A Transition-Based Fault-Tolerant Routing Methodology For InfiniBand Networks*

J. M. Montañana, J. Flich, A. Robles, P. López, and J. Duato
Dept. of Computer Engineering (DISCA)
Universidad Politécnica de Valencia
Camino de Vera, 14, 46021–Valencia, Spain
E-mail: jmontana@gap.upv.es

Abstract

Currently, clusters of PCs are considered a cost-effective alternative to large parallel computers. As the number of elements increases in these systems, the probability of faults increases dramatically. Therefore, it is critical to keep the system running even in the presence of faults. The interconnection network plays a key role in its performance. InfiniBand (IBA) is a new standard interconnect suitable for clusters. Most of the fault-tolerant routing strategies proposed for massively parallel computers cannot be applied to IBA because routing and virtual channel transitions are deterministic, which prevents packets from avoiding the faults.

A possible approach to provide fault-tolerance in IBA consists of using several disjoint paths between every source-destination pair of nodes and selecting the appropriate path at the source host. However, to this end, a routing algorithm able to provide enough disjoint paths, while still guaranteeing deadlock freedom, is required. In this paper we address this issue, proposing a simple and effective fault-tolerant methodology for IBA networks that can be applied to any network topology and meets the trade-off between fault-tolerance degree and the number of network resources devoted to it. Preliminary results show that the proposed methodology scales well and supports up to three faults in 2D and five in 3D tori using only two virtual channels.

1 Introduction

Over the recent years, there is a trend in using clusters of PCs for building large systems. Some examples are cluster-based commercial Internet portal servers like AOL, Google, Amazon or Yahoo. Also, clusters of PCs are currently being considered as a cost-effective alternative for small and large-scale parallel computing. Each time, more cluster-based systems are included into the top500 list of supercomputers. In particular, the Virginia Tech's X [24] (with 2,200 processors) occupies the third position in the list.

InfiniBand[13] is a standard interconnect technology for interconnecting processor nodes and I/O nodes to build a system area network (SAN). The InfiniBand Architecture (IBA) is designed around a switch-based interconnect technology with high-speed serial point-to-point links connecting multiple independent and clustered hosts and I/O devices. Therefore, this interconnect technology is suitable to build large clusters.

In many cluster-based systems, it is critical to keep the system running even in the presence of faults. These systems use a very large number of components (processors, switches, and links). Each individual component can fail, and thus, the probability of failure of the entire system increases. Although switches and links are robust, they are working close to their technological limits, and therefore they are prone to faults. Increasing clock frequency leads to a higher power dissipation, and a higher heating could lead to premature faults. So, fault-tolerant mechanisms in cluster-based systems are becoming a key issue.

Most of the fault-tolerant routing strategies proposed in the literature for massively parallel computers are not suitable for clusters (see chapter 6 in [10] for a description of some of the most interesting approaches). This is because they often require certain hardware support that is not provided by the current commercial interconnect technologies [1, 13]. Additionally, these routing strategies have been normally designed for specific regular network topologies, like meshes and tori. However, the switch interconnection pattern in clusters may be irregular. Furthermore, they cannot be applied to IBA because routing is deterministic, which

*This work was supported by the Spanish MCYT under Grant TIC2003-08154-C06-01 and the Generalitat Valenciana under Grant CTIDIB/2002/288, and the JCC de Castilla-La Mancha under Grant PBC-02-008.

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

prevents packets from circumventing the faulty components found along their paths. Also, some of these routing strategies need to perform virtual channel transitions when the packet is blocked due to a fault. However, virtual channels in IBA cannot be selected at routing time. Additionally to the best of our knowledge, there are no proposals focused on providing fault tolerance in IBA.

In IBA routing and virtual channel selection is performed based on the destination local ID (DLID) and the service level (SL) fields of the packet header. These two fields are computed at the source node and do not change along the path. Therefore, IBA routing is a kind of source routing with the routing info distributed. As a consequence, a possible way to provide fault-tolerance in IBA would be to have several alternative paths between every source-destination pair, selecting one of them at the source host. In fact, IBA provides a mechanism supported by hardware [13], referred to as Automatic Path Migration (APM), which may be used for this aim. According to this mechanism, at connection setup time, the source node is given two sets of path information for each destination, one for the primary path and another one for the alternate path. APM provides a fast mechanism for migration from the primary to the alternate path when a faulty component is detected in the network. Once path migration is accomplished, the alternate path is converted into the new primary path. Therefore, the subnet manager could reload the alternate path variables with new ones and re-enable the APM mechanism.

2 Motivation

To take advantage of the APM mechanism, it is necessary to apply an effective methodology able to provide enough alternative paths to tolerate the occurrence of a certain number of faulty components. Furthermore, the provided alternative paths should be disjoint.

As the network topology changes in the presence of faults, it seems more appropriate to apply a generic routing algorithm[1] in order to compute the paths. Indeed, as we want to provide several paths per each source-destination pair, the routing algorithm should provide some degree of adaptivity. Among the existing proposals for generic routing, up*/down* [22] is the most popular routing scheme. Despite the fact that there exist several generic routing schemes able to achieve higher performance, either they require support for distributed adaptive routing [23], which is not provided by IBA switches, or they exhibit a high computational cost [4], which prevents them from being applied to large network sizes. Up*/down* is able to provide alternative paths between most pairs of nodes. However, this routing scheme does not guarantee on its own the existence of enough disjoint paths. Therefore, it is worth analyzing the fault-tolerance capabilities of up*/down* and proposing strategies to improve them.

In this paper, we propose a simple and effective fault-tolerant methodology for IBA networks, based on the methodology presented in [20]. The objective of this methodology will be to obtain the maximum number of disjoint paths in order to tolerate the maximum number of faults. This strategy will also use an underlying generic routing algorithm in order to prevent deadlocks, and thus, it can be applied to any network topology. The proposed methodology will use virtual lanes (VLs) and service levels (SLs) in order to increase its fault-tolerance capacity. However, VLs and SLs in IBA could be needed for other purposes (mainly QoS). In fact, IBA switches support up to 15 VLs and 16 SLs. Thus, the number of VLs and SLs dedicated to fault-tolerance issues should be bounded.

Therefore, the goal of the paper is twofold. First, the fault-tolerance capabilities of a generic routing algorithm like up*/down* will be analyzed. And second, a new fault-tolerant methodology will be presented to improve these capabilities. This methodology will be evaluated in terms of fault-tolerance, resource needs, and performance degradation.

The rest of the paper is organized as follows. In Section 3, fault models and related work are described. In Section 4, the new methodology will be presented. In Section 5, an analysis of the up*/down* routing algorithm and the proposed methodology in terms of fault-tolerance, resource requirements, and performance degradation will be presented. Finally, in Section 6 some conclusions will be drawn.

3 Fault Models and Related Work

Using fault-tolerant mechanisms will ensure, in the case that a component fails, that the system will keep running in a degraded mode until the failed component is repaired. In order to deal with permanent faults in a system, two fault models can be used. The first one is the static fault model. In a static fault model, once a fault is detected, all the running processes in the system are halted, the network is emptied, and a special application is run in order to deal with the new faulty component. This application detects where the fault is and computes new routing tables in order to avoid the faulty component. The static fault model has a practical interest when checkpointing techniques are applied.

The second one is the dynamic fault model. Once a fault is found, actions are taken in order to properly handle the faulty component. For instance, a source node that detects a faulty component along a path used to reach a certain destination node could switch to a different path that does not use the faulty component. Therefore, the system keeps working, the network is not emptied, and checkpointing is not

[1] Generic routing algorithms can be applied to any topology.

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

needed.

Basically, there are three ways to tolerate a fault: component redundancy, fault-tolerant routing algorithms, and network reconfiguration. Using component redundancy has been the easiest and most costly way to provide fault-tolerance. Components in the system are replicated and once a faulty component is detected it is switched to its redundant copy.

When using fault-tolerant routing algorithms, redundancy is achieved by an efficient use of the existing network resources. However, in some cases, additional hardware is required, which increases the cost. Most of the fault-tolerant routing algorithms have been proposed for the dynamic fault model avoiding faulty components by using alternative paths. Strategies for fault-tolerant routing differ by the number of supported faults and resources needed for guaranteeing deadlock-freedom in the presence of faults.

A large number of routing algorithms for fault-tolerance in multiprocessor systems have been proposed. In [6, 15, 16] adaptive routing is used together with the status of links. Some solutions consist of using non-fault-free routing algorithms together with virtual channels. Some of these solutions are based on block faults [2, 5](often disabling some healthy nodes), whereas others allow individual faults [9, 12]. Finally, other routing solutions [8, 11] are based on misrouting and backtracking of packets to support the maximum number of faults. All these strategies require specialized routing circuits, and often the use of additional virtual channels.

With reconfiguration[3], once a fault is detected, the new topology is discovered and new routing tables are computed. This approach is suitable for switch-based networks (Myrinet [1], IBA [13]). When using reconfiguration, any number of faults is supported without using additional resources. The main drawback of reconfiguration is the high delay of packets that are discarded and resent during the reconfiguration process. This may prevent its application to environments with hard time constraints (real-time applications and QoS).

4 Description of the Methodology

The main goal of the methodology is to provide a set of alternative paths between each pair of hosts. To this end, the APM mechanism provided by IBA is used to switch to a fault-free alternate path. To provide reliability, while still guaranteeing deadlock freedom, some virtual channels and service levels will be required. Therefore, it will be important to meet the trade-off between the achieved fault-tolerance degree and the number of network resources used.

The proposed methodology is composed of six different stages as shown in Figure 1. The key point to achieve a good set of paths for fault-tolerance is to have the maximum freedom degree when choosing those paths. The main limitation in the path selection is due to the routing restrictions imposed to guarantee deadlock freedom. For this, at the first stage, the methodology computes a certain number of disjoint paths between every pair of hosts only depending on the network topology and without taking into account the occurrence of possible deadlocks. So, in this stage, the methodology does not limit the use of any possible path. By doing this, full flexibility is achieved and the best set of paths can be obtained. At a latter stage, deadlock situations will be solved without modifying the physical paths.

The number of computed alternative paths will depend on the number of failures to be supported. In particular, under a dynamic fault model, $n + 1$ disjoint paths for every pair of nodes are required support n faults. However, notice that the switch with the lowest number of ports connecting switches, and with at least one host attached to it, will bound the number of supported faults.

The way paths are computed is critical for the methodology. As the network size increases, a larger number of possible paths can be computed, and therefore the computation time could be extremely high. Thus, an efficient way of computing paths should be used in order to keep the computation time low. The methodology takes advantage of the fact that each possible disjoint path for a particular source-destination pair will use different ports both at the source switch and the destination switch. So, the methodology will be focused on obtaining the shortest disjoint paths by searching within the subset of paths that accomplish this property.

In the case of 2D Torus networks and other planar topologies, the methodology can be improved even more in order to effectively compute paths. The order in which links are used by the different disjoint paths at source and destination switches is important to quickly compute paths. For instance, in Figure 2.a we can observe that three disjoint paths are already computed for a particular source-destination pair. However, it is impossible to obtain a fourth disjoint path as some links are not reachable without sharing a link or switch with the already computed paths. A way to solve this problem and at the same time compute paths in a quick way is using links at the source switch in the clockwise direction and the links at the destination switch in the counter-clockwise direction. Figure 2.b shows an example of four possible disjoint paths that follow this rule. Notice that the first computed path for a source-destination pair may affect the chances of obtaining the remaining three disjoint paths. In the case that the four disjoint paths are not achieved, the process will be repeated by using different ports at starting and ending switches when computing the first path. For other network topologies, some similar rules should be extracted in order to keep a low computation time.

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

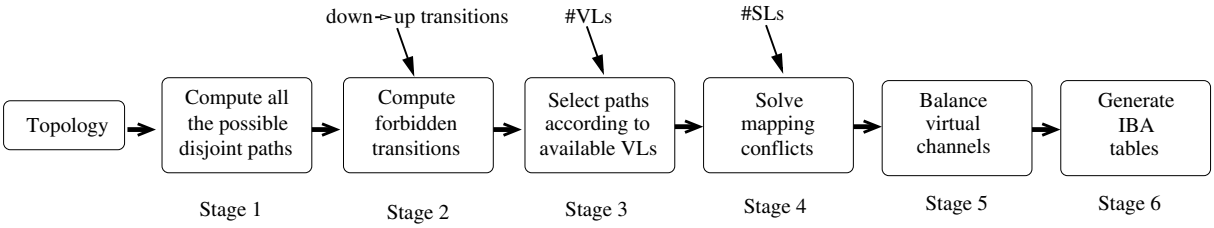

Figure 1. Stages of the proposed methodology.

The computational cost of the algorithm is O(n^2l^2), where n represents the number of switches and l the number of ports of each switch used to connect to other switches. This is because we have to compute certain number of disjoint paths for every source-destination pair (i.e., n^2 pairs). Moreover, for each pair of nodes it may be needed to repeat the searching of disjoint paths so many times as the number of source-destination pairs of ports, (i.e., l^2).

Notice, that to bound the complexity of the first stage of the methodology, we must limit the length of the disjoint paths to be searched. In particular for 2D tori, we have limited the length of the paths according to the topological distance between nodes plus 4.

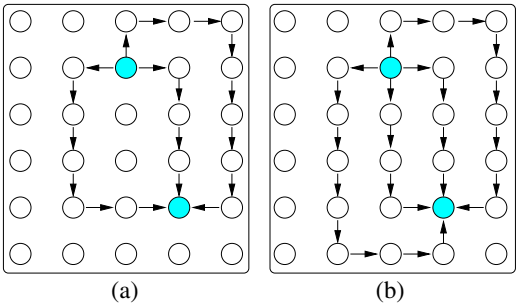

Figure 2. Example of disjoint paths.

The second stage of the methodology is focused on analyzing deadlocks. For this, the methodology builds and searches for possible cyclic channel dependencies. In particular, an underlying routing algorithm is used in order to detect possible cycles. We have chosen in this paper the up*/down* routing scheme[2] because it can be applied to any topology. We check whether every computed routing path contains any forbidden transition according to the up*/down* rule (a link in the up direction is used after having used one in the $down$ direction), which could lead to a cyclic channel dependency and, therefore, to a deadlock. In order to guarantee deadlock freedom, we must remove those transitions.

Cyclic channel dependencies can be removed by using virtual channels [7]. An easy way of doing this is to route the packets through a new different virtual channel every time a forbidden transition is found. In turn, cyclic channel dependencies between virtual channels can be removed by forcing their use in increasing order. This effectively enables the use of the computed paths without leading to deadlocks.

In our methodology, a packet will be injected into the network through VL0. It will continue to be routed through VL0 until the first forbidden transition has to be crossed. Then, the packet will be routed through VL1. If the packet has not to cross more forbidden transitions, it will continue to be routed through VL1 until it reaches the destination host. Otherwise, the packet will change from VL1 to VL2 when the next forbidden transition must be crossed, and so on. Notice that paths that cross the greatest number of forbidden transitions will impose the upper bound on the number of virtual channels to be used.

In IBA, the use of virtual channels can be devoted to other purposes like QoS. Therefore, the number of virtual channels for fault-tolerance issues should be limited. Hence, at the third stage of the methodology (Figure 1) we will only select those paths that traverse a number of forbidden transitions that can be supported by the number of VLs available for fault-tolerance. Notice that some alternative paths may be removed, thus affecting the fault-tolerance degree.

Virtual channel selection in IBA is based on the use of service levels (SLs). By means of SL-to-VL mapping tables, SLs are used to select the VLs to use at each switch. This table returns, for a given input port and a given SL, the VL to be used at the corresponding output port. The SL is placed at the packet header and it cannot be changed by the switches. Therefore, we should also assign the proper SL that must be used for a given path. According to the proposed methodology, every path will be labeled with a SLx where x is the number of forbiden transitions crossed. However, the fact of fixing a path with a unique SL can lead to a mapping conflict. It occurs when two packets labeled with the same SL enter a switch through the same input port but through different VLs, and they need to be routed to the same output port but also through different VLs. The problem is that the SL-to-VL mapping table does not use the input VL in order to determine the output VL. Both packets have the same SL because they have to cross the same num-

[2]The assignment of up and $down$ directions to links has been performed according to the DFS methodology [19], instead of the original BFS methodology [22] due to the smaller number of forbidden transitions imposed.

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

ber of forbidden transitions. The conflict arises when the number of forbidden transitions crossed by each path up till then is different. Figure 3 shows an example. Let A and B be two routing paths labeled with SL1 (both have to cross one forbidden transition), which cross switch V using the same input and output ports. Let us assume that, at switch V, A has already crossed its forbidden transition, whereas B still has not crossed it. Thus, A must use VL1, whereas B must use VL0. However, at switch V a mapping conflict arises as it is not possible to distinguish both paths because both of them are labeled with the same SL.

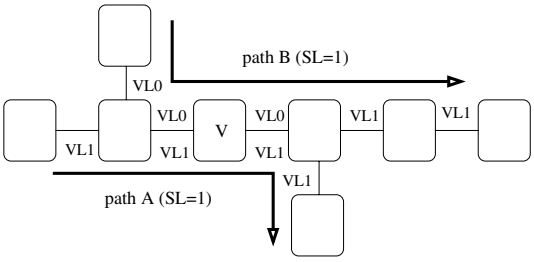

Figure 3. Mapping conflict example.

Hence, at the fourth stage of the methodology (Figure 1), we will solve these mapping conflicts in the following way. Firstly, we will try to find an alternative path for one of the paths causing the conflict, preferably of the same length. Notice that the new path should be disjoint to the rest of paths between the same source-destination pair. If it is not possible and there is still one new service level available, then it will be used, labeling one of the paths with this additional service level. Otherwise, one of the paths that incur in the mapping conflict will be discarded. Notice that this will lead to a decrement in the network reliability. Therefore, the number of mapping conflicts solved in this way will depend on the trade-off between the required network reliability and the number of available service levels.

At the fifth stage, we will balance virtual channel utilization. Due to the second stage, VL0 will be the most utilized virtual channel. All packets must use VL0, whereas VL1 is used only by some packets after crossing the first forbidden transition and so on. This causes a traffic unbalance over the virtual channels, increasing the head-of-line blocking at each switch. This will negatively influence network performance, as shown in [20]. In order to better balance the traffic over the virtual channels, we can distribute the traffic initially injected by VL0 (packets that are not going to traverse any forbidden transition), among the rest of VLs. This can be easily done by modifying the SL level of the packets. Notice that this will be carried out only if it does not introduce new mapping conflicts. These packets do not cross any forbidden transition and they are routed through the same assigned VL until being delivered, without switching to any other VL. Hence, balancing virtual channel utilization cannot lead to deadlock.

Finally, at the sixth stage, the routing information is generated. This includes generating the forwarding tables, the SL-to-VL mapping tables, and the tables that map destination ID's with SLs at source hosts. This is performed by using the destination renaming mechanism [17].

5 Evaluation

In this section, we will evaluate the proposed methodology. The fault-tolerant routing algorithm resulting from applying this methodology will be referred to as TFTR (Transition-Based Fault-Tolerant Routing). In particular, we are interested in the trade-off between the fault-tolerance degree and the resources required by TFTR. For comparison purposes, we will also analyze the fault-tolerance of up*/down* routing. First, we will introduce the scenario where TFTR and up*/down* will be analyzed and later we will present their evaluation results.

5.1 Evaluation Model

Often, clusters and networks of workstations are arranged on regular network topologies when the performance is the primary concern. Low dimensional tori (2D and 3D) are one of the most widely used topologies in commercial parallel computers. Furthermore, recent proposals, such as Alpha 21364 [18] and BlueGene/L [14], use 2D and 3D tori, respectively. For this reason, we have performed a preliminary analysis of TFTR on torus topologies. In particular, we will evaluate routing algorithms in 2D Torus and 3D Torus networks[3]. We have analyzed networks of 16(4×4), 25(5×5), 36(6×6), 49(7×7), and 64(8×8 and $4 \times 4 \times 4$) nodes.

In the analysis, we will only consider link failures. Indeed, only faults on links connecting switches will be taken into account. Note that a switch failure can be viewed as if all its links had failed. Moreover, a failure[4] in a link connecting a host to a switch does not change the topology. The number of disjoint paths depends on the maximum degree of any switch in the network. Therefore, at maximum there will be 4 disjoint paths in a 2D Torus (6 in a 3D Torus). Hence, at most, 3 faults can be tolerated (5 in a 3D Torus). Therefore, the methodology will be focused on obtaining the maximum number of disjoint paths.

[3]Notice that the e-cube algorithm could also be applied to tori. However, in IBA and in the absence of faults, this routing scheme already needs two VLs and a number of SLs that grows exponentially with the number of dimensions as shown in [21].

[4]Notice that to cope with failures in links connecting switches to hosts, it would be necessary to use either CAs with several ports or more than one CA per host. This issue is out of the scope of this paper.

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

Different number of SLs will be granted in order to evaluate the fault-tolerance degree achieved by the methodology. Only two virtual channels will be used.

The methodology also will be evaluated in terms of network performance. In particular, network performance degradation due to faults will be analyzed when applying the proposed methodology.

5.2 Evaluation Results

In Table 1 the number of alternative disjoint paths obtained by the TFTR and up*/down* routing algorithms is shown. One virtual channel is used for up*/down* and two virtual channels for TFTR. The Table shows the percentage of source-destination pairs for which one, two, three or four disjoint paths could be computed. Table shows results for any combination of routing algorithm and topology.

		Percentage of pairs of nodes with n disjoint paths			
Torus	SLs	n=1	n=2	n=3	n=4
TFTR (2 VLs)					
$4 \times 4, 5 \times 5$	3	0.00	0.00	0.00	100,00
$6 \times 6, 7 \times 7$	4	0.00	0.00	0.00	100,00
8×8	5	0.00	0.00	0.00	100,00
9×9	6	0.00	0.00	0.00	100,00
10×10	7	0.00	0.00	0.00	100,00
up*/down* (1 VL)					
4×4	1	19,17	30,00	33,33	17,50
5×5	1	22,00	38,00	30,00	10,00
6×6	1	25,87	42,54	25,08	6,51
7×7	1	26,53	47,11	21,77	4,59
8×8	1	28,13	49,80	18,65	3,42
9×9	1	28.09	52.78	16.48	2.65
10×10	1	28.83	54.55	14.51	2.12
Torus	SLs	n=1	n=2	n=5	n=6
TFTR (2 VLs)					
$4 \times 4 \times 4$	3	0.00	0.00	0.00	100,00

Table 1. Percentage of pairs of nodes with n disjoint paths. Unbounded number of SLs.

As we can observe, TFTR obtains for all 2D topologies four disjoint paths (and six paths for 3D Torus) for every source-destination pair with only two virtual channels. So, fault-tolerance in 2D and 3D tori can be achieved by TFTR with only two virtual channels. Notice that up*/down* is not able to find a sufficient number of disjoint paths for every source-destination pair. Moreover, for all network sizes there is a significant number of source-destination pairs with only one up*/down* path. So, up*/down* will not even tolerate one fault.

Figure 4 shows the percentages of singular cases when using up*/down* and TFTR for different number of faults (only for 2D tori). A singular case is defined as the fault combination that can not be solved by the routing algorithm. That is, the routing algorithm is not able to obtain a valid path for a particular source-destination pair. Notice, though, that the routing algorithm could continue to provide healthy paths for the rest of source-destination pairs. Fault combinations that lead to unconnected nodes are not considered as singular cases. All fault combinations have been examined except for 5 and 6 faults, where a random set of fault combinations has been analyzed. Error bars are plotted in these cases.

We can observe that TFTR does not present any singular case for one, two, and three fault cases, as it is able to obtain four disjoint paths. Therefore, the methodology is 3-fault tolerant with 2 VLs for 2D tori. This is a reasonable fault-tolerance degree taking into account the analyzed network sizes, and the fact that the mean time between failures is much greater than the mean time to repair. However, from 4 faults upwards, the proposed methodology is not able to tolerate all fault combinations. In particular, for 4 faults up to 30% of fault combinations are not supported. This percentage increases as the number of faults in the network increases. For 6 faults, almost all fault combinations are not tolerated by the methodology when it is applied to 2D tori. However, notice that the up*/down* routing algorithm has a pretty worse behavior. Practically, all fault combinations even with one fault are not tolerated by this algorithm. Notice that, when a singular case arises, the only solution will be only to launch a network reconfiguration process to compute new routing tables. Both up*/down* or TFTR can be used to compute new routing tables, because they are valid for any network topology (the new topology resulting from the occurrence of faults). If TFTR is used, once the reconfiguration process is finished, the network will be able to tolerate up to 3 additional faults, and so on. Also, notice that the occurrence of a low percentage of singular cases with TFTR may contribute to postpone the need of carrying out the network reconfiguration.

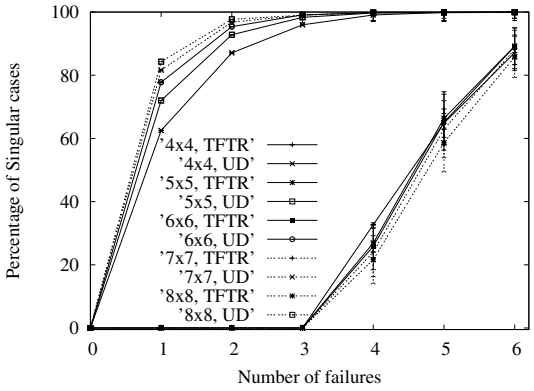

Figure 4. Percentage of singular cases.

Once we know that two virtual channels are enough to tolerate three faults for 2D tori and 5 faults for 3D tori, when

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

using the TFTR strategy, we should analyze the number of service levels that will be required in order to correctly use these virtual channels. Table 1 shows the number of required service levels (in an unbounded scenario). The need of SLs increases as the network size increases. In particular, 3 SLs are required for a 4×4 Torus network and 7 SLs for a 10×10 Torus network. The reason for this increase is because as paths get longer, the probability of mapping conflicts increases, and thus, additional SLs are required. It is significant also that for the 3D torus network, only 3 SLs are required. 3D tori allow more paths to be computed and therefore, mapping conflicts can be avoided by alternative paths. Moreover, it can be deduced that a network with 64 nodes should be designed by using a 3D torus (rather than a 2D torus requiring 5 SLs) in order to obtain a fault-tolerant system with few SLs (just 3 SLs).

Taking into account that IBA allows up to 16 SLs, it seems too large the number of SLs required by TFTR for some topologies. Therefore, a bounded scenario where the number of SLs is limited should be also evaluated. For this reason, Table 2 shows the number of disjoint paths obtained by TFTR when the number of SLs is bounded. Notice that results for just one SL are not shown as this does not allow to use VL transitions.

As can be observed, when restricting the number of SLs, the number of available disjoint paths decreases for some source-destination pairs. In particular, from 4×4 to 7×7 2D networks, some source-destination pairs were granted only 2 disjoint paths. Although the percentage of these cases is low, it limits the fault-tolerance of TFTR to one fault. For 8×8 Torus network and 4 SLs, the methodology is 2-fault tolerant. Therefore, the methodology effectiveness is sensitive to the number of SLs used.

		Percentage of pairs of nodes with n disjoint paths			
	VLs=2				
Torus	SLs	n=1	n=2	n=3	n=4
4×4	2	0.00	0.83	4.58	94.58
5×5	2	0.00	1.17	15.17	83.67
6×6	2	0.00	1.75	23.02	75.24
6×6	3	0.00	0.08	4.52	95.40
7×7	2	0.00	2.93	29.97	67.09
7×7	3	0.00	0.40	20.87	88.61
8×8	2	0.00	4.32	30.53	65.13
8×8	3	0.00	0.77	14.26	84.97
8×8	4	0.00	0.00	5.03	94.97

Table 2. Percentage of pairs of nodes with n disjoint paths with a bounded number of SLs.

Finally, Figure 5 shows the performance degradation suffered by the network in the presence of faults when the TFTR algorithm is used. Every source node will use the shortest disjoint path that does not traverse the faulty links. For a particular number of faults, different random combinations of faults are injected. For every combination of faults the network is simulated (with the simulation tool presented in [20]) obtaining its throughput. Error bars are shown for every number of faults.

As can be observed, throughput decreases as the number of faults increases. However, it can be noticed that performance degradation is relatively lower in larger networks. For instance with 6 faulty links, throughput decreases up to 28% for the 4×4 torus and only a 18% for the 8×8 torus. This is because the same number of faulty links affects a lower percentage of paths in larger networks.

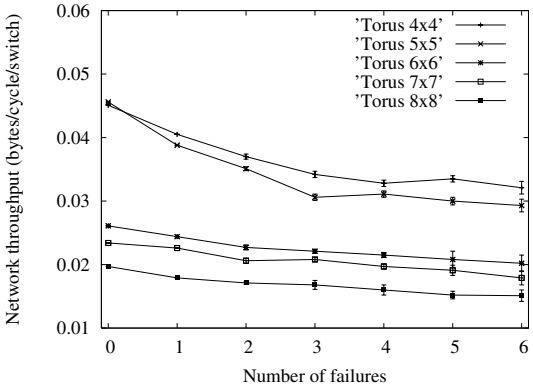

Figure 5. Throughput degradation.

6 Conclusions

Cluster-based systems are becoming an interesting alternative to parallel computers. Many of these systems are continously working ($24 \times 7 \times 365$). Thus, fault-tolerance becomes a key issue in the design of these systems. Moreover, nodes of clusters are usually interconnected by means of some high-performance network. Among these networks, IBA is one of the most promising alternatives.

Most of the fault-tolerant routing algorithms proposed in the literature for massively parallel computers cannot be applied to IBA-based systems as they require certain hardware support that IBA does not provide. A possible approach to provide fault-tolerance in IBA consists of providing several disjoint paths between every source-destination pair and selecting the appropriate fault-free path at the source host by using the APM mechanism.

In this paper, we have proposed a simple and effective methodology to design fault-tolerant routing strategies for IBA that provides several disjoint paths between every source-destination pair. The proposed methodology makes use of some network resources (VLs and SLs), and guarantees deadlock freedom by using the up*/down* routing scheme.

Preliminary analysis of the resulting fault-tolerant routing strategy (referred to as TFTR) on 2D and 3D Torus networks shows that 2 VLs are enough to tolerate up to 3 link

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

failures. Moreover, the number of required SLs ranges from 3 for the 4×4 Torus up to 7 for the 10×10 Torus. This can be considered an interesting result if we take into account that the up*/down* routing algorithm is not able even to tolerate one single failure. From 4 faults upwards, TFTR shows a progressive increase in the number of combinations of faults that are not tolerated as the number of faults increases. In order to keep the network running when the number of faults exceeds the maximum number of faults tolerated by the routing algorithm, a network reconfiguration process must be carried out to obtain new routing tables if a non-supported case is detected. However, once new tables have been computed, TFTR can be used again in order to tolerate additional faults.

As future work, we plan to analyze more deeply 3D tori and to evaluate other regular topologies as 2D and 3D mesh networks. Also, Multistage networks (MINs), like Fat-Trees topologies, will be analized. We also plan to improve the path searching method in order to obtain a set of disjoint paths that supports a larger number of faults in a bounded scenario of SLs.

References

[1] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. Seizovic and W. Su, "Myrinet - A gigabit per second local area network," *IEEE Micro*, pp. 29–36, Feb. 1995.

[2] R.V. Boppana and S. Chalasani, " Fault-tolerant wormhole routing algorithms for mesh networks," in IEEE Trans. on Computers, vol. 44 no 7., pp. 848-864, July 1995.

[3] R.Casado, A Bermúdez, J. Duato, F.J. Quiles, and J.L. Sánchez, "A protocol for deadlock-free dynamic reconfiguration in high speed local area networks," in IEEE Trans. on Parallel and Distributed Systems, Vol. 12, No. 2, pp. 115-132, 2001.

[4] L.Cherkasova, V. Kotov, and T. Rokicki, " Fibre channel fabrics: Evaluation and design," in Proc. of 29 th Int. Conf. On System Sciences, Feb. 1995.

[5] A. A. Chien and J. H. Kim, "Planar- adaptive routing: Low-cost adaptive networks for multiprocessors," in Proc. of the 19th Int. Symp. on Computer Architecture, pp. 268-277, May 1992.

[6] W.J. Dally and H. Aoki, "Deadlock-free adaptive routing in multicomputer networks using virtual channels," in IEEE Trans. on Parallel and Distributed Systems, vol 4, no 4. pp 466-475, April 1993.

[7] W. J. Dally and C. L. Seitz, "Deadlock-free Message Routing in Multiprocessors Interconnection Networks", *IEEE Trans. on Computers*, vol. C-36, no. 5, pp. 547-553, May 1987.

[8] B.V. Dao, J. Duato, and S. Yalamanchili, "Configurable flow control mechanisms for fault-tolerant routing," in Proc. of the 22nd Int. Symp. on Computer Architecture, pp 220-229, June 1995.

[9] J. Duato, "A theory of falult-tolerant routing in wormhole networks," in Proceedings of the Int. Conf. on Parallel and Distributed Systems, pp. 600-607, December 1994.

[10] J. Duato, S. Yalamanchili, and L. Ni. Interconnection Networks. An Engineering Approach. Morgan Kaufmann Publishers, 2003.

[11] P.T. Gaughana and S. Yalamanchili, "A family of faulty-tolerant routing protocols for direct multiprocessor networks," in IEEE Trans. on Parallel and Distributed Systems, vol 6. no 5. pp 482-497, May 1995.

[12] C.J. Glass and L.M.Ni, "The turn model for adaptive routing," in Proceedings of the 19th Int. Symp. on Computer Architecture, pp. 278-287, May 1992.

[13] InfiniBand^{SM} Trade Association, *InfiniBand^{TM} Architecture. Specification Volume 1. Release 1.0*. Available at `http://www.infinibandta.com/`.

[14] IBM BG/L Team, "An Overview of BlueGene/L Supercomputer", ACM Supercomputing Conference, 2002.

[15] S. Konstantinidou and L. Snyder, "The Chaos router," in IEEE Trans. on Computers, vol. 43 no.12, pp. 1386-1397,Dec. 1994.

[16] D.H. Linder and J.C. Harden, "An Adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes," in IEEE Trans. on Computers, vol. C-40 no 1, pp.2-12, Jan. 1991

[17] P. López, J. Flich, and J. Duato, Deadlock-free Routing in InfiniBand^{TM} through Destination Renaming, in *Proc. of 2001 Int. Conf. on Parallel Processing (ICPP'01)*, Sept. 2001.

[18] S.S. Mukherjee, P. Bannon, S. Lang, A. Spink, D. Webb, "The Alpha 21364 Network Architecture," in IEEE Micro, Jan-Feb 2002.

[19] J. C. Sancho and A. Robles, "Improving the Up*/Down* Routing Scheme for Networks of Workstations", in Proc. of the *Euro-Par 2000*, Aug. 2000.

[20] J.C. Sancho, A.Robles, J. Flich, P.Lopez, and J. Duato, "Effective methodology for deadlock-free minimal routing in InfiniBand networks," in Proc. of the 2002 Int. Conf. on Parallel Processing, IEEE Computer Society, 2002.

[21] J.C. Sancho, A. Robles, P. Lopez, J. Flich, and J. Duato, "Routing in InfiniBand^{TM} Torus Network Topologies," in IEEE Proc. of Int. Conf. on Parallel Processing, 2003.

[22] M. D. Schroeder et al., "Autonet: A High-speed, Self-configuring Local Area Network Using Point-to-point Links", *Journal on Selected Areas in Comm.*, vol. 9, no. 8, pp. 1318-1335, Nov. 1991.

[23] F. Silla and J. Duato, "High-Performance Routing in Networks of Workstations with Irregular Topology", *IEEE Trans. on Parallel and Distributed Systems*, vol. 11, no. 7, pp. 699-719, Jul. 2000.

[24] http://www.top500.org

0-7695-2132-0/04/$17.00 (C) 2004 IEEE