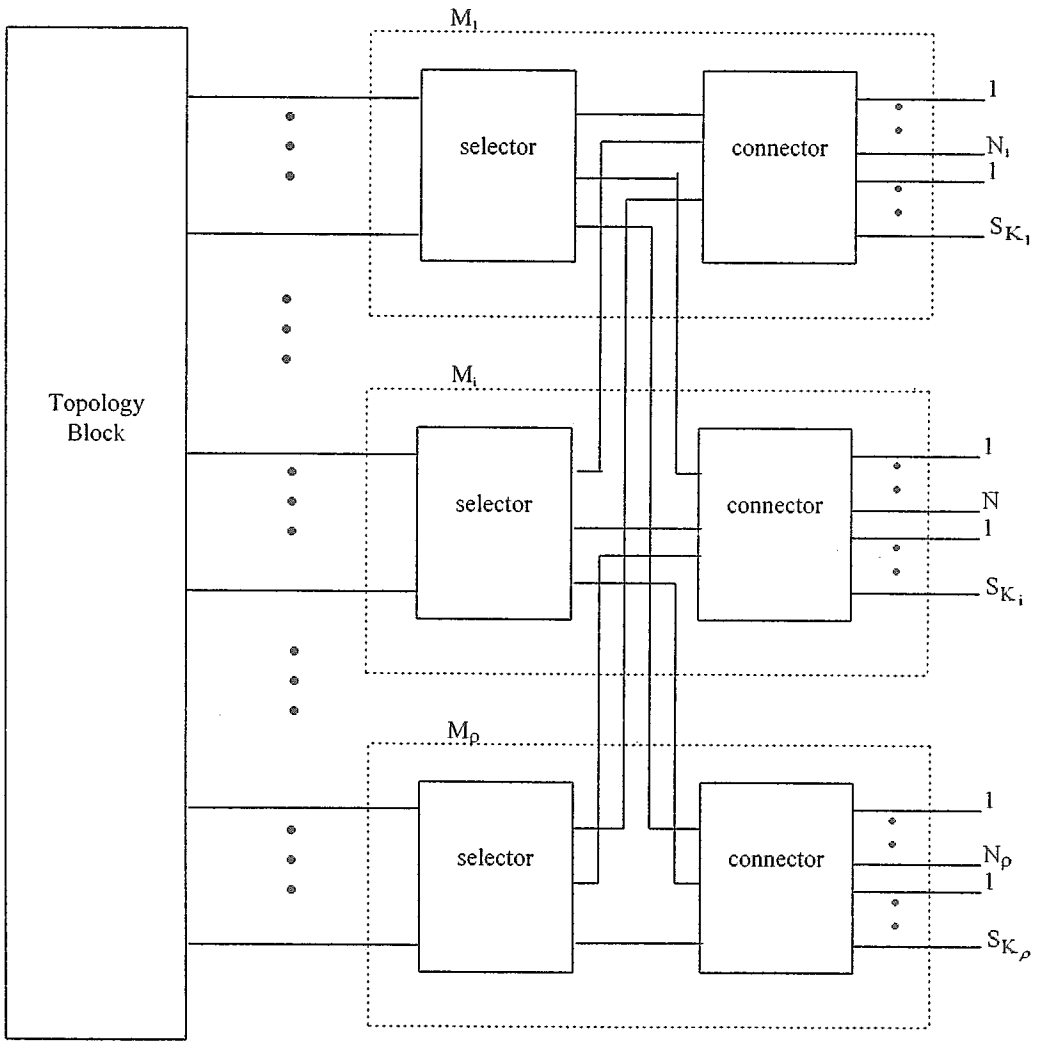


Fig. 2- The system architecture

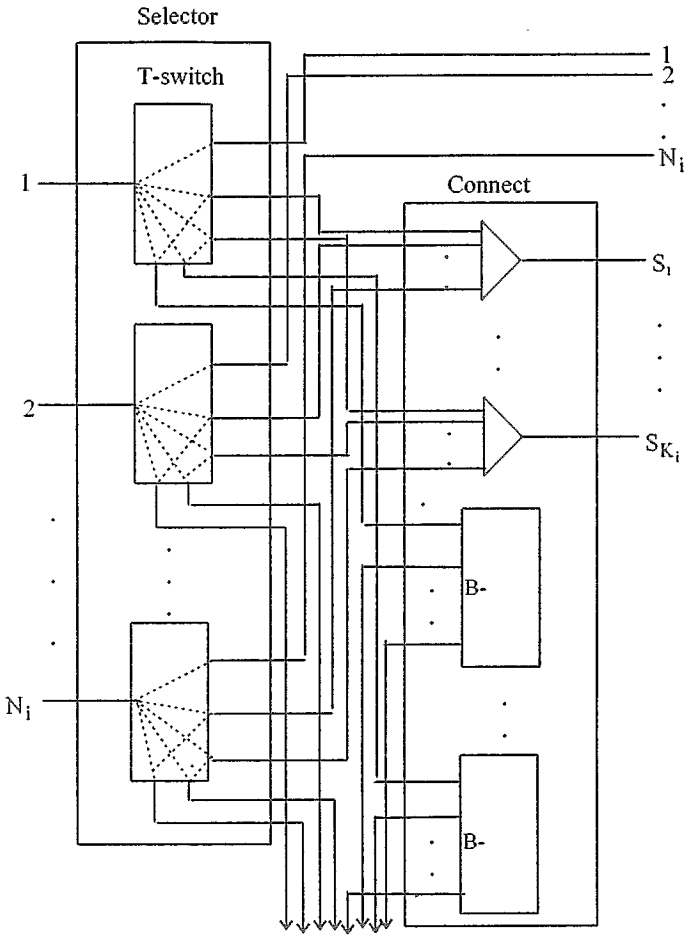
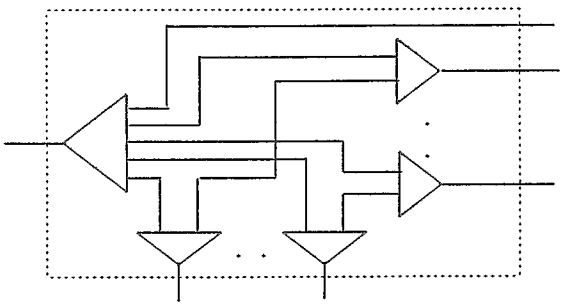
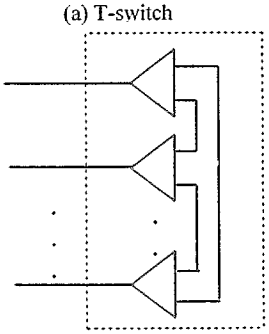


Fig. 3- The configuration of a module M_i.



(a) T-switch



(b) B-switch

Fig. 4- The configuration of a T-switch and B-switch.

The system is constructed by taking ρ modules. Figure 5 depicts the case of N=9, K=3 and ρ=3.

2.2 Reconfiguration Strategy

This subsection examines a reconfiguration strategy for dealing with node failures by allocating spares. Assuming a node fails, it loses both its computational and communication capabilities. Any original node may be either in the *Busy* state or *down* state. Any spare node during operation must be in one of the following states:

- *Ready*: The spare is idle and ready to replace a faulty node.
- *Local-replaced*: The spare is employed to replace a faulty node in the same module.
- *Remote-replaced*: The spare is employed to replace a faulty node in the remote module.
- *Down*: The spare has failed.

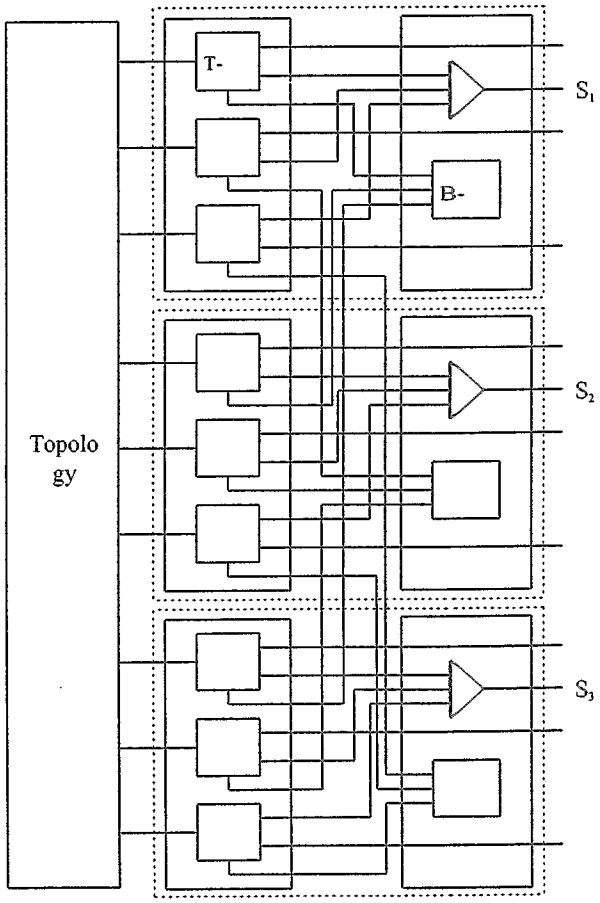


Fig. 5- The system configuration where N=9, K=3 and ρ=3.

Initially, any spare node is in the *Ready* state. When the system detects a failed node, an available spare is selected to replace it, designing the spare node's state as *Local-replaced* if both nodes are in the same module, or as *Remote-replaced* if they are both in distinct modules. A spare S_p in *Remote-replaced* state may be redesignated a *Local-replaced* state if no spares in the module are in *Ready* state and there exists an available spare S_q in the remote module in *Ready* state. The S_q may serve to replace the tasks of S_p and designates its state as *Remote-replaced* state. Then, S_p serves to replace the faulty node in the same module and designates its state as *Local-replaced.*

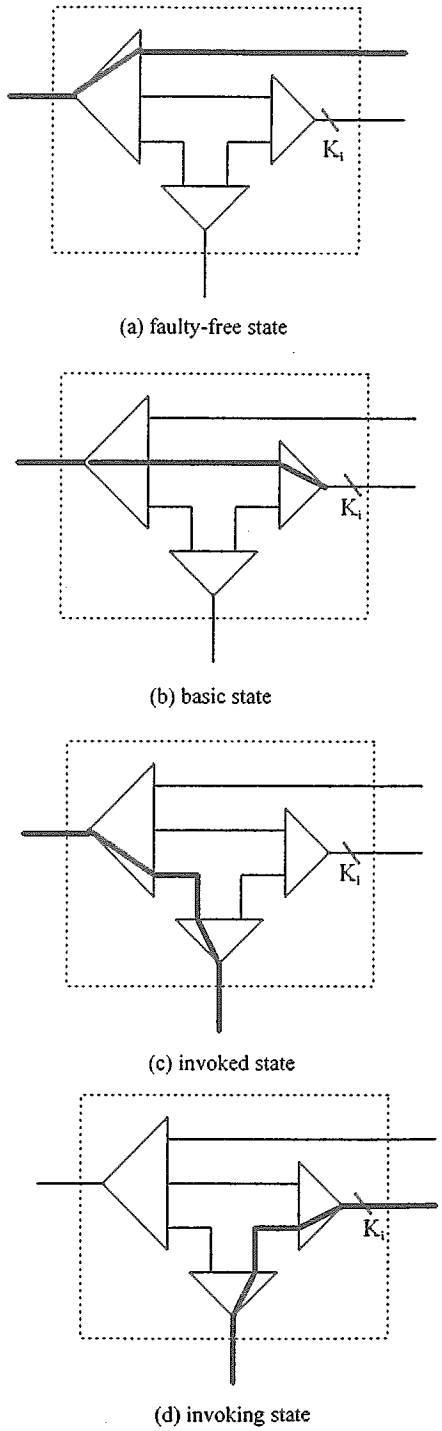
Any T-switch may be in any of the following states:

- *Faulty-free state*: If no faulty node exists in a module and all the spare in that module nodes are available (Figure 6(a)).
- *Basic state*: If some node has failed while the local spare is in *Ready* (Figure 6(b)).
- *Invoked state*: If some node has failed while the local spare is in *Busy* (Figure 6(c)).
- *Invoking state*: The local spare is supporting some remote module (Figure 6(d)).

Reconfiguration Algorithm:

This algorithm is employed to select an available spare node from a module to replace a faulty node. Let the faulty node i be in module M and the number of faulty nodes be not greater than K, then perform the following steps in order:

1. An available spare node in module M, exists, select it to replace i and designates its state as *Local-replaced.*
2. If a spare node S_p in M is in the *Remote-replaced* state, locate an available spare S_q from a remote module to inherit the tasks of S_p. Then, S_p replaces node i. Both S_p and S_q are designated as the *Local-replaced* and *Remote-replaced* states, respectively.
3. Find an available spare S_p from a remote module to replace node i and designate its state as *Remote-replaced.*



(a) faulty-free state

(b) basic state

(c) invoked state

(d) invoking state

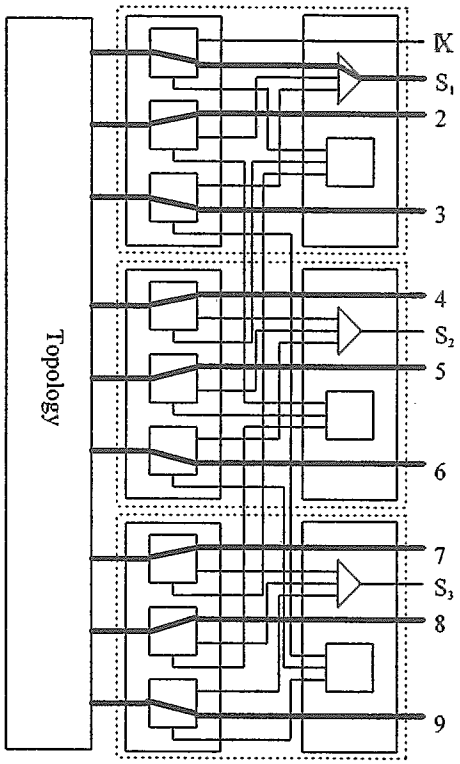
Fig. 6- The possible states of a T-switch.

The above reconfiguration procedure succeeds as long as the number of node failures is not greater than K. Some nodes

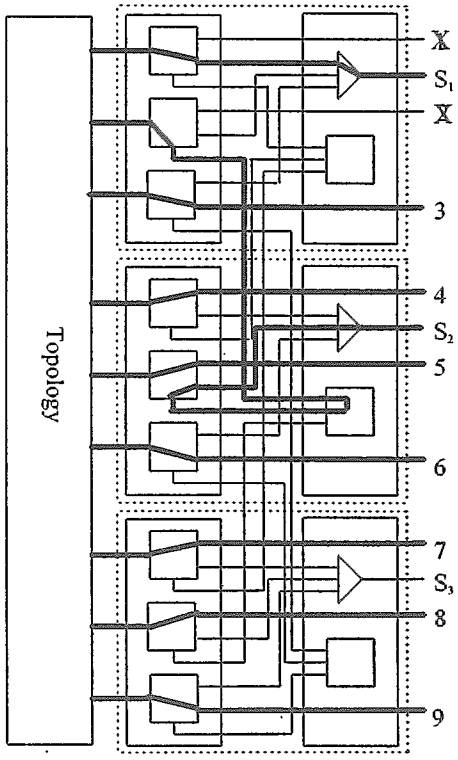
may be reassigned their tasks during reconfiguration, but the probability of requiring task reassignment is low. The probability depends on the distribution of the faulty nodes. Normally, if the number of faulty nodes in any module is not more than the spare nodes available, task reassignment is not required. However, even if such is not the case, reassignment, at most once per reconfiguration process. Thus, the proposed reconfiguration strategy is simple and rapid.

Although the implementation details have not been discussed, it is assumed thar nonfaulty nodes can recognize node failure. When such failure is identified, the reconfiguration can be performed within the module, i.e., decisions concerning which spare node to attach and which switches to change can be made locally.

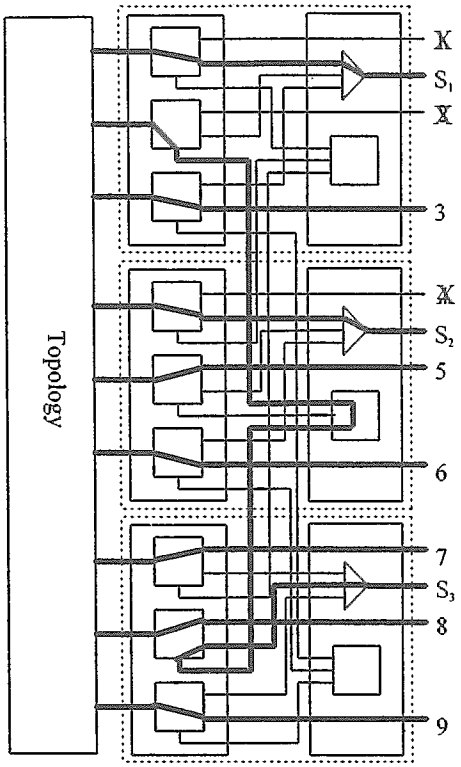
Consider for of the reconfiguration process for the configuration in Figure 7. If node 1 fails to be replaced by S_1, Figure 7(a) clearly indicates that node 1 has been removed, and that its tasks and address are inherited by node S_1. Note that only the T-switch is involved. If node 2 subsequently fails, any spare node in *ready* state can replace it (see Figure 7(b), let node S_2 replace node 2). Since the spare node, S_1, in the same module as node 2, is busy, it can invoke any other spare node in the remote module to replace the failed node 2 and in this case both T-switch and B-switch are involved. If node 4 becomes faulty, however, some nodes need their tasks reassigning so that node 4 may be replaced by some ready spare node. In this case, node 4 cannot be directly replaced by a spare node since the related T-switch of node S_2 has been set in *invoked* state. Fortunately, reconfiguration may be achieved simply by reassigning the tasks of certain nodes. Figure 7(c) illustrates node S_2 has been reassigned to replace node 4, and the original tasks of S_2 can be transferred to any other spare node (assuming that S_3 inherits the tasks).



(a) S_1 replaces node 1.



(b) S_1 and S_2 replace node 1 and node 2, respectively.



(c) S_1, S_2 and S_3 replace node 1, node 2 and node 4, respectively

Fig. 7- The reconfiguration process when node 1, 2, and 4 fail in order.

3. Reliability and Cost

Evaluating the reliability of systems need provided only on the assumption of node failure, since the probability of such failure is usually much higher than that of link and switch failures [6]. Chau and Liestman's formula [3] serves here to evaluate the system reliability of these types of fault-tolerant designs. The formula is as follows. Let c be the coverage factor, M the number of original nodes in a module, k the number of spare nodes in a module, r the reliability of a single node, $RM_{M,k}$ the reliability of a module with k spare nodes, and $RS_{N,K}$ the reliability of a system with K spare nodes. For global and module-sharing schemes,

$$RS_{N,K} = RS_{N,K-1} + \binom{N+K-1}{K} \cdot r^N \cdot (1-r)^K c^K \quad (1)$$

, where $RS_{N,0} = r^N$.

For modular schemes,

$$RM_{M,k} = RM_{M,k-1} + \binom{M+k-1}{k} \cdot r^M \cdot (1-r)^k c^k, \text{ where } RM_{M,0} = r^M \quad (2)$$

and

$$RS_{N,K} = (RM_{M,k})^\rho .$$

If r is evaluated by $e^{-\lambda t}$, where λ is the failure rate over time t, we illustrate the reliability of module-sharing and modular schemes versus time is illustrated in Fig. 8, where N=16, M=8, K=4, k=2, λ=0.1, ρ=2 and c=1. Typically, it is assumed that t=1 represents one million hours. The module-sharing scheme achieves higher reliability than the modular one at the same level of hardware cost (described in the following).

As depicted in Figure 8, the system contains N original nodes with K spare nodes, and each original or spare node has d data links incident to it. Therefore, the number of nodes and data links for the proposed scheme herein are $(N+K)$ and $d(N+K)$, respectively. For a fault-tolerant design that can achieve fully utilization of spares, the system reliability depends on the number of spare nodes, *i.e.*, the more there are of embedded spare nodes, the more reliable the system. The number of switches also increases relatively. Thus, switch complexity plays a key role in the measuring system. For simplicity, the switch complexity is calculated by the number of 2-to-1 switches. Specifically, each d-line, 1-to-Y or Y-to-1 switch may be constructed from d(Y-1) 2-to-1 switches, where Y is an integer greater one. Consider the proposed scheme illustrated in Figure 9. Clearly, the system can be divided into K modules containing N/K original nodes with one spare node each. Hence, the number of switches employed for the system is dK(N+3N/K-1). Comparing that number to the dK(2N-1) and dK(2N+K-1) 2-to-1 switches ultilized in the configurations suggested by Alam and Melhem [1], and by Chau and Liestman [3], respectively, it is clear that the proposed scheme requires fewer switches. Figure 9 depicts the comparison in detail.

4. Concluding Remarks

While a number of fault-tolerant designs for multiprocessors employ hardware

switches, the switch complexity may not be practical for many applications. In this paper, we have presented a module-sharing approach to design effective fault-tolerant systems. The main advantages of the proposed scheme are (1) modular design requires less hardware and (2) spare-sharing achieves high reliability. Task reassignment is performed during reconfiguration, at most one step, rendering reconfiguration a rapid and simple it.

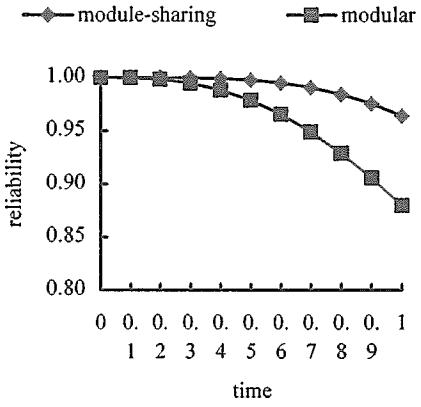


Fig. 8- The reliability curves of module-sharing and modular schemes.

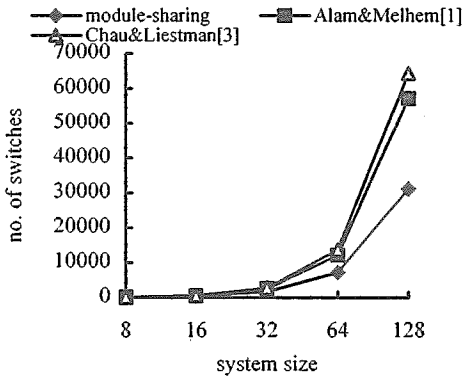


Fig. 9- the number of switches required for module-sharing, Alam & Melem's[1] and Chau & Liestman's[3] schemes where K=N/4.

References

1. Alam and R. G. Melhem, "An efficient modular spare allocation scheme and its application to fault tolerant binary hypercubes," *IEEE Tran. Parallel Distributed Syst.*, Vol. 2, No. 1, pp. 117-126, 1991.
2. Becker and H. U. Simon, "How robust is the n-cube?," *in Proc. 27th Symp. Foundations Computer Science*, pp. 283-291, 1986.
3. Chau and A. L. Liestman, "A proposal for a fault-tolerant binary hypercube architecture," *in Proc. 19th Int. Symp. Fault Tolerant Computing*, pp. 323-330, 1989.
4. Chen and S. J. Upadhyaya, "Reliability, reconfiguration, and spare allocation issues in binary-tree architectures based on multiple-level redundancy," *IEEE Tran. Comput.*, Vol. 42, No. 6, pp. 713-723, 1993.
5. Dutt and J. P. Hayes, "Designing fault-tolerant systems using automorphism," *J. Parallel and Distributed Comput.*, pp. 249-268, July 1991.
6. Dutt and J. P. Hayes, "Some practical issues in the design of fault-tolerant multiprocessors," *IEEE Tran. Comput.*, Vol. 41, No. 5, pp. 588-598, 1992.
7. M. Hassan and V. K. Agrawal, "A fault-tolerant modular architecture for binary trees," *IEEE Tran. Comput.*, Vol. C-35, No. 4, pp. 356-361, 1986.
8. Horowitz and A. Zorat, "The binary tree as an interconnection network: Applications to multiprocessor systems and VLSI," *IEEE Tran. Comput.*, No. 4, pp. 247-253, 1981.
9. Pease, "The indirect binary n-cube multiprocessor array," *IEEE Tran. Comput.*, Vol. C-26, pp. 458-473, 1977.
10. Preparata and J. Vuillemin, "The cube-connected cycles: A versatile network for parallel computations," *Commun. ACM*, Vol 24, No. 5, pp. 300-309, 1981.
11. Rennels, "On implementing fault-tolerance in binary hypercube," *in Proc. 15th Int. Symp. Fault Tolerant Computing*, pp. 344-349, 1985.
12. Yang, L. P. Zu and Y. N. Wu, "A reconfigurable modular fault-tolerant hypercube architecture," *IEEE Tran. Parallel Distributed Syst.*, Vol. 5, No. 10, pp. 1018-1032, 1994.