Multi-homed fat-tree routing with InfiniBand

Abstract—**For clusters where the topology consists of a fat-tree or more than one fat-tree combined into one subnet, there are several properties that the routing algorithms should support, beyond what exists today. One of the missing properties is that current fat-tree routing algorithm does not guarantee that each port on a multi-homed node is routed through redundant spines, even if these ports are connected to redundant leaves. As a consequence, in case of a spine failure, there is a small window where the node is unreachable until the subnet manager has rerouted to another spine.**

In this paper, we discuss the need for independent routes for multi-homed nodes in fat-trees by providing real-life examples when a single point of failure leads to complete outage of a multi-port node. We present and implement the methods that may be used to alleviate this problem and perform simulations that demonstrate improvements in performance, scalability, availability and predictability of InfiniBand fat-tree topologies. We show that our methods not only increase the performance by up to 52.6%, but also, and more importantly, that there is no downtime associated with spine switch failure.

I. Introduction

The fat-tree topology is one of the most common topologies for high performance computing clusters today where, for example, it is used in the currently fastest supercomputer in world - MilkyWay-2 [1]. Moreover, for clusters based on InfiniBand (IB) technology the fat-tree is the dominating topology. Fat-tree IB systems include large installations such as Stampede, TGCC Curie and SuperMUC [2]. There are three properties that make fat-trees the topology of choice for high performance interconnects: deadlock freedom, the use of a tree structure makes it possible to route fat-trees without special considerations for deadlock avoidance; inherent fault-tolerance, the existence of multiple paths between individual source destination pairs makes it easier to handle network faults; full bisection bandwidth, the network can sustain full speed communication between the two halves of the network.

For fat-trees, as with most other topologies, the routing algorithm is crucial for efficient use of the underlying topology. The popularity of fat-trees in the last decade led to many efforts to improve their routing performance. These proposals, however, have several limitations when it comes to flexibility and scalability. This also includes the current approach that the OpenFabrics Enterprise Distribution [3], the de facto standard for InfiniBand system software, is based on [4], [5]. One problem is the static routing used by IB technology that limits the exploitation of the path diversity in fat-trees as pointed out by Hoefler et al. in [6]. Another problem with the current routing is its shortcomings when routing oversubscribed fat-trees as addressed by Rodriguez et al. in [7]. A third problem, and the one that we are analyzing in this paper, is that for hosts with two or more host-channel adapter ports connected to the same fat-tree-based subnet, the routing algorithm does not guarantee that independent (relative to single points of failure) root switches are chosen for the corresponding down paths.

In this paper, we discuss the need for independent routes for multi-homed nodes in fat-trees by providing real-life examples when a single point of failure led to fatal consequences. We present and implement the methods that may be used to alleviate this problem and perform experiments and simulations that demonstrate the usefulness of our approach. There are two key aspects of the redundant path implementation:

- Identifying paths that need to be routed in a mutually redundant way.
- Ensuring that the paths are in fact redundant.

The rest of this paper is organized as follows: we discuss related work in Section II and follow with introducing the InfiniBand Architecture and fat-tree routing in Section III. We continue with a discussion of our enhancements in Section IV. Next, we describe the experimental setup in Section V followed by the experimental analysis in Section VI. Finally, we conclude in Section VII.

II. Related Work

There was much research done in the topics of fat-tree routing, multipathing, dynamic reconfiguration and fault-tolerance. First of all, there are proposals [4] and proprietary implementations of adaptive routing algorithms available [8] that extend IB's destination routing capabilities such that traffic directed to a given endpoint can traverse different paths through the network. However, the presented adaptive routing is reactive (loss of throughput during the adaptation phase), not deterministic, is only available for new switch units (if some switches do not support adaptive routing, it leads to overall slowdown of the adaptive routing manager) and is not yet widely available. Additionally, it does not give any guarantee that the chosen paths will be mutually independent and some transport layers like IB Reliable Connected (RC) cannot be routed in an adaptive way due to the possible out of-order delivery [9].

Next, there were proposals to use other routing algorithms [10]–[13] to achieve multi-pathing. However, when these algorithms are applied to regular topologies like fat-trees, they may not take all the properties of a regular topology into account and cannot deliver optimal performance, and, again, there is no guarantee that a multi-homed host can be reached using independent paths.

Furthermore, there were proposals to add LID Mask Control (LMC) support to fat-tree routing [14], however, these attempts failed because the Open Subnet Manager (OpenSM) still does not support LMC for fat-tree routing. Moreover, LMC is not a solution in a multi-homed environment and it does not guarantee that the chosen path will use independent switches.

Later, there was research done on multipath routing for Extended Generalized Fat-Trees (XGFTs) [15]. However, it was shown that XGFTs cannot be used to represent many real-life topologies [9] and the aim of that work is purely theoretical as the authors failed to consider how real enterprise systems are built.

Lastly, there were several proposals to use oblivious routing in fat-tree systems [5], [7], [16]–[18]. The most successful one is [5], which is the default fat-tree routing for OpenSM. These routing algorithms assume that there are enough links in a fat-tree to reconfigure the routing without much performance loss in case of failures. However, all of these algorithms work on the port level, that is, they treat each port as an independent node and do not consider the extra fault-tolerance possibilities that are provided by the additional ports at the same node. We will show that using such an oblivious routing algorithm may lead to a total lack of connectivity even in cases where a node is multi-homed.

Unlike previous research on IB routing algorithms, we discuss and analyze various enterprise fat-trees where nodes are multi-homed, i.e. a single node is connected to two or more parts of the fat-tree through multiple ports. We focus on real-life enterprise systems where fault-tolerance, reachability and performance are of the utmost importance. Our work is partially based on [19], [20] where we also analyzed fat-trees, however, we did not focus on the multi-homing aspect. In this work we widen the scope and, first, consider the enterprise fat-trees with multi-homed nodes and, second, propose and evaluate a fault-tolerant routing algorithm.

III. Technical Background

The InfiniBand Architecture (IBA) supports a two-layer topological division. At the lower layer, IB networks are referred to as subnets, where a subnet consists of a set of hosts interconnected using switches and point-to-point links. At the higher level, an IB fabric constitutes one or more subnets, which are interconnected using routers. Hosts and switches within a subnet are addressed using LIDs and a single subnet is limited to 49151 LIDs. Whereas LIDs are the local addresses valid only within a subnet, each IB device also has a 64-bit *Global Unique IDentifier* (GUID) burned into its non-volatile memory. A GUID is used to form a GID - an IB layer-3 address. A GID is created by concatenating a 64-bit subnet ID with the 64-bit GUID to form an IPv6-like 128-bit address. In this paper, we will focus on port GUIDs, i.e. the GUIDs assigned to every port connected to the IB fabric.

A. Subnet Management

Every IB subnet requires at least one subnet manager (SM), which is responsible for initializing and bringing up the network, including the configuration of all the IB ports residing on switches, routers, and host channel adapters (HCAs) in the subnet. At the time of initialization the SM starts in the *discovering state* where it does a sweep of the network in order to discover all switches and hosts. During this phase it will also find other SMs and negotiate who should be the master SM. When this phase is completed the SM enters the *master state*. In this state, it proceeds with LID assignment, switch configuration, routing table calculations and deployment, and port configuration. At this point the subnet is up and ready for use. After the subnet has been configured, the SM is responsible for monitoring the network for changes.

B. Fat-Tree Routing

A major part of the SM's responsibility is routing table calculations. Routing of the network aims at obtaining full connectivity, deadlock freedom, and proper load balancing between all source and destination pairs in the local subnet. Routing tables must be calculated at network initialization time and this process must be repeated whenever the topology changes in order to update the routing tables and ensure optimal performance.

In case of the fat-tree routing algorithm, the routing function iterates over an array of all leaf switches. When a leaf switch is selected, for each end-node port connected to that switch (in port numbering sequence), the routing function routes towards that node. All of that is performed by $route\_to\_cns$ function, for which a pseudocode is presented in Algo. 1. When routing the particular LIDs, the function goes up one level to route the downgoing paths, and next, on each switch port, it goes down to route the upgoing paths. This process is repeated until the root switch level is reached. After that the paths towards all nodes are routed and inserted into the linear forwarding tables (LFTs) of all switches in the fabric. The function route_downgoing_by_going_up() is a recurrence function whose main task is to balance the paths and call the route_upgoing_by_going_down function, which routes the upward paths in the fat-tree towards destination through the switch from which it was invoked.

Algorithm 1 $route\_to\_cns()$

```
Require: Addressing is completed
Ensure: All $hca\ ports$ are routed
1: for $sw_{leaf} = 0$ to $max\_leaf\_sw$ do
2:     for $sw_{leaf}.port = 0$ to $max\_ports$ do
3:         $hca\_lid = sw_{leaf}.port-> remote\_lid$
4:         $sw_{leaf}.routing\_table[hca\_lid] = sw_{leaf}.port$
5:         $route\_downgoing\_by\_going\_up()$
6:     end for
7: end for
```

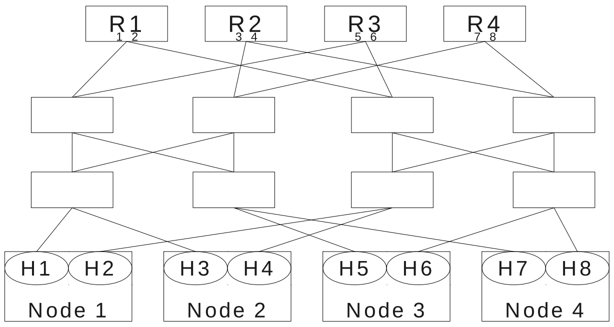

Figure 1: Test results from a real cluster

There are several problems with the design of $route\_to\_cns()$ function. First, it is oblivious and routes the end-ports without any consideration as to which node they belong. Second, it depends on the physical port number for routing. This is a major problem, and a possible consequence is presented on Fig. 1 which depicts a scenario with four two-port nodes connected to a small fat-tree. In this case, ports $H1$, $H2$, $H5$ and $H6$ are connected to port 1 on each switch while the rest of the ports ($H3$, $H4$, $H7$ and $H8$) are connected to port 2 on each switch. The downward routes are presented for each root switch. Because the current fat-tree routing routes on leaf switch basis, after routing 4 end-ports (traversing through the second leaf switch in this case), it will wrap around and start assigning the paths again from the leftmost root switch. Therefore, each pair of end-ports ($H1$ and $H2$, $H3$ and $H4$, $H5$ and $H6$, and $H7$ and $H8$) will be routed through the same root switch. This is a very popular scenario that provides the user with counter-intuitive behavior: a node that has built-in physical fault-tolerance (2 end-ports connected to different switches) has a single point of failure that is one of the root switches. There are many variations of this problem and, depending on the physical cabling, the single point of failure may be located on any switch in a fat-tree.

C. Subnet Reconfiguration

During normal operation the SM performs periodic *light sweeps* of the network to check for topology changes. If a change is discovered during a light sweep or if a message (trap) signaling a network change is received by the SM, it will reconfigure the network according to the changes discovered. The reconfiguration includes the steps used during initialization. Whenever the network changes (e.g. a link goes down, a device is added, or a link is removed) the SM must reconfigure the network accordingly. Reconfigurations have a local scope, i.e. they are limited to the subnets in which the changes occurred, which means that segmenting a large fabric with routers limits the reconfiguration scope.

IB is a lossless networking technology, and under certain conditions it may be prone to *deadlocks* [21], [22]. Deadlocks occur because network resources such as buffers or channels are shared and because IB is a lossless network technology, i.e. packet drops are usually not allowed. The necessary condition for a deadlock to happen is the creation of a cyclic credit dependency. This does not mean that when a cyclic credit dependency is present, there will always be a deadlock, but it makes the deadlock occurrence possible.

IV. Motivation and Design

A. Motivation

There are three main reasons that motivated us to create a multi-homed fat-tree routing and all come from real-world experience that we gained when designing and working with enterprise IB fabrics.

First, as it was mentioned before, the current fat-tree routing is oblivious whether ports belong to the same node or not. This makes the routing depend on the cabling and may be very misleading to the fabric administrator, especially when recabling is not possible due to a fixed cable length as often happens in enterprise IB systems. Furthermore, this requires the fabric designers to connect the end-nodes in such a way that will make the fat-tree routing algorithm route them through independent paths. Whereas this is a simple task for very small fabrics, when a fabric grows, it quickly becomes infeasible. Additionally, the scheme will break in case of any failure as usually the first thing that the maintenance does when a cable or a port does not work, is to reconnect the cable to another port, which changes the routing.

Second, due to the bandwidth limitations of the 8x PCI Express 2.0, it does not make sense to utilize both HCA ports on the same node with QDR speeds. This means that one of the ports is the active port and the second one is a passive port. The passive port will start sending and receiving traffic only if the original active port fails. Only with the advent of 8x PCI Express 3.0, where the bandwidth is doubled compared to 8x PCI Express 2.0, two QDR ports may be used simultaneously on the same card, which allows all ports connected to the fabric to be in an active state. These hardware limitations (active-passive case) mean that for today's routing only the active ports are important when routing and balancing the traffic because the passive ports do not generate anything (apart from management packets). The classic fat-tree routing algorithm deals with such a situation in an oblivious manner. The assumption here is that every port connected to the fabric is independent and has the same priority when being routed and what matters is the switch number and switch port number to which the node port is connected. Multi-homed fat-tree routing algorithm is not oblivious in such a case and first routes the active port on each node, therefore making sure that the path for that port will be optimal.

Third, because of doubled bandwidth of PCI Express 3.0, all ports at all nodes need to be considered with equal weights when doing routing. Classic fat-tree routing is able to do that, since it is the original assumption, however, it does not provide for any fault-tolerance that comes from the fact that each node has two or more ports. It means that even though the node ports are connected to different leaves, their paths may meet higher in the fabric and a single point of failure may exist. Therefore, to obtain optimal fault-tolerance, it is necessary to remove the single point of failure by making sure that ports belonging to the same node take mutually independent paths, which is what mFtree routing does. What needs to be taken into account is to distinguish between a single point of failure that is always fatal for an end-port (local link, local leaf switch) and a single point of failure that can eventually be recovered from by rerouting. In other words, we wanted to ensure that no single point of failure will impact both redundant paths even for a very short time before the SM has been able to reroute and reconfigure.

B. Design

As mentioned earlier, to obtain multi-homed routing in fat-tree topologies, one must abandon the fat-tree routing algorithm that is optimized for shift all-to-all communication patterns. In other words, a new routing logic is needed to make the routing algorithm less oblivious than the classic fat-tree routing and keep the same level of determinism.

The sample pseudocode for the auxiliary function $route\_multihomed\_cns()$ is presented in Algo. 2 and Algo. 3 presents the code for the main function $route\_hcas$. There were also changes done in $route\_downgoing\_by\_going\_up()$ that are presented in4. The code for auxiliary functions is presented, so that the exact flow of the algorithm can be analyzed.

Algorithm 2 $route\_multihomed\_cns()$

```
Require: Addressing is completed
Ensure: All $hca\_ports$ are routed through independent spines
 1: for $sw_{leaf} = 0$ to $leaf\_sw\_num$ do
 2:     for $sw_{leaf}.port = 0$ to $max\_ports$ do
 3:         $hca\_node = sw_{leaf}.port- > remote\_node$
 4:         if $hca\_node.routed ==$ true then
 5:             $continue$
 6:         end if
 7:         $route\_hcas(hca\_node)$
 8:     end for
 9: end for
```

As seen on Algo. 2, there are many similarities when compared to the classic fat-tree routing. An iteration is done over all leaf switches and then over all leaf switch ports, so the routing can be deterministic. However, the first major difference is that mFtree routing does not simply take the LID of the remote port connected to the leaf switch, but uses the whole node as a parameter to the main routing, which is presented on Algo. 3.

Having the end-node, we iterate over all its ports, and route each port in a usual way using the $route\_downgoing\_by\_going\_up()$ function. When all ports on the node are routed, we mark the node as $routed$, so that when that node is encountered on another leaf switch, it is not routed. For 2-port nodes, this saves half of the loop iterations.

Algorithm 3 $route\_hcas(hca)$

```
Require: Node that is to be routed
Ensure: All $hca\_ports$ belonging to the node with hca_lid are routed
 1: for $hca\_node.port = 0$ to $port\_num$ do
 2:     $hca\_lid = hca\_node.port- > lid$
 3:     $sw_{leaf} = hca\_node.port- > remote\_node$
 4:     $sw_{leaf}.port = hca\_node.port- > remote\_port\_number$
 5:     $sw_{leaf}.routing\_table[hca\_lid] = sw_{leaf}.port$
 6:     $route\_downgoing\_by\_going\_up()$
 7: end for
 8: $hca\_node.routed =$ true
 9: $clear\_redundant\_flag()$
```

Algorithm 4 $route\_downgoing\_by\_going\_up())$

```
Require: Current hop switch
Ensure: Best effort is done to find an upward redundant switch
Ensure: Switches on the path are marked with a redundant flag
 1: $group_{min} = 0$
 2: $redundant\_group = 0$
 3: for $port\_group = 0$ to $port\_group\_num$ do
 4:     if $group_{min} == 0$ then
 5:         if $group_{min}- > remote\_node.redundant$ then
 6:             $group_{min} = port\_group$
 7:         end if
 8:     else if $port\_group.cnt_{down} < group_{min}.cnt_{down}$ then
 9:         $group_{min} = port\_group$
10:         if $group_{min}- > remote\_node.redundant$ then
11:             $min\_redundant\_group = group_{min}$
12:         end if
13:     end if
14: end for
15: if $group_{min} == 0$ then
16:     $fallback\_normal\_routing(hca\_lid)$
17: else if $group_{min}- > remote\_node.redundant$ then
18:     $group_{min} = min\_redundant\_group$
19:     $group_{min}- > remote\_node.redundant =$ false
20: end if
```

A major change was done to the algorithm that selects the next-hop upward switch in $route\_downgoing\_by\_going\_up()$ function. Normally, the port group with lowest downward counters is selected, but in the case of mFtree algorithm, redundancy is

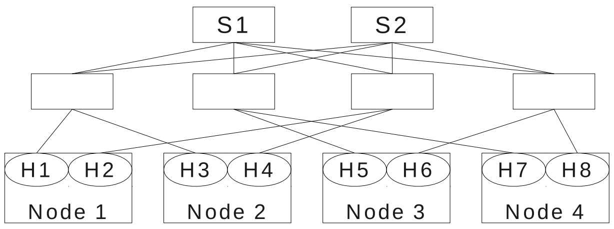

Figure 2: Experimental cluster

considered to be the decisive factor. Upward node can only be chosen as the next-hop if it does not route any other ports belonging to that end-node (in other words, the $redundant$ flag is true). The redundant flag is cleared before the next end-node is routed. If there are no nodes that are redundant as may happen in heavily oversubscribed fabrics or in case of link failures, mFtree falls back to normal fat-tree routing.

We may observe the similarity of this routing function to the routing function presented in Algo. 1.

V. Experiment setup

To evaluate our proposal we have used a combination of simulations and measurements on a small IB cluster. In the following subsections, we present the hardware and software configuration used in our experiments.

A. Hardware Setup

Our test bed consisted of four two-port nodes and six switches. Each node is a Sun Fire X2200 M2 server [23] that has a dual port Mellanox ConnectX DDR HCA with an 8x PCIe 1.1 interface, one dual core AMD Opteron 2210 CPU, and 2GB of RAM. The switches were: two 36-port Sun Datacenter InfiniBand Switch 36 [24] QDR switches which acted as the fat-tree roots; two 36-port Mellanox Infiniscale-IV QDR switches [25], and two 24-port SilverStorm 9024 DDR switches [26], all of which acted as leaves with the nodes connected to them. The port speed between the QDR switches was configured to be 4x DDR, so the requirement for the constant bisectional bandwidth in the fat-tree was assured. The cluster was running the Rocks Cluster Distribution 5.3 with kernel version 2.6.18-164.6.1.el5-x86_64, and the IB subnet was managed using a modified version of OpenSM 3.2.6 (with and without our mFtree implementation). The topology on which we performed the measurements is shown in Fig. 2. We used *perftest*, a tool provided with OFED, to measure the downtime that occurred when one of the spine switches went down. *Perftest* was modified to support regular bandwidth reporting and continuous sending of traffic at full link capacity.

B. Simulation Setup

To perform large-scale evaluations and verify the scalability of our proposal, we use an InfiniBand model for the OMNEST simulator [27] (OMNEST is a commercial version of the OMNeT++ simulator). The IB model consists of a set of simple and compound modules to simulate an IB network with support for the IB flow control scheme, arbitration over multiple virtual lanes, congestion control, and routing using linear routing tables. The model supports instances of HCAs, switches and routers with linear routing tables. The network topology and the local routing tables were generated using OpenSM coupled with ibsim and IBMgtSim and converted into OMNEST readable format in order to simulate real-world systems.

In each of the simulations, we used a link speed of 20 Gbit/s (4x DDR) and Maximum Transfer Unit (MTU) equal to 2048 bytes. Furthermore, we use uniform and non-uniform (shift and recursive-doubling) traffic patterns. For the uniform traffic pattern, each source randomly chooses a destination from the list of available destinations. This may lead to a situation, where more than one source chooses the same destination at the same time, which may cause slight congestion. The probability of this happening is inversely proportional to the number of end-ports, and in large fabrics is quite low.

The non-uniform traffic patterns can represent collective communications and are named shift and recursive-doubling. The patterns were simulated by translating their algorithm into sequences of destinations specific for each end-port and their implementation was described in a paper by Zahavi [9]. A random node-ordering of the MPI node-number to cluster end-ports was used in the translation and during the simulation, the end-ports progress through their destinations sequence independently when the previous message has been sent to the wire, which is one of the features of the IB model implemented in OMNEST simulator.

C. Topologies

There were 5 simulation topologies of varying size. There were two 648-port fat-trees: the 2-stage one built with 18 spines and 36 leaves, and the 3-stage one built with 18 spines, 36 middle-stage switches and 27 2-switch modules (54 separate leaf switches). Each switch in a 2-switch module has 36 ports: 12 ports connected to end-nodes, 12 horizontal links connected to the neighboring switch in the module and 12 uplinks to the middle-stage switches. Next, the 3-stage 432-port and 216-port topologies are variations of the 3-stage 648-port fabric. The 432-port and 216-port fabrics are 2/3 and 1/3 of size the original 3-stage 648-port fabric, respectively. Last, the 384-port fabric is 2-stage oversubscribed fat-tree that consists of 8 spines and 16 leaf switches. The oversubscription ratio is 1.33:1.

VI. Performance Evaluation

Our performance evaluation consists of measurements on an experimental cluster and simulations of large-scale topologies. For the cluster measurements we use the *average per node throughput* and *time* as our main metrics. However, we are not comparing the network performance here, but the network downtime is of main interest. For the simulations

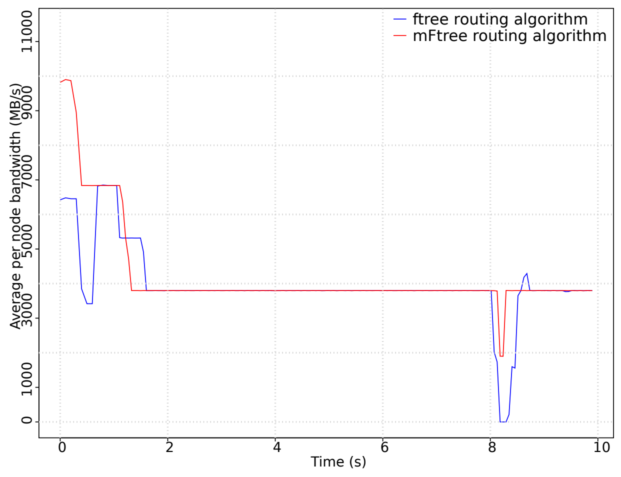

Figure 3: Results from the hardware cluster

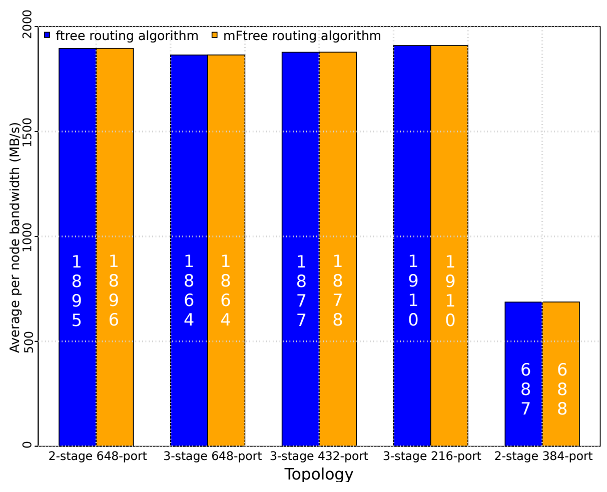

Figure 4: Test results for uniform traffic

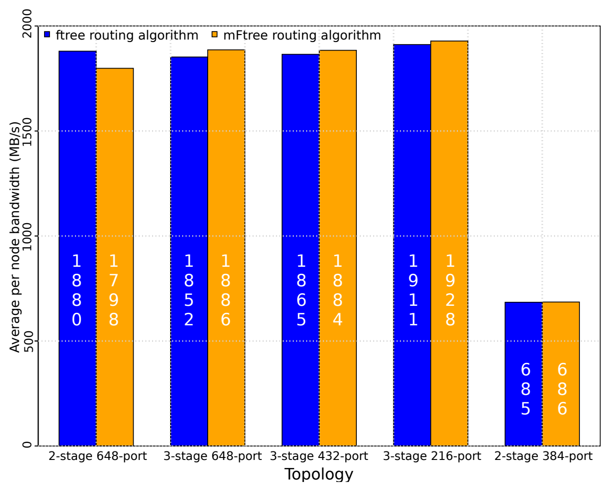

Figure 5: Test results shift traffic pattern

we use the *achieved average throughput per end node* as the metric for measuring the performance of the mFtree algorithm on the simulated topologies. In both the experimental cluster and in the simulations, all traffic flows are started at the same time and they are based on transport layer of the IB stack.

A. *Experimental results*

The main aim of the experiment was to show that applications experience less downtime with mFtree than with the classic fat-tree routing algorithm. In this experiment we rebooted the switch *S2* and measured how much time it took before the application started sending the traffic through the backup path. The whole experiment was conducted with two flows present in the network: port *H4* was sending traffic to port *H2* and *H3* was sending traffic to port *H1*. For classic fat-tree routing, both flows: $H4 \rightarrow H1$ and $H3 \rightarrow H2$ were routed through *S1* whereas for mFtree the flow $H4 \rightarrow H1$ was routed through *S1* and flow $H3 \rightarrow H2$ was routed through *S2*. In each case, we failed (rebooted) switch *S1* and measured how much time passed before normal operation was resumed.

The results are presented on Fig. 3. We observe that for mFtree routing, only half of the throughput (from 3795 MB/s down to 1897 MB/s) is lost as only one flow needs to be rerouted. For classic fat-tree routing, we observe that both flows must be rerouted, which means that even though a node receiving the traffic has two independent ports, a failure of a single switch makes the whole node unreachable.

The observation that needs to be highlighted is the fact that there is no downtime for connectivity between any pair of nodes with multiple ports operating in active-active mode when using the mFtree routing algorithm. The classic fat-tree routing engine that routes both traffic pairs through the same spine switch experiences a complete loss of service despite of the fact that the destination node has multiple ports. This means that the classic fat-tree routing algorithm is unable to use multiple ports on a node to provide for additional fault-tolerance.

B. *Simulation results*

The simulations were to show that using our routing algorithm does not deliver worse performance than the classic fat-tree routing algorithm when the tree size increases. We performed the simulations in an active-active mode, which means that no port was unused (passive).

The uniform traffic, for which the results are presented on Fig. 4, is considered to be very synthetic and not demanding on the routing algorithm. However, we used it to show the baseline performance as this traffic pattern has a predictable and easily understandable behavior, and is general rather than specific to a given application. Under uniform traffic conditions, where each source chooses the destination from all other possible destinations (apart from the ports located on the same end-node), the performance

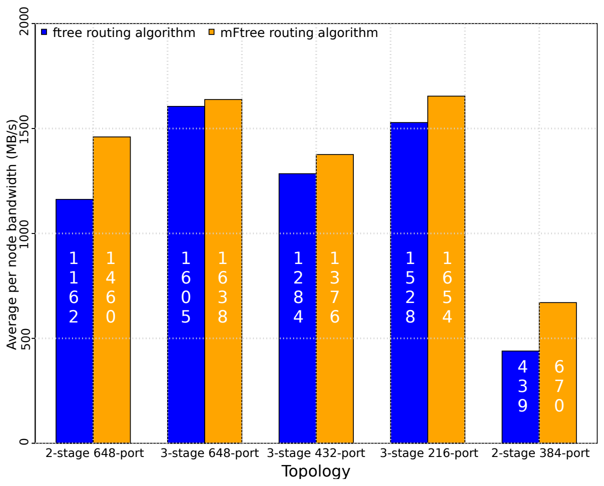

Figure 6: Test results for recursive traffic pattern

of both the routing algorithms is equal. This means that mFtree routing algorithm does not create any additional overhead when routing and we do not sacrifice performance for redundancy.

The results for the shift traffic pattern shown on Fig. 5 start to exhibit slight differences between the two algorithms. The classic fat-tree routing, which was specifically designed for the shift traffic pattern delivers slightly better performance (on average 82 MB/s or 4.5% higher per node) only on a 2-stage 648-port fat-tree. However, for any 3-stage fabric, it is the mFtree routing algorithm that delivers slightly better performance (ranging from 0.8% to 1.8% on average per node). For the oversubscribed 384-port fabric the performance delivered by mFtree and classic fat-tree is equal. The slight performance increase that is observed for mFtree is caused by better traffic distribution while traversing middle-stage switches in the upward direction. The 3-stage fabrics are built in such a way that each pair of interconnected leaf switches share four middle-stage switches. The end-nodes are connected in such a way that end-port:A is connected to the first leaf switch and the end-port:B is connected to the second leaf switch. What happens is that for the classic fat-tree routing, when routing in the upward direction, the traffic to port A and port B may traverse the same switch and even use the same link, which leads to slight congestion. For mFtree routing this does not happen as the traffic for end-port:A and end-port:B is separated from each other from the very beginning, which means that the traffic distribution is better. However, the influence of this phenomenon is minor because such a traffic overlap does not occur often for the shift traffic pattern we generated.

The doubling recursive pattern, for which the results are shown on Fig. 6, is the most demanding one from the patterns we tested, and the differences in performance here are significant. What is first noticeable is that the overall performance is lower for both algorithms. However, when we directly compare classic fat-tree routing and mFtree, we will observe that mFtree delivers better performance in each case. The differences are especially visible for 2-stage fabrics: for the 648-port fabric, the performance delivered by mFtree routing is 25.6% higher than for the classic fat-tree routing and for the 384-port fabric, the performance is 52.6% higher. For the 3-stage fabrics, the performance differences are 2%, 7.1%, 8.2% for the 648-port, 432-port and 216-port fabrics, respectively. Such performance differences are explained by the fact that the doubling recursive pattern is constructed in such a way that end-port:A at every end-node only sends to end-ports:A on other nodes. Furthermore, both $Node1 : portA$ when sending to $Node2 : portA$ (due to regularity of the classic fat-tree routing) uses second left-most spine as does $Node1 : portB$ when sending to $Node2 : portB$. For mFtree routing, the spines chosen will be always different. This is especially important in the oversubscribed fabric where the congestion not only occurs on the upward links, but also on some of the downward links.

Table I: Execution time comparison for different algorithms in seconds.

Topology	minhop	DFSSSP	ftree	mFtree
2-stage 384-port	5.12	5.04	0.02	0.02
2-stage 648-port	10.96	10.86	0.14	0.12
3-stage 216-port	5.38	5.14	0.08	0.02
3-stage 432-port	11.04	11.72	0.18	0.16
3-stage 648-port	16.5	19.38	0.38	0.36
3-stage 3456-port	105.86	286.38	11.66	10.58
4-stage 4608-port	401.1	1671.54	59.96	81.04

C. Execution time

Another important metric of a routing algorithm is its execution time, which is directly related to the reconfiguration time of the fabric in case of topology changes. For comparison, we added two more routing algorithms: minhop and deadlock-free single-source shortest-path routing (DFSSSP), and added two large topologies to observe how does the execution time scale with regard to the node number: a 3-stage 3456-port fat-tree and a 4-stage dual-core 4608-port fat-tree. The results are presented in Table I. As we may observe, the fat-tree routing and mFtree routing are comparable when it comes to the execution time. For a smaller number of ports (up to 4608), mFtree is in fact slightly faster than classic fat-tree. This happens because the ratio of ports to switches is low (i.e. there are many more ports than switches). However, the problem encountered when routing a 4608-port topology is that it contains 1440 24-port switches. What is not optimized is clearing the switch redundancy flag in function $clear\_redundant\_flag()$ from Algorithm 3. The loop in that function iterates over all the switches regardless whether a particular switch was on the path or not. This could be optimized by creating a list of switches that are on the path and making sure only those switches are iterated upon.

When it comes to other routing algorithms, we observe that they have extremely long execution times for fat-tree topologies. This is especially true to DFSSSP whose execution time explodes 1671.5 seconds, which means almost 28 minutes of downtime in case of a failure.

VII. Conclusions and Future Work

In this paper, we presented the new mFtree routing algorithm. We showed that it improves the network performance compared to the current OpenSM fat-tree routing by up to 52.6%. Most importantly, however, mFtree routing algorithm gives much better redundancy than classic fat-tree routing, which means that multi-homed nodes will suffer no downtime in case of switch failures.

In future we plan to optimize the routing algorithm, so its execution time is shorter on larger fabrics. Furthermore, we will be working with other enhancements to the fat-tree routing.

References

[1] J. Dongarra, "Visit to the National University for Defense Technology Changsha, China," Report, June 2013, http://www.netlib.org/utk/people/JackDongarra/PAPERS/tianhe-2-dongarra-report.pdf.

[2] "Top 500 supercomputer sites," http://top500.org/, June 2013.

[3] "The OpenFabrics Alliance," http://openfabrics.org/.

[4] C. Gómez, F. Gilabert, M. E. Gómez, P. López, and J. Duato, "Deterministic versus Adaptive Routing in Fat-Trees." [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.96.5710

[5] E. Zahavi, G. Johnson, D. J. Kerbyson, and M. Lang, "Optimized Infiniband fat-tree routing for shift all-to-all communication patterns," in *Concurrency and Computation: Practice and Experience*, 2009.

[6] T. Hoefler, T. Schneider, and A. Lumsdaine, "Multistage switches are not crossbars: Effects of static routing in high-performance networks," in *Cluster Computing, 2008 IEEE International Conference on*, 29 2008-Oct. 1 2008, pp. 116–125.

[7] G. Rodriguez, C. Minkenberg, R. Beivide, and R. P. Luijten, "Oblivious Routing Schemes in Extended Generalized Fat Tree Networks," *IEEE International Conference on Cluster Computing and Workshops*, 2009.

[8] "InfiniBand in the Enterprise Data Center," White paper, Mellanox Technologies, April 2006, http://www.mellanox.com/pdf/whitepapers/InfiniBand__EDS.pdf.

[9] E. Zahavi, "Fat-trees routing and node ordering providing contention free traffic for mpi global collectives," in *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, may 2011, pp. 761 –770.

[10] P. Lopez, J. Flich, and J. Duato, "Deadlock-free routing in infiniband/sup tm/ through destination renaming," *Parallel Processing, International Conference on, 2001.*, pp. 427–434, 3-7 Sept. 2001.

[11] J. Sancho, A.Robles, J. Flich, P.Lopez, and J. Duato, "Effective methodology for deadlock-free minimal routing in infiniband networks," in *Proceedings of the 2002 International Conference on Parallel Processing*. IEEE Computer Society, August 2002, pp. 409–418.

[12] J. C. Sancho, A. Robles, and J. Duato, "Effective strategy to compute forwarding tables for infiniband networks," in *ICPP*, L. M. Ni and M. Valero, Eds. IEEE Computer Society, 2001, pp. 48–60.

[13] J. Domke, T. Hoefler, and W. Nagel, "Deadlock-Free Oblivious Routing for Arbitrary Topologies," in *Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium.* IEEE Computer Society, May 2011, pp. 613–624.

[14] X.-Y. Lin, Y.-C. Chung, and T.-Y. Huang, "A Multiple LID Routing Scheme for Fat-Tree-Based Infiniband Networks," *Proceedings of IEEE International Parallel and Distributed Processing Symposiums*, 2004.

[15] S. Mahapatra, X. Yuan, and W. Nienaber, "Limited multi-path routing on extended generalized fat-trees," in *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, ser. IPDPSW '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 938–945. [Online]. Available: http://dx.doi.org/10.1109/IPDPSW.2012.115

[16] X. Yuan, W. Nienaber, Z. Duan, and R. Melhem, "Oblivious routing for fat-tree based system area networks with uncertain traffic demands," in *Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, ser. SIGMETRICS '07. New York, NY, USA: ACM, 2007, pp. 337–348. [Online]. Available: http://doi.acm.org/10.1145/1254882.1254922

[17] S. Öhring, M. Ibel, S. Das, and M. Kumar, "On Generalized Fat Trees," in *Proceedings of 9th International Parallel Processing Symposium*, 1995, pp. 37–44.

[18] C. E. Leiserson, Z. S. Abuhamdeh, D. C. Douglas, C. R. Feynman, M. N. Ganmukhi, J. V. Hill, D. Hillis, B. C. Kuszmaul, M. A. St. Pierre, D. S. Wells, M. C. Wong, S.-W. Yang, and R. Zak, "The network architecture of the connection machine cm-5 (extended abstract)," in *Proceedings of the fourth annual ACM symposium on Parallel algorithms and architectures*, ser. SPAA '92. New York, NY, USA: ACM, 1992, pp. 272–285. [Online]. Available: http://doi.acm.org/10.1145/140901.141883

[19] Omitted for blind review.

[20] Omitted for blind review.

[21] W. J. Dally and B. Towles, *Principles and practices of interconnection networks*. Morgan Kaufmann, 2004.

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

[23] "Sun Fire X2200 M2 server," Oracle Corporation, November 2006, http://www.sun.com/servers/x64/x2200/.

[24] "Sun Datacenter InfiniBand Switch 36," Oracle Corporation, http://www.sun.com/products/networking/datacenter/ds36/.

[25] "MTS3600 36-port 20 Gb/s and 40Gb/s InfiniBand Switch System," Product brief, Mellanox Technologies, 2009, http://www.mellanox.com/related-docs/prod_ib_switch_systems/PB_MTS3600.pdf.

[26] "SilverStorm 9024 Switch," Qlogic, http://www.qlogic.com/Resources/Documents/DataSheets/Switches/Edge_Fabric_Switches_datasheet.pdf.

[27] E. G. Gran and S.-A. Reinemo, "Infiniband congestion control, modelling and validation," in *4th International ICST Conference on Simulation Tools and Techniques (SIMUTools2011, OMNeT++ 2011 Workshop)*, 2011.