The Journal of Supercomputing, 27, 5–17, 2004

An Augmented *k*-ary Tree Multiprocessor with Real-Time Fault-Tolerant Capability

BABACK A. IZADI bai@engr.newpaltz.edu
Department of Electrical and Computer Engineering, State University of New York, 75 South Manheim Blvd., New Paltz, NY 12561

FÜSUN ÖZGÜNER ozguner@ee.eng.ohio-state.edu
Department of Electrical Engineering, Ohio State University, 2015 Neil Ave., Columbus, OH 43210-1277

Abstract. We present a real-time fault-tolerant design for an l-level k-ary tree multiprocessor and examine its reconfigurability. The k-ary tree is augmented by spare nodes and spare links. By utilizing the capabilities of wave-switching communication modules of the spare nodes, faulty nodes and faulty links can be tolerated. We consider two modes of operations. In the strict mode, the multiprocessor is under heavy computation or hard deadline and therefore we use a fast and local reconfiguration scheme to tolerate the faulty nodes. In the relaxed mode, where light computation or soft deadline is encountered, a global reconfiguration scheme is used to maximize the utilization of spare nodes, both in this mode as well as in the next strict mode. Both theoretical and simulation results are examined. Our simulation results, in the relaxed mode of operation, reveal that our approach can tolerate significantly more faulty nodes than other approaches, with a low overhead and no performance degradation.

Keywords: real time, fault tolerance, k-ary tree, augmented multiprocessor, reconfiguration, wave switching

1. Introduction

Today, researchers are using hundreds of thousands of processors to design multicomputers with petascale computing power [1]. Petascale multicomputers have been cited as having the capacity to solve a number of complex applications, including some with real-time implication. Some of these parallel computations are tree structured. Therefore, static and dynamic tree embedding into parallel machines based on popular topologies, such as the hypercube and the mesh, have been extensively investigated [2–4]. Alternatively, the tree has been suggested as a simple and effective topology to build such multicomputers [5–7]. A disadvantage of this topology is that a single faulty node or faulty link can disable the operation of the entire tree. Hence, fault-tolerant trees have been examined by a number of researchers where spare nodes and spare links are used to replace the faulty ones [8–13]. In a real-time environment, faulty components have to be replaced with spares in a manner that also satisfies the required completion deadline of active tasks. Two modes of operation is generally considered: the strict mode and the relaxed mode. The strict mode pertains to tasks whose computational requirements are heavy or have a hard completion deadline. The relaxed mode, on the other hand, consists of tasks with a soft completion deadline or a light computational load. Therefore, in the strict mode of operation, in order to allow for fast reconfiguration, spare replacement of faulty components should result in very few changes in the system interconnections. A common approach to accommodate this mode of operation is to replace each faulty component with the local spare using a distributed reconfiguration algorithm [14]. On the other hand, in the relaxed mode of operation, a global reconfiguration algorithm is applied to maximize the probability that in the next strict mode of operation, there exists a local spare for every faulty component.

In this paper, we present a two-stage redundant scheme for the k-ary tree. The objectives of the scheme are two fold. First, facilitate real-time fault tolerance by allowing the system to operate in either the strict mode or the relaxed mode. Second, utilize the spare network to tolerate a large number of faculty nodes and faulty links.

The rest of the paper is organized as follows. In the next section, notation and definitions that are used throughout the paper are given. An overview of our approach is presented in Section 3. In Section 4, we examine the reconfigurability of the scheme. Both theoretical and simulation results are presented. Finally, concluding remarks are discussed in Section 5.

2. Notation and definitions

Each node of an l-level k-ary tree is represented by $P(i,j)$ where i is the level number and j is the index of the node in the level i. Similarly, the spare node j in the level i is denoted by $S(i,j)$ at stage one and $SS(i,j)$ at stage two. The link connecting any two nodes P and Q is represented by $P \rightarrow Q$. Finally, we define the connection requirement (C_R) of a spare node α in a cluster with multiple faulty nodes, as the number of edge-disjoint paths that must be constructed within the spare network from α to other unassigned spare nodes, in order to tolerate the faulty nodes within the cluster.

3. Overview of the TECKT

In our scheme, at stage one, an enhanced cluster K-ary tree (ECKT) [8] with l levels is constructed by assigning one spare node to the regular nodes of each sub-tree with l_{c1} levels. Each of these sub-trees is called a cluster. Each spare node in a cluster is connected to every regular node of its cluster via an intra-cluster spare link. Furthermore, the spare nodes of neighboring clusters are interconnected using inter-cluster spare links; two clusters are declared neighbors if there exists at least one regular node in each cluster with a direct link between them. Figure 1 depicts a 6-level enhanced cluster 2-ary trees with $l_{c1} = 3$. In the figure, the regular links and the spare links are shown with heavy and light lines, respectively, and a cluster is highlighted with the dotted line.

At stage two, the process is repeated by assigning one spare node at stage two to each sub-tree of spare nodes of stage one with l_{c2} levels. We call the overall structure the two-stage enhanced cluster K-ary tree (TECKT). Hence, the spare nodes at stage one and stage two are interconnected as a $k_{s1} = k^{l_{c1}}$-ary tree with $l_{s1} = l/l_{c1}$ levels and

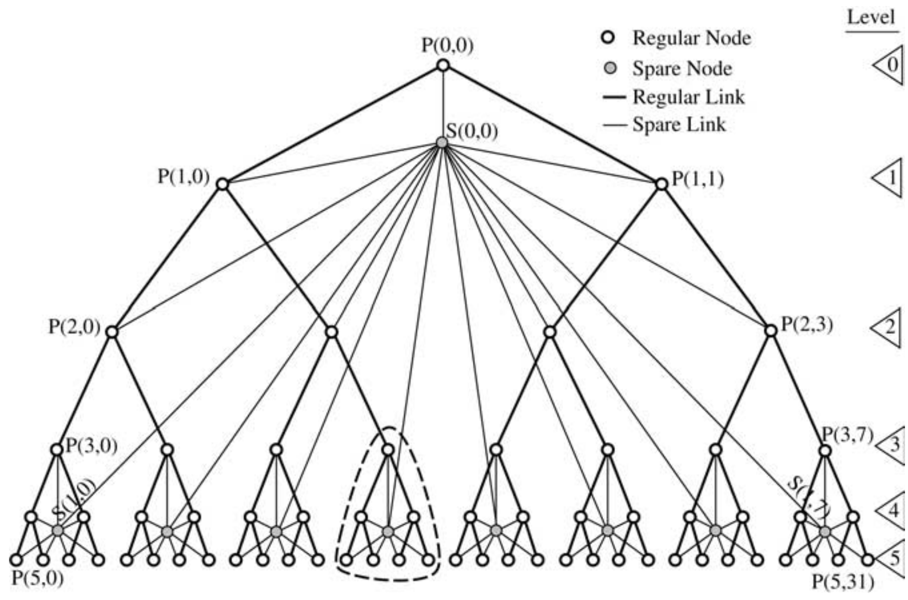

Figure 1. A 6-level enhanced cluster 2-ary tree.

a $k_{s2} = k^{(l_{c1} \times l_{c2})}$-ary tree with $l_{s2} = l/l_{c1} \times l_{c2}$ levels, respectively; and, the number of spare nodes at stage one and stage two are $(k^l - 1)/(k^{l_{c1}} - 1)$ and $(k^l - 1)/(k^{(l_{c1} \times l_{c2})} - 1)$, respectively. Figure 2 depicts a two-stage four-level two-ary enhanced cluster tree with $l_{c1} = l_{c2} = 2$. In the figure, the regular links are shown with heavy lines. The spare links at stage one and stage two are shown with light and dashed lines, respectively. A cluster at stage one in the figure is highlighted with the dashed triangle. Note that in Figure 2, the spare nodes at stage one are interconnected as a

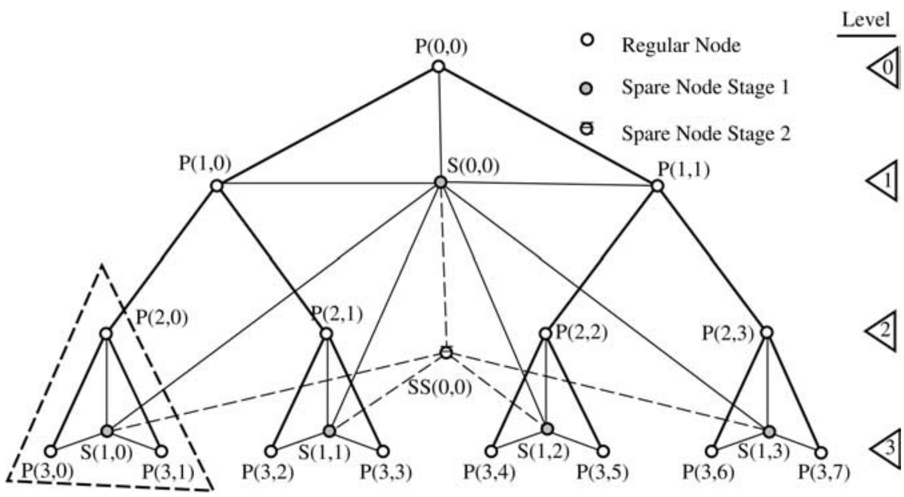

Figure 2. A two-stage enhanced cluster two-ary tree with $l = 4$ and $l_{c1} = l_{c2} = 2$.

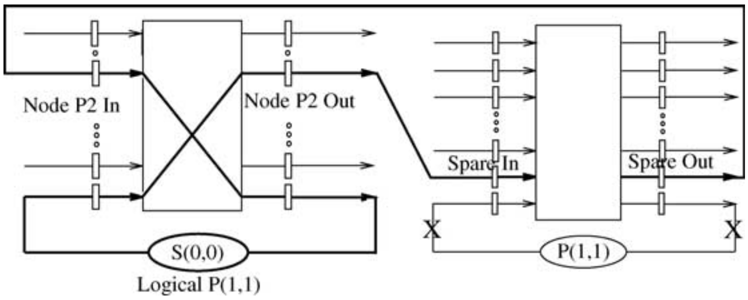

Figure 3. Spare node S(0, 0) replacing faulty node P(1, 1).

four-ary tree ($k^{l_{c1}} = 2^2$) with two levels ($l_{s1} = 4/2$). At stage two, since ($l_{s2} = 4/(2 \times 2) = 1$), there is only one spare node.

We next describe how a TECKT tolerates faulty nodes and faulty links. Each node is made of a computation module and a wave-switching communication module [15]. Wave-switching implements circuit-switching and wormhole-switching concurrently; permanent connections and long messages use the circuit-switched segment while short messages are transmitted using the wormhole-switching. We assume that faulty nodes retain their ability to communication. This is a common assumption since the hardware complexity of the communication module is much lower than the computational module. Therefore, the probability of failure in the communication module is much lower than the computation module. This assumption may be avoided by duplicating the communication module in each node.

To tolerate a faulty node, the computation module of the spare node logically replaces the computation module of the faulty node. In addition, if the spare node resides in the cluster of faulty node, the new communication module consists of the functional communication module of the faulty node merged with the appropriate routine channel of the local spare node. Figure 3 illustrates how local spare node S(0,0) replaces faulty node P(1, 1). The heavy lines in Figure 3 represent effective permanent connections after the reconfiguration; the permanent connections are established by utilizing the circuit-switched routing channels of the communication modules. If the assigned spare node and the faulty node belong to different clusters, a dedicated path is constructed by linking the appropriate circuit-switched routing channels of the intermediate spare nodes. For example, in Figure 2, if the node P(1, 1) is faulty and the spare S(0, 0) is not available, the spare S(1, 1) can replace it by linking the appropriate circuit-switched routing channels of P(1, 1), S(0, 0), and S(1, 1). The dedicated path then becomes an extension of the communication module of the faulty node and the spare node functionally replaces the faulty one. Furthermore, due to the capabilities of the circuit-switched routing modules, the physical location of the faulty node and its assigned spare node becomes irrelevant. The TECKT can tolerate both intra-cluster and inter-cluster link failures. This is accomplished by logically replacing the routing channel that connects the processor to a faulty link with the circuit-switched routing channel that connects it to a spare link, and hence utilizing the spare links to establish a parallel path to the faulty link and bypassing it. Figure 4 illustrates the reconfiguration of the TECKT in Figure 2

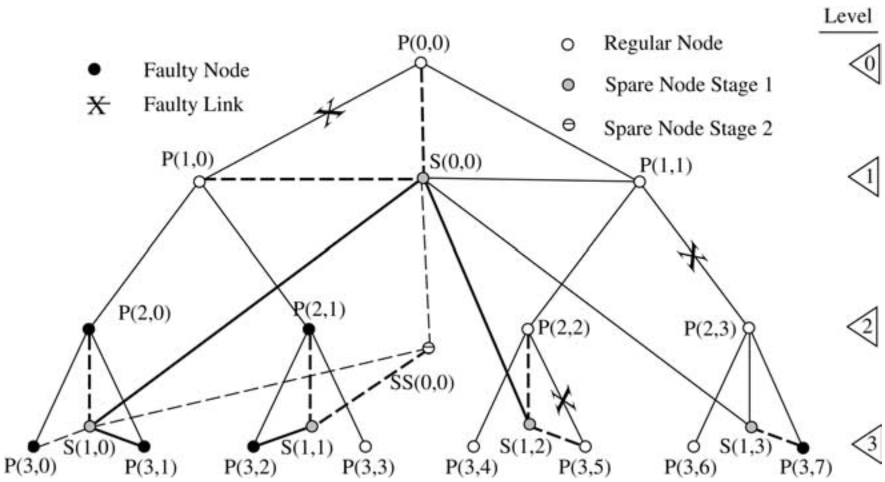

Figure 4. The TECKT in presence of faults.

in the presence of indicated faulty nodes and faulty links. For the sake of clarity, only active spare links are shown in the figure and bypass paths are drawn in a variety of line styles to distinguish them. As shown in the figure, spare nodes $S(0,0)$, $S(1,0)$, $S(1,1)$, $S(1,2)$, $S(1,3)$, and $SS(0,0)$ logically replace faulty nodes $P(3,0)$, $P(2,0)$, $P(3,2)$, $P(3,1)$, $P(3,7)$, and $P(2,1)$, respectively. Also, faulty links $P(1,0) \rightarrow P(0,0)$, $P(2,2) \rightarrow P(3,5)$, and $P(1,1) \rightarrow P(2,3)$ are bypassed using parallel paths $P(0,0) \rightarrow S(0,0) \rightarrow P(1,0)$, $P(2,2) \rightarrow S(1,2) \rightarrow P(3,5)$, and $P(2,3) \rightarrow S(1,3) \rightarrow S(0,0) \rightarrow P(1,1)$, respectively. Figure 5 illustrates how spare nodes $SS(0,0)$ and $S(1,1)$ replace faulty nodes $P(2,1)$ and $P(3,2)$, respectively, by merging their communication module. The heavy and dashed lines in Figure 5 pertain to similar lines in Figure 4 and represent effective permanent circuit-switched connections after the reconfiguration.

Considering only the faulty nodes in Figure 4 and examining the cluster associated with the spare node $S(1,0)$, two out of three faulty nodes of the cluster have to be assigned to available spare nodes from other fault-free clusters via edge-disjoint paths through the spare node $S(1,0)$. Therefore, the C_R (connection requirement) of the spare node $S(1,0)$ is said to be two.

4. Reconfigurability of the TECKT

To allow for fast reconfiguration in the strict mode of operation, the reconfiguration algorithm should result in minimum changes in system interconnections. Hence, under strict mode of operation, each faulty node of a cluster is replaced by the local spare node at stage one, as illustrated by the example in Figure 6. The algorithm is applied distributively, allowing each spare node at stage one to monitor the status of the regular nodes within its cluster, and replace the faculty node as outlined in the

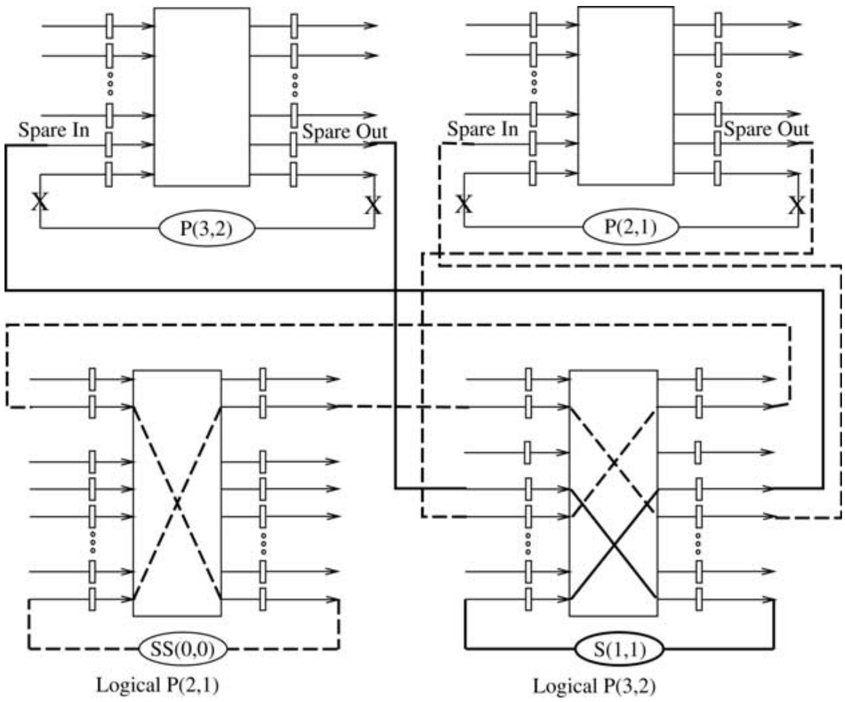

Figure 5. Replacing faulty nodes P(2, 1) and P(3, 2) with spare nodes SS(0, 0) and SS(1, 1).

previous section. The reconfiguration algorithm under the strict mode of operation fails if more than one node becomes faculty in a cluster.

The reconfiguration algorithm in the relaxed mode of operation assigns each detected faulty node to a spare node at stage two so that every healthy regular node

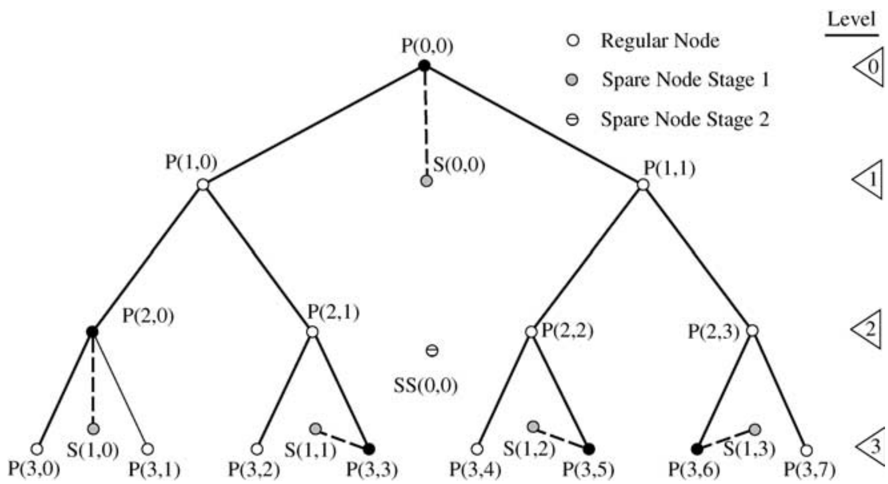

Figure 6. Reconfiguration of the TECKT in strict mode.

would have an available spare node at stage one in its local cluster for possible reconfiguration in the next strict mode of operation. For example, in Figure 2, in the relaxed mode of operation, the task of faulty node P$(3,3)$ is assigned to the spare node SS$(0,0)$ while in the strict mode of operation, it would be assigned to the local spare node, S$(1,1)$. Under relaxed mode of operation, if there is no unassigned spare node at stage two, a spare node from stage one is assigned to the faulty node; details of reconfiguration algorithm under the relaxed mode of operation is discussed later in this section. We next establish theoretical and simulation results pertaining to the relaxed mode of operation.

Let us group the spare nodes into three sets: S_S (set of source nodes), S_U (set of used nodes), and S_T (set of target nodes). A source node is a spare node at stage one in a cluster with multiple faulty nodes. The set S_S then represents the spare nodes with a C_R greater than 0. S_T is the set of unassigned spare nodes, and S_U consists of spare nodes that have been assigned to faulty nodes. For example, considering only the faulty nodes in Figure 4, after assigning faulty node P$(2,1)$ to spare node SS$(0,0)$ and the local spare node of each faulty cluster to a local faulty node, $\mathrm{S}_S = \{\mathrm{S}(1,0)\}$, $\mathrm{S}_U = \{\mathrm{S}(1,1), \mathrm{S}(1,3), \mathrm{SS}(0,0)\}$, and $\mathrm{S}_T = \{\mathrm{S}(0,0), \mathrm{S}(1,2)\}$. During the reconfiguration algorithm, the spare nodes are dynamically assigned to the various sets. To illustrate this, suppose the C_R of a spare node $\alpha \in \mathrm{S}_S$ is greater than 0, and there is a dedicated path from α to $\beta \in \mathrm{S}_T$. Consequently, β replaces a faulty node in the cluster of α via the dedicated path. β is then called used and is assigned to S_U. Also, the C_R of α is reduced by one. If the C_R of α becomes zero, it is also marked as used and is assigned to S_U. The TECKT is called reconfigured when S_S becomes an empty set.

From the previous section, it follows that the reconfigurability of the TECKT, under the relaxed mode of operation, is a function of the number of dedicated and edge-disjoint paths, within the spare trees at stages one and two, that can be established between the local spare nodes (nodes in S_S) of the clusters with multiple faulty nodes and the available spare nodes (nodes in S_T). Obviously, if the spare nodes are interconnected as a complete graph, the TECKT can tolerate up to $(((k_{s1})^{l_{s1}} - 1)/(k_{s1} - 1)) + (((k_{s2})^{l_{s2}} - 1)/(k_{s2} - 1))$ faulty nodes regardless of fault distribution. Hence, the reconfigurability of the TECKT is a direct consequence of the connectivity of the topology that interconnects the spare nodes. Let us represent the topology of the graph connecting the spare nodes by $G = (V, E)$, where $V = \mathrm{S}_S \bigcup \mathrm{S}_U \bigcup \mathrm{S}_T$ and E consists of the appropriate spare links. Let the C_R of a node $n \in \mathrm{S}_S$ be represented by $\mathrm{C}_R(n)$, and let us denote the sum of the C_Rs of all nodes in a set P as $\sum_{n\in P} \mathrm{C}_R(n)$. Since the number of faulty nodes cannot exceed the number of spare nodes, $|S_T| \geq \sum_{n\in \mathrm{S}_S} C_R(n)$. The following theorem examines the connectivity of G as it pertains to the reconfigurability of the TECKT.

Theorem 1 *Consider a graph $G(V, E)$, where $V = S_S \bigcup S_U \bigcup S_T$. The necessary and sufficient condition for every node $n \in S_S$ to have $C_R(n)$ edge-disjoint paths to $C_R(n)$ nodes in S_T is that the minimum number of edges leaving any subset of nodes $P \subseteq V$ be greater than or equal to $\sum_{n\in(P\bigcap S_S)} C_R(n) - |P \bigcap S_T|$.*

Proof: We first prove the necessary condition: if from every node $n \in \mathrm{S}_S$, there exists $\mathrm{C}_R(n)$ edge-disjoint paths to $\mathrm{C}_R(n)$ nodes in S_T, then the minimum number of edges leaving any subset of nodes $P \subseteq V$ must be greater than or equal to $\sum_{n \in (P \cap S_S)} C_R(n) - |P \cap S_T|$, which is the sum of the C_Rs of S_S nodes within P minus the number of S_T nodes in P. Let us consider a subset $P_1 \subseteq S_S$. Each of the edge-disjoint paths from a node in S_S to a node in S_T must be carried over at least one edge in the cut set $(P_1, V - P_1)$. Therefore, the sum of the C_Rs of the nodes in P_1, which represents the total required number of edge-disjoint paths from the nodes in P_1 to the nodes in S_T, must be smaller than or equal to the number of edges in the cut set $(P_1, V - P_1)$. Now, let us consider a subset $P \subseteq V$ and denote the graph interconnecting the nodes of P as g. Obviously, g is a subgraph of G. Within g, there exists only $|P \cap S_T|$ target nodes. Therefore, at most $|P \cap S_T|$ of the edge-disjoint paths may exist in g. The rest of the paths must then be carried over the cutset $(P, V - P)$. Therefore, the necessary condition follows.

We next prove the sufficient condition: if the minimum number of edges leaving any subset of nodes $P \subseteq V$ is greater than or equal to $\sum_{n \in (P \cap S_S)} C_R(n) - |P \cap S_T|$, every node $n \in S_S$ would have $C_R(n)$ edge-disjoint paths to $C_R(n)$ nodes in S_T. Let us create a new graph $G' = (V', E')$ by adding two nodes s and t to G as specified below and depicted by Figure 7. Each node in S_T is connected to t via a single edge. Each node $n \in S_S$ is connected to s via $C_R(n)$ parallel edges. Let the sum of the C_R of all nodes in S_S be L. The number of edge-disjoint paths between s and t in G', according to Menger's theorem [16], is equal to the size of the mincut in G'. We will show that there always exists an (s, t) mincut in G' whose size is equal to L. The mincut in G' may exist at s, t, G, or some combination of them. By construction, the size of the cut at s equals L. Similarly, the cut size at t is greater than or equal to L since $|S_T| \geq \sum_{n \in S_S} C_R(n) = L$. Per stated condition, for $P = s \cup S_S$ or $P = s \cup S_S \cup S_U$, the cut $(P, V' - P)$ must have a cutsize greater than or equal to L. Consider a general cut in G' crossing L_1 of the edges connecting s to S_S nodes, L_2 edges of G, and L_3 of the edges connecting S_T nodes to t (Figure 7). The number of S_T nodes on the unshaded side of the cut is L_3. The sum of the C_Rs of S_S nodes within the same side of the cut is $L - L_1$. Therefore, the stated condition can be formulated as $L_2 \geq (L - L_1) - L_3$ or $L_1 + L_2 + L_3 \geq L$. From this inequality, it follows that any cut in G'

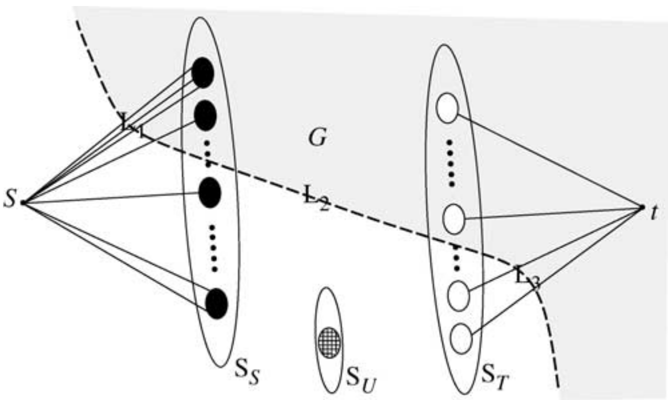

Figure 7. A cut in graph G'.

has a cutsize greater than or equal to L. Therefore, L is the size of the mincut. Hence, there exist L edge-disjoint paths between nodes s and t. Each of these s-t edge-disjoint paths must pass through a unique node in S_T because each node in S_T is connected to t via a single edge. Since there only exists L edges from s (one per path), the number of edge-disjoint paths from s that passes through each node $n \in S_S$ is equal to $C_R(n)$. Therefore, each node $n \in S_S$ can make $C_R(n)$ edge-disjoint paths to $C_R(n)$ distinct nodes in S_T. □

We next establish a necessary condition for the reconfigurability of the TECKT based on Theorem 1.

Theorem 2. *The necessary condition for the TECKT to reconfigure in the presence of faulty nodes is that the number of faulty nodes in each sub-tree with l_i levels ($l_i = j \times l_{c1} \times l_{c2}; j = 1, 2, \ldots$) be less than or equal to* $((k_{s1})^{(l_i/l_{c1})} - 1)/(k_{s1} - 1) + ((k_{s2})^{(l_i/(l_{c1} \times l_{c2}))} - 1)/(k_{s2} - 1) + 2$.

Proof: Let us examine the spare trees of the TECKT at stage one and stage two, starting at the leaf nodes and moving toward the root node. The number of spare nodes at stages one and two of a sub-tree of the TECKT with $l_i = j \times l_{c1} \times l_{c2}$ levels are $((k_{s1})^{(j \times l_{c2})} - 1)/(k_{s1} - 1)$ and $((k_{s2})^{j} - 1)/(k_{s2} - 1)$, respectively. Two such spare sub-trees with $k_{s1} = 2$, $k_{s2} = 4$, and $l_{c2} = 2$ are depicted in Figure 8 for $j = 1$ and $j = 2$, with dotted lines around the smaller and larger subsets, respectively. Since only two additional inter-cluster spare link exists that connect the sub-tree to external spare nodes, the maximum number of faulty nodes that can be tolerated in the sub-tree is $((k_{s1})^{(j \times l_{c2})} - 1)/((k_{s1} - 1)) + ((k_{s2})^{j} - 1)/((k_{s2} - 1)) + 2$. □

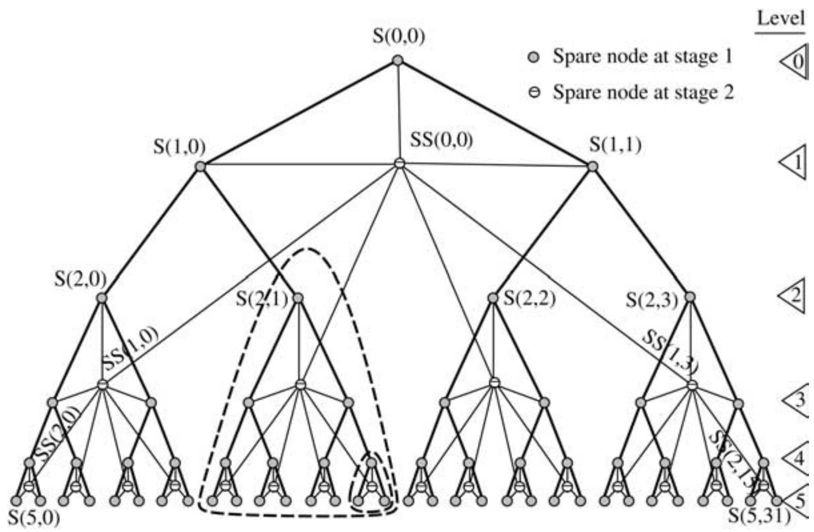

Figure 8. A spare tree at stages one and two.

Examining Figure 8, it is obvious that Theorem 1 is violated for a cluster with more than three faulty nodes; the C_R of the local spare node must be at most two since only two inter-cluster spare links are crossed. Other examples of the TECKT will yield similar results. Therefore, under a fixed number of faulty nodes per cluster, no theoretical lower bound on the number of tolerated faulty nodes per cluster can be established.

We next examine the simulation results based on the following reconfiguration algorithm. An optimal reconfiguration algorithm can be developed by utilizing the maxflow algorithm. Here, optimality is measured as the ability to assign a spare node to every faulty node whenever such an assignment is feasible *vis-à-vis* Theorem 1. The main drawback to a reconfiguration using the above algorithm is that a digraph representation of the spare network has to be constructed [17] and the spare node assignment has to be done by the host processor. To overcome these deficiencies, we next present a near-optimal reconfiguration algorithm, which is called Alloc-Spare-TECKT. The algorithm consists of four parts as specified below:

1. *Early abort:* The solvability test based on Theorem 2 is made to determine whether the reconfiguration is feasible.
2. *Assignment at stage two:* We utilize Lee's path-finding algorithm [18] to find a set of candidate spare nodes at stage two that can be assigned to the faulty node. The algorithm begins by constructing a breadth-first search of minimum depth $d \times (1 \le d \le 2(l_{s2} - 1))$ in the spare tree of stage two from the local spare node of a faulty cluster. If a free spare node is found, a path is formed to the faulty node. The algorithm guarantees that a path to a spare node will be found if one exits and the path will be the shortest possible [18]. Once a path is formed, the links associated with that path are deleted from the spare tree, resulting in a new structure. If there still remain some uncovered faulty nodes, a solvability test based on Theorem 2 is performed on the new structure, and this step is repeated for a higher depth d in stage two of the spare tree.
3. *Local assignment:* If all spare nodes at stage two are assigned and there still remain some faulty nodes, the local spare node of every faulty cluster is assigned to a faulty node within the cluster.
4. *Assignment at stage one:* If there remain additional faulty nodes, we apply Lee's path-finding algorithm to both stages one and two from the local spare node of a faulty cluster with an unassigned faulty node. Reconfiguration fails if $d > 2(l_{s1} + l_{s2} - 1)$, which is the longest acyclic path in the spare tree.

To illustrate our reconfiguration algorithm, let us consider Figure 9. The algorithm avoids the early abort of step one, since the total number of faulty nodes and spare nodes in the subtree with $l_i = j \times l_{c1} \times l_{c2} = 1 \times 2 \times 2 = 4$ levels are the same. During the second step, the algorithm randomly selects the faulty node $P(2,1)$. A breadth-first-search of depth one, at stage two, from the local spare node $S(1,1)$ yields the available spare node $SS(0,0)$. Hence, the spare node $SS(0,0)$ is assigned to the faulty node $P(2,1)$. Since no more unassigned spare node is available at stage two, the algorithm moves on to step three, where randomly faulty nodes $P(3,7)$, $P(3,2)$, and $P(2,0)$ are assigned to their local spare nodes $S(1,3)$, $S(1,1)$ and $S(1,0)$, respectively.

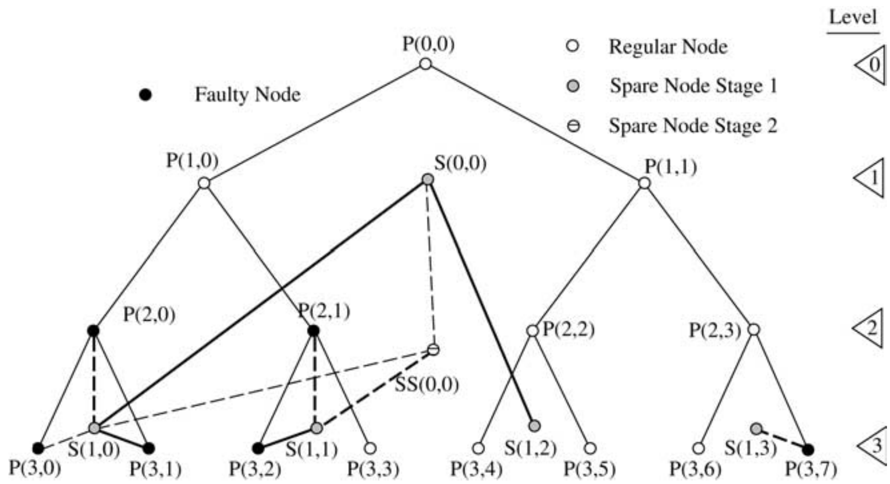

Figure 9. Reconfiguration of the TECKT in relaxed mode.

Finally, during step four, faulty nodes P$(3,0)$ and P$(3,1)$ are assigned to spare nodes S$(0,0)$ and S$(1,2)$ using a breadth-first- search of depth two from the local spare node S$(1,0)$.

We simulated the reconfigurability of a 9-level 3-ary tree with $l_{c1} = l_{c2} = 3$. The simulation results for the cluster approach [17] (a local reconfiguration scheme), the enhanced cluster approach [8], and the two-stage enhanced cluster approach are shown in Figure 10. The result suggests that the TECKT, under the relaxed mode of operation, can tolerate nearly 200 faulty nodes 90% of the time. Hence, the two-stage approach improves the reconfigurability of the given tree by almost a factor of two over the enhanced cluster approach and a factor of four over the local reconfiguration scheme. The approaches in Raghavendra et al. [11] and Lowrie and Fuchs [13] perform slightly worse and slightly better than the local reconfiguration scheme, respectively. Finally, the approach proposed in Kwan and Toida [10] only tolerates two faulty nodes and the scheme in Dutt and Hayes [12] tolerates a fixed number of faulty nodes at the expense of large node degrees.

In the presence of faulty links, no theoretical lower bounds on the number of tolerated faulty links can be established, since more than one faulty link sharing a regular node results in a failed reconfiguration. For example, in Figure 2, if links P$(1,0) \rightarrow$ P$(0,0)$ and P$(0,0) \rightarrow$ P$(1,1)$, are faulty, the spare link P$(0,0) \rightarrow$ S$(0,0)$ has to be used by both of them, which is not possible. Hence, only a simulation result may be attained. Also, since each intra-cluster link failure can only be replaced by one spare link parallel path, no distinction between the strict mode and relaxed mode can be made. The attained simulation results for the TECKT are similar to the ECKT [8]. The TECKT can tolerate a few more faulty links than the ECKT due to additional bypass paths at the second stage.

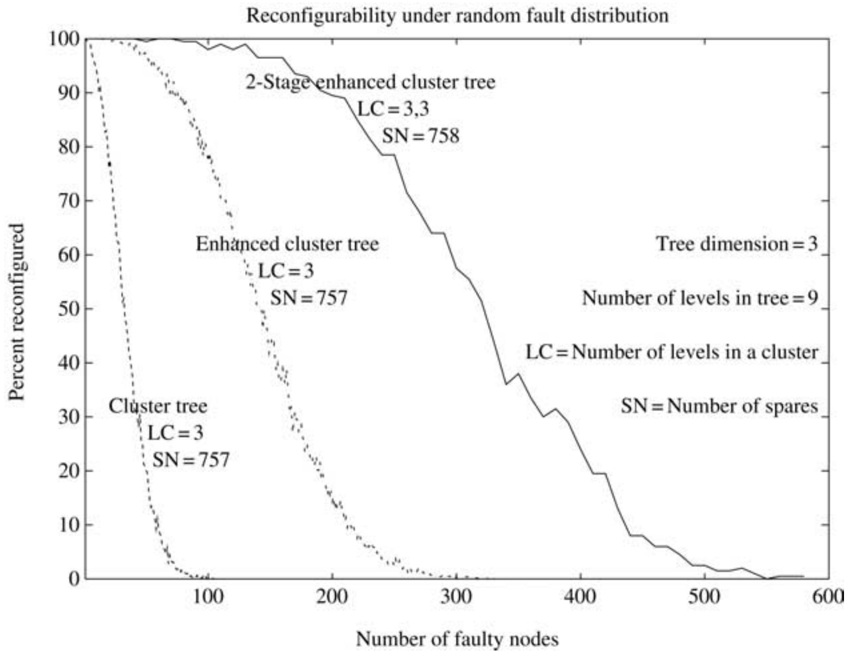

Figure 10. Reconfiguration of a tree under random fault distribution.

5. Conclusions

In this paper, we proposed a scheme to allow a tree-based multiprocessor tolerate faulty nodes in real time. During the strict mode of operation, the scheme uses local reconfiguration, which is the fastest and involves the fewest switch changes. Then, in the next relaxed mode of operation the tasks of local spare nodes, at stage one, are transferred to the spare nodes at the second stage by applying a global reconfiguration scheme. If a node becomes faulty during the relaxed mode of operation, the scheme tries to assign a spare node at stage two to replace it. This is done to maximize the probability that in the next strict mode of operation local spare nodes may be available to replace potential faulty nodes.

Our result indicates that no theoretical lower bound on the number of tolerated faulty nodes or faulty links can be established. However, our simulation results in the relaxed mode of operation and under random distribution of faulty nodes reveal that the TECKT can tolerate a relatively large number of faulty nodes. Compared to other proposed schemes, the TECKT can tolerate significantly more faulty nodes for the similar overhead.

References

1. J. Dongarra and D. Walker. The quest for petascale computing. *IEEE Computing in Science and Engineering*, 32–39, May 2001.

2. S. Bhatt, F. Chung, F. Leighton, and A. Rosenberg. Efficient embedding of trees in hypercubes. *SIAM Journal of Computing*, 21(1):151–162, 1992.
3. K. Li. Determining the expected load of dynamic tree embedding in hypercubes. *Proceedings of 17th International Conference on Distributed Computing Systems*, pp. 508–515, 1997.
4. S. Lee and H. Choi. Embedding of complete binary trees in meshes with row-column routing. *IEEE Transactions on Parallel and Distributed Systems*, 7(5):493–497, 1996.
5. C. E. Leiserson. The network architecture of the connection machine CM-5. In *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 272–285, June 1992.
6. Meiko World Incorporated. *Computing Surface 2 Reference Manuals*, Preliminary Edition, 1993.
7. H. L. Muller, P. W. Stallard, and D. H. Warren. An evaluation study of a link-based data diffusion machine. In *Proceedings of the 8th International Parallel Processing Symposium*, pp. 115–128, April 1994.
8. B. Izadi and F. Özgüner. Reconfigurable *k*-ary tree multiprocessors. *International Journal of Parallel and Distributed Systems and Networks*, 3(4):227–234, 2000.
9. J. P. Hayes. A graph model for fault-tolerant computing systems. *IEEE Transactions on Computers*, c-25:875–884, September 1976.
10. C. L. Kwan and S. Toida. An optimal 2-FT realization of binary symmetric hierarchical tree systems. *Networks*, 12(12):231–239, 1982.
11. C. Raghavendra, A. Avizienis, and M. D. Ercegovac. Fault tolerance in binary tree architectures. *IEEE Transactions on Computers*, c-33:568–572, June 1984.
12. S. Dutt and J. Hayes. On designing and reconfiguring *k*-fault-tolerant tree architectures. *IEEE Transactions on Computers*, 39:490–503, April 1990.
13. M. B. Lowrie and W. K. Fuchs. Reconfigurable tree architecture using subtree oriented fault tolerance. *IEEE Transactions on Computers*, c-36:1172–1182, October 1987.
14. R. Libeskind-Hadas, N. Shrivastava, R. Melhem, and C. Liu. Optimal reconfiguration algorithms for real-time fault-tolerant processor arrays. *IEEE Transactions on Parallel and Distributed Systems*, 6:498–510, May 1995.
15. J. Duato, P. Lopez, and S. Yalamanchili. Deadlock- and livelock-free routing protocols for wave switching. In *Proceedings of the 11th International Parallel Processing Symposium*, pp. 570–577, April 1997.
16. C. J. Colbourn. *The Combinatorics of Network Reliability*, Oxford University Press, 1987.
17. B. Izadi. Design of fault-tolerant distributed memory multiprocessors. Ph.D. thesis, the Ohio State University, 1995.
18. C. Y. Lee. An algorithm for path connection and its applications. *IRE Transactions on Electronic Computers*, ec-10:346–365, 1961.