Making an SCI Fabric Dynamically Fault Tolerant

Håkon Kvale Stensland[1,2], Olav Lysne[1,2], Roy Nordstrøm[3], and Hugo Kohmann[3]
[1] *Simula Research Laboratory, Lysaker, Norway*
[2] *University of Oslo, Department of Informatics, Oslo, Norway*
[3] *Dolphin Interconnect Solutions, Oslo, Norway*

Abstract

In this paper we present a method for dynamic fault tolerant routing for SCI networks implemented on Dolphin Interconnect Solutions hardware. By dynamic fault tolerance, we mean that the interconnection network reroutes affected packets around a fault, while the rest of the network is fully functional. To the best of our knowledge this is the first reported case of dynamic fault tolerant routing available on commercial off the shelf interconnection network technology without duplicating hardware resources. The development is focused around a 2-D torus topology, and is compatible with the existing hardware, and software stack. We look into the existing mechanisms for routing in SCI. We describe how to make the nodes that detect the faulty component do routing decisions, and what changes are needed in the existing routing to enable support for local rerouting. The new routing algorithm is tested on clusters with real hardware. Our tests show that distributed databases like MySQL can run uninterruptedly while the network reacts to faults. The solution is now part of Dolphin Interconnect Solutions SCI driver, and hardware development to further decrease the reaction time is underway.

1. Introduction

Scalable Coherent Interface (SCI) [1] is a mature interconnect technology, originally standardized by IEEE in 1992 as IEEE Standard 1596. The intention with SCI was to design an interconnect technology between processors, memory and I/O devices. Today SCI is primarily used as an interconnect technology for clusters, distributed databases and in embedded solutions. In this paper we describe a mechanism that adds fault tolerance to SCI interconnects.

Fault-tolerant routing methods can be divided into three groups: Reconfiguration-based, Source-based and Switch-based. The reconfiguration-based methods include static and dynamic reconfiguration. Static reconfiguration involves halting the network and reconfiguring the routing-function in order to avoid the failed component while the network is down. Dynamic reconfiguration means reconfiguring the interconnection network while it is up and running. Although several methods for dynamic reconfiguration have been described in literature, e.g. the Double Scheme [2], none of them have yet been implemented in a real system.

A source-based rerouting approach is based upon the principle that the sender detects the problem, and resends the packet using another path. In [3] intermediate nodes on the path to the destination is utilized to adaptively route packets to the destination. In switch-based fault tolerance, the source does not have to know of the fault in the network, because the switches hide the network fault by rerouting messages around the faulty components. In [4, 5] virtual channels and a number of Up* / Down* graphs is used to achieve the redundant properties. Both these solutions use Virtual Channels, and neither of them has been implemented on real systems.

By dynamic fault tolerance we mean the ability of the network to handle a fault dynamically, without stopping the network operation. Dynamic fault tolerance methods include dynamic reconfiguration, source-based rerouting methods, and switch based rerouting methods. In this work we describe a dynamic method that is partly switch based, and partly source based. The reason for this duality is that the SCI topologies studied are direct networks, where the nodes act as both sources and switches.

There exist methods for fault tolerance for existing point-to-point technologies, such as Myrinet [6], and InfiniBand [7]. However, to the best of our knowledge, the existing work has taken a static reconfiguration approach, by stopping the network (and the applications), reconfiguring, and starting again. The

978-1-4244-1694-3/08/$25.00 ©2008 IEEE

traditional approach to dynamic fault tolerance in the industry has been to duplicate all interconnect resources. This approach is expensive, and the resources in the network might not be fully utilized. Therefore we want to use the inherent redundant resources in a single network to provide fault tolerance.

SCI is a ring-based technology, and from one point of view, the closest approach to ours is found in the Element Interconnect Bus (EIB) in the Cell Broadband Engine. EIB contains two pairs of counter-rotating rings for communication between the PPE (PowerPC core), memory controller, I/O controller and the eight synergistic processor elements (SPE) [8]. This is exploited to give the Cell a degree of fault tolerance. In our framework, this would, however, correspond to having several redundant interconnection networks, thus the method would not be applicable when there is a requirement of a single fabric.

Our effort can be seen as an exercise in providing dynamic fault tolerance to an exiting and mature technology that has not been designed for such advanced features. This means that some of the techniques we use had to be implemented in the drivers, rather than in specialized hardware. Still we were able to achieve the main objective of the fault tolerant routing method developed, namely that the local switch based re-routing in the network should take less than 1 second. The target value is set to 1 second because this enable us to support uninterrupted operation of distributed database solutions like MySQL Cluster [9]. Hardware support for our techniques is currently under development, and this will substantially reduce the reaction time of the method.

The paper is structured as follows. In section 2 we give a brief overview of SCI and the technology at hand. In section 3 we present the alternative methods for fault tolerant routing in SCI, and give a brief overview of our approach. In section 4 we discuss the necessary changes to the routing tables and in section 5 we present the needed changes in the SCI driver. Measurement results and performance evaluation is given in section 6 before we conclude in section 7.

2. Scalable Coherent Interface

The SCI standard is designed with unidirectional point-to-point links, and with a design-goal to implement a fully hardware based distributed shared memory. The basic topology in the SCI standard is a unidirectional ring. A ring topology does, however, have limited scalability. For that reason, there exist line cards with more than one SCI interface, allowing a node to be part of more than one SCI ring. This feature can be used to build more complex ring based topologies such as is illustrated in Figure 1.

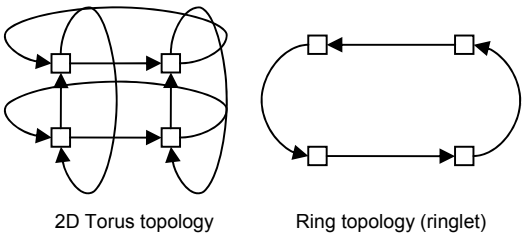

Figure 1. Example of topologies in SCI

The line cards that we have used for this development contain two SCI interfaces, and this allowed us to build a two dimensional torus. The cards contain two Dolphin Interconnect Solutions Link Controller 3 (LC3) [10] and a PCI to SCI Bridge (PSB66) [11]. The architecture of the card is illustrated in Figure 2. The link controllers and the PSB66 are connected with a bus called B-link. The B-link is a back-end interface between a PSB and up to eight link controllers. The primary topologies used in Dolphin's SCI implementation today are: a single ring, or several rings in a 2-D or 3-D torus topology, meaning that there are either two or three link controllers on each card. In the experiments we report in this paper, we have used cards with two LC3 chips, and the system was interconnected into a 2D torus (Figure 1).

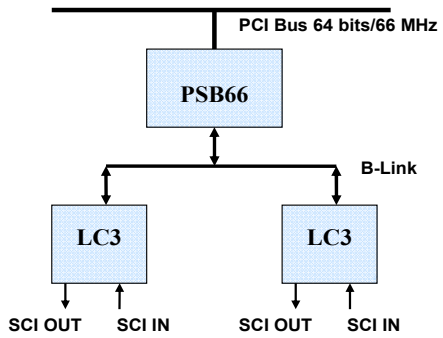

Figure 2. Architecture on Dolphin 2-D SCI Card

The current routing algorithm in the 2-D topology is based upon dimension-order routing. A packet will be sent to the recipient nodes coordinate in one dimension before the dimension is changed. To ease the explanation, we will throughout the paper assume that the horizontal (X-direction) ring is first by default as illustrated by the dashed arrow in Figure 4 (the dotted arrow in figure 4 will be explained below).

The fault-model assumed by our approach is inherited from properties of the SCI hardware. If a link in a ring fails, then the entire ring will fail. The reason for this is that the SCI ring protocol consists of packets and echoes, and no packet transmission over the link is complete until the echo has been received by the sender of the packet. Since the ring is unidirectional, all packet transmissions will be affected – either directly or indirectly because of broken transmission of the echo. If a node dies in a 2-D cluster, both rings connected to the node will be broken for the same reason.

As illustrated in Figure 3, a link controller has two sets of routing tables: One table is called the Link Routing Table. This table decides if a packet should be taken off the SCI ring and sent up to the B-link. The other table called the B-link Routing Table is used to decide which link controller the packet will be sent out on. All packets are routed based on the destination address.

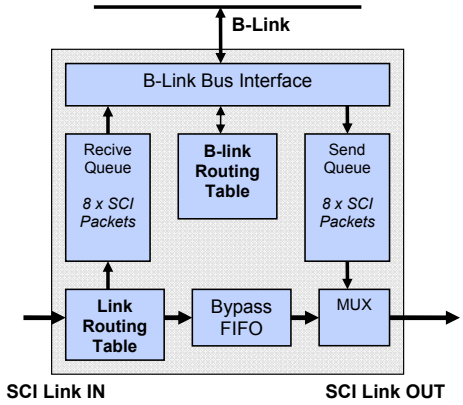

Figure 3. Routing-tables in Dolphin Link Controller 3

3. Alternatives for fault-tolerant routing in SCI

SCI is a mature technology that was specified at a time when fault tolerance in the interconnect was not considered as important as it is today. When we set out to implement fault tolerant routing on existing SCI hardware, the task was therefore to utilize the existing features of the components in order to extract the fastest possible reaction to faults that these components allowed.

There are three possible approaches to fault tolerant routing in an SCI interconnection network. The first is to enable fault tolerant routing based upon a static reconfiguration of the routing tables. In this solution, the routing function is controlled by an application running on a centralized front-end node. Through an alternative infrastructure, e.g. an Ethernet connecting all of the nodes, the cluster nodes report any issues (i.e. bad cables) to the front-end. In case a problem is reported, all communication in the cluster is halted, while the front-end calculates and distributes new routing tables to all nodes in the cluster. If the front-end is unable to reach some nodes, they will be considered dead and removed from the routing tables. Communication can resume when all nodes accept the new routing tables.

In this solution, routing tables need to be calculated, e.g. using Dijkstra's shortest-path-first algorithm [12]. The drawback for static reconfiguration is a longer reconfiguration time. Measurements we have conducted have shown that communication downtime in a standard 4-node cluster, when an SCI link is broken is measured to almost four seconds, thus a factor 4 above our threshold. Furthermore, this approach will affects all communicating pairs, not only those that would communicate through the failed component.

Another approach is the use of redundant hardware, where two ore more SCI adapters can operate as one virtual adapter, thus duplicating the entire interconnect. This solution is also available commercially. Its drawback is that it will increase the overall cost, because of the need for two sets of adapters, and a dual fabric. Furthermore, redundant hardware is unable to offer any protection in the case of a dead node, as both adapters will lose power.

The third approach, which we develop in this paper, is to route affected packets around the faulty area without halting operation in the rest of the cluster. When a node detects a problem with a ring, packets bound for the problematic ring will be sent downstream, so that it can be routed by the next node, and finally be routed to the destination as illustrated by the dotted path between node A and C in Figure 4. The approach guarantees support for one failure, support for two or more faults are not guaranteed. This solution is implemented and evaluated in this paper.

The implementation consists of two parts. The first part consists of the changes done to the routing tables to make the SCI-nodes ready to accept packets that are on their way around a fault. The second part consists of support in Dolphin's SCI driver for making local rerouting decisions in the node that detects the fault.

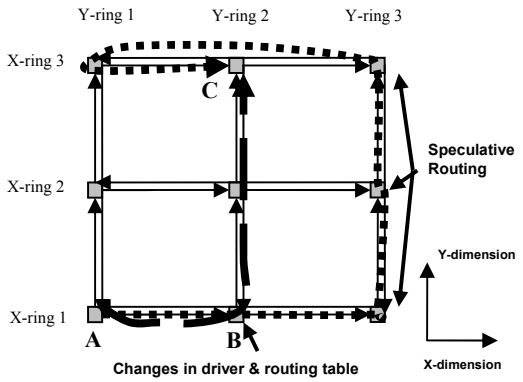

Figure 4. Routing in a 9-node 2-D SCI topology

4. Changes to the routing algorithm

Several aspects of the routing had to be modified in the implementation of our approach. First the Link Routing Tables needed to be modified to enable the packets to continue on the current dimension if there is a problem with the next dimension's ring. In all dynamic fault tolerant routing, this is generally the easy part, as this is done in the node that detects the fault. The more complex problem is related to the nodes that have no information of the faulty ring. When a fault occurs, they need to be able to handle packets that they do not see under normal operation, and they need to be able to do this without being notified of the nature of the fault.

The main idea of our algorithm is to let a packet that is traveling in the X-direction, and that should have been forwarded onto a faulty ring in the Y-direction, simply continue one more step in the X-direction, as is illustrated in figure 4. The dashed arrow from node A to node C represents the normal path according to dimension-order routing. If the Y-ring (labeled Y-ring 2) fails, node B will detect that its vertical ring has stopped working, and will continue forwarding the affected packets along the X-dimension (the dotted arrow). The node downstream from B will not know of the fault, but by inserting speculative routing entries on the downstream node, we are able to route the packets towards their destination.

An important part of this algorithm is to set up a downstream node to route packets that was supposed to do a dimension change on the upstream node. This must, however, be done in such a way that it does not interfere with the regular routing in the fault free case. The important observation in this case is the following: Assume two adjacent nodes A and B on a ring. Assume further that A is upstream of B. Then under normal fault free operation, node B will never see any packet that is supposed to be taken off the ring by A. Furthermore, B is the only node on the ring that can be described in this way, as all other nodes on the ring will see packets that B send to A. Therefore, the routing entries must be set up in the following way:

- All necessary routing entries for dimension order routing with the X-dimension first must be present.
- On rings in the X-dimension, all nodes must have additional link routing table entries that pick up packets that under normal operation should have been picked up by its upstream node.
- On a ring in the Y-dimension, all nodes must have additional link routing tables for packets that travel in this Y-dimension ring because faults, but that would not be on this ring under normal fault free operation.

The only status available to an SCI node is the status of its own link controllers. If a problem is detected on a ring, the driver will disable the faulty link controller. When this is done, the local routing tables need to be changed. How this is done is handled in the next section.

SCI fabrics are prone to deadlocks if the routing algorithm is not carefully selected. In the topologies that we consider here, dimension order routing is used in the fault free case, and it is well known that this routing algorithm is deadlock free. In the presence of faulty rings, however, we modify the routing algorithm so that it is no longer strictly dimension order. It can be shown that with a faulty ring, the routing algorithm remains deadlock free. Space restrictions disallows us to repeat the full analysis here, but the methodology is based on buffer dependency graphs, and is similar the one that is described in [13]

5. Implementation in the driver

In the previous section we described the part of the rerouting mechanism that can be set up a priori. There are, however, parts of the mechanism that need to be handled by the drivers at the time when the fault is detected. The most obvious part is that when a node detects that one of its rings is dead, the Link Routing Table on the link controller connected to the functional ring must be altered. This is necessary to ensure that the node lets packets destined for nodes on the dead ring pass on to its downstream node. This section gives an overview of how this is handled, and also explains the limitations of the existing hardware with

respect to reaction time. Hardware support for the functionality that we describe here is under development, and this is expected to give near to immediate reaction to faulty rings.

The error handling in Dolphin Interconnect Solutions SCI driver is divided into two sections as shown in Figure 5. The first section is called CableNotOk and has the responsibility for detecting faulty SCI rings on the node that detects the fault. The error is detected by checking if certain interrupt bit is set. The second section is called ReadyToGo, and has the responsibility to calculate new routing-tables, and set up the card.

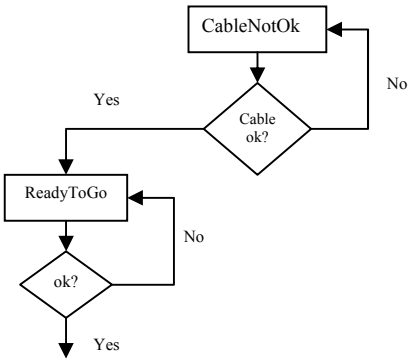

Figure 5. Simplified error handling overview in Dolphin's SCI driver

The first section, CableNotOk is called when the card receives a CableNotOk interrupt related to the link controller. When the driver enters the CableNotOk-section a timer is initialized. This timer is currently set to 50 milliseconds. During this time, the CableNotOk handler in the driver will disable all interrupts, while all link controllers are checked. If a connectivity problem is detected, the affected link controller will be disabled, and status stored before continuing with re-enabling interrupts. If no problems are detected, the driver will go directly to enabling interrupts. After the previously started timer is finished, the driver will check for any remaining errors. If errors are found, a new pass through the CableNotOk handler will be forced. If no problems are present, the driver will continue to the next section.

The next section of the driver is called ReadyToGo, and it is responsible for the initialization of the link controllers, changing of routing tables in the link controllers and checking the connectivity on the SCI link. ReadyToGo is also controlled by a timer, similarly to the CableNotOk handler. The default value to this timer is currently set to 200 milliseconds. The first event in ReadyToGo is the initialization of the link controllers with basic information like node ID, link frequency and the current topology selected. Next step is to calculate the routing tables. These calculations are based on a normal default routing table, and the status stored away in the CableNotOk handler. After the routing tables are completed, the link controllers are set up, and the process to check the SCI link is started. The link controllers will send a probe request to itself around the SCI ring. This is done to make sure that the ring is initialized and ready.

A workaround was also needed to handle side-effect of our fault-tolerant routing caused by the SCI *scrubber*. The scrubber is a hardware mechanism built to protect SCI rings from packets without a valid destination. A potential problem occurs when a Y-ring is broken, and a node upstream of this broken Y-ring sends packets to a node in the broken Y-dimension. In this case the upstream node would continue to forward the packet on the X-dimension, until the packet has traveled around the ring back to the sender, as illustrated by the dashed arrow in Figure 6. A packet that travels around the entire ring without being taken by any node will be removed from the ring by the scrubber. The scrubber can not be disabled. This was solved by using the low-level probe mechanism in the SCI driver to query the upstream nodes Y-ring status. In Figure 6, node C would probe node D, and detect the broken Y-ring. A local list with broken links will then be updated, and another reset of the driver will be issued. This is done to incorporate the broken Y-rings status into the routing table. If a broken Y-ring is detected on the upstream node, all packets for destinations on the affected Y-ring will be sent directly out on the local Y-ring.

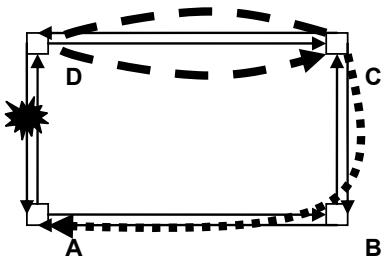

Figure 6. Workaround to support broken Y-rings

A final feature that was implemented in the driver is the ability to re-enable a SCI link when a cable is re-inserted. When a SCI link is disabled, the link controllers bit in the IO interrupt control register is

masked, and a watchdog is initialized. This watchdog checks the masked register every second to se if there is a change in cable status. If a change is detected, the driver will initialize a re-routing process. This feature is, however, not the focus of this paper.

6. Performance Evaluation

To evaluate the performance of the fault-tolerant routing algorithm, two test clusters are set up as illustrated in Figure 7. The first cluster is a 4 node cluster, with Intel Xeon 2,8GHz CPUs, 1GB RAM, Intel E7520 Lindenhurst chipset, Dolphin Interconnect PCI (64-bit) SCI 2D Card (D334), and run the Linux 2.6.9-42.EL x86_64 kernel.

The second cluster is a 9-node cluster with AMD Athlon 64 X2 4200+ dual-core CPUs, 2GB RAM, nVidia nForce 570 SLI chipset, Dolphin Interconnect PCI Express SCI 2D Card (D352) and run the Linux 2.6.9-42.0.3-EL x86_64 kernel.

The driver version is Dolphin DIS 3.1.10. The tests are done with two tools: A tool called scibench2 to measure bandwidth and one-way latency, and a tool called downtime to measure the communication downtime while the cluster is doing rerouting. Both tools are available as open source in Dolphin's SCI driver.

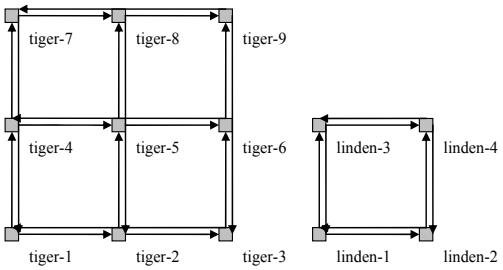

Figure 7. Cluster setup with hostname

The first test is on the 9-node cluster illustrated in Figure 7. We have communication from tiger-1 to tiger-8, from tiger-3 to tiger-7 and from tiger-1 to tiger-3. During the test, a cable on the X-ring between tiger-1, tiger-2, and tiger-3 is removed.

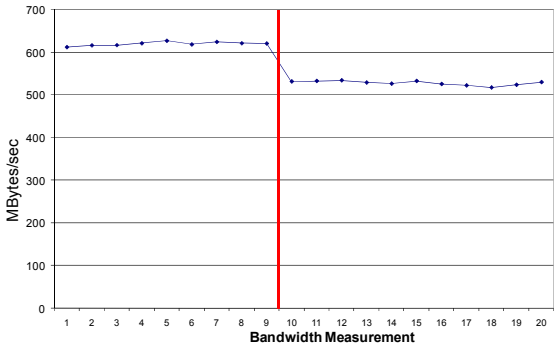

Figure 8. 9-nodes – Total bandwidth – Ring down

Figure 8 is a plot of total bandwidth in the cluster during the test. Before the ring is broken, an even amount of bandwidth is available. After the error has occurred, a drop from around 620MB/sec to around 530MB/sec can be observed. This is expected to be due to more competing traffic in the fabric, and because we are removing 1/6 of the available aggregate bandwidth in the cluster. The average communication downtime seen in during this test is *401.15 milliseconds*, while the worst case communication downtime is measured to *597.16 milliseconds*.

The average latency measured in Figure 9 is showing a small increase on about 0.20 microseconds. Increase can be explained with the increased amount of competing traffic, since a ring is missing, and the fact that some of the traffic gets a longer path after the reconfiguration.

MySQL Cluster has also been tested with the 9-nodes and 4-nodes. Cables have been removed, and the database has continued to run uninterrupted in all the scenarios tested. We have also tested powering down a node. This obviously requires MySQL to reconfigure, because of the lost node, but the rerouting process was finished well before the 1000 millisecond deadline.

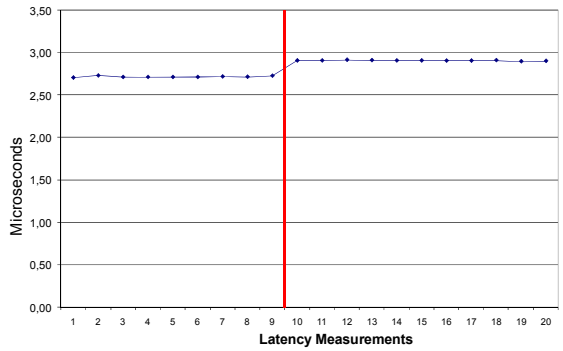

Figure 9. 9-nodes - Average latency - Ring down

The second test is done on the 4-node cluster. In this test we run communication from linden-1 to linden-4. In this test, we did a complete power down of the node linden-2. This disabled both the X-dimension link from linden-1 to linden-2 and the Y-dimension link from linden-2 to linden-4. The length of the packets path will be the same before and after node linden-2 is powered down.

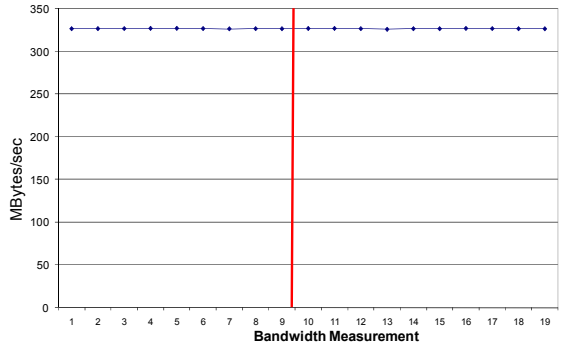

Figure 10. 4-nodes – Total bandwidth - Node down

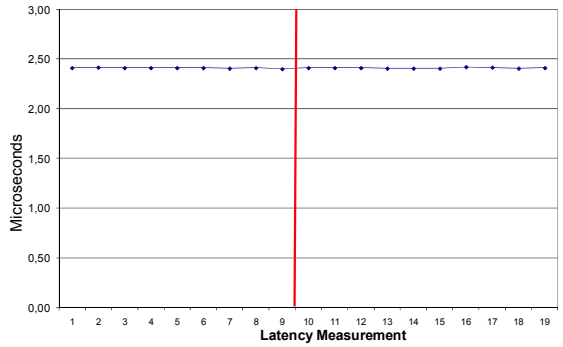

Figure 11. 4-nodes - Average latency - Node down

In Figure 10 and Figure 11 total bandwidth and average latency for the second test shows no degradation in either bandwidth or latency after the power to linden-2 is switched off.

The average communication downtime in this test is *770.87 milliseconds*. And the worst-case is *805.50 milliseconds*. This is a higher number than the first test, and is due to the fact that two of the remaining nodes in the cluster have to do a remote probe for Y-ring status, and since is has changed, another run through the rerouting mechanism must be taken. In order to incorporate the remote Y-ring status in the routing tables. We have made an analysis on where in the reconfiguration process most of the time is used, and this is in the CableNotOk timer, currently set to 50 milliseconds, and in the ReadyToGo timer currently set to 200 milliseconds. Both these timers are to make sure that all the nodes in the cluster are in a ready before the process continues.

Tests show that the extra pass through the rerouting algorithm, whenever a Y-ring is broken does increase the reconfiguration time. We will now discuss some alternative solutions to probing the downstream nodes Y-ring status:

1) Add an extra redundant Y-ring: The first alternative is to add an extra Y-ring in the cluster. On a 3x3 cluster this would require three extra nodes with extra hardware and cables. These nodes would also only be available for forwarding packets. It would have been impossible with existing hardware to filter out only the traffic affected by the broken Y-ring. This could have resulted in this ring quickly becoming a hotspot.

2) Use 3D cards and utilize the Z-ring as a redundant Y-ring: This alternative suggests using SCI-cards with three link controllers, and use the third controller (Z-dimension) as a redundant Y-ring. This is easy to implement from a technical standpoint. The negative with is that it would increase the cost, since it requires more expensive hardware.

3) Make hardware changes for fault tolerant routing: This could easily be solved if the existing SCI hardware had the ability to support two addresses per node, and utilize one address to route X-dimension first, and one address to route Y-dimension first. When an error is detected, the destination address on the packets could have been changed, and the packet handled differently. This option has already been implemented. A revised version of Dolphin Interconnect Solutions LC3, called P2S is currently being tested. This new revision increases the routing tables (Routing RAM) from 256 entries to 8192

entries, and enables a node the have two addresses. The 14th bit in the Routing RAM is reserved to indicate which of the nodes two addresses to use. If bit 14 is set to 0, then X-dimension is routed first, and if it is set to 1, Y-dimension is routed first.

7. Conclusion and further work

In this paper we have developed a dynamically fault-tolerant routing method for SCI networks. This method has been implemented on existing hardware from Dolphin Interconnect Solutions, and is now a part of the official Dolphin SCI driver. Performance tests show very little degradation in performance before and after a fault occurs. Tests have also shown that we are able to support rerouting faster than the 1 second goal that was set to support the distributed MySQL database server.

Samples of the next revision SCI hardware are up and running in the lab at Dolphin, and it is expected to be available to the market shortly. This new hardware has hardware support for dynamic fault-tolerant routing, by implementing the ability to use two addresses per node, and the ability to use independent routing tables for both addresses. The limitation of the current generation of SCI hardware is the need to probe the downstream node on the X-dimension link, and ask for cable status on the Y-ring. This step is the most time-consuming step in the process. In the new SCI revision, P2S does not require this step, because the hardware will automatically change and route Y-dimension first if an error occurs. With the probe for remote Y-ring status step removed, fast re-routing times as low as 100 milliseconds is expected.

8. References

[1] IEEE, *Std. 1596 "Scalable Coherent Interface (SCI)"*. IEEE, 1992.

[2] T. M. Pinkston, R. Pang, and J. Duato. *"The Double Scheme: Deadlock-free Dynamic Reconfiguration of Cut-Through Networks"*. in *International Conference on Parallel Processing*. 2000.

[3] M. E. Gómez, N. A. Nordbotten, J. Flich, P. López, A. Robles, J. Duato, T. Skeie, and O. Lysne, *"A Routing Methodology for Achieving Fault Tolerance in Direct Networks"*. IEEE Transactions on Computers, 2006. vol. 55, no. 4: pp. 400-415.

[4] O. Lysne and T. Skeie, *"Load Balancing of Irregular System Area Network through Multiple Roots"*. International Conference on Communication in Computing, 2001.

[5] I. Theiss and O. Lysne, *"FRoots, a Fault Tolerant and Topology-Flexible Routing Technique"*. 2005.

[6] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W. Su, *"Myrinet: A Gigabit-per-Second Local Area Network.* IEEE MICRO, 1995.

[7] InfiniBand Trade Association, *"InfiniBand Architecture Specification"*.

[8] D. C. Pham, T. Aipperspach, D. Boerstler, M. Bollinger, R. Chaudhry, D. Cox, and P. Harvey, *"Overview of the Architecture, Circuit Design, and Physical Implementation of a First-Generation Cell Processor"*. IEEE Journal of Solid-State Circuits, 2006. vol. 41, no. 1: pp. 179-196.

[9] MySQL AB, *"MySQL Cluster Architecture Overview"*. 2004.

[10] Dolphin Interconnect Solutions, *"Link Controller 3 Specification D666 - LC3"*. 2002.

[11] Dolphin Interconnect Solutions, *"PSB66 Specification D667"*. 2001.

[12] E. W. Dijkstra, *"A note on two problems in connexion with graphs"*, in *"Numerische Mathematik 1"*. 1959. pp. 269-271.

[13] O. Lysne and S. Gjessing, *"Constructing SCI-Configurations that are free from Deadlocks"*. In *International Workshop on SCI-based High-performance Low-Cost Computing*. 1996