

Reconfiguration in a Circuit-Switched k -ary Tree Multiprocessor

Baback A. Izadi

Electronic Engineering Technology Department
DeVry Institute of Technology
Columbus, OH 43209 U.S.A
bai@devrycols.edu

Füsün Özgüner

Department of Electrical Engineering
The Ohio State University
Columbus, OH 43210 U.S.A.
ozguner@ee.eng.ohio-state.edu

Abstract *In this paper, we present a strongly fault-tolerant design for the l -level k -ary tree multiprocessor and examine its reconfigurability. Our design assigns one spare node to the regular nodes of each subtree with l_c levels. Moreover, spare nodes are interconnected to form a spare tree. Our approach utilizes the circuit-switched capabilities of the communication modules of the spare nodes to tolerate a large number of faulty nodes and faulty links. Both theoretical and simulation results are examined. Compared with other proposed schemes, our approach can tolerate significantly more faulty nodes and faulty links with a low overhead and no performance degradation.*

Keywords: fault tolerant, k -ary trees, spare allocation, reconfiguration, augmented multiprocessors

1 Introduction

The tree is a useful topology for networks of hierarchical computing systems and has been used in the design of multicomputers [1]. One attractive feature of the tree-based architecture with n nodes is that any two nodes can exchange information in $O(\log n)$ time. A major disadvantage of this topology is that a single faulty node or faulty link can disable the operation of the entire tree. Hence, some form of fault tolerance is essential.

Fault-tolerant trees have been examined by a number of researchers where spare nodes and spare links are used to replace the faulty ones. An optimal 1-fault-tolerant k -ary tree with l -levels was proposed by Hayes [2]. The scheme for $k = 4$ and $l = 3$ is depicted in Figure 1. Upon a node

failure, using the spare nodes and the spare links (depicted by dashed lines in the figure), a k -ary tree with l -levels is maintained. In the literature, a reconfigurable system that retains the same service level, as well as keeping the same system topology after the occurrence of faults, is called a strongly fault-tolerant [3] system. Hence the reconfigured system can support the original task partition and it can use faster deterministic routers instead of slower adaptive ones. Figure 1 represents a strongly fault-tolerant design. Kwan and

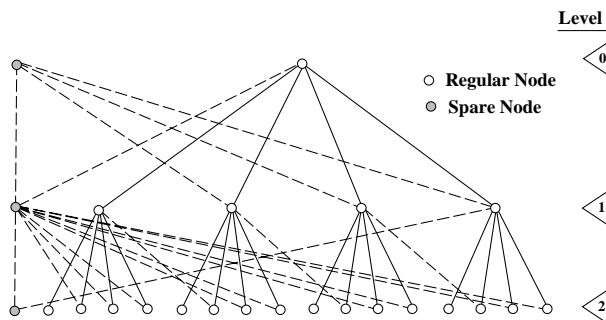


Figure 1: Optimal 1-fault-tolerant tree proposed by Hayes

Toida [4] extended the approach to design an optimal 2-fault-tolerant binary tree. To be able to tolerate more faulty nodes, other researchers [5, 6, 7] have proposed schemes based on a covering technique where a node u is said to cover a node v , if node u replaces node v when node v becomes faulty. Then, the node covering u replaces u and so on. The schemes require large number of spare links. To reduce the number of required spare links, some researchers [8, 9, 10, 11] have pro-

posed local reconfiguration schemes where a spare node is assigned to each set of nodes. One faulty node from each set is tolerated. A common drawback to the proposed schemes is the large number of spare links required and low utilization of spare nodes.

In this paper, we propose a global reconfiguration scheme that utilizes circuit-switched communication to make the k -ary tree strongly fault tolerant. In our scheme, one spare node is assigned to each group of regular nodes called a cluster (Figure 2); each spare node 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 with a direct link between them. We call the resulting topology the *Enhanced Cluster K -ary Tree* (ECKT). Our approach differs from others in the way we link the spare nodes to the faulty regular nodes: we utilize the circuit-switched capabilities of various spare nodes' communication modules to construct edge-disjoint paths between multiple faulty regular nodes of a cluster and multiple unassigned spare nodes in different clusters. We use this property to show that our approach has a significant higher fault-tolerance capability than other approaches. Faulty links are bypassed by establishing parallel paths using spare 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 ECKT. 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 $S(i, j)$. The link connecting any two nodes P and Q is represented by $P \rightarrow Q$. Finally, we define the *connec-*

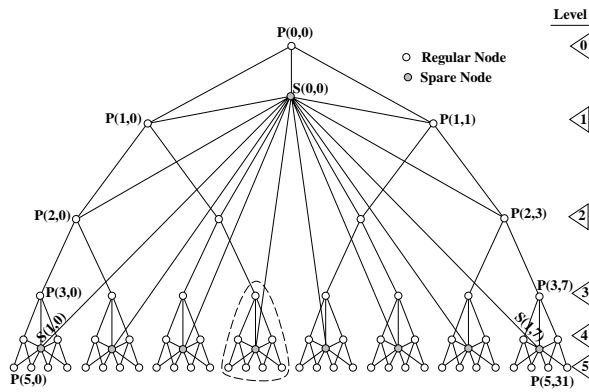
tion requirement (C_R) of a spare node α in a cluster with multiple faulty nodes, as the number of edge-disjoint paths that must be constructed within the spare network from α to other spare nodes in the fault-free clusters, in order to tolerate the faulty nodes.

3 Overview of the ECKT

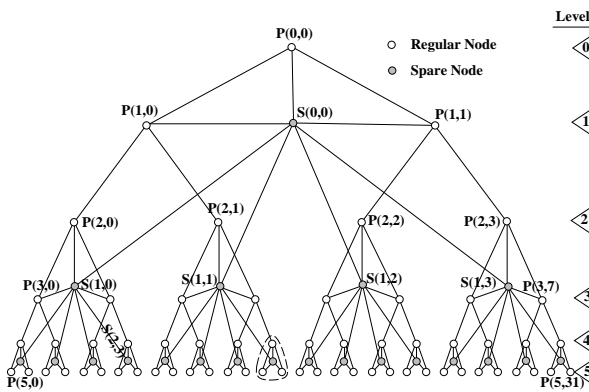
An ECKT with l levels is constructed by assigning one spare node to the regular nodes of each sub-tree with l_c levels. We call such a sub-tree a cluster. The number of regular nodes in one cluster is then $\frac{k^{l_c}-1}{k-1}$. The degree of a regular node at the root is $k+1$, at a leaf is 2 and elsewhere is $k+2$. The network interconnecting the spare nodes (the spare tree) is connected as a k_s -ary tree with l_s levels where $k_s = k^{l_c}$ and $l_s = \frac{l}{l_c}$. Hence, the number of spare nodes is $\frac{k_s^{l_s}-1}{k_s-1}$ and the number of spare links within the spare tree is $\frac{k_s^{(l_s-1)}-1}{k_s-1}k_s$. Figures 2(a) and 2(b) depict 6-level 2-ary enhanced cluster trees with $l_c = 3$ and $l_c = 2$ respectively. In the figure, the regular and the spare links are shown with heavy and light lines respectively. A cluster in Figure 2 is highlighted with the dotted line. Note that in Figure 2(a) the spare nodes are interconnected as an 8-ary tree ($k_s = 2^3$) with 2 levels ($l_s = \frac{6}{3}$) and in Figure 2(b) as a 4-ary tree with 3 levels.

We next describe how an ECKT tolerates faulty nodes and faulty links. We assume that a given faulty node retains its communication capability. This is a common assumption since in today's distributed memory multiprocessors, the computation and the communication modules of a node are separate. Furthermore, since the complexity of the computation module is much greater than that of the communication one, the probability of a failure in the computation module is much higher. This assumption may be avoided by duplicating the communication module in each node.

To facilitate reconfiguration, the routing channels described below are used at each node. The block diagrams of a regular node router and a spare node router for an ECKT are depicted in Figure 3. Each node consists of a computation module and a communication module. The communication module is made of several routing channels, and



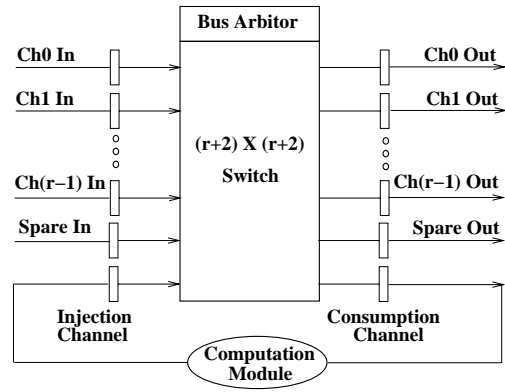
(a)



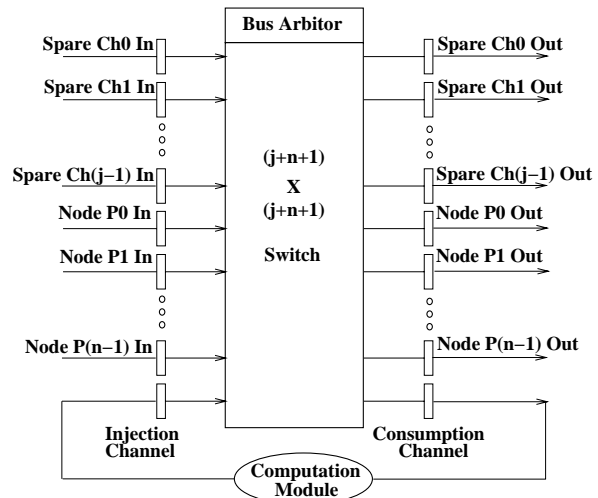
(b)

Figure 2: A 6-level enhanced cluster 2-ary tree with a) $l_c = 2$ b) $l_c = 3$

each routing channel consists of a *channel in* and a *channel out*. For example the routing channel 0 of the regular node depicted in Figure 3(a) is made of Ch0 In and Ch0 Out. Each regular node router consists of $r + 1$ routing channels allowing it to connect to one of its r neighboring regular nodes as well as its designated spare node. The connection to a neighboring regular node is provided via Ch X In and Ch X Out channels, where $X = 0$ for the parent node, $X = 1$ for the leftmost child node, $X = 2$ for the next child node to the right and so on. The connection to the designated spare node is facilitated using Spare In and Spare Out channels. Figure 3(b) illustrates the communication module of a spare node with n regular nodes in its designated cluster and j neighboring spare



(a)



(b)

Figure 3: Block diagram of a) Regular node b) Spare node

nodes. The node routing channel P X (Node P X In and Node P X Out) allows the spare node to be connected to the regular node X within its cluster. Similarly, connection to another neighboring spare node is provided using the appropriate Spare Ch X In and Spare Ch X Out.

Connecting a spare node to a regular node is done to tolerate a node failure. If the spare node resides in the cluster of the faulty node, the spare routing channel (Spare In and Spare Out) of the faulty node is connected to the appropriate node routing channel of the spare node. For example in Figure 2(a), if the node P(1, 1) becomes faulty,

the spare node $S(0,0)$ replaces it as illustrated in Figure 4. The heavy lines in the figure represent the permanent connections made after the reconfiguration. Hence, the computation module of the spare node replaces the computation module of the

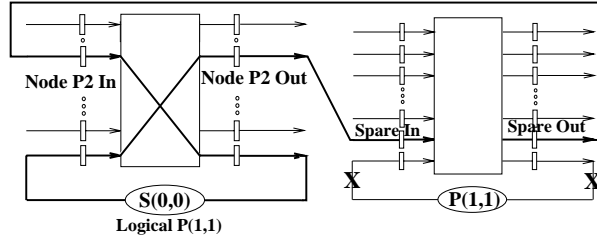


Figure 4: Spare node $S(0,0)$ replacing faulty node $P(1,1)$

faulty node. In addition, the new communication module consists of the functional communication module of the faulty node merged with the appropriate routine channel of the spare node. Due to the circuit-switched capabilities of the communication modules, the cost of communication is nearly constant between any two given nodes. Therefore, replacing $P(1,1)$ with $S(0,0)$ has negligible effect in the performance of the system.

If the assigned spare node and the faulty node belong to different clusters, a dedicated path to connect them needs to be established. Such a path can be constructed by linking the appropriate routing channels of the intermediate spare nodes. For example in Figure 2(a), if the node $P(2,1)$ is faulty and the spare $S(0,0)$ is not available, the spare $S(1,1)$ can replace it by linking the appropriate spare routing channels of $S(0,0)$ and $S(1,1)$. Moreover, the proper node routing channel of $S(0,0)$ is linked with the spare routing channel of $P(2,1)$. 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.

There are three cases which may involve routing data through a faulty node. The first one is when the message originates from the faulty node. Here, its spare replacement sends its data via its Injection Channel (Figure 3(b)) out to the appro-

priate Node PX Out channel. The message is then routed to the *communication module* (CM) of the faulty node via Spare In. Depending on the final destination node, any of the channel outs may be selected. The second case is when the destination of the message is the faulty node. Once the data reaches the CM of the faulty node, it is automatically routed to the channel Spare Out instead of the Consumption Channel. The spare node's CM would consequently receive the message using the appropriate channel (Node PX In). The message is then sent to the computation module of the spare node via the Consumption Channel. The third case is when the faulty node is neither the source nor the destination of the message but is used merely as an intermediate switch connection. In this case, the spare node is not involved at all, and routing is done normally. Therefore, each spare node has dual functions. One is to be the logical replacement for a faulty node. The other is to be an intermediate switch connection. Both functions may be active at the same time. Note that the connection of the spare node router to the active node(s) is established during the reconfiguration and remains intact thereafter.

Upon detecting a node failure, the spare node within the respective cluster logically replaces the faulty node. If the local spare node is not available, an available spare node from a different cluster may be used. Every spare node consequently activates its links to the active channels and disables the rest. The spare node's CM then forwards messages bound for other spare nodes as well as forwarding/receiving its data to/from other regular nodes via the CM of the faulty node as discussed above. Therefore, no modification to the available computation or communication algorithm is needed.

The enhanced cluster approach can tolerate both intra-cluster and inter-cluster link failures. This is accomplished by utilizing the spare links to establish a parallel path to the faulty link. To tolerate an intra-cluster link failure, using the spare links and through the local spare node, a parallel path to the faulty link is established. For example, in Figure 2(a), if the link $P(3,1) \rightarrow P(4,2)$ is faulty, it can be bypassed by the path $P(3,1) \rightarrow S(1,1) \rightarrow P(4,2)$ as specified below. The CM of $S(1,1)$ is

configured so that the node routing channel 0 and the node routing channel 1 are interconnected. In addition, regular nodes $P(3, 1)$ and $P(4, 2)$ logically replace their routing channels, which connect them to the faulty link, with their spare routing channels. Consequently, all messages that are bound for the faulty link are sent through the path $P(3, 1) \rightarrow S(1, 1) \rightarrow P(4, 2)$. To tolerate an inter-cluster link failure, more than one spare node CM is utilized. For example, in Figure 2(a), the path $P(2, 0) \rightarrow S(0, 0) \rightarrow S(1, 0) \rightarrow P(3, 0)$ can bypass the inter-cluster faulty link $P(2, 0) \rightarrow P(3, 0)$ as specified below. The node $P(3, 0)$ is connected to the spare node $S(1, 0)$ by linking $P(3, 0)$'s Spare In and Spare Out to $S(1, 0)$'s Node P0 Out and Node P0 In respectively. Similarly, the spare routing channel of the node $P(2, 0)$ is connected to the node P1 routing channel of the spare node $S(1, 0)$. Finally, the spare nodes $S(0, 0)$ and $S(1, 0)$ are interconnected by linking their appropriate "spare channel out"s and "spare channel in"s. Reconfiguration fails if either of the two nodes at the end of a faulty link is also faulty because the pertinent spare link has to be used to tolerate both the faulty node and the faulty link. The reconfiguration also fails if two faulty links have a common regular node, since the spare link that connects to the shared regular node has to be used in more than one dedicated parallel path.

Example: Figure 5 illustrates the reconfiguration of the ECKT in Figure 2(a) in the presence of indicated faulty nodes and faulty links. For the sake of clarity, non-active spare links are deleted from Figure 5. As shown in the figure, spare nodes $S(0, 0)$, $S(1, 1)$, $S(1, 6)$, $S(1, 7)$, $S(1, 3)$, $S(1, 0)$, $S(1, 2)$, $S(1, 4)$ and $S(1, 5)$ logically replace $P(1, 1)$, $P(2, 1)$, $P(2, 3)$, $P(3, 7)$, $P(4, 8)$, $P(5, 2)$, $P(5, 9)$, $P(5, 19)$ and $P(5, 20)$ respectively. Also, intra-cluster faulty links $P(3, 1) \rightarrow P(4, 2)$, $P(4, 4) \rightarrow P(5, 8)$, $P(3, 3) \rightarrow P(4, 7)$ and inter-cluster faulty link $P(2, 0) \rightarrow P(3, 0)$ are bypassed using parallel paths $P(3, 1) \rightarrow S(1, 1) \rightarrow P(4, 2)$, $P(4, 4) \rightarrow S(1, 2) \rightarrow P(5, 8)$, $P(3, 3) \rightarrow S(1, 3) \rightarrow P(4, 7)$ and $P(2, 0) \rightarrow S(0, 0) \rightarrow S(1, 0) \rightarrow P(3, 0)$ respectively.

Considering only the faulty nodes in Figure 5 and examining the cluster associated with the spare $S(0, 0)$, two out of three faulty nodes of the cluster

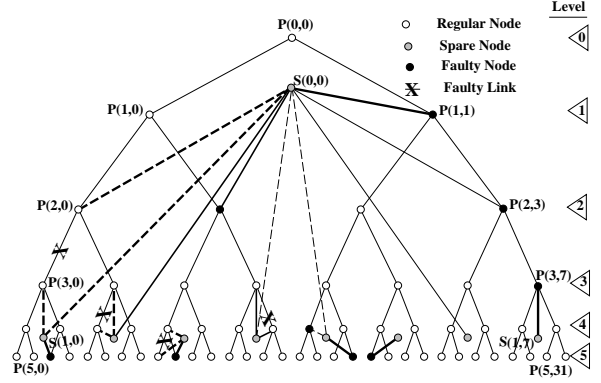


Figure 5: The ECKT in presence of faults

have to be assigned to available spare nodes from other fault-free clusters via edge-disjoint paths through the spare $S(0, 0)$. Therefore, the C_R (connection requirement) of the spare $S(0, 0)$ is said to be 2. Note that, in general, the C_R of a spare node is equal to the number of faulty nodes that reside within its cluster minus one.

4 Reconfigurability of the ECKT

We first examine reconfiguration in the presence of only faulty nodes. Let's 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 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 and have a C_R of 0. For example, considering only the faulty nodes in Figure 5, after assigning the local spare node to a local faulty node in each faulty cluster, $S_S = \{S(0, 0)\}$, $S_U = \{S(1, 0), S(1, 2), S(1, 4), S(1, 5), S(1, 7)\}$, and $S_T = \{S(1, 1), S(1, 3), S(1, 6)\}$. During the reconfiguration algorithm, which is discussed later in this section, the spare nodes are dynamically assigned to the various sets. To illustrate this, suppose the C_R of a spare node $\alpha \in S_S$ is greater than 0 and there is a dedicated path from α to $\beta \in S_T$. 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 ECKT is called reconfigured when S_S becomes an empty set.

From the previous section, it follows that the reconfigurability of the ECKT is a function of the number of dedicated and edge-disjoint paths, within the spare tree, 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) of the fault-free clusters. However, spare nodes do not have to be interconnected as a tree. Obviously, if the spare nodes are interconnected as a complete graph, the ECKT can tolerate up to $\frac{k_s^{l_s}-1}{k_s-1}$ faulty nodes regardless of fault distribution. Hence, the reconfigurability of the ECKT is a direct consequence of the connectivity of the topology which interconnects the spare nodes. Let's represent the topology of the graph connecting the spare nodes by $G = (V, E)$, where $V = S_S \cup S_U \cup S_T$ and E consists of the appropriate spare links. Let the C_R of a node $n \in S_S$ be represented by $C_R(n)$ and denote the sum of the C_R 's of all nodes in a set P as $\sum_{n \in P} C_R(n)$. Since the number of faulty nodes can't exceed the number of spare nodes, $|S_T| \geq \sum_{n \in S_S} C_R(n)$. The following theorem examines the connectivity of G as it pertains to the reconfigurability of the ECKT.

Theorem 1 Consider a graph $G(V, E)$, where $V = S_S \cup S_U \cup S_T$. 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 $\sum_{n \in (P \cap S_S)} C_R(n) - |P \cap S_T|$.

Proof: We first prove the necessary condition: if from every node $n \in S_S$, there exists $C_R(n)$ edge-disjoint paths to $C_R(n)$ 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_R 's of S_S nodes within P minus the number of S_T nodes in P . Let's 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 cutset $(P_1, V - P_1)$. Therefore, the sum

of the C_R 's 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 cutset $(P_1, V - P_1)$. Now, let's 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's create a new graph $G' = (V', E')$ by adding two nodes s and t to G as specified below and depicted by Figure 6. 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 [12], 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 cutsize at t is greater than or equal to L , since $|S_T| \geq \sum_{n \in S_S} C_R(n) = L$. Per the 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

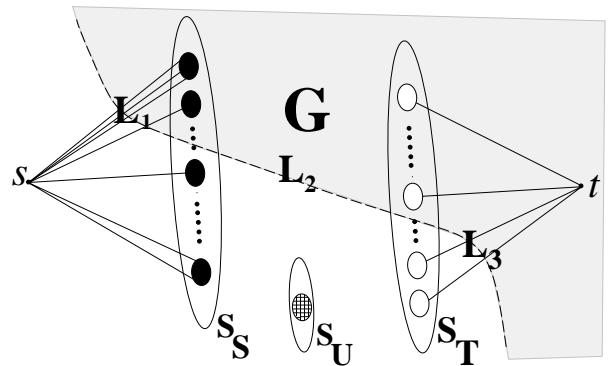


Figure 6: A cut in graph G'

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 6). The number of S_T nodes on the unshaded side of the cut is L_3 . The sum of the C_R 's 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 has a cutsize greater than or equal to L . Therefore, L is the size of the mincut. Hence, L edge-disjoint paths exist between nodes s and t . Then, each of the 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 exist 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 apply Theorem 1 to the spare tree and examine the reconfigurability of the ECKT.

Theorem 2 *The necessary condition for the ECKT to reconfigure in the presence of faulty nodes is that the number of faulty nodes in each sub-tree with l_i levels ($l_i = j \times l_c; j = 1, 2, \dots$) be less than or equal to $\frac{k_s^{l_i} - 1}{k_s - 1} + 1$.*

Proof: Let's examine the ECKT by starting at the leaf nodes and moving toward the root node. The number of spare nodes of a sub-tree with $l_i = j \times l_c$ levels is $\frac{k_s^j - 1}{k_s - 1}$; two such sub-trees with $l_i = l_c$ and $l_i = 2l_c$ are depicted in Figure 7, with dotted lines around the smaller and larger subsets respectively. Since only one additional inter-cluster spare link exists that connects the sub-tree to an external spare node, the maximum number of faulty nodes that can be tolerated in the sub-tree is $\frac{k_s^j - 1}{k_s - 1} + 1$. ■

Examining the cluster shown by the dotted line in Figure 7, it is obvious that Theorem 1 is violated for a cluster with more than two faulty nodes; the C_R of the local spare node must be at most 1 since only one inter-cluster spare link is crossed. Other examples of the ECKT will yield similar results. Therefore, under a fixed number of faulty nodes per cluster, no theoretical lower bound on

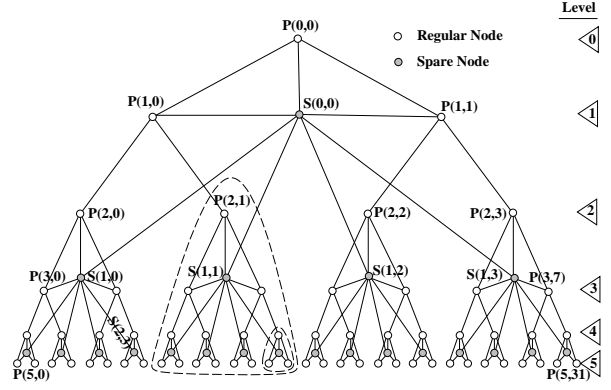


Figure 7: An ECKT with $k = 2$, $l = 6$ and $l_c = 3$

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/mincut algorithm. Here, optimality is measured as the ability to assign a spare node to every faulty node whenever such an assignment is feasible vis-a-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 [11] 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-ECKT*. The algorithm consists of three parts as specified below:

1. Early Abort: In this part, the solvability test based on Theorem 2 is made to determine whether reconfiguration is feasible.

2. Local Assignment: In this step, the local spare node of every faulty cluster is assigned to a faulty node within the cluster. If all faulty nodes are covered, the ECKT is reconfigured.

3. Non-Local Assignment: To find a set of candidate spare nodes that can be assigned to a faulty node, we utilize Lee's path-finding algorithm [13]. The algorithm begins by constructing a breadth-first search of minimum depth d ($1 \leq d \leq 2(l_s - 1)$) in the spare tree from the local spare node of a faulty cluster with a non-zero C_R . If a free spare node is found, a path is formed to the source node. The algorithm guarantees that a path to a spare node will be found if one exists and the path

will be the shortest possible [13]. Therefore, all faulty nodes that are one link away from available spare nodes (at depth 1) are assigned first. 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 remains some uncovered faulty nodes, a solvability test to check the C_R of neighboring spare nodes, similar to Early Abort, is performed on the new structure and Step 3 is repeated for a higher depth d . Reconfiguration fails if $d > 2(l_s - 1)$, which is the longest acyclic path in the spare tree. ■

Alloc-Spare may be applied distributively. Step 1 can be performed by having each spare node check its own node degree as dictated by Theorem 2 and broadcast that information along with its C_R to its neighboring spare nodes. Each spare node consequently checks for solvability based on Theorem 1. Step 2 is done by the local spare node of each faulty cluster. For Step 3, each spare node with a non-zero C_R makes a breadth-first search within the spare tree to locate the unassigned spare node(s).

We implemented algorithm Alloc-Spare-ECKT for an enhanced cluster 3-ary tree with $l = 9$ and $l_c = 3$. Hence, 757 spare nodes are interconnected as a 27-ary spare tree with $l_s = 3$. The simulation result for up to 400 randomly placed faulty nodes is shown in Figure 8. 1000 simulation runs were

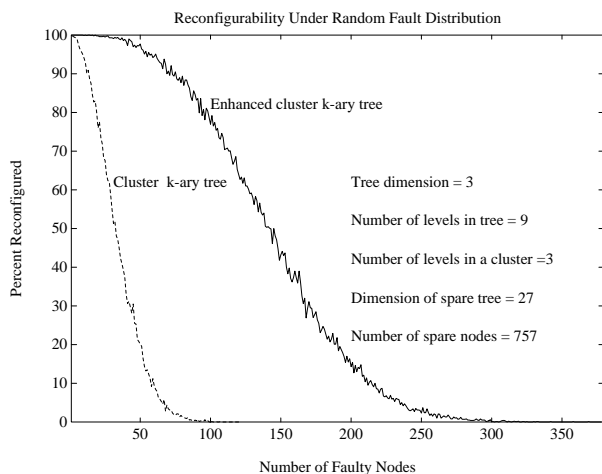


Figure 8: Tolerating faulty nodes only

performed for each given number of faulty nodes. The other plot in the figure (labeled as k -ary clus-

ter tree) pertains to the result of the local reconfiguration scheme [11, 14, 8, 10]. The result indicates nearly 100% reconfigurability for the ECKT in the presence of about 30 faulty nodes and nearly 90% reconfigurability in the presence of nearly 100 faulty nodes. The approaches in [5] and [7] perform slightly worse and slightly better than the local reconfiguration scheme respectively. Finally, the approach proposed in [4] only tolerates 2 faulty nodes and the scheme in [6] tolerates a fixed number of faulty nodes at the expense of large node degrees.

We next investigated the effect of node degree and cluster size on the fault tolerance of the ECKT. Our simulation results reveal that for a given ECKT, the best reconfiguration is achieved when the degree of the spare tree ($k_s = k^{l_c}$) and the spare tree levels ($l_s = \frac{l}{l_c}$) are kept to a minimum. However, the former requires l_c to be the minimum and the latter needs it to be the maximum. Hence, for a given ECKT, l_c should be selected such that there is a balance.

We next consider reconfiguration of the ECKT under 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. Our reconfiguration algorithm utilizes unassigned spare links to establish a parallel path to each faulty link. We implemented the algorithm for an enhanced cluster 3-ary tree with $l = 9$ and $l_c = 3$. The simulation result for up to 200 randomly placed faulty links is shown in Figure 9. The result indicates that the given ECKT can tolerate nearly 20 faulty links 90% of the time.

By combining the reconfiguration algorithm, which tolerates faulty nodes and faulty links, a combination of faulty nodes and faulty links may be tolerated. We assumed the probability of a link failure to be the same as a node failure. After randomly allocating the faulty elements, the algorithm first tries to tolerate all faulty links. If successful, it then uses Alloc-Spare-ECKT to tolerate the faulty nodes. The simulation result for an enhanced cluster 3-ary tree with $l = 9$ and $l_c = 3$ is shown in Figure 10. The result indicates that nearly 50 faulty elements are tolerated by the ECKT 90% of the time.

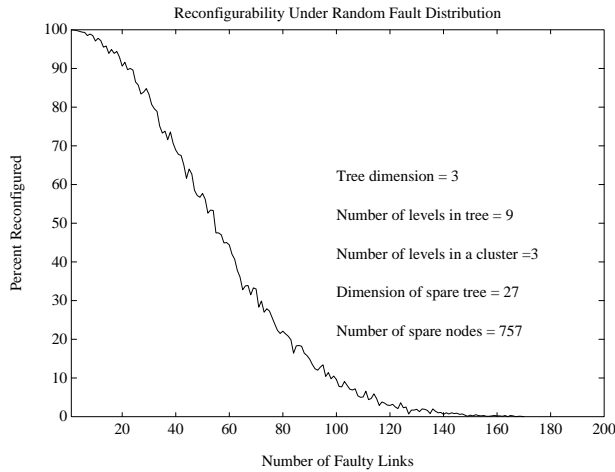


Figure 9: Tolerating link failures only

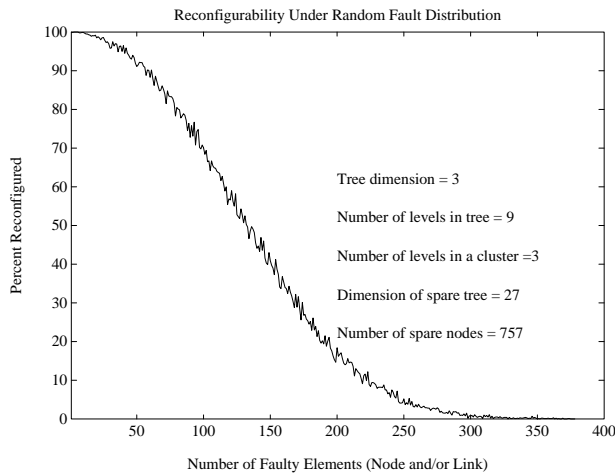


Figure 10: Tolerating faulty nodes and links

5 Conclusion

In this paper, we presented a strongly fault-tolerant design for the k -ary tree multiprocessor and examined its reconfigurability. Our scheme is global in the sense that any spare node can replace any faulty node. However, our reconfiguration algorithm can be implemented in a distributed manner. By combining the reconfiguration algorithms for faulty nodes and faulty links, combinations of node and link failures are tolerated.

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 under random distribution of faulty

nodes and faulty links reveal that the ECKT can tolerate relatively large number of faulty nodes and faulty links. Compared to other proposed schemes, the ECKT can tolerate significantly more faults for the same overhead.

References

- [1] C. Leiserson et. al., *The Network Architecture of the Connection Machine CM-5*. Cambridge, MA: Thinking Machine Corporation, 1992.
- [2] J. P. Hayes, "A graph model for fault-tolerant computing systems," *IEEE Transactions on Computers*, vol. c-25, pp. 875–884, September 1976.
- [3] N. Tzeng, "A cube-connected cycles architecture with high reliability and improved performance," *IEEE Transactions on Computers*, vol. 42, no. 2, pp. 246–253, 1993.
- [4] C. L. Kwan and S. Toida, "An optimal 2-FT realization of binary symmetric hierarchical tree systems," *Networks*, vol. 12, no. 12, pp. 231–239, 1982.
- [5] C. Raghavendra, A. Avizienis, and M. D. Ercegovic, "Fault tolerance in binary tree architectures," *IEEE Transactions on Computers*, vol. c-33, pp. 568–572, June 1984.
- [6] S. Dutt and J. Hayes, "On designing and reconfiguring k -fault-tolerant tree architectures," *IEEE Transactions on Computers*, vol. 39, pp. 490–503, April 1990.
- [7] M. B. Lowrie and W. K. Fuchs, "Reconfigurable tree architecture using subtree oriented fault tolerance," *IEEE Transactions on Computers*, vol. c-36, pp. 1172–1182, October 1987.
- [8] A. S. Hassan and V. K. Agarwal, "A fault-tolerant modular architecture for binary trees," *IEEE Transactions on Computers*, vol. c-35, pp. 356–361, April 1986.

- [9] A. D. Singh, "Interstitial redundancy: An area efficient fault tolerance scheme for large area VLSI processor arrays," *IEEE Transactions on Computers*, vol. 37, pp. 1398–1410, November 1988.
- [10] Y. Chen and S. J. Upadhyaya, "Reliability, reconfiguration, and spare allocation issues in binary-tree architectures based on multiple-level redundancy," *IEEE Transactions on Computers*, vol. 42, pp. 713–723, June 1993.
- [11] B. Izadi, "Design of fault-tolerant distributed memory multiprocessors," *Ph.D. thesis, the Ohio State University*, 1995.
- [12] C. J. Colbourn, *The Combinatorics of Network Reliability*. Oxford University Press, 1987.
- [13] C. Y. Lee, "An algorithm for path connection and its applications," *IRE Transactions on Electronic Computers*, vol. ec-10, pp. 346–365, 1961.
- [14] A. D. Singh, "A reconfigurable modular fault tolerant binary tree architecture," *Proceedings of the IEEE International Symposium on Fault Tolerant Computing*, pp. 298–304, 1987.