EE482 Research Paper Report: Fault-Tolerant Routing

Shu-Yi Yu <*syu@crc.stanford.edu*>
Robert Wei-Je Huang <*weije@crc.stanford.edu*>
Pranab Bhooma <*pranab@corp.cirrus.com*>

I. INTRODUCTION

Fully adaptive minimal-path routing scheme is widely used as the backbone for different kinds of routing algorithms in hypercube topology. However, as the number of processors in a cube-based multicomputer increases, the probability of processor or link failure also increases. With the presence of faults, there's an urgent need to develop fault-tolerant routing schemes to keep the healthy part of the network operating in a gracefully degraded mode with minimal overhead. In this report, we present two different algorithms for fault-tolerant routing with different goals in the presence of failures. In the first algorithm, a fault-free degree indicator concept called "safety vector" is studied and used to represent the minimal-path routing adaptivity in each node. These distributed safety vectors can be set up iteratively in n-1 rounds, where n is the dimension of the cube. A fault-tolerant routing algorithm targeting at finding minimal routing paths in a faulty network and reducing the resulting overhead is then established on the bases of these safety vectors. In the second algorithm, a dynamically configurable flow control mechanism is introduced to contain the faulty parts and avoid the deadlocks due to routing under faulty channels is developed. Finally, we will also compare the pros and cons of these two algorithms.

II. Fault-Tolerant Routing Using Safety Vectors in Cube-Based Network

This section studies the algorithm proposed in "Adaptive Fault-Tolerant Routing in Cube-Based Multicomputers Using Safety Vectors" by Jie Wu, IEEE Trans. on Parallel and Distributed Systems, April 1998.

1. Problem Definition

The problem is to route the packets using minimal paths in an n-dimensional hypercube with some link or node failures. The resulting overhead under fault-free conditions should be as small as possible.

Assumptions are:

(1) All faults are fail-stop. There are no malicious faults.

(2) Fault detection and diagnosis algorithms exist. The faulty nodes and links can be detected on the fly.

(3) Each node knows the safety status of all its neighbors and can distinguish an adjacent faulty link from an adjacent faulty node.

2. Proposed Solution

The proposed solution in this paper begins with the introduction of the *safety vector* concept.

2.1 Safety Vector Concept in the Binary n-cube

For each node in the binary n-cube, a safety vector is introduced to represent the minimal-path routing adaptivity to any node in the network with given hamming distance. The key concepts are defined as follows:

(1) *Hamming distance*: in an n-cube: the distance is defined as the number of different coordinates between the source and destination nodes.

(2) *Dynamic adaptivity*: a routing algorithm is k-adaptive at an intermediate node if it can use up to k neighbors that are along the minimal paths between this node and the destination node. In fault-free hypercubes, the k value of an intermediate node is the hamming distance between this node and the destination node. However, k is normally smaller than the hamming distance in the presence of faults.

(3) *Safety vectors*: an n-bit safety vector is associated with each node in an n-hypercube. Each element u_k in the vector is either 1 or 0 and represents the minimal-path routing capability of the corresponding node to any k-hamming-distance node. In fact, $u_k=1$ if and only if the minimal-path routing from this node can be guaranteed to any destination whose hamming distance is k apart. On the contrary, $u_k=0$ means that portions or all of the destinations k-distance away cannot be reached through minimal-path routing.

2.2 Calculation of the safety vector

Based on the definition of safety vectors and the topological properties of a binary hypercube, the k-th element u_k of a safety vector can be determined from the (k-1)-th element u_{k-1} of the neighboring safety vectors, except for the first bit which can be decided locally. The algorithm, called *Global-Status* (GS) algorithm, of computing safety vectors $u=(u_1,u_2,...,u_n)$ proceeds as follows:

(1) Initial condition: if the node itself is faulty, the safety vector is (0,0,...,0). If the node has a faulty link, set u_1=0. Otherwise, set u_1=1. In other words, the first bit of every healthy node in the network is set up locally according to the link status at the first stage.

(2) For k=2 to n: iteratively set up the rest of the bits. For each iteration, the node collects all the (k-1)-th bits (u_{k-1}) from the neighbor's safety vectors. If the link to one of the neighbors is corrupted, the node assigns a zero vector associated with the corresponding neighbor. After the collection is done, the node calculates the sum of the u_{k-1}s. If the sum is greater than n-k, set u_k=1. Otherwise, u_k=0.

The reason for the second part of the algorithm is obvious. In order to guarantee the minimal-path routing capability to any k-distance nodes, at least one of the u_{k-1} from any set of k neighboring nodes should be 1 to start the minimal-path routing. This leads to the fact that there are at most (k-1) 0's (or equivalently, at least (n-k+1) 1's) in the n neighboring u_{k-1}'s in the binary n-cube. The complexity of the algorithm is n-1 iterations, where n is the dimension of the cube.

2.3 Fault-tolerant Routing Algorithm

The proposed algorithm in this paper uses the constructed safety vector of each node to conservatively prevent from routing through faulty parts of the network. The approach goes as follows:

(1) Compute the hamming distance between the source and destination nodes. Assume the distance = k.

(2) The node check its own safety vector and see if the u_k=1. If u_k=1, the minimal path routing is guaranteed and it selects from one of the neighbors along the minimal path with u_{k-1}=1 as the next hop.

(3) If u_k=0, check the u_{k-1} of the neighbors along the minimal paths. If one of them is 1, select the corresponding neighboring node as the next hop.

(4) If u_k=0 and all u_{k-1}s from the neighboring nodes along the minimal paths are 0, minimal-path routing in this case would probably hit the faulty nodes or links, and we thus conservatively choose the suboptimal path from non-minimal-path routing. The node checks the u_{k+1}s from the neighbors not on the minimal path. If one of them is 1, the suboptimal routing can be guaranteed, and we select the corresponding node as the next hop.

(5) If none of the above condition holds, the routing failure due to the faulty network is declared, and it is concluded that the source and destination might be disconnected.

2.4 Extensions to the n-dimensional Mixed-Radix Hypercube Network

To extend the safety vector concept described above to the mixed-radix n-cube, the only difference for the algorithm is in the iterative calculation of the safety vectors. Here, to compute the u_k, the node collects all the u_{k-1} information from nodes with hamming distance 1 (nodes differ in only one dimension). For example, node (000) in a 3x3x4 cube collects data from (001), (002), (010), (020), (100), (200), and (300). Then it calculates the equivalent u_{k-1} for each dimension by finding the minimum of u_{k-1} along the nodes on the corresponding dimension. For the above example, there are 3 dimensions and hence 3 equivalent u_{k-1}s will be generated as min{u_{k-1} of (001) and (002)}, min{u_{k-1} of (010) and (020)}, and min{u_{k-1} of (100), (200), and (300)}. Then it follows the same criteria in the binary n-cube to find the sum of these equivalent u_{k-1}s and get the u_k. For the routing algorithm, the node looks at the computed equivalent safety vectors along each dimension instead of vectors of its direct neighbors.

3. Discussions and Potential Improvements

(1) *Complexity*: O(N) iterations needed to compute the safety vectors. Since normally the dimension of the cube N is not very huge. In addition, the routing can be fast since once the safety vectors are constructed, the only operations are table-lookup and simple combinational logic. Therefore, the resulting timing overhead is not a big issue here. However, each node has to pay more space for the storage of safety vectors of the neighbors and its own. This extra storage per node increases with N^2 since there are N neighboring N-bit vectors needed with N being the number of dimensions.

(2) *When to update the safety vectors*: The overhead on the traffic is the propagation of the safety vector information. Periodic updating can be wasteful and creating a lot of redundant traffic when the probability of fault is low. Hence, demand-driven or state-change-driven strategies are preferred for updating safety vectors.

(3) *Routability in the presence of faults*: This algorithm is actually a quite conservative way to prevent routing through the faulty area. Since the zero value in the k-th bit of the safety vector does not imply the disability to route through minimal paths for arbitrary k-distance source-destination pairs, this algorithm may impede some routable traffic from successfully going through. One possible solution to this is to have more global information in one node instead of neighboring safety vectors only. If the node can see the safety vectors of the nodes 2-distance apart and declare the failure of minimal-path routing or suboptimal-path routing after trying all possible 2-distance neighbors, the resulting routability in the presence of fault could be greatly increased. However, this solution pays the extra storage space per node and increases the traffic overhead.

(4) *Deadlock*: Routing in the presence of fault may break the original order of dimensions in deadlock-free

routing and thus is prone to deadlock. This algorithm does not consider the potential deadlock when the packet has to route through non-minimal paths. In the next section, we will study another algorithm which avoids the deadlock while handling the routing in a faulty network.

III. Dynamically Configurable Message Flow Control for Fault-Tolerant Routing

In this section, we study the algorithm proposed in " Dynamically Configurable Message Flow Control for Fault-Tolerant Routing" by B.V. Dao, J. Duato, and S. Yalamanchilli, IEEE Transactions on Parallel and Distributed Systems, Jan. 1999.

1. Problem Formulation

1.1 Tradeoff: Performance and Reliability

Fault-tolerant routing protocols in interconnection networks rely heavily on the network flow control mechanisms used. However, different flow control protocols target on different figures of merits, and thus designers often run into a dilemma to select a proper scheme to balance the tradeoff between performance and reliability. Optimal flow control schemes, such as *wormhole switching* (WS), realize very good performance by reducing latencies, but are prone to deadlock in the presence of faults (as mentioned in the end of last section). On the other hand, conservative flow control mechanisms, such as *pipelined circuit switching* (PCS), improve the reliability through the creation of the dedicated end-to-end path prior to message transmission at the expense of performance degradation. A practical engineering problem is thus to select a proper flow control under the presence of faults. In this paper, the target is emphasized on how to develop a fault-tolerant routing algorithm while keeping the faulty network deadlock-free.

1.2 Assumptions and models

In this paper, the author basically constructs the fault-tolerant routing scheme on the basis of a cube-based, deadlock-free (using Duato's algorithm) network with virtual-channel, flit-based message transmission and adaptive routing in the fault-free situation.

For the virtual channel model, a virtual channel trio model (V_i^D, V_i^C, V_i^*), representing *data channel*, *corresponding channel*, and *complimentary channel*, is being used. They are used to transmit data, header, and special control flits, respectively. The corresponding channels and the complimentary channels essentially form a control network for coordinating fault recovery and adaptive routing of header flits, including limited and controlled backtracking of header flits. Note that the control flits (ACK, NACK, credits…) travel opposite to the corresponding packets.

For the fault model, this paper assumes the existence of fault detection mechanisms and relies on using such information for robust, reliable communication. Failures can be either static (failures that present when the system is power on) or dynamic (random failures during operation) on idle links and routers, and dynamic on busy links.

Either the entire processing element and its associated router can fail or a physical link may fail. When a node and its router fail, all physical links incident on the failed node are also marked as being *faulty*.

In addition to the *faulty* channels, the *unsafe channel* concept is introduced in the presence of fault. Healthy channels ending in nodes which are adjacent to the faulty nodes and/or communication channels may be marked *unsafe*. For the channels other than being *faulty* or *unsafe*, a *safe* status is marked.

Another main constraint is that the network assumed to have from low to moderate number of faults.

2. Solution

The solution proposed in this paper operates dynamically in two phases (also called as *Two Phase Protocol*), with the introduction of the *Scouting Switching* flow control mechanism.

2.1 Scouting Switching (SS) Flow Control

The first data flit is constrained to remain K links behind the routing header. Every time the header successfully crosses a channel, a positive acknowledgment is propagated back to the location of the first data flit. If the routing header backtracks, a negative acknowledgment is transmitted. When the header reaches a destination node, it sends K consecutive positive acknowledgments. The data flits can only advance after the next possession of the next K links are acknowledged.

Such an approach enables the header to backtrack if a faulty component is encountered. When K=0, the flow control is equivalent to the wormhole switching, while large K can ensure path setup prior to data transmission and thus better reliability in the presence of fault. The parameter, K, is referred to as the *scouting distance* or *probe lead*. By dynamically modifying K, we can gain improved run-time tradeoffs between fault tolerance and performance.

The time-space diagram comparison of WS, SS, and PCS is shown in figure 1.

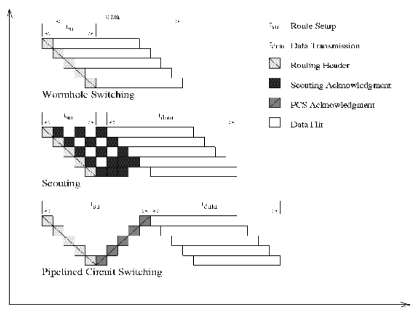

Fig.1 Time-space diagram of WS, Scouting, and PCS

2.2 Key Approach: Dynamical Switching Between Different Flow Control Mechanisms

Routing in the presence of faults demands a greater level of flexibility than required in fault-free networks. However, designing routers based on the relatively rare occurrence of faults requires that all message traffic be penalized, even the messages that route through the fault-free portions of the network. Overhead may arise due to the setting up of a fault-free path prior to data transmission (e.g. PCS), marking processors and channels faulty to construct convex fault regions or increasing the number of virtual channels for routing messages around the faulty components. Hence, the author proposed a configurable flow control concept to get the optimal balance between tradeoffs of different protocols:

(1) When the messages are in fault-free segments of the network, they are routed using protocols based on optimistic flow control such as wormhole switching.

(2) When the message encounters faulty networks, it adopts more conservative schemes such as misrouting, backtracking, SS flow control mechanism with dynamically configuring probe lead (K).

The distinguishing features of this approach are:

(1) It does not rely on additional virtual channels over that already needed for fully adaptive routing in PCS and WS.

(2) With low to moderate number of faults, the performance is considerably better than that for conservative fault-tolerant routing algorithms with equivalent reliability,

(3) It is based on a more flexible fault model, i.e., supports link and/or node faults and does not require convex fault regions,

(4) It can utilize existing techniques for recovery from dynamic or transient failures of links or switches, and

(5) It provides routing protocols greater control over hardware message flow control, opening up new avenues for optimizing message passing performance in the presence of faults.

2.3 Two Phase Protocol

In the absence of faults, the deadlock-free routing function based on Duato's Protocol (DP) are employed. The order of channel for allocation is (1) *safe adaptive channels* (safe C2 channels), (2) *safe deterministic channels* (safe C1 channels). In addition, the flow control in fault-free network is wormhole switching, which is equivalent to K=0 in scouting switching, to get the optimal performance.

In faulty networks, if none of the safe channels are not available (by being faulty or unsafe), profitable (or productive) safe or unsafe adaptive channels are preferred. As soon as a message enters an unsafe channel, the flow control switches to scout switching, whereby allocating a counter for K. And by dynamically controlling the values of K, the header can backtrack if it gets blocked or maximum number of K is reached.

If there are no more profitable channels, the routing goes into detour construction. The detour is complete when all the misroutes made during the construction of the detour have been corrected or when the destination node is reached. (There is no SS flow control during detour construction)

When the detour is complete SS acknowledgments start to flow again, and data flits can resume progress.

Note that in this approach, the deadlock-free is still guaranteed in the presence of faulty C1 channels since the flow control protocol switches to the conservative scouting switching as soon as the header travels through an unsafe channel, which is in the vicinity of the faulty region. The payload flits do not advance before the header clears the any blocking along the next paths, and thus avoids the deadlock.

3. Strong Points vs. Weak Points

From low to moderate number of faults, configurable flow control mechanisms can lead to deadlock-free fault-tolerant routing protocols whose performance is superior to more conservative routing protocols with comparable reliability. In a network with a large number of faults, TP's partially optimistic behavior results in severe performance degradation. With conservative routing protocols, no data is transmitted until a path has been setup between the source and the destination. Also, since the information on the faulty region is absolutely local, the header may try a long erroneous paths before finally finding itself going in the wrong way and backtracking to source nodes. This situation gets worse when the number of faults increases. One possible way to solve this is to add more global information on each node informing headers of how far they can go without touching next unsafe/faulty channel and use it as a supplement to the routing decision.

Two-phase protocol (TP) does not require any complex renumbering scheme to provide fault tolerance, does not require the construction of convex regions, does not require additional virtual channels, and the dynamic fault-tolerant version of TP does not rely on time-outs or padding of messages. It does, however, result in a more complex channel model, which can affect link speeds.

The critical issues can be summarized as follows:

(1) It's not reliable with large number of faults. Overhead increases non-linearly with the increase in number of faults, without the reliability of delivery.

(2) Relies heavily on maximum number of misroutes allowed. If misroutes are sufficiently chosen the message does not reach the destination, while loading the network.

(3) There is not enough information supplied for choosing maximum number of backtracks and misroutes.

(4) Overhead increases for small messages for maintaining Control and Corresponding Flits.

(5) It is too long before it can be realized if a not cannot be reached.

(6) Link speed is reduced for the use of positive and negative acknowledgments.

(7) Assures Deadlock freedom by the use DP, controlled detour construction.

IV. Comparison and Conclusion

Fault tolerant routing is very critical as it directly relates to the performance of the system. But the choice of the network heavily depends on,

(1) Number of Faults in the Network.

(2) Reliability of the Network preferred.

(3) Complexity allowed that could be afforded.

(4) Deadlock issue.

The first algorithm, being very conservative, can be used if we want to have highly reliable routers with large number of faults, with minimal path routing. Although the complexity and the latency is attractive, there is no deadlock avoidance here and the extra traffic for global safety information propagation can be critical weakness of this mechanism. Another issue is the routing disability for some traffic and fault combinations due to the pessimistic nature of the theory. A potential improvement to this is suggested in the Section 3 of Part II.

The second algorithm is more suitable for low to moderate number of faults, and as an alternative to the existing WS, PCS, FCS routing and mainly for the purpose of deadlock avoidance. On the other hand it is not so reliable, reduction in link speed, fails with huge number of faults. Also, the resulting long misrouting latency is crucial, and a possible way to solve this is discussed in Section 3 of Part III.

V. Reference

1. J. Wu, "Adaptive Fault-Tolerant Routing in Cube-Based Multicomputers Using Safety Vectors", IEEE Transactions on Parallel and Distributed Systems, April 1998

2. Y. Lan, "A Fault-Tolerant Routing Algorithm in Hypercubes", Proc. of 1994 International Conference on Parallel Processing, Aug. 1994

3. T.C. Lee and J.P. Hayes, "A Fault-Tolerant Communication Scheme for Hypercube Computers", IEEE Transactions on Computers, Oct. 1992

4. Binh Vien Dao, Jose Duato and Sudhakar Yalamanchilli, "Dynamically Configurable Message Flow Control for Fault-Tolerant Routing", IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, Vol. 10, No. 1: JANUARY 1999, pp. 7-22

5. S. Chalasani and R. Boppana, "Fault-Tolerant Wormhole Routing in Tori," Proc. Eighth ACM Conf. Supercomputing, July 1994.

6. A.A. Chien and J.H. Kim, "Planar-Adaptive Routing: Low-Cost Adaptive Networks for Multiprocessors," J. ACM, vol. 42, no. 1, pp. 91-123, 1995.