InfiniBand Network Analysis and Monitoring using OpenSM

A Thesis

Presented in Partial Fulfillment of the Requirements for the Degree Master of Science in the Graduate School of The Ohio State University

By

Nishanth Dandapanthula, B.Tech

Graduate Program in Computer Science and Engineering

The Ohio State University

2011

Master's Examination Committee:

Dr. D.K. Panda, Advisor

Dr. P. Sadayappan

Abstract

As InfiniBand (IB) clusters grow in size and scale, predicting the behavior of the IB network in terms of link usage and performance becomes an increasingly challenging task. The IB specification provides a detailed subnet management infrastructure to handle various aspects of the network. Open Subnet Manager (OpenSM) is an implementation of this specification and is available as a part of the Open Fabrics software package. Although many studies have identified the various subnet management phases, there exist no standard benchmarks that quantitatively identify the time spent in these phases and analyze the functional intricacies of OpenSM. There currently exists no open source tool that allows users to dynamically analyze and visualize the communication pattern and link usage on the IB network.

In this thesis , we design a set of micro-benchmarks using OpenSM to study the various phases of subnet management at a finer granularity. We design and develop a scalable InfiniBand Network Analysis and Monitoring tool - *INAM*. INAM monitors IB clusters in real time by querying various subnet management entities in the network. It is also capable of capturing the pattern of traffic on a subset of links in the network. This allows users to visualize and analyze the network communication characteristics of a job in a high performance computing environment. We demonstrate the effectiveness of INAM using a wide set of micro benchmarks and applications.

Dedicated to my family

Acknowledgments

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of those who made it possible. I would like to acknowledge and thank the invaluable support and guidance of my advisor, Dr D.K Panda. I am deeply indebted to him for his guidance and advice. This has been a privilege to work with him and I got to learn a lot from him during this time.

I would like to acknowledge and thank Dr. P. Sadayappan for agreeing to be a a part of my thesis defense committee.

I would also like to acknowledge and thank Hari Subramoni and Jerome Vienne for their valuable support, brain storming sessions and making complex things seem simple.

I don't have enough words to express my deep gratitude towards my parents, Hari babai, Indu pinni, Nandu babai and Vandana pinni. Nothing that I will ever do can even come close to their love and support for me and the sacrifices they have made to see me here.

Finally, I thank all my friends UV gang Jadiya, Banni, Dpac, Mutthu, Rajachan, Pattu and Gult gang- Guruji, Srinath, Ashish and Praneeth for sharing this journey and making it a memory to be cherished forever.

Vita

December 16, 1987 ..... Born - Warangal, India

2009 ..... B.Tech., Information Technology.,
VIT University,
Vellore, India.

2010-2011 ..... Graduate Research Associate,
The Ohio State University

Publications

Research Publications

N. Dandapanthula, H. Subramoni, J. Vienne, K. Kandalla, S. Sur, D. K. Panda, and R. Brightwell "INAM - A Scalable InfiniBand Network Analysis and Monitoring Tool, 4th Int'l Workshop on Productivity and Performance". *(PROPER 2011), in conjunction with EuroPar, Aug. 2011. To Appear*

Fields of Study

Major Field: Computer Science and Engineering

Studies in High Performance Computing: Prof. D. K. Panda

Table of Contents

List of Tables

Table **Page**

List of Figures

Figure **Page**

Chapter 1: INTRODUCTION

Across various enterprise and scientific domains, users are constantly looking to push the envelope of achievable performance. The need to achieve high resolution results with smaller turn around times has been driving the evolution of enterprise and supercomputing systems over the last decade. Interconnection networks have also rapidly evolved to offer low latencies and high bandwidths to meet the communication requirements of distributed computing applications. InfiniBand has emerged as a popular high performance network interconnect and is being increasingly used to deploy some of the top supercomputing installations around the world. According to the Top500 [25] ratings of supercomputers done in June'11, 41.20% of the top 500 most powerful supercomputers in the world are based on the InfiniBand interconnects. Recently, InfiniBand has also started to make in-roads into the world of enterprise computing.

1.1 InfiniBand - An Overview

InfiniBand Architecture (IBA) [12] defines a switched network fabric for interconnecting processing nodes and I/O nodes, using a queue-based model. The I/O devices and compute nodes are connected to the switch fabric using one or more channel adapters (CA). Every fabric may consist of one or more subnets which are

interconnected by routers. Each subnet is managed in an autonomous way. InfiniBand specifications do not enforce the usage of any specific network topology or routing algorithm. IB has multiple transport services including Reliable Connection (**RC**) and Unreliable Datagram (**UD**), and supports two types of communication semantics: Channel Semantics (Send-Receive communication) over RC and UD, and Memory Semantics (Remote Direct Memory Access - RDMA communication) over RC. Both semantics can perform zero-copy data transfers, i.e, the data can directly be transferred from the application source buffers to the destination buffers without additional host level memory copies. IB also proposes link layer Virtual Lanes (VL) that allows the physical link to be split into several virtual links, each with their specific buffers and flow control mechanisms. This possibility allows the creation of virtual networks over the physical topology. However, current generation InfiniBand interfaces do not offer performance counters for different virtual lanes.

1.2 OFED

OFED, short for OpenFabrics Enterprise Distribution, is an open source software for RDMA and kernel bypass applications. It is needed by the HPC community for applications which need low latency and high efficiency and fast I/O. A detailed overview of OFED can be found in [21]. OFED provides performance monitoring utilities which present the port counters and subnet management attributes for all the device ports within the subnet. Some of the attributes which can be obtained from these utilities are shown in Table 1.1.

IBA Architecture specifications define how the various network management activities should be undertaken and describe the structure of the control packets to be

Utility	Attribute	Description
perfquery	XmtData	The number of 32 bit data words sent out through that port since last reset
perfquery	RcvData	The number of 32 bit data words received through that port since last reset
perfquery	XmtWait	The number of units of time a packet waits to be transmitted from a port
smpquery	LinkActiveSpeed	The current speed of a link
smpquery	NeighborMTU	Active maximum transmission unit enabled on this port for transmit

Table 1.1: Sample of attributes provided by utilities inside OFED

used to exchange information among the entities to maintain compatibility between the various vendor implementations. It specifies a few management classes which coordinate to create a subnet management scheme which monitors the subnet and takes measures to conform to any changes in the topology. OFED includes such an InfiniBand subnet manager called OpenSM which configures an InfiniBand fabric. It comprises of the subnet manager (SM) which scans the fabric, initiates the subnet and then monitors it. The subnet management agents (SMA), which are similar to daemons, are deployed on every device port of the subnet to monitor their respective hosts. All management traffic including the communication between the SMAs and the SM is done using subnet management packets (SMP). They use UD queue pairs for their communication.The subnet management traffic uses the Virtual Lane (VL) 15 rather then cluttering the normal links [20]. IB specifications enforce the usage of VL 15 for subnet management traffic. The remaining VLs from 1 to 14 are available for general purpose traffic and the functionality of VL 15 is independent of the general subnet traffic. A detailed account of the functionalities and working of OpenSM is described in chapter 2.

1.3 Network Topologies and Routing Schemes

The classic Ethernet architectures use hierarchical switched network, whereas InfiniBand uses a switched fabric topology. The prominent network topologies used in contemporary scenarios include, but are not limited to, Fat-Tree topology [2], Mesh Topology and 3D-Torus topology [28]. All communications within these networks either begin or end at a Channel Adapter (CA). Even after the topology is set up and every device on the subnet is connected, at this stage there are no routing or forwarding tables set up on these devices. Thus any communication among the various subnet devices is not possible since the subnet is not initiated. InfiniBand also supports static routing. Thus, any changes to the topology have to be reflected in the forwarding tables which warrants the need for a subnet manager. The functionality of an SM is to discover the network, configure the devices with the forwarding tables and then monitor the subnet for any changes.

OpenSM supports several routing engines for the purpose of configuring these networks to enable the communication among various components of the subnet. A few prominent ones are explained below [15]. The user has the option of specifying which routing algorithm to use.

1.3.1 Min Hop Algorithm

In the default case, where the user fails to provide the routing algorithm to be used by OpenSM for the purpose of network configuration, MinHop routing scheme is invoked. In the initial phase, a MinHop matrix is created and populated with values which signify the number of hops required by each port to get to each Local Identifier (LID). After the MinHop matrix table is populated, the whole subnet is traversed

switch after switch. At every switch, the decision regarding which outgoing port to be used for each of the neighboring LIDs of that switch is taken. Since, there is a chance of multiple outgoing ports having the same number of hops to a LID, each port keeps track of the number of LIDs reachable from it using a counter. This approach tries to balance the link subscription which defines the number of routes per link at a switch level [10].

1.3.2 UPDN Unicast Routing Algorithm

Since the approach followed by MinHop routing scheme for the purpose of equalizing the link subscription at the switch level causes circular dependencies among the switch buffers which might in turn lead to a deadlock in the subnet, the UPDN algorithm restricts the paths which can be taken to reach a particular LID and thus avoids the chances of a deadlock occurring within the subnet. Initially the algorithm tries to detect the root nodes automatically by conducting a statistical analysis of the number of hops from a particular CA to a switch. Here zero or more root nodes may be found. If no root nodes are found, OpenSM falls back to MinHop. All the root nodes are ranked zero and a then a Binary First Search (BFS) is applied to rank the consequently traversed switches incrementally. At the culmination of the ranking phase, another BFS is started from each device on the subnet to configure the forwarding tables on the subsequently traversed switches.

1.3.3 LASH Unicast Routing Algorithm

LASH stands for Layered Shortest Path Routing. A data structure consisting the shortest paths among all the source and destination pairs is created. Using a channel dependency graph to avoid the occurrence of deadlocks, LASH assigns the routes to

various virtual lanes (VL). If adding a route can cause a potential deadlock, then the route is added to the next VL. Finally the number of routes per virtual lane are arranged and balanced such that the number of routes per lane average out. LASH is a topology independent scheme and performs well in irregular topologies and is very flexible. The number of VLs used quantifies the optimality of the implementation of LASH. The distribution of traffic is even and is spread out through the entire network. [23] provides more details about LASH.

1.3.4 DOR Unicast Routing Algorithm

Dimension order routing algorithm is optimal for a k-ary n-cube topology. Each port within the subnet must be cabled to represent a dimension of the k-ary n-cube subnet. It is based on the MinHop routing scheme and thus uses shortest paths from source to destination. To take advantage of the symmetry in a mesh or hypercube topology, DOR routes packets using the shortest path in the order of lowest dimension first. [16] describes DOR at a greater detail and provides a fix to the problem of circular buffer dependency with a new approach.

1.3.5 Modular Routing Engine

In order to facilitate the use of novel routing practices and schemes, OpenSM provides the Modular routing engine structure. By converting the routing scheme into a static routing file whose format is dictated by OpenSM, this structure allows OpenSM to use the new routing module. This support is not available for multicast call backs yet. This functionality loads the (LID forwarding tables) LFTs or the MinHop matrices as in case of MinHop routing scheme as per the routing file provided by the user.

Apart from the above mentioned routing schemes, OpenSM also introduced DnUp routing scheme which is a minor variation of UpDn scheme but is more adapted to the subnets in which up links and CA nodes are connected to the same switch nodes. Another scheme introduced recently is the Torus-2QoS Unicast routing scheme. OpenSM documentation [15] describes the above mentioned routing schemes at a finer detail.

1.4 Motivation

The goal of this work is to analyze and monitor an InfiniBand Subnet. A major part of this depends on tracking the performance counters and subnet management attributes such as the number of packets sent, number of packets receives and the number of packets dropped etc. The best way to obtain these values is to query the subnet manager for them as its the job of the subnet manager to keep track of the performance counters and subnet management attributes of all the devices in the subnet. This process requires some level of interaction with the InfiniBand subnet manager (OpenSM) and an understanding of how OpenSM monitors and obtains this data. Since there exist no detailed studies regarding this scheme, we needed to do an in depth analysis of the working and functionalities of OpenSM. The benchmarks designed in this work can be used to validate the performance of OpenSM using various routing schemes. This could aid in decisions such as choosing the optimal routing scheme for a particular cluster.

The main motivation behind developing a scalable InfiniBand Network Analysis and Monitoring Tool (INAM) was to provide the user with a seamless and real time view of the subnet as a whole. As INAM shows real time statistics of all the counters on a subnet device, the user can determine the cause of a link, node or port failure

using information such as the amount of traffic flowing through the device at the time of failure, the utilization of a link etc. Since INAM monitors and reports the status of the subnet at every moment, it can be configured to notify the system administrator regarding any changes in the subnet such as a node, link or port going down, topology changes etc. such that an appropriate action can be taken without delay. While designing topology aware algorithms, the visualization of the communication pattern provided by INAM could help in verifying the correctness of the algorithms. The utility of INAM includes but is not limited to the above mentioned functionalities and thus would be of formidable help in various real world scenarios.

1.5 Problem Statement

1.5.1 Analysis and Evaluation of OpenSM

There have been several attempts to evaluate the InfiniBand subnet management mechanism in the past [5] [6]. But these studies were based on simulations and the networks used were based on irregular topologies. [27] models the subnet management mechanism for Fat Tree InfiniBand networks using OpenSM. In the above mentioned study, Vishnu et al. also present the time taken by OpenSM in various subnet management phases at a broad level on a small scale InfiniBand Cluster. As the contemporary InfiniBand clusters have grown in size and scale, there exist no current studies which quantify the performance of OpenSM at a larger scale. Further more, the behavior of OpenSM with various configuration options, using different routing schemes, under a varied set of scenarios, at a finer granularity, is unclear. Thus the main objective is to understand the functionality, working and behavior of OpenSM in the current InfiniBand cluster scenario.

Approach Followed

In this thesis, we take up the challenge of designing a set of micro benchmarks which can measure the time taken by OpenSM in the various phases of configuring the fabric. We compare the amount of time taken by OpenSM in each phase when the system size is varied (number of nodes in the cluster). We compare this for the five algorithms supported by OpenSM (MinHop, UPDN, FTREE, LASH and DOR). We also compare the total time taken by OpenSM to configure the subnet by varying the system size and the routing algorithm. Experiments were conducted on the behavior of OpenSM when multiple cards were active on the nodes. We also conduct experiments on whether the number of nodes attached to a leaf switch affect the performance of OpenSM. We discuss how the minimum number of hops between any two nodes varies with the routing algorithm and the topology of the network. Furthermore, we see how OpenSM behaves in terms of reconfiguration time when a failure occurs in the subnet. We compare the reconfiguration time in various scenarios by varying the system size, the routing algorithm used and the number of HCAs active per node. We also discuss the experiments we conducted to understand the impact of the measures taken by OpenSM, when failures occur, in the subnet on any ongoing parallel application.

1.5.2 Network Monitoring and Analysis of an IB Network

Different factors can affect the performance of applications utilizing IB clusters. One of these factors is the routing of packets / messages. Due to static routing, it is important to ensure that the routing table is correctly programmed. Hoefler et al. showed, in [9], the possible degradation in performance if multiple messages traverse

the same link at the same time. Unfortunately, there do not exist any open-source tools that can provide information such as the communication matrix of a given target application or the link usage in the various links in the network, in a user friendly way.

Most of the contemporary network monitoring tools for IB clusters have an overhead attached to them which is caused by the execution of their respective daemons which need to run on every monitored device on the subnet. The purpose of these daemons is to gather relevant data from their respective hosts and transmit it to a central daemon manager which renders this information to the user. Furthermore, the task of profiling an application at the IB level is difficult considering the issue that most of the network monitoring tools are not highly responsive to the events occurring on the network. For example, to reduce the overhead caused by constant gathering of information at the node by the daemons, a common solution is to gather the information at some time intervals which could be anywhere between 30 seconds to 5 minutes. The interval at which the information is gathered is refereed to as the sampling frequency. The sampling frequency is directly proportional to the amount of data transferred between the daemons and the daemon manager and thus is directly proportional to the overhead created by the daemons. This also increases the amount of processing done by the daemons at their respective nodes which also leads to an overhead. All these factors cause a trade off with the responsiveness of the network monitoring tool. This method has an additional disadvantage in that, it does not allow us to monitor network devices such as switches and routers where we will not be able to launch user specified daemon processes.

As IB clusters grow in size and scale, it becomes critical to understand the behavior of the InfiniBand network fabric at scale. While the Ethernet ecosystem has a wide variety of matured tools to monitor, analyze and visualize various elements of the Ethernet network, the InfiniBand network management tools are still in their infancy. To the best of our knowledge, none of the available open source IB network management tools allow users to visualize and analyze the communication pattern and link usage in an IB network.

These lead us to the following broad challenge - *Can a low overhead network monitoring tool be designed for IB clusters that is capable of depicting the communication matrix of target applications and the link usage of various links in the InfiniBand network?*

Approach Followed

In this thesis, we address this challenge by designing a scalable InfiniBand Network Analysis and Monitoring tool - *INAM*. INAM monitors IB clusters in real time and queries the various subnet management entities in the IB network to gather the various performance counters specified by the IB standard. We provide an easy to use web interface to visualize the performance counters and subnet management attributes of the entire cluster or a subset of it on the fly. It is also capable of capturing the communication characteristics of a subset of links in the network, thereby allowing users to visualize and analyze the network communication characteristics of a job in a high performance computing environment. Our experimental results show that INAM is able to accurately visualize the link usage within a network as well as the communication pattern of target applications.

1.6 Organization of Thesis

The remainder of this thesis is organized as follows. In chapter 2, we describe the functionality of OpenSM and dig deeper into its working and the various phases it follows in the course of configuring the fabric. It gives a brief overview of the InfiniBand subnet management infrastructure and provides a detailed account of the components and the intricate functionalities of OpenSM. Chapter 3 focuses on how the set of micro benchmarks, which are designed to quantify the performance of OpenSM, are designed and how they measure the time spent by OpenSM in the various subnet management phases. We discuss how we go about measuring the reconfiguration time taken by OpenSM in case of failures. In Chapter 4 we present the design of INAM. We evaluate and analyze the working of INAM in various scenarios and describe the related tools which exist and discuss their advantages and disadvantages when compared to INAM. Finally we summarize our conclusions and possible future work in chapter 5.

Chapter 2: BACKGROUND

2.1 Introduction to OpenSM

OpenSM is an implementation of the InfiniBand Subnet Manager and Administration. OpenSM implements an InfiniBand compliant Subnet Manager (SM) according to the InfiniBand Architecture Specification chapters: Management Model (13), Subnet Management (14), and Subnet Administration (15) [12]. It runs on top of Open Fabrics verbs layer and performs the tasks required by the InfiniBand specification for bringing InfiniBand network to operational state. At least one OpenSM entity is required per InfiniBand subnet to initialize the InfiniBand hardware. OpenSM can by default attach itself to the first available port or it can display the available ports and ask the user to choose among those ports or can take the port specified by the user. It then configures the IB fabric connected to that port only and ignores all the other fabrics connected to other ports. The Subnet Manager (SM), the Subnet Management Agent (SMA), the Subnet Management Database (SMDB) and the Subnet Administration (SA) are the main components involved in InfiniBand subnet management. Section 2.2 gives a detailed overview of each of the above mentioned components.

OpenSM uses management datagrams (MAD) which are also known as subnet management packets to interact with the various components involved in the subnet management phase. These SMPs comprise a critical component in the functionality of OpenSM as they play a part in every aspect of the network as listed below [1].

1. network discovery monitoring
2. configuration of ports and I/O devices
3. responding to queries posed to the SA by other subnet management components
4. performance and baseboard management

OpenSM passes through certain phases as a part of its functionality of configuring the subnet fabric. The default phases comprise of scanning the IB fabric, initializing it and then sweeping the fabric occasionally for any changes in the topology. There are certain parameters associated with these phases that can be tweaked (for example the time between the sweep operations) as per the requirement of the situation. Since the default installation of OpenSM performs well for a system size of up to a couple of hundred nodes, the configuration file has various options which can be varied to tune OpenSM as per the requirements of the user. A few relevant options which we use as a part of our experiments are mentioned here. A complete list of these options can be found at [15].

1. *-c, –create-config file name* – A template is created by OpenSM by dumping all its configuration options into this file.
2. *-r, –reassign_lids* – All end node LIDs are reassigned. If not specified, the previously existing LID assignments are preserved.

3. *-R, –routing_engine* Routing engine names – The default routing engine used is Min Hop algorithm. There are other routing engines which are supported by OpenSM. They are MinHop, UPDN, file, FTREE, LASH and DOR.

4. *-A, –ucast_cache* – Routing recalculation is avoided by activating the unicast routing cache when no topology change is detected or in cases where the change does not need the recalculation of the routes (for example if a node reboots we need not compute the routes when the node goes down and when the node reboots again).

5. *-o, –once* – This option causes OpenSM to configure the subnet once, then exit.

6. *-s, –sweep interval value* – The number of seconds between subnet sweeps. 0 disables sweeping. The default interval is 10 seconds.

7. *-t, –timeout value* – Specifies milliseconds used for transaction timeouts. Default timeout is 200 milliseconds.

8. *-i, -ignore-guids equalize-ignore-guids-file* – This is the option where the mentioned guids are ignored for link load equalization algorithm.

OpenSM also provides a test program called osmtest for validating the InfiniBand Subnet Manager and Subnet Administrator. The osmtest suite can create an inventory file of all available nodes, ports, and Path Records, including all their fields and then compare it to the previously saved records to check if there have been any changes to the fabric [22]. It has the following test flows.

1. Multicast Compliance test

2. Event Forwarding test

3. Service Record registration test

4. RMPP stress test

5. Small SA Queries stress test

2.2 Components and Working of OpenSM

OpenSM delegates its functionalities in a hierarchical manner. The important components are the SM, SMA, SMP and Subnet Management Interface (SMI). Their hierarchy is mentioned in Figure 2.1. The SM plays a key role in the network discovery process and it is involved in the monitoring phase of the subnet where the subnet is monitored for any changes in topology. The SM has all the data related to the fabric configuration in the SMDB. The SA is responsible for responding to the queries on SMDB.

The SM is typically a software entity running on a compute node within the subnet, though it is capable of residing within any device on the subnet. On a switch the SM can reside on the management port which is known as the port 0 of the switch. The SM follows the following phases in the process of configuring the subnet. Initially it focuses on discovering and maintaining the subnet it manages. In the next phase, it assigns the Subnet IDs and LIDs to all the ports in the subnet. Finally it calibrates the route to be taken between any two nodes in conjunction with the routing algorithm being used and then periodically checks the subnet for any changes.

An SMA resides on every device populating the subnet irrespective of whether it is a CA, a router or a switch. It monitors the status of the device it resides on and

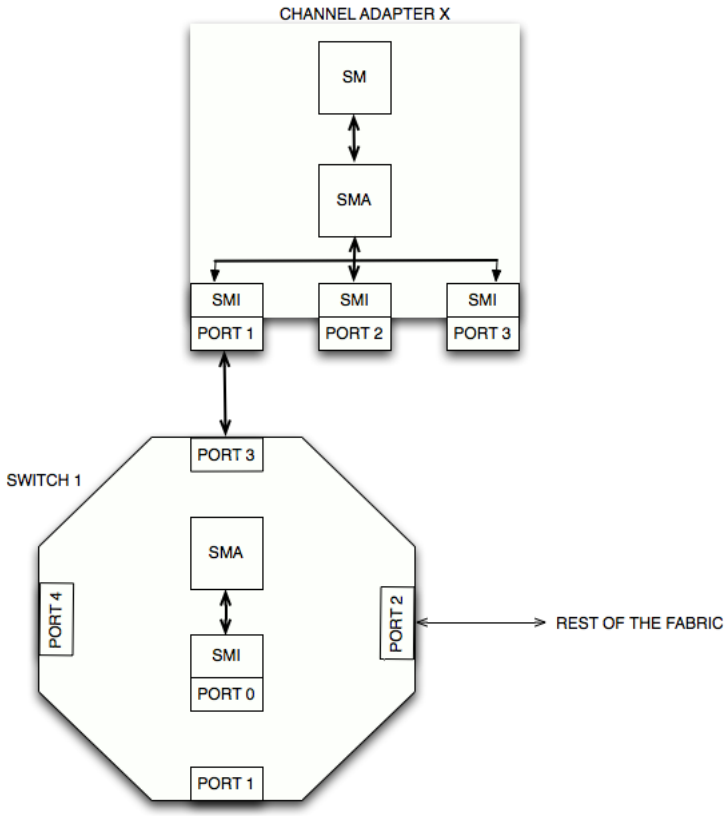

Figure 2.1: SM, SMA and SMI

reports any and all changes to the SM by using SMPs. SMPs are control packets and they use the unreliable datagram service. All communications between the SM and the SMAs are in the form of these SMPs which use the management virtual lane (VL15) so as not to interfere with the ongoing cluster traffic [6]. The SMI is a UD Queue Pair. As shown in Figure 2.1, an SMI is associated with each port on a CA. On the other hand, if the device is a switch, the SMI only exists on the port 0 of the switch which is the management port. Its sole purpose is to assist the SM and the SMAs within CAs, routers, and switches to send and receive SMPs [12].

OpenSM attaches to a specific IB port on the local machine and configures only the fabric connected to it. Initially when OpenSM is started, none of the devices know their LIDs and none of the switches have their forwarding tables set up. The SM then probes for network devices populating that subnet. It exchanges control packets with the SMAs present in every subnet device. The SMPs can be either LID routed (LSMP) or Direct routed (DSMP). LSMPs are routed by switches by using the forwarding table by doing a look up. DSMPs have to be processed at each intermediate subnet manager interface (SMI) as they have the information about the output port they need to be forwarded to at each hop. These are used during subnet discovery before the forwarding tables at the switches. No processing at the SMI is required for an LSMP. The data packets take the route taken by the LSMPs after the DSMPs are used for discovering the topology and updating the switch routing tables. The header of the SMP states the operation the packet needs to do. The operations involved in subnet management are as follows.

1. *Get:* The Get operation is used to get the information about the CA, Switch or a Router port

2. *Set:* The Set operation is used by the subnet manager to set the attributes of a port at the end of the subnet discovery

3. *GetResp:* The GetResp operation is the response to the Get SMP of a subnet manager

4. *Trap:* A trap message is sent if a change is detected in the topology during the sweep phase by the SMA. The SMA sends a trap message if its local node's state has changed

5. *TrapRepress:* SMA sends trap messages to the SM until it receives a TrapRepress from the SM [27]

There may be many instances of SMs running on the same subnet. Out of these one of them is the master SM and the others are standby SMs or slave SMs. The standby SMs periodically poll on the master SM and wait for a successful response. In case of failure of the Master SM, one of the standby SMs becomes the master SM. The Master SM keeps a counter which is updated every time the SM is involved in a subnet management event. The slave SMs keep polling this counter periodically and if they detect that the counter has not changed in a particular time interval, then the next slave SM with the highest priority takes over.

2.3 Phases in OpenSM

The initial phase for OpenSM while configuring the fabric is the topology discovery phase where the SM sends direct routed SMPs to every port and processes the responses. A heavy sweep is done initially to discover all the nodes and switches in the subnet. Alternatively a new heavy sweep can also be triggered when the OpenSM

process receives a *HUP* signal, which is sent out if a trap was received or a topology change was found.

In SM and subnet LID configuration phase, LIDs are assigned to all the ports in the subnet by the SM. In the path computation phase, valid paths between each pair of end nodes are computed as per the routing algorithm being invoked. In the path distribution phase, configuring the forwarding table at each switch is done. After the configuration is done, a light sweep is done to check if there have been any changes to the subnet. Each of these phases is discussed in detail in the following sections.

2.3.1 Subnet Discovery

OpenSM sends a *Get SMP* with NodeInfo attribute to get the kind of end node (Switch, CA or Router) and a *Get SMP* with PortInfo to get the port attributes. It sends a *Set SMP* to set the LIDs and other port attributes. OpenSM provides an option of Trap based subnet discovery. In order to allow a switch to notify the subnet manager about any topology changes which may have occurred, SM initializes every port on the switch, even though it may not have any active connection with a CA. The algorithm is as follows [5].

```
if AttributeID = SwitchInfo then
    if SwitchInfo.PortStateChange then
        delete the topology database
        send a Get(NodeInfo)
    endif
elseif AttributeID = NodeInfo then
    if sender not visited then
        add this node to the topology DB
        for each port in sender do
            Send a Get(PortInfo)
        endfor
    endif
elseif AttributeID = PortInfo then
    if management port then
        send a Set(PortInfo)
    endif
    if PortInfo.PortState is not DOWN then
        add this port to the sender ports list
        send a Get(NodeInfo)
    endif
endif
```

A DSMP is used for the subnet discovery process and it's significant components are shown in Figure 2.2. The DSMP also mentions the subnet operation to be executed. The DrSLID and the DrDLID fields signify the directional source and destination lids. The Initial and return path array fields keep track of the lids and ports traversed as the forwarding tables are not yet set at each switch. A special LID called Permissive LID (PLID) which has the value FFFFh is used for the discovery process. The hop pointer and hop counter keep track of the current position and the maximum number of hops allowed, respectively. An example of subnet discovery process can be drawn from Figure 2.1.

Initially, when the SM starts it is not aware of its current status. The SM on CA-X queries the host channel adapter (HCA) to obtain ports on that device and then sets it's SM bit to 1. An option to select the port to which the SM binds is given to the

user, otherwise it selects the best available port. Initially, an SMP (SUBN-GET) is sent out by the SM through the SMA to the port to which it is bound on that device (ex: Port 1) using the SMI, with the DrSLID and DrDLID set to PLID, the initial path array set to 0h,1h, the return path array set to 0h,0h and the hop count set to 1 and the hop pointer set to 0. At port1 of CA-X, the return path array is modified as 1h,0h (as the PLID in the DrDLID field indicates that the particular SMP is being used for discovery process) and is forwarded to the next port connected (In this case, Port 4 of switch 1).

As, the DrDLID is set to PLID, this packet is internally forwarded to the management port (port 0) after the return path array is modified as 1h, 3h and the hop pointer is incremented to 1. The SMI at port 0 of switch1 processes the SMP and then forwards it to the switch's SMA. The SMA prepares a response SMP after executing a SUBN-GET and includes the attributes of all the 4 ports on that switch in the response SMP and uses the return path array to send this information to the SM after verifying if the hop pointer and hop counter are equal. If yes, the SMP cannot proceed to the next switch in the fabric and has to return to the SM.

The SM stores this information and similarly traverses the rest of the fabric by incrementing the hop counter. This completes the subnet discovery process. If the packet is not a DSMP then there is no processing involved at the SMI. IBA specifies the actions needed to taken by the SM in all possible scenarios which may be encountered during the discovery phase.

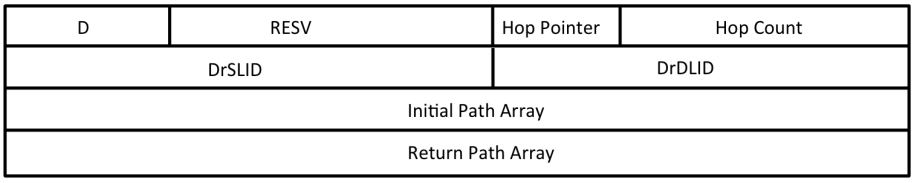

Figure 2.2: DLID SMP Outline

2.3.2 SM and Subnet LID Configuration

After the fabric is scanned, the SM configures the LIDs on each port traversed using the SUBN-SET operation. Unique LIDs are assigned to all the ports in the subnet. This phase is referred to as the SM and subnet LID configuration phase. All the LIDs are reconfigured every time a heavy sweep occurs as a result of some changes which were detected in the topology. Since, this process consumes a lot of time, there is an option which enables us to refrain OpenSM from reconfiguring all the LIDs every time a heavy sweep is done. For example, if certain number of nodes are restarted in a subnet and we know in all certainty that the nodes are going to get back to their previous stable working state, then we can avoid the overhead caused by reconfiguring all the Subnet LIDs by refraining OpenSM from doing so.

2.3.3 Path Computation

Paths between all pairs of source and destination nodes are calculated. As InfiniBand specification does not impose any particular routing algorithm or path computation methodology, OpenSM selects the one with the least number of hops by default or uses the option provided by the user. In case of a tie in the default case, one of the

available paths is chosen randomly. Section 1.3 talks about the routing algorithms supported by OpenSM in detail.

2.3.4 Path Distribution

The forwarding tables on each switch are configured in this phase. The LSMPs and the data packets then refer to these forwarding tables for traversing the subnet. OpenSM performs the forwarding table configuration phase once the path computation phase is complete. This policy is optimal, because during the subnet discovery phase, nodes are not ready for communication, unless the whole subnet is in ACTIVE state. When the forwarding tables are being updated, the network is in an unstable state and thus there is a possibility of a deadlock occurring among the data packets being transmitted. Hence, typically no data packets are allowed during this phase [4]. After the successful completion of this stage, the subnet is set to ACTIVE state.

After the path distribution phase, the subnet is in ACTIVE state. The SM then periodically monitors the subnet for any changes. This is called a light sweep. The interval at which the light sweep is done can be modified as per the user's requirement. The default time interval is 10 seconds. If the light sweep detects a change in the topology, a heavy sweep is triggered. A heavy sweep is essentially the discovery phase.

2.4 Failures

2.4.1 Failure Detection

Failures in a huge subnet constitute of a node, switch or port failures. After a subnet is configured or after the subnet enters ACTIVE state, periodical light sweeps are initiated to check for any changes in the topology. On a different note, the SMAs keep monitoring their respective devices for any change of state. When an

unexpected event is detected by the SMA, it notifies the particulars of this event to the SM through a SubnTrap message which is a management datagram (MAD). The prominent fields in a Trap message are the type of device from which the trap message originated (Switch, Node etc), the device LID, the trap number and the notice field. Typical trap messages generated by the SMA have the trap_num element in the trap message set to 128 or 144. These two traps signify two different events in the subnet.

Trap 128 is generated when an SMA detects a change in the link state of at least one of its ports. This message has the elements of prod_type, type, issuer_lid and trap_num in its notice attribute. The type field is set to 1 which indicates that this is a subnet management trap message. The prod_type identifies the type of device generating the trap (switch/CA). The issuer_lid specifies the LID of the device generating the trap message. Trap 144 is generated when a change is detected in the device/port on which the SM is running. This message has similar attributes as the Trap 128 message. It is triggered by events such as change in the priority of the master SM or failure of master SM. When the SMA is not in a position to send out trap messages, it can optionally store any unconventional events which might be sent to the SM at a later point of time. Figure 2.3 shows an instance of how failures are detected by OpenSM.

In Figure 2.3, when a port fails in the leaf node, the switch attached to that node (Switch 1) detects a change in the link state of one of its ports. The SMA on switch 1 then prepares a trap 128 message to propagate this issue to the SM residing on the SM node. If one of the ports on switch 1 fail, then switch 2 detects the change in the link state of one of its ports and generates a trap 128 message. OpenSM does not differentiate between node, switch or port failures. They are all treated in terms of

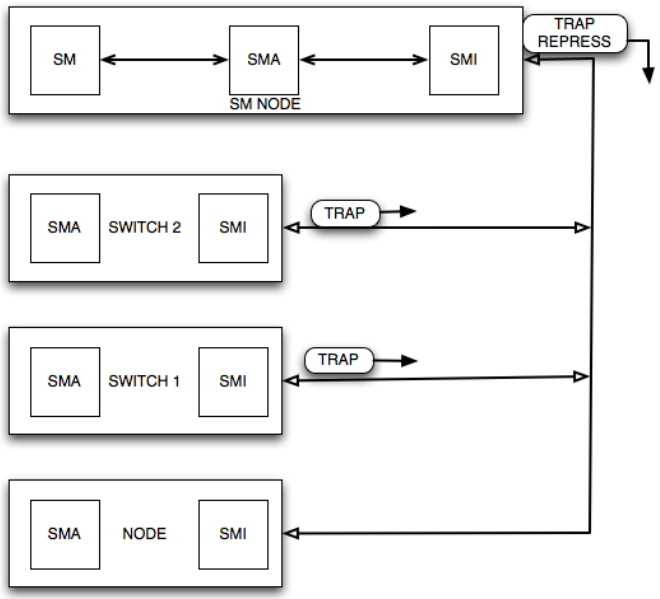

Figure 2.3: Failure Detection

the trap messages as link failures. The location of the failure in the subnet does not carry any weightage.

2.4.2 Failure Handling

When the SubnTrap (Notice) is received by the SM, it checks to see if the trap numbers in the notice attribute are 128 or 144. Both the trap messages (128 and 144) trigger a heavy sweep within the subnet by setting the *force_heavy_sweep* flag. This triggers the function *trap_rcv_process_request* which forces a heavy sweep. In some cases, the SMA may send repetitive trap messages to the SM. The SM can reply by sending a SubnTrapRepress (Notice) message to convey to the SMA in question that it has received the trap message from that SMA and is acting on it, thus refraining it form sending more duplicate trap messages. The frequency at which the trap

messages are generated can also be limited by varying the SubnetTimeout attribute. This can be useful to avoid the VL15 packet drops because of excessive generation of trap messages. A heavy sweep initiates the reconfiguration of subnet. The various phases it passes through are as follows.

1. Initiate heavy sweep
2. Configure SM LID
3. Subnet LID Configuration
4. Configure switches for Unicast and Multicast
5. Subnet Up

Chapter 3: ANALYSIS AND EVALUATION OF OPENSM

We designed a set of benchmarks to quantify the performance of OpenSM based on the time spent by OpenSM in the various phases of configuring the fabric. The phases monitored by the benchmarks at the highest granular level are the discovery phase, the SM LID configuration and subnet LID configuration phase and the routing phase. Initial prerequisite is that the cluster or subnet should not have any other instance of OpenSM running. The benchmarks are described as follows.

3.1 Designing Benchmarks to Quantify the Performance of OpenSM

The benchmarks are designed to run OpenSM with the option -o which means that the SM will do the topology discovery, route computation and the configuration of the switches only once and then exit. We use the option -v to moderate the amount of verbosity with which the log is written. Another option -R is used to facilitate the option of the user choosing which routing algorithm to use. The user can also mention the number of iterations these benchmarks have to run. The benchmarks will average out the time taken by OpenSM in each phase based on the number of iterations. We parse through the log file and obtain the time stamps of events which mark the beginning and ending of the phases. Using the time stamps of these events

the time taken by OpenSM in various phases is obtained. The benchmarks and their functionalities are explained in the following sections.

3.1.1 Subnet_Discovery_Time

This benchmark calculates the time taken by OpenSM in the subnet discovery phase. The discovery phase consists of the time taken by OpenSM to bind the Subnet Manager to a port and to complete the heavy sweep. The benchmark takes the name of the routing algorithm and the number of iterations the benchmark has to run as parameters. The five routing algorithms (MinHop, DOR, UPDN, LASH and fat tree) which are supported by OpenSM can be used in the benchmark. A sample output for a 1,000 iterations while using MinHop routing algorithm is shown in Figure 3.1.

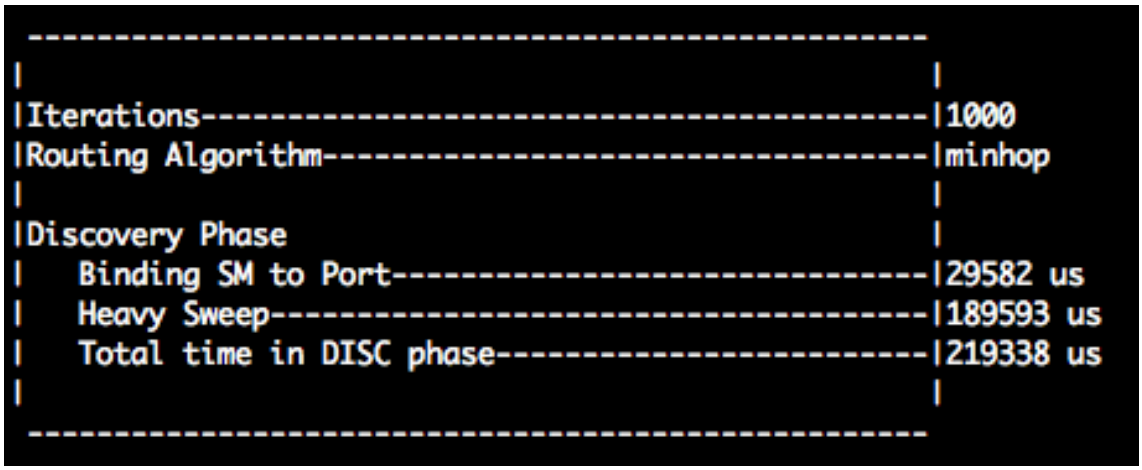

Figure 3.1: Sample Output of Subnet_Discovery_Time Benchmark

3.1.2 Subnet_LID-Configuration_Time

This benchmark calculates the time taken by OpenSM once the SM enters the MASTER phase. The significance of the MASTER phase is that this is the first

and only instance of SM running on that subnet. If any other instances of SM are started while this instance is running, then they fall back to STANDBY phase. The MASTER phase consists of the time taken by OpenSM to set up the QOS and in the phase of SM and Subnet LID configuration. The parameters taken by this benchmark are the number of iterations the benchmark has to run and the routing algorithm. A sample output for a 1,000 iterations while using DOR routing scheme is shown in Figure 3.2.

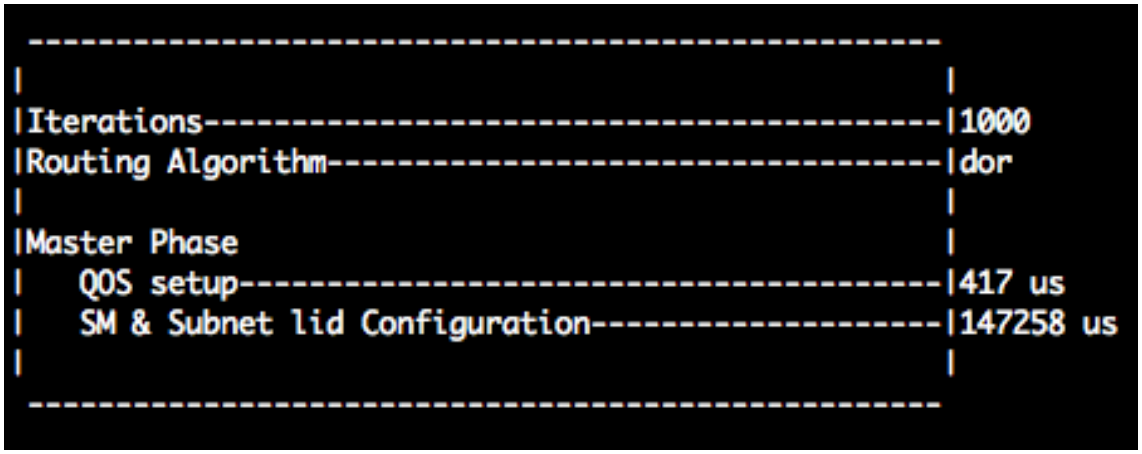

Figure 3.2: Sample Output of Subnet_LID-Configuration_Time Benchmark

3.1.3 Subnet_Routing_Time

This benchmark calculates the time taken by OpenSM in the Routing Phase. The routing phase consists of the time taken by OpenSM in calculating the routes, updating the switch tables and configuring the links and ports. The parameters taken by this benchmark are the number of iterations the benchmark has to run and the

routing algorithm. A sample output for a 1,000 iterations while using LASH routing scheme is shown in Figure 3.3.

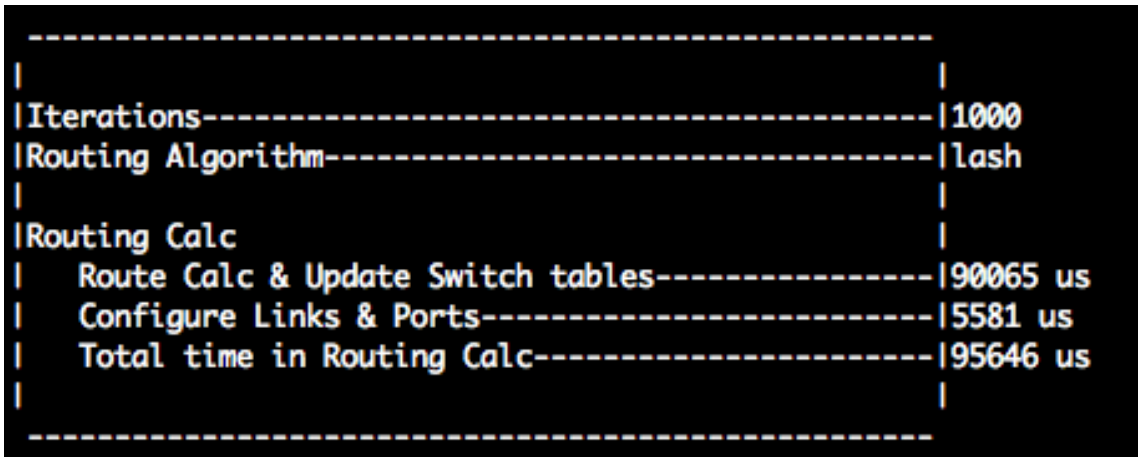

Figure 3.3: Sample Output of Subnet_Routing_Time Benchmark

3.1.4 Subnet_Total_Time

This Benchmark sums up the total time taken by OpenSM to configure the subnet. The parameters taken by this benchmark are the same as the other benchmarks. This benchmark also displays the time taken by OpenSM in all its sub phases as well. A sample output for a 1,000 iterations while using FTREE routing algorithm is shown in Figure 3.4.

3.2 Design - Quantifying the reconfiguration time of OpenSM

We designed a benchmark to calculate the time taken by OpenSM to reconfigure the subnet in case of a failure. When an HUP signal is sent to OpenSM, a heavy sweep is triggered just like the trap messages. Since OpenSM does not differentiate

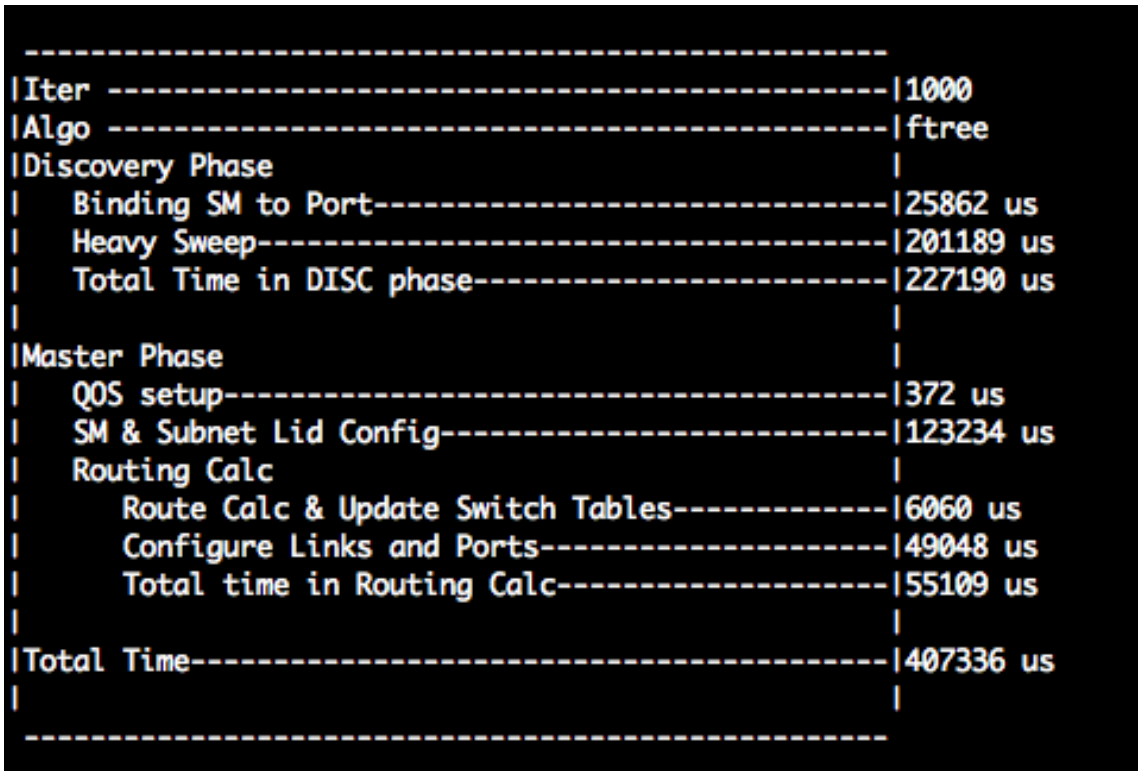

Figure 3.4: Sample Output of Subnet_Total_Time Benchmark

between a switch, node or port failures, we could mimic these failures using the HUP signal. We designed a second benchmark to understand the impact of the measures taken by OpenSM to handle the failures, on any ongoing computation. A detailed description of these benchmarks is provided in the following sections

3.2.1 Subnet_Reconfigure_Time

This benchmark measures the time taken by OpenSM to reconfigure the subnet when a failure occurs. The parameters taken by the benchmark as input are the routing algorithm to be used, the path to the location where the log files are saved

and the path where OpenSM is installed. The benchmark starts OpenSM in verbose mode and logs the results. It obtains the process ID (pid) of OpenSM and then sends out an HUP signal to that pid to mimic a failure. Then we query the log to find the time taken by OpenSM to reconfigure the subnet. A sample output is shown in Figure 3.5.

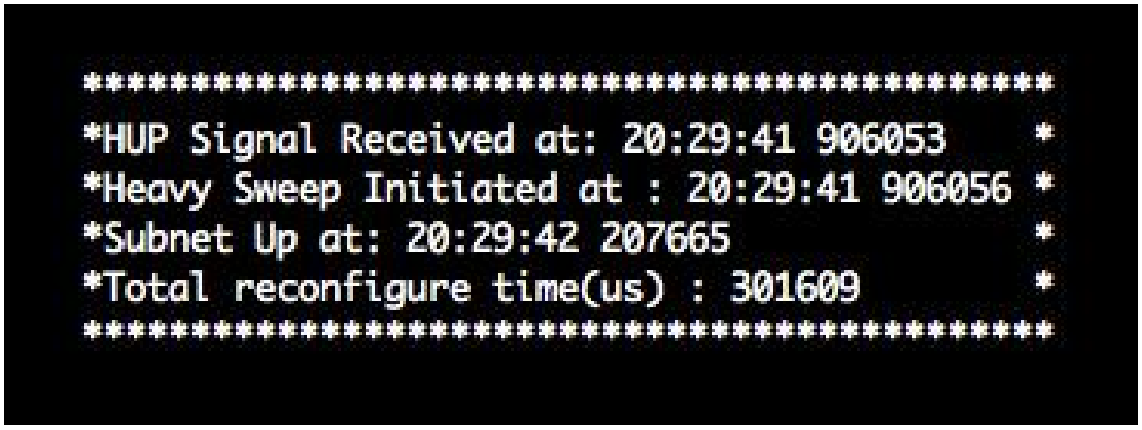

Figure 3.5: Sample Output of Subnet_Reconfigure_Time Benchmark

3.2.2 Impact_perf

This benchmark runs an IMB alltoall benchmark and introduces failures periodically. The parameters taken as input by this benchmark are the number of failures to be introduced, the path to the build of MVAPICH2 [17], the path to an IMB build, the path to hostfile and the path to the message length file. It runs an IMB alltoall benchmark once without introducing any failures and distributes the failures evenly in the second run by taking the output of the first IMB alltoall run. The aim of this benchmark is to reflect the impact of the reconfiguration process undertaken by

OpenSM on any ongoing computation. The failures are introduced by using the HUP signal methodology used previously. The output is an IMB alltoall output.

3.3 Evaluation and Experiments

3.3.1 Experimental setup

The experimental setup is a cluster of 71 nodes which are all dual Intel Xeons E5345 connected to an InfiniBand Switch which has an internal topology of a Fat Tree. The number of nodes were varied by unloading the OpenIB service. The topology was moderated by using a custom made tool which depicts a human readable format of the subnet topology using the *ibnetdiscover* functionality provided by OpenSM. The nodes were made undetectable to OpenSM by removing the InfiniBand drivers on the entire line card on a step by step basis for every leaf node.

In the first set of experiments, we compared the amount of time taken by OpenSM in each phase when the system size is varied (number of nodes in the cluster). We conducted this experiment for all the five routing algorithms supported by OpenSM (MinHop, UPDN, FTREE, LASH and DOR). In the second experiment, we compared the performance of the various routing algorithms supported by OpenSM to find the routing scheme which provides optimal performance and is best suited for our cluster.

In the first set of experiments, we see that the total time taken by OpenSM to configure the subnet increases as the system size increases. We can also see that SM/Subnet LID configuration phase takes the least time when compared to all the other phases. The subnet discovery phase takes up a major 75% of the total time taken by OpenSM to configure the fabric. As we already know the number of device ports in the subnet from the discovery phase, the subnet discovery phase just has to

assign LIDs to these ports. This explains the amount of time taken in the second phase. In the second experiment we found that OpenSM configures the experimental subnet the fastest when FTREE routing algorithm is used. This can vary from cluster to cluster as it depends on the internal topology of the InfiniBand switch used in the cluster. For example, if a cluster with a hypercube or mesh topology is used, we could see the optimal performance with other routing algorithms such as DOR or LASH. Thus the following experimental results are cluster dependent.

3.3.2 Performance of OpenSM for Various Routing Schemes

DOR

In Figure 3.6, we use the set of benchmarks discussed previously to obtain the time taken by OpenSM in its various phases while OpenSM is configured to run with direction oriented routing algorithm. The time taken by OpenSM in the subnet discovery increases by 15% when the system size varies from 8 nodes to 16 nodes where as the variation is not as prominent and is not uniform when the system size is varied from 16 nodes to 64 nodes as there is a minor drop when the system size varies from 16 to 32 nodes. The SM and Subnet LID configuration phase shows a steady increase as the system size increases. The time taken in the routing phase decreases by 22% as the system size increases from 8 to 16 nodes and then steadily increases as the system size increases from 16 to 64 nodes. This is because of a functionality of DOR where it uniformly distributes the load on each link. We see a similar spike in the LASH algorithm which is also optimized for 3D mesh and 3D torus topologies and LASH too equilizes the number of paths assigned to each VL to distribute the load uniformly. Finally, the total time taken by OpenSM increases as the system size increases from 8 to 64 nodes. The variation when the system size increases from 8 to

16 nodes and 32 to 64 nodes is much larger then the variation when the system size increases from 16 to 32 nodes.

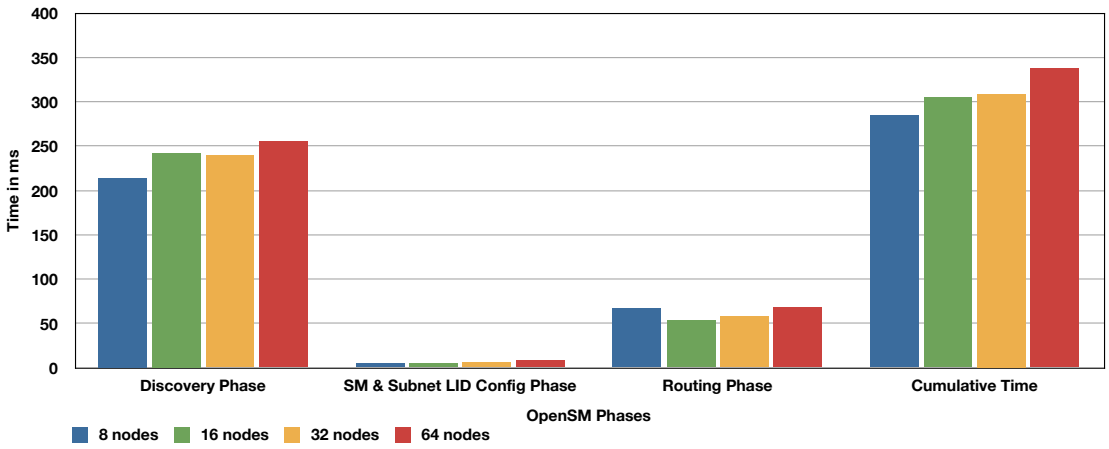

Figure 3.6: Performance of OpenSM with DOR Routing

FTREE

In Figure 3.7, we plot the time taken by OpenSM in its various phases by using the set of benchmarks discussed previously in the design section. Here OpenSM is configured to run using the Fat Tree routing algorithm. The time spent in the subnet discovery phase increases by 11% when the system size increases from 8 to 16 nodes. It varies steadily with minor increments after that. The behavior of SM and subnet LID configuration phase is consistent with the behavior seen previously when DOR scheme is used. Finally, the total time taken to configure the subnet in this case increases steadily as the system size is varied from 8 to 64 nodes.

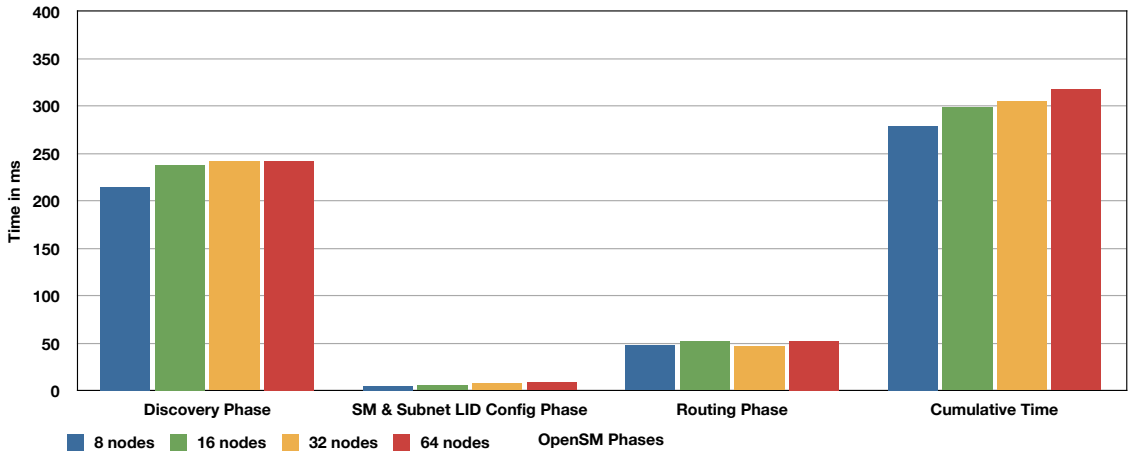

Figure 3.7: Performance of OpenSM with FTREE Routing

LASH

Figure 3.8 is obtained by using the set of benchmarks, mentioned in the previous section, while OpenSM is configured with layered shortest path routing algorithm. The time taken in the subnet discovery phase increases while the system size increases from 8 to 64 nodes. The time taken in the SM and subnet LID configuration phase increases steadily with the system size. But, we see an interesting behavior in the routing phase. The time taken drops by 27% when the system size increases from 8 nodes to 16 nodes. It then increases steadily after that. We see a similar behavior in the DOR routing scheme which is also optimized for 3D-torus and mesh topologies. The time taken in the routing phase is the same for both when the system size is 8 and 64 nodes. The total time taken by OpenSM to configure the fabric increases when the system size varies from 8 to 64 nodes.

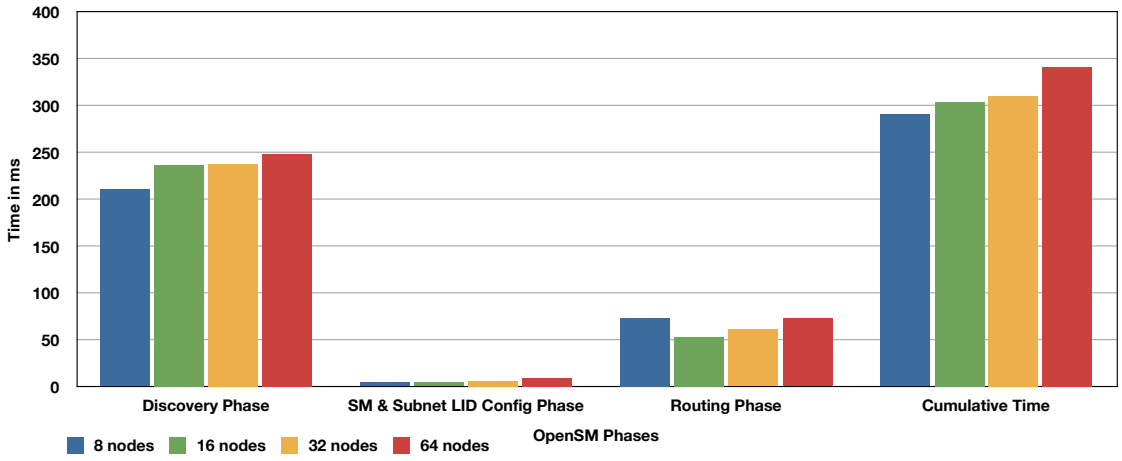

Figure 3.8: Performance of OpenSM with LASH Routing

MINHOP

In the case of Figure 3.9, OpenSM is configured using MinHop routing algorithm which is the default routing algorithm to which OpenSM falls back to when the option -R is not mentioned. We see an interesting behavior for the time taken in the routing phase as it stays almost uniform as the system size is varied. This behavior is because of the calculation of the MinHop matrix which does not incur much overhead with an increase in system size. The time taken in the SM and subnet LID configuration phase and the total time taken by OpenSM to configure the subnet show a steady increase with the system size.

UPDN

Here, in Figure 3.10, OpenSM is configured to use UPDN routing algorithm which is also a unicast routing algorithm. The total time taken by OpenSM to configure the subnet is similar to that of the pattern seen with all the FTREE and MinHop routing schemes. We see a spike of 27% in the routing phase when the system size varies

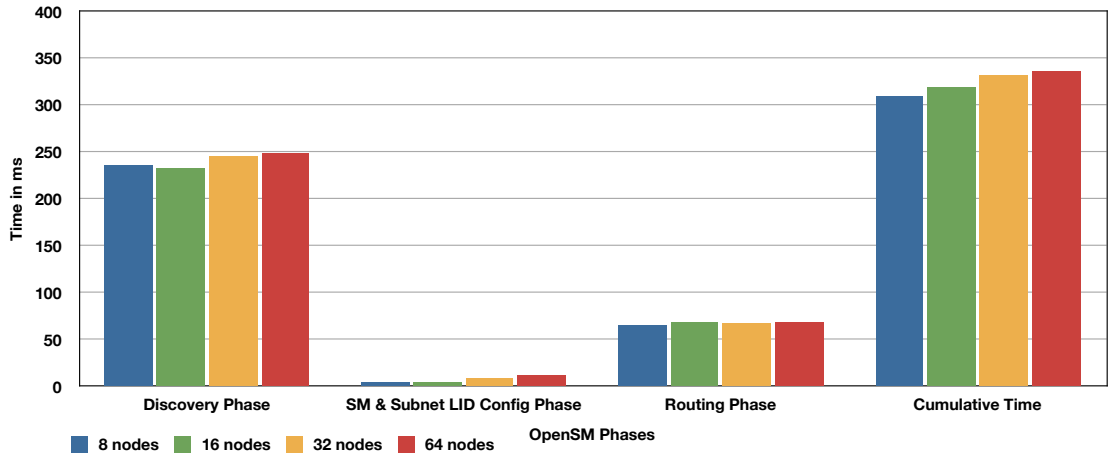

Figure 3.9: Performance of OpenSM with MINHOP Routing

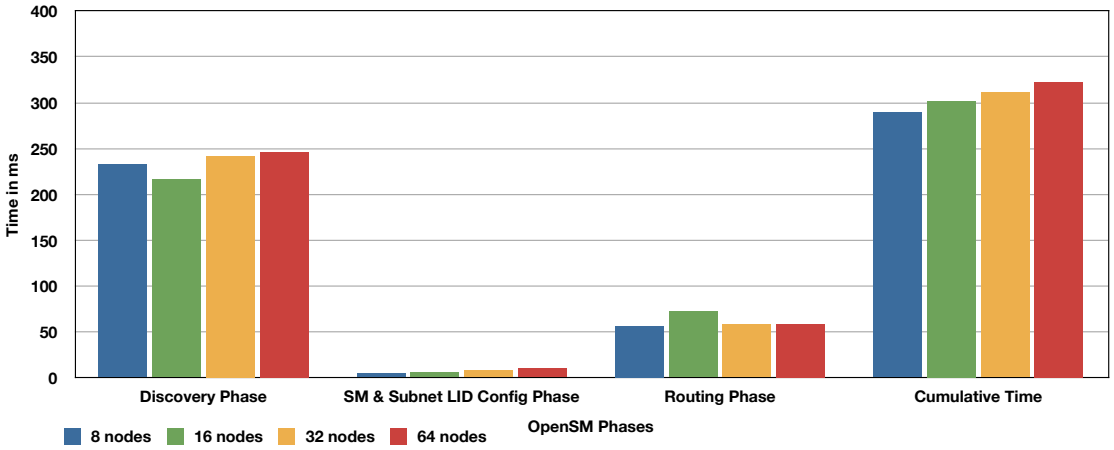

Figure 3.10: Performance of OpenSM with UPDN Routing

from 8 to 16 nodes. It then drops by 25% when the system size varies from 16 to 32 nodes. This behavior can be traced back to the Binary First Search (BFS) scheme used by UpDn. An initial BFS is done to rank the devices in the subnets according to hierarchy and then a second BFS is done to distribute the paths. The behavior of the time taken in the SM and subnet LID configuration phase is consistent with all the other routing algorithms.

3.4 Analysis and Observation

Our experiments show that route calculation and updating the switch forwarding tables with that information contributes to a major portion of the total time taken by OpenSM to configure the subnet. Another major aspect which affects the performance of OpenSM is the way in which OpenSM does the Subnet LID configuration as we expect the time taken in this phase to increase steadily as the system size increases. Since we already have path traversed from initial path array and return path array SM and Subnet LID config phase does not take as long as Discovery phase even though the process is the same as we already know the path traversed. In the following sections, we analyze the time taken by OpenSM in each of its phases for the various system sizes, for all routing algorithms. This is to determine which routing algorithm provides the optimal performance with OpenSM for a particular subnet.

3.4.1 Comparison of Routing Algorithms

In this section we analyze the time taken by each routing algorithm for a specific phase of OpenSM and then compare the different routing algorithms for that specific phase. The time taken in the subnet discovery phase is independent of the routing algorithm with which OpenSM is configured. The same is the case with time taken in

the SM and subnet LID configuration phase because it always increases steadily when the system size is varied from 8 to 64 nodes irrespective of which routing algorithm is used.

In Figure 3.11, we can see that FTREE routing algorithm performs considerably better in the routing phase when compared to other routing algorithms as the system size is varied from 8 to 64 nodes. As expected, the time taken is maximum when the system size is 64 nodes. In Figure 3.12, we can see that the cumulative time taken by OpenSM to configure the subnet is the least when FTREE routing algorithm is used for all systems sizes under consideration. We also see that the total time taken increases linearly as the system size increases. From the following graphs we can conclude that, for this particular experimental setup, FTREE is the optimal routing scheme.

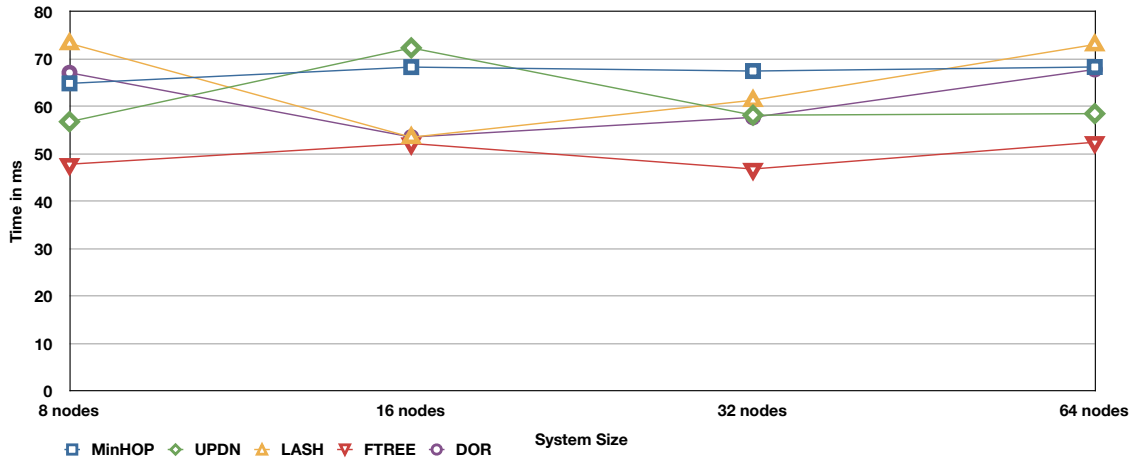

Figure 3.11: Time Taken by OpenSM in Routing Phase for Various Routing Algorithms

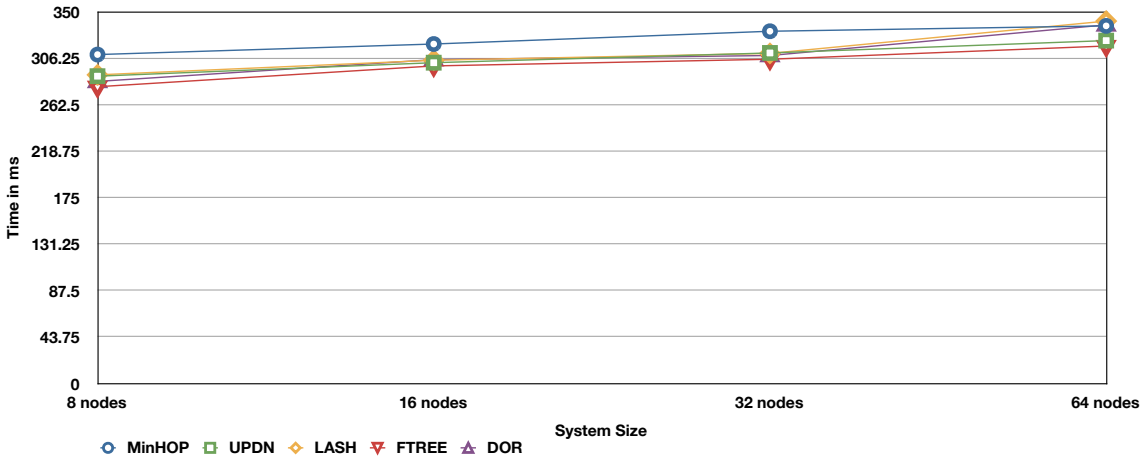

Figure 3.12: Cumulative Time Taken by OpenSM for Various Routing Algorithms

3.4.2 Analysis of FTREE routing scheme

After the LID assignment and configuration phase is complete, OpenSM starts configuring the switch tables. The final phase where the switch tables are configured takes up 55% of the total time spent in the routing phase. In this phase, the following functions are called to configure the subnet with FTREE routing scheme.

- *ucast_mgr_route* - starts building the routes with the specified routing algorithm. It falls back to MinHop routing algorithm if no routing algorithm is specified
- *construct_fabric* - called to starting the FatTree fabric construction. This function performs the following tasks
  - Populating the FatTree Switch and CA tables.
  - Reading GUID files provided by user (skip if not provided)
  - Ranking FatTree
  - Populating CA & switch ports

- *fabric_rank* - determines the maximum rank of a switch present in that particular subnet
- *fabric_mark_leaf_switches* - determines and marks the leaf switches in the fabric
- *fabric_make_indexing* - starts the indexing process for the FatTree
- *fabric_dump_general_info* - provides general fabric topology information which includes the FatTree rank (root to leaf switches), enumerating the number of switches at a particular rank all the way until the maximum switch rank, number of CAs, CA ports, switches
- *fabric_validate_topology* - makes sure that the topology of the given switch is FatTree indeed. If not, the fabric is configured with MinHop
- *osm_ucast_mgr_process* - FTREE tables are configured for all switches

3.4.3 Analysis of UPDN routing scheme

In case of UPDN routing scheme, a major part of the time spent in the routing phase is spent to configure the switch tables. This takes up 47% of the total time taken in the routing phase. The sequence of functions called is as follows.

- *ucast_mgr_route* - starts building the routes with the routing algorithm specified
- *osm_ucast_mgr_build_lid_matrices* - starts the switch's Min Hop table assignments as UpDn is dependent on the MinHop algorithm. It then finds the number of CAs and stores them in the cl_map.
- *updn_build_lid_matrices* - Ranking all port guids in the list

- *updn_subn_rank* - Ranking the subnet
- *updn_subn_rank* - Sets all switches to MinHop tables
- *updn_set_min_hop_table* - Initiates Min Hop Table of all switches and conducts a BFS through all port guids in the subnet
- *osm_ucast_mgr_process* - UPDN tables are configured on all switches

3.4.4 Analysis of LASH routing scheme

When LASH routing scheme is used, the major fraction (48%) of the total time taken in the routing phase is spent in the part where the switch tables are configured for unicast and multicast. The flow of function calls is as follows.

- *ucast_mgr_route* - starts building the routes with the routing algorithm specified
- *osm_ucast_mgr_build_lid_matrices* - Starting switches' Min Hop Table Assignment
- *discover_network_properties* - determines the minimum and maximum operational virtual lanes
- *connect_switches* - tries to connect to all switches in the subnet in pairs by determining the shortest path
- *osm_lash_process_switch* - determines whether the connection which was being established in the previous section was a success or failure
- *lash_core* - determines the required number of lanes and the available number of lanes and evaluates whether LASH can be used or not

- *ucast_mgr_route* - builds the forwarding tables for LASH

3.4.5 Impact of the Number of Nodes on a Leaf Switch on Performance of OpenSM

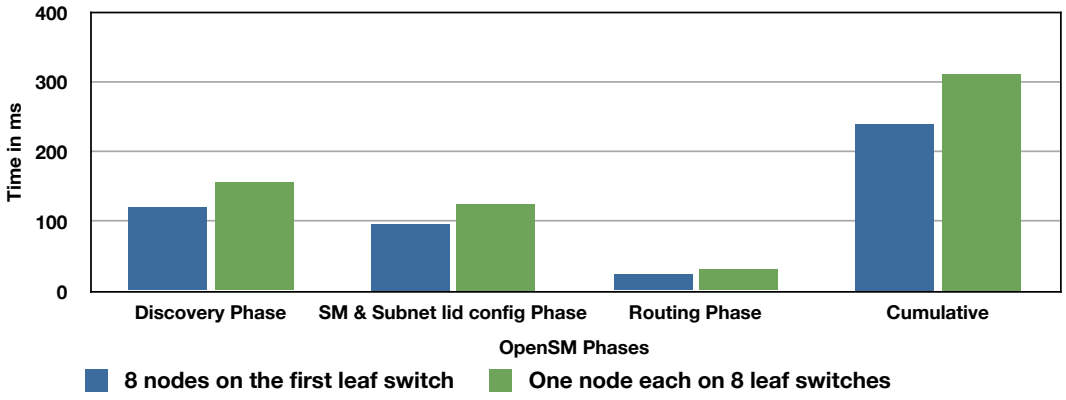

Figure 3.13: Impact of Number of Nodes per Leaf Switch on the Performance of OpenSM

The performance is also affected based on the number of nodes active on a particular leaf switch. As expected, the total time taken by OpenSM increases every time a new node is added to the subnet or to a leaf switch. But, the unanswered question is whether the node placement on the leaf switches affect the performance of OpenSM or not. Our experimental set up consisted of 11 leaf switches and 71 nodes. In the first case we just used 8 nodes and all of them were connected to a single leaf switch and all other leaf switches did not have any nodes connected to them. In the second case we had 8 leaf switches and we had one node connected to each of these switches. Then we measured the time taken by OpenSM in each phase while configuring these subnets using the benchmarks mentioned previously. As expected, the time taken by

OpenSM increased by 30% in the second case as compared to the first case. This behavior was consistent across all the routing algorithms. Figure 3.13 shows the above described behavior.

3.4.6 Variation in Minimum Number of Hops

Figure 3.14 shows the distribution of the minimum number of hops between all possible pairs of nodes in the subnet. The minimum number of hops between any pair of nodes is always the same irrespective of the routing algorithm used by OpenSM. The actual route taken by the packets may be different from the minimalistic route. This behavior can be attributed to the contention among the links and traffic hot spots in the subnet. In Figure 3.14, shown below, we can see that in a FAT Tree topology setting, the minimum number of hops between any two nodes can be divided into two categories based on whether the two nodes are located on the same leaf switch or are located on different leaf switches.

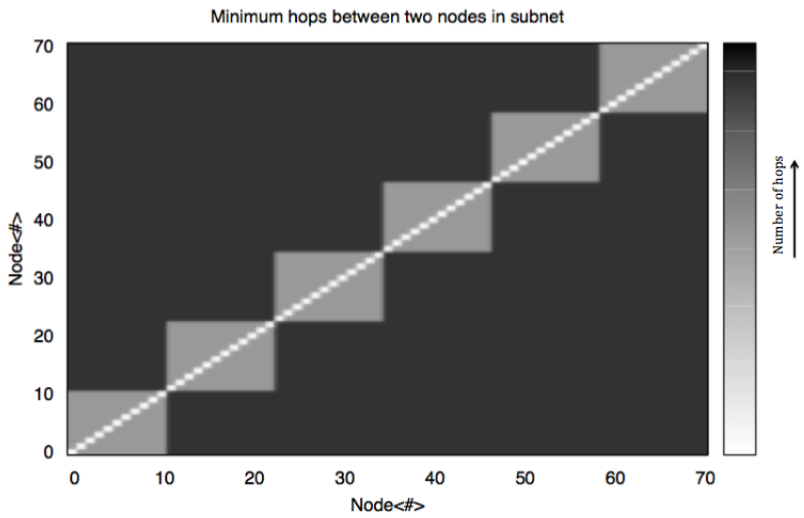

Figure 3.14: Minimum Number of Hops Between any Two Nodes in a Subnet

3.4.7 Reconfiguration Time of Various Routing Algorithms

Figure 3.15 depicts the time taken by OpenSM to reconfigure the subnet because of the occurrence of a failure. We used the Subnet_Reconfigure_Time benchmark mentioned in the previous section to quantify this value. We calculated the reconfiguration time by varying the factors listed below. It can be seen that FTREE routing algorithm takes the least amount of time to reconfigure the subnet in all of the following cases. We can see that the number of channel adapters per node is directly proportional to the time taken by OpenSM in configuring the subnet.

1. Routing Algorithm
2. System Size
3. Number of HCAs per node

3.4.8 Impact of Reconfiguration on Ongoing Computation

Figure 3.16 shows the time taken to complete an IMB alltoall as the number of failures are increased and when caching the unicast routes is toggled. We used the benchmark Impact_perf mentioned in section 3.2.2 to obtain these values. The message size used for the IMB alltoall is 512K and the system size used is 512 cores. We observed that OpenSM sweeps impact performance of parallel applications and Unicast route caching helps reduce performance impact. We initially calculated the time taken for an IMB alltoall (512 KB message size & 512 cores system size) operation to complete when there are no failures in the network. Then we used this to distribute the number of failures evenly while conducting this experiment. Thus, this shows that when a job is running on x nodes on a cluster of size y nodes,

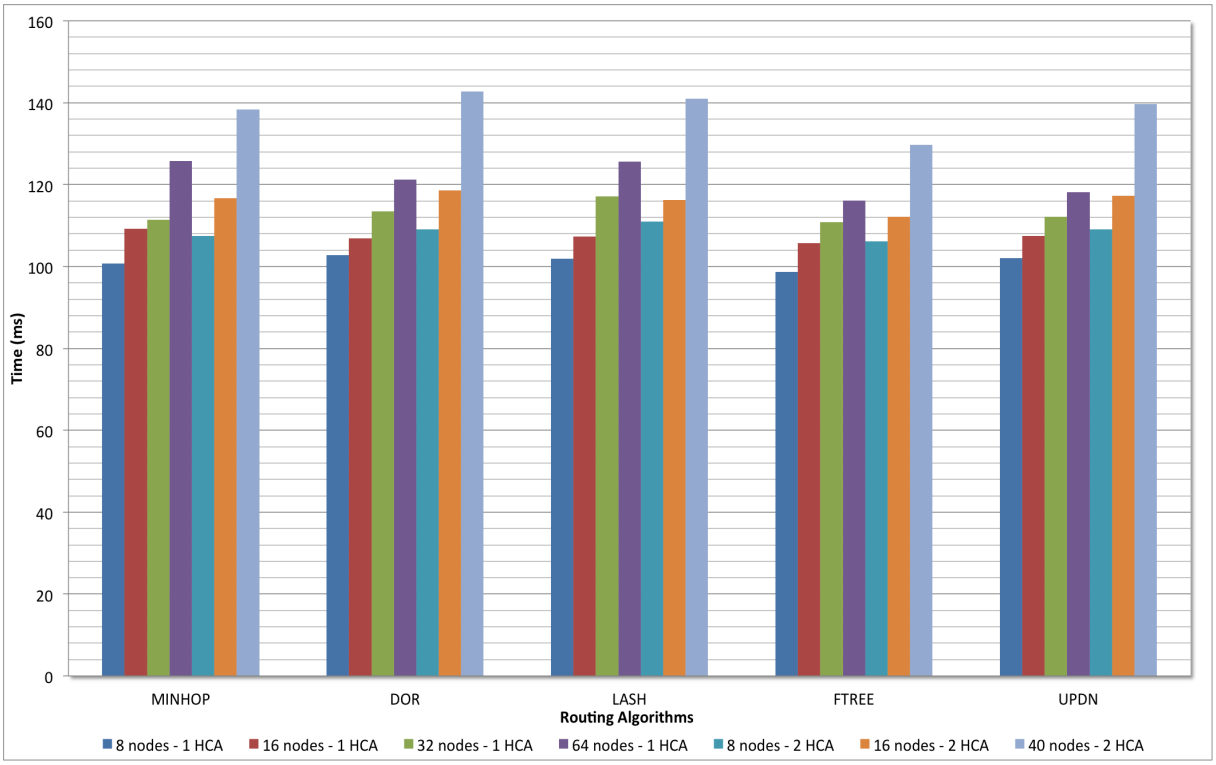

Figure 3.15: Time Taken by OpenSM to Reconfigure the Subnet for Various Routing Algorithms

this job would be impacted because of the heavy sweeps done by OpenSM if a node from the other "y-x" nodes fails.

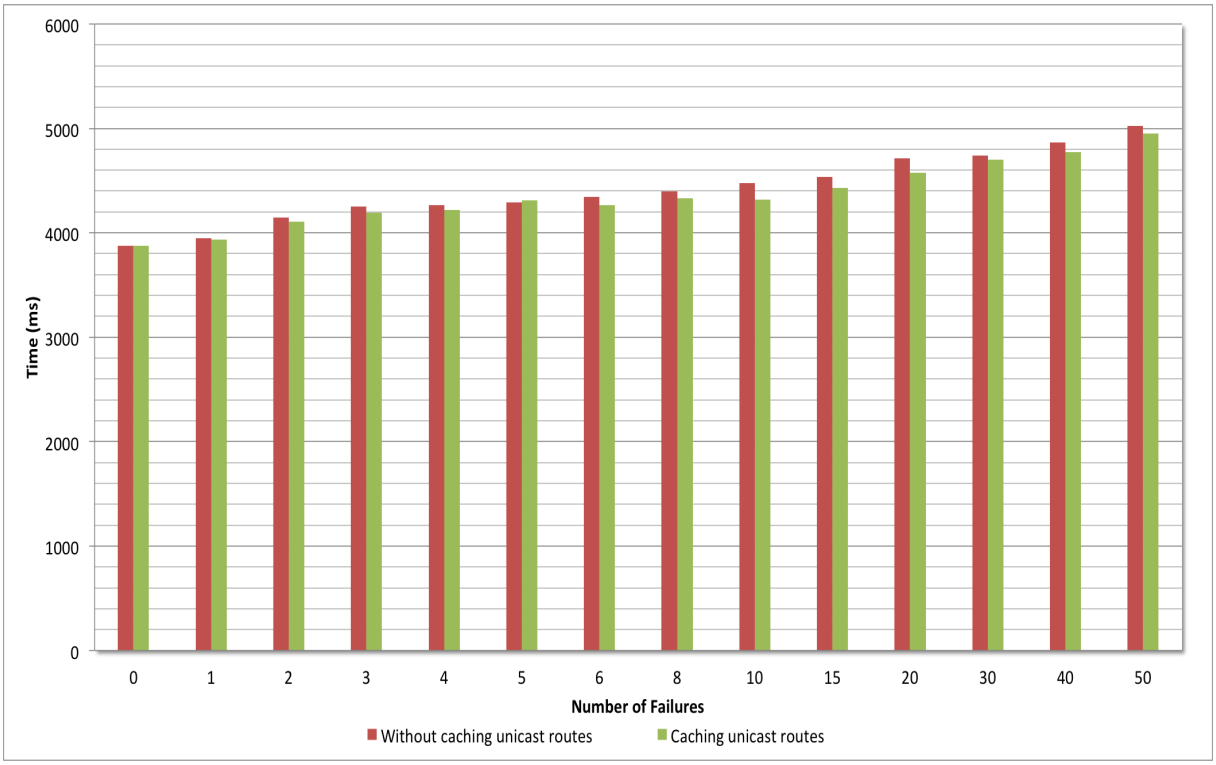

Figure 3.16: Impact of Reconfiguration on Ongoing Computation With and Without Caching Unicast Routing

3.5 Related Work

Earlier studies have evaluated OpenSM and its performance on certain networks but have not focused on bottlenecks of OpenSM and its performance at scale. [5] and [6] present a simulation based study to evaluate the subnet management mechanism. In [27], the authors focus on evaluating the subnet discovery time and the

various management phases within the discovery phase of OpenSM by varying the number of ports per switch, number of switches and the number of nodes. But these evaluations have been performed on a very small scale InfiniBand cluster. This analysis may not be consistent with the performance of OpenSM on a large scale InfiniBand cluster. There has not been much study on the potential bottlenecks of OpenSM and to study this we need to evaluate the performance of OpenSM the at a higher granularity. This work focuses exactly on those three aspects, bottlenecks, granularity and scalability.

Chapter 4: SCALABLE INFINIBAND NETWORK ANALYSIS AND MONITORING

4.1 Design And Implementation of INAM

We describe the design and implementation details of our InfiniBand Network Analysis and Monitoring tool (INAM) in this section. Figure 4.1 presents the overall framework for INAM. For modularity and ease of portability, we separate the functionality of INAM into two distinct modules - the InfiniBand Network Querying Service (INQS) and the Web-based Visualization Interface (WVI). INQS acts as a network data acquisition service. It retrieves the requested information regarding ports on all the devices of the subnet to obtain the performance counters and subnet management attributes. This information is then stored in a database using MySQL methods [18]. The WVI module then communicates with the database to obtain the data pertaining to any user requested port(s) in an on-demand basis. The WVI is designed as a standard web application which can be accessed using any contemporary web browser. The two modes of operation of the WVI include the live observation of the individual port counters of a particular device and the long term storage of all the port counters of a subnet. This information can be queried by the user in the future. INQS can be ported to any platform, independent of the cluster size and the Linux

distribution being used. INAM is initiated by the administrator and there exists a connection thread pool through which individual users are served. As soon as a user exits the application, the connection is returned to the pool. If all the connections are taken up, then the user has to wait. Currently the size of this connection pool is 50 and can be increased based on the requirement of the user.

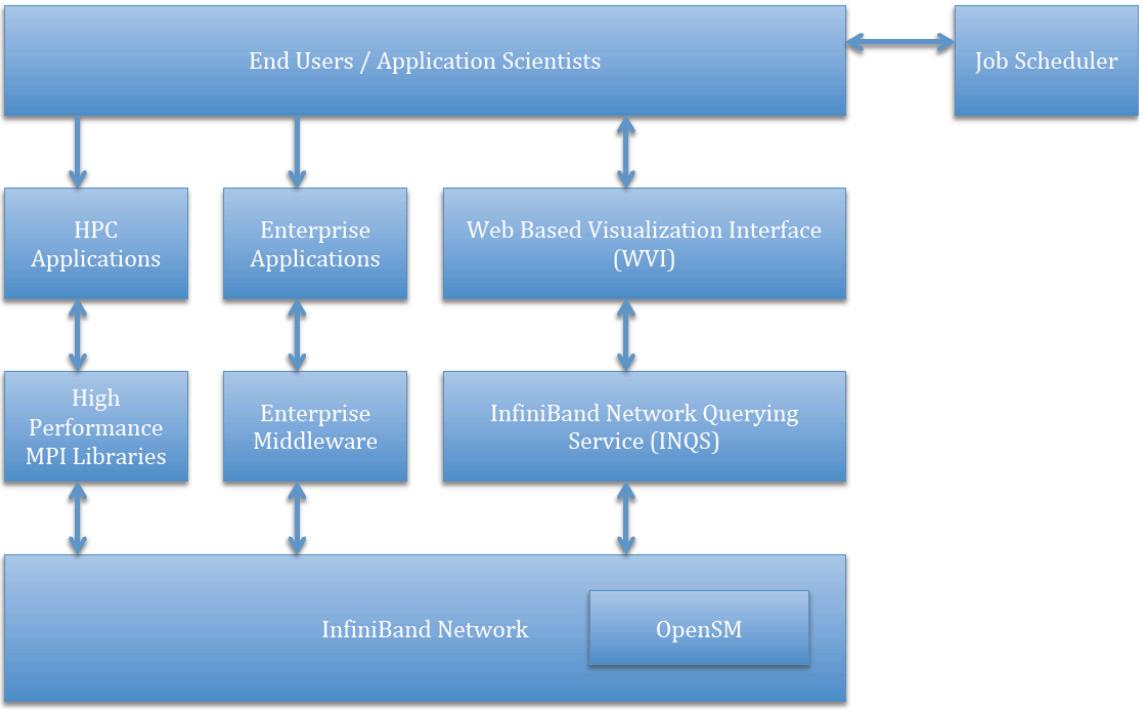

Figure 4.1: The INAM Framework

As we saw in chapter 1, a major challenge for contemporary IB network monitoring tools is the necessity to deploy daemon processes on every monitored device on the subnet. The overhead in terms of CPU utilization and network bandwidth caused by these daemons often cause considerable perturbations in the performance of real user applications that use these clusters. INAM overcomes this by utilizing the Subnet Management Agents (SMA) which are required to be present on each IB enabled

device on the subnet. The primary role of an SMA is to monitor and regulate all IB network related activities on their respective host nodes. The INQS queries these SMAs to obtain the performance counters and subnet management attributes of the IB device(s) on a particular host. The INQS uses Management Datagram (MAD) packets to query the SMAs. As MAD packets use a separate Virtual Lane (VL 15), they will not compete with application traffic for network bandwidth. Thus, compared to the contemporary InfiniBand network management tools, INAM is more responsive and and causes less overhead.

INAM is also capable of monitoring and visualizing the utilization of a link within a subnet. To obtain the link utilization, the *XmtWait* attribute alone or *XmtData* / *RcvData* and *LinkActiveSpeed* attributes in combination are used. The *XmtWait* attribute corresponds to the period of time a packet was waiting to be sent, but could not be sent due to lack of network resources. In short it is an indication of how congested a link is. The *LinkActiveSpeed* attribute indicates the speed of the link. This can be used in combination with the change in *XmtData* or *RcvData* attribute to see whether the link is being over utilized or not. In either case, we update a variable called the link utilization factor to depict the amount of traffic in the link. There is also an option to use just the INQS as a stand alone system to save the device port information and the link usage information over a period of time (time can be varied depending on the memory available) to analyze the traffic patterns over an InfiniBand subnet. INQS initially creates a dynamic *MySQL* database of all the available LID-Port combinations, along with the physical links interconnecting these ports. The LID-Port combination signifies all combinations of the device LIDs in the subnet and their respective ports. This information is updated periodically and thus

adapts to any changes in the network topology. The frequency at which the data is collected from the subnet and the frequency at which the data is displayed on the WVI can both be modied as per the requirement of the user. The overhead here would be associated with the WVI module. The display frequency can be reduced to 1 second and this would serve the users for all practical purposes. If this display frequency is less then 1 second, then we see a drop in the responsiveness of the dynamic graphs generated.

The WVI interacts with the database and displays the information requested by the user in the form of a graphical chart. Dynamic graphs are generated by using *HighCharts Js* [8]. We use a push model instead of a pull model to update the data in the WVI. The connection between the MySQL database and the WVI is kept open and hosting server pushes data to the browser as soon as the database is updated by INQS. This technique removes the overhead on the web server caused by the browser constantly polling the database for new data. This is implemented using a methodology called Comet [7]. This makes the web server stable and provides high availability even when deployed on large InfiniBand clusters with heavy data flow. The rest of the functionalities of the web server are implemented using Java 2 Platform Enterprise Edition (J2EE) [13]. The communication pattern of an MPI job is created by WVI by querying the database and then by using the canvas element of HTML5 [11] to chart out the physical topology and connections between the ports on a subnet.

4.1.1 Features of INAM

INAM can monitor an InfiniBand cluster in real time by using the functionalities provided by Open Fabrics Enterprise Distribution (OFED) stack. It can also monitor the link utilization on the fly and provide a post mortem analysis of the communication pattern of any job running on the IB cluster.

The user can select the device he wants to monitor through a dynamically updated list of all the currently active devices on the subnet. An option to provide a list of all the port counters which need to be compared in real time, is given to the user. A detailed overview of all the subnet management attributes of a particular port in a subnet can also be obtained. The attributes are divided into four main categories which are *Link Attributes*, *Virtual Lane Attributes*, *MTU Attributes* and *Errors and Violations*. INAM also provides dynamic updates regarding the status of the master Subnet Manager(SM) instance to the user. If there is a change in the priority of SM or if the Master SM instance fails or if a new slave SM takes over as a Master SM instance, the status is updated and the user is notified. This can help to understand the fail-over properties of OpenSM. Further more, a user can ask INAM to monitor the network for the time period of an MPI job and then it helps the user understand the communication pattern of that job using a color coded link utilization diagram.

4.2 Experimental Results

In this section, we describe the experimental setup, provide the results of our experiments, and give an in-depth analysis of these results.

4.2.1 Experimental Setup

The experimental setup is a cluster of 71 nodes (8 cores per node with a total of 568 cores) which are all dual Intel Xeons E5345 connected to an InfiniBand Switch which has an internal topology of a Fat Tree. We used a part of this cluster to show the functionality of INAM. This set up comprises of 6 leaf switches and 6 spine switches with 24 ports each and a total of 35 leaf nodes equipped with ConnectX cards. The functioning of INAM is presented using a series of benchmarks in varied scenarios. The first set of results are obtained using a *bandwidth sharing benchmark* to create traffic patterns which are verified by visualizing the link usage using INAM. The second set of benchmarks shows similar network communication patterns with *MPI_Bcast* configured for diverse scenarios. The third set of experiments verifies the usage of INAM using the LU benchmark from the SpecMPI suite.

4.2.2 Visualizing Port Counters

The user can select the device they want to monitor through a dynamically updated list of all the currently active devices on the subnet. The user can also provide a list of all the port counters they want to compare in real time. Figure 4.2 depicts how INAM allows users to visually compare multiple attributes of a single port. In this example, we show how two attributes - transmitted packets and received packets, of user selected port can be compared. Figure 4.3 shows a subset of the subnet management attributes which can be monitored by INAM. The list is dynamic and is updated as soon as a change is detected. Details regarding the physical link state are shown in the leftmost image and information regarding the SM is depicted in the image at the centre. The right most image shows the VL attributes of a port. The

device list at the top is dynamic and is updated if a new device is added or if an existing device is removed.

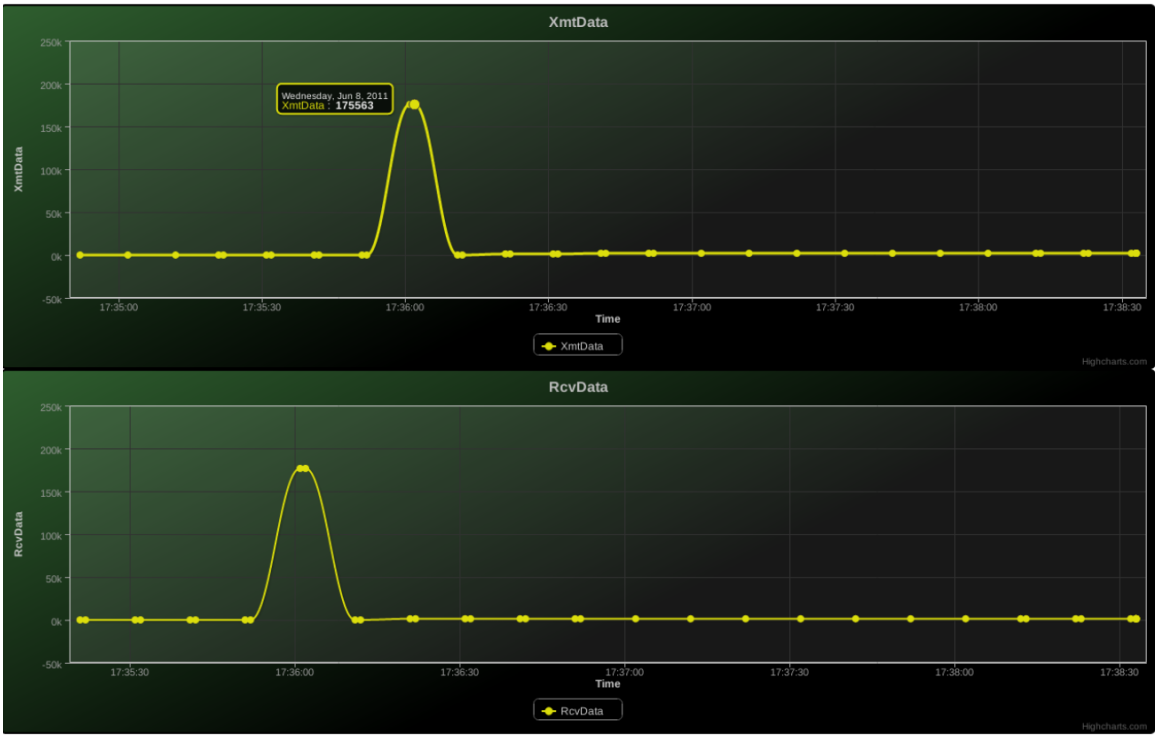

Figure 4.2: Monitoring the XmtData and RcvData of a port

4.2.3 Point to Point Visualization: Synthetic Communication Pattern

We create custom communication patterns using the bandwidth sharing benchmark mentioned in [26] to verify the functioning of INAM. The benchmark in question enables us to mention the number of processes transmitting messages and the number of processes receiving messages at leaf switch level and thus creating a blocking point to point communication pattern. We created various test patterns, each incrementally more communication intensive then the previous pattern, to help us notice a

Configure

Switch InfinIO3008 #3	Switch InfinIO3008 #4	Switch InfinIO3008 #6	Switch InfinIO3008 #8
ws26 HCA1	ws25 HCA1	ws5 HCA1	ws3 HCA1

Submit

LinkWidthEnabled	1X or 4X
LinkWidthSupported	1X or 4X
LinkWidthActive	4X
LinkSpeedSupported	2.5 Gbps
LinkState	Active
PhysLinkState	LinkUp
LinkDownDefState	Polling
LinkSpeedActive	5 Gbps
LinkSpeedEnabled	2.5 Gbps

LID	2
GUID	0x2c902002135dd
Activity Count	787060
Priority	0
Status	3

VLCap	VL0-7
VLHighLimit	0
VLArbHighCap	8
VLArbLowCap	8
VLStallCount	7
OperVLs	VL0-3

Figure 4.3: Monitoring the Subnet Management Attributes of a port

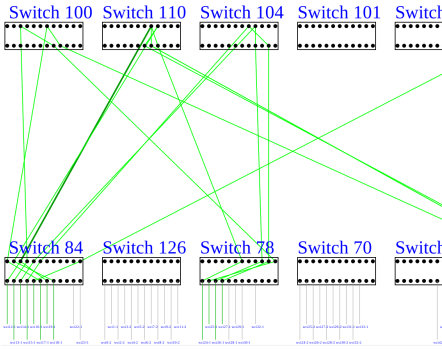

Figure 4.4: INAM depiction of network traffic pattern for 16 processes

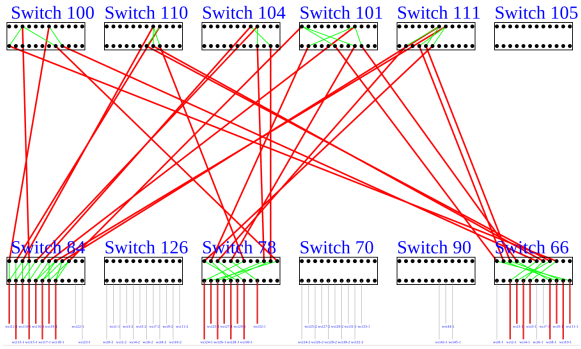

Figure 4.5: INAM depiction of network traffic pattern for 64 processes

difference in the pattern using INAM. Two of those patterns are mentioned in detail in the consequent sections.

Test Pattern 1

The first test pattern is visualized in Figure 4.4. The process arrangement in this pattern is such that 8 processes, one per each of the 8 leaf nodes connected to leaf switch 84, communicate with one process, on each of the four leaf nodes connected to

the each of the two switches 78 and 66. The thick green line indicates that multiple processes are using that link. In this case, it can be observed that the thick green line originating from switch 84 splits into 2 at switch 110. The normal green links symbolize that the links are not being over utilized, for this specific case.

Test Pattern 2

Figure 4.5 presents the network communication for test pattern 2. The process arrangement in this pattern is such that 32 processes, four per each of the 8 leaf nodes connected to leaf switch 84, communicate with two processes, on each of the eight leaf nodes connected to the each of the two switches 78 and 66. 32 processes send out messages from switch 84 and 16 processes on each of the switches 78 and 66 receive these messages. This increase in the number of processes per leaf node explains the exorbitant increase in the number of links being overly utilized. Figure 4.5 also shows that all of the inter switch links are marked in thick lines, thus showing that each link is being used by more then one process. The links depicted in red indicate that the link is over utilized. Since each leaf node on switch 84 has four processes and each leaf node on the other switches have two processes, the links connecting the leaf nodes to the switch are depicted as thick red lines.

4.2.4 Link Utilization of Collective Operations: Case Study with MPI_Bcast Operation

In this set of experiments, we evaluate the visualization of the *One-to-All* broadcast algorithms typically used in MPI libraries, using INAM. MVAPICH2 [17] uses the tree-based algorithms for small and medium sized messages, and the scatter-allgather algorithm for larger messages. The tree-based algorithms are designed to achieve

lower latency by minimizing the number of communication steps. However, due to the costs associated with the intermediate copy operations, the tree-based algorithms are not suitable for larger messages and the scatter-allgather algorithm is used for such cases. The scatter-allgather algorithm comprises of two steps. In the first step, the root of the broadcast operation divides the data buffer and scatters it across all the processes using the binomial algorithm. In the next step, all the processes participate in an allgather operation which can either be implemented using the recursive doubling or the ring algorithms.

We designed a simple benchmark to study the link utilization pattern of the MPI_Bcast operation with different message lengths. For brevity, we compare the link utilization pattern with the binomial algorithm with 16KB message length and we study the scatter-allgather (ring) algorithm with a data buffer of size 1MB. We used six processes for these experiments, such that we have one process on each of the leaf switches, as shown in Figure 4.6. In our controlled experiments, we assign the process on switch 84 to be the root (rank 0) of the MPI_Bcast operation, switch 126 be rank 1 and so on until the process on switch 66 is rank 5. Figure 4.6 shows a binomial traffic pattern for a broadcast communication on 6 processes using a 16KB message size. The binomial communication pattern with 6 processes is as follows:

- Step1: Rank0 → Rank3
- Step2: Rank0 → Rank1 and Rank3 → Rank4
- Step3: Rank1 → Rank2 and Rank4 → Rank5

In Figure 4.6, a darker color is used to represent a link that has been used more than once during the broadcast operation. We can see that processes with ranks

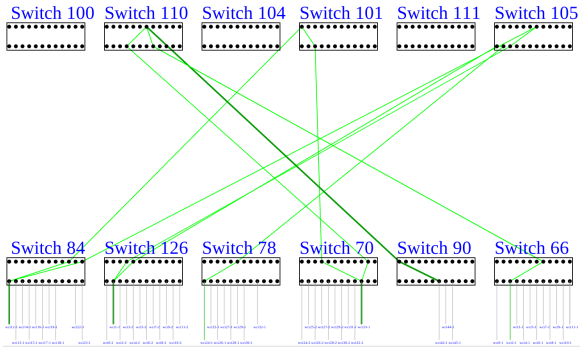

Figure 4.6: Link utilization of binomial algorithm

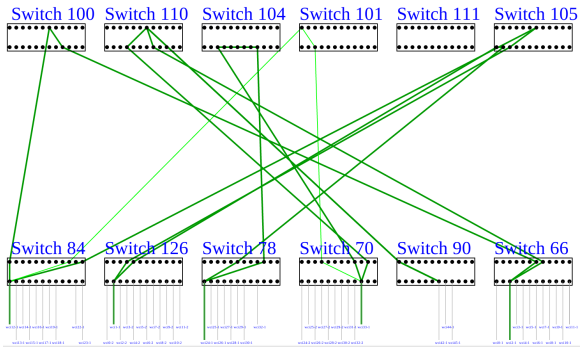

Figure 4.7: Link utilization of scatter-allgather algorithm

0 through 4, the link connecting the compute nodes to their immediate leaf-level switches are used more than once, because these processes participate in more than one send/recv operation. However, process P5 receives only one message and INAM demonstrates this by choosing a lighter shade. We can also understand the routing algorithm used between the leaf and the spine switches by observing the link utilization pattern generated by INAM. We also observe that the process with rank4, uses the same link between switches 90 and 110 for both its send and receive operations. Such a routing scheme is probably more prone to contention, particularly at scale when multiple data streams are competing for the same network link.

Figure 4.7 presents the link utilization pattern for the scatter-allgather (ring) algorithm with 6 processes. We can see that the effective link utilization for this algorithm is considerably higher when compared to the binomial exchange. This is because the scatter-allgather (ring) algorithm involves a higher number of communication steps than the binomial exchange algorithm. With 6 processes, the ring algorithm comprises of 6 communication steps. In each step, process Pi communicates with its immediate

logical neighbors processes $P(i-1)$ and $P(i+1)$. This implies that each link between the neighboring processes are utilized exactly 6 times during the allgather phase.

4.2.5 Application Visualization: SpecMPI - LU

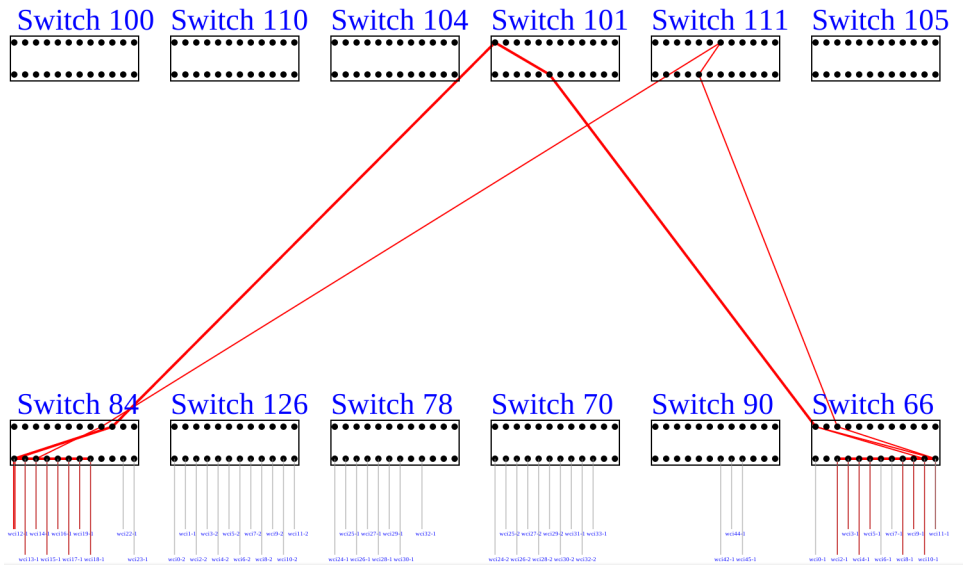

Figure 4.8: INAM depicting the communication pattern using LU benchmark

In this experiment, we ran the LU benchmark (137.lu medium size - mref) from the SpecMPI suite [19] on a system size of 128 processes using 16 leaf nodes with 8 nodes on each of the two leaf switches. The prominent communication used by LU comprise of MPI_Send and MPI_Recv. The communication pattern is such that each process communicates with its nearest neighbors in either directions (p2 communicates with p1 and p3). In the next step, p0 communicates with p15, p1 communicates with p16 and so on. This pattern is visualized by INAM and is shown in Figure 4.8. It can be seen that a majority of the communication is occurring on an intra-switch level.

4.2.6 Overhead of Running INAM

Since we use the subnet management agent (SMA), which acts like daemons monitoring all the devices of a subnet, we do not need to use any additional daemons installed on every device to obtain this data. This is a major advantage as it avoids the overhead in the contemporary approach caused by the daemons which are installed on every device. The user just needs to have the service *opensmd* started on the subnet. Since the queries used communication through Virtual Lane 15 for the purpose of data acquisition, there is no interference with the generic cluster traffic. For verifying this, we compared the performance of an IMB alltoall benchmark while toggling the data collection service on and off by using messages of 512 KB and 16 KB for various system sizes and then plot the percentage overhead. The results obtained are shown in Figure 4.9 which shows that the overhead is minimal even though the service is on. There is not much increase in the overhead when the message size is increased from 16 KB to 512 KB. Thus using the daemons already existing on each of the nodes boosts the responsiveness of the tool.

4.3 Related Tools

There is a plethora of free or commercial network monitoring tools that provide different kinds of information to the system administrators or the users. But only a few of them provide specific information related to IB network. We focus here on three popular network monitoring tools: *Ganglia* [14], *Nagios*[3] and *FabricIT* [24].

Ganglia is a widely used open-source scalable distributed monitoring system for high-performance computing systems developed by the University of California inside the Berkeley Millennium Project. One of the best features of Ganglia is to offer an

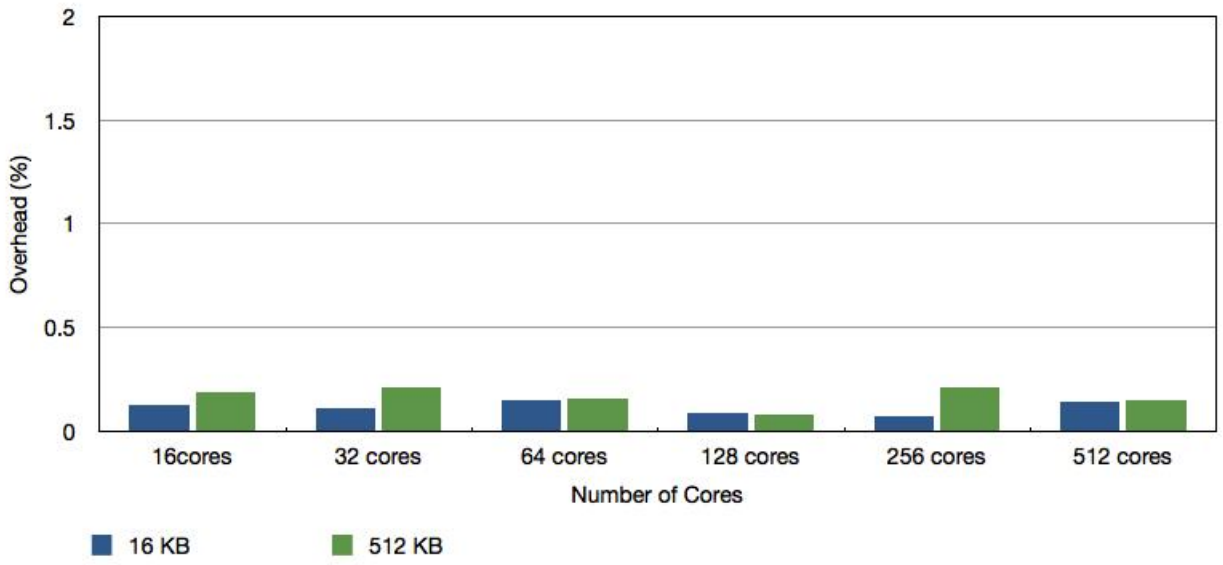

Figure 4.9: Overhead caused by running INAM

overview of certain characteristics within all the nodes of a cluster, like memory, CPU, disk and network utilization. At the IB level, Ganglia can provide information through perfquery and smpquery. Nevertheless, Ganglia can't show any information related to the network topology or link usage. Furthermore, to get all the data, Ganglia need to run a daemon, called *gmond*, on each node, adding an additional overhead.

Nagios is another common open-source network monitoring tool. Nagios offers almost the same information as Ganglia through a plug-in called "InfiniBand Performance Counters Check". But, as Ganglia, Nagios can't provide any information related to the topology.

FabricIT is a proprietary network monitoring tool developed by Mellanox. Like INAM, FabricIT is able to provide more information than Ganglia or Nagios, but the

free version of the tool does not give a graphical representation of the link usage or the congestion.

INAM is different from the other existing tools by the richness of the given information and also its unique link usage information, giving all the required elements to users to understand the performance of applications at the IB level.

Chapter 5: CONCLUSION AND FUTURE WORK

In this thesis, We focused on three aspects of OpenSM which are: bottlenecks, performance evaluation at higher granularity and performance evaluation at a finer scale. We design a set of micro benchmarks which can measure the time taken by OpenSM in the various phases of configuring the fabric. The results have been obtained by tweaking various aspects of OpenSM. We analyze these performances in various scenarios. We compare the amount of time taken by OpenSM in each phase when the system size is varied (number of nodes in the cluster). We conduct this experiment for all the five routing algorithms supported by OpenSM (MinHop, UPDN, FTREE, LASH and DOR).

We also compared the total time taken by OpenSM to configure the subnet by varying the system size for all the routing algorithms. We see that, for our experimental set up, OpenSM has a better performance when it's configured with FTREE routing scheme. We also see that OpenSM reconfigures the network in the least amount of time when FTREE routing scheme is mentioned. Another result worthy of mentioning is that, the process of reconfiguration undertaken by OpenSM in case of a failure effects an ongoing parallel application running on that subnet. We developed a web application which renders a global view of the subnet.

We have presented INAM - a scalable network monitoring and visualization tool for InfiniBand networks which renders a global view of the subnet through a web-based interface (WVI) to the user. INAM depends on many services provided by the open-source OFED stack to retrieve necessary information from the IB network. INAM also has an on line data collection module (INQS) which runs in the background while a job is in progress. After the completion of the job, INAM presents the communication pattern of the job in a graphical format. The overhead caused by this tool is very minimal and it does not require the user to launch any special processes on the target nodes. Instead, it queries on the IB devices directly through the network and to gather data.

In future, we would like to extend this work to do an on line analysis of the traffic patterns on a cluster. If next generation InfiniBand devices offer performance counters for each virtual lane, we could leverage it to study link utilization and network contention patterns in a more scalable fashion. Another dimension would be to create a time line graphical pattern to depict the exact amount of data being communicated in the subnet during a particular interval. We also plan to look into reducing the failure recovery time and the associated overhead. Another issue is to experiment with the osm_event_plugin provided by OpenSM and design an efficient way to log important events so as to obtain a better understanding of OpenSM. We would also like to extend the functionality of INAM such that the user can monitor and compare various counters from different ports. We would also like to show if the links are used multiple times simultaneously when the communication matrix is generated.

Bibliography

[1] Linux InfiniBand Project. http://infiniband.sourceforge.net. 14

[2] *IEEE Trans. Comput.*, 34(10), 1985. 4

[3] Wolfgang Barth. *Nagios. System and Network Monitoring.* No Starch Press, U.S. Ed edition, 2006. 63

[4] A. Bermúdez, R. Casado, F. J. Quiles, and J. Duato. Handling Topology Changes in InfiniBand. *IEEE Trans. Parallel Distrib. Syst.*, 18(2):172–185, 2007. 24

[5] A. Bermúdez, R. Casado, F. J. Quiles, T. M. Pinkston, and J. Duato. On the InfiniBand Subnet Discovery Process. In *CLUSTER*, pages 512–, 2003. 8, 20, 49

[6] A. Bermudez, R. Casado, F.J. Quiles, T.M. Pinkston, and J. Duato. Evaluation of a subnet management mechanism for InfiniBand networks. In *Parallel Processing, 2003. Proceedings. 2003 International Conference on*, pages 117 –124, oct. 2003. 8, 18, 49

[7] DWR - Direct Web Remoting. http://directwebremoting.org/dwr/. 54

[8] HighCharts JS - Interactive JavaScript Charting. http://www.highcharts.com/. 54

[9] T. Hoefler, T. Schneider, and A. Lumsdaine. Multistage Switches are not Crossbars: Effects of Static Routing in High-Performance Networks. In *Proceedings of the 2008 IEEE Cluster Conference*, Tsukuba, Japan, Sep. 2008. 9

[10] T. Hoefler, T. Schneider, and A. Lumsdaine. Optimized Routing for Large-Scale InfiniBand Networks. In *17th Annual IEEE Symposium on High Performance Interconnects (HOTI 2009)*, Aug. 2009. 5

[11] HTML5 - Canvas Element. https://developer.mozilla.org/en/HTML/Canvas. 54

[12] InfiniBand Trade Association, InfiniBand Architecture Specification, Volume 1, Release 1.0. http://www.infinibandta.com. 1, 13, 18

[13] Enterprise Edition (J2EE) Overview Java 2 Platform. http://java.sun.com/j2ee/overview.html. 54

[14] M. L. Massie, B. N. Chun, and D. E. Culler. The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. *Parallel Computing*, 30(7), July 2004. 63

[15] Mellanox OFED Stack for Linux Users Manual, Revision 1.5.2. http://www.mellanox.com/related-docs/prod_software/Mellanox_OFED%20_Linux_user_manual_1_5_2.pdf. 4, 7, 14

[16] J.M. Montanana, M. Koibuchi, H. Matsutani, and H. Amano. Balanced Dimension-Order Routing for k-ary n-cubes. In *Parallel Processing Workshops, 2009. ICPPW '09. International Conference on*, pages 499 –506, sept. 2009. 6

[17] MVAPICH2: High Performance MPI over InfiniBand and iWARP. http://mvapich.cse.ohio-state.edu/. 33, 59

[18] MySQL. http://www.mysql.com/. 51

[19] M. S. Mller, G. Matthijs van Waveren, Ron Lieberman, Brian Whitney, Hideki Saito, Kalyan Kumaran, John Baron, William C. Brantley, Chris Parrott, Tom Elken, Huiyu Feng, and Carl Ponder. SPEC MPI2007 - an application benchmark suite for parallel systems using MPI. *Concurrency and Computation: Practice and Experience*, pages 191–205, 2010. 62

[20] OFED. Open Fabrics Alliance. http://www.openfabrics.org/downloads/management/README. 3

[21] Open Fabrics Alliance. http://www.openfabrics.org/. 2

[22] OpenSM Release Notes, Revision 3.3. git://git.openfabrics.org/ sashak/management.git. 15

[23] T. Skeie, Olav Lysne, and Ingebjrg Theiss. Layered shortest path (lash) routing in irregular system area networks, 2002. 6

[24] Mellanox Technologies. Fabric-it. http://www.mellanox.com/pdf/prod_ib_switch_systems/pb_F 63

[25] Top500. Top500 Supercomputing systems. http://www.top500.org, November 2010. 1

[26] J Vienne, Maxime Martinasso, Jean-Marc Vincent, and Jean-Franois Méhaut. Predictive models for bandwidth sharing in high performance clusters. In *Proceedings of the 2008 IEEE Cluster Conference*, Tsukuba, Japan, Sep. 2008. 57

[27] A. Vishnu, Amith R Mamidala, Hyun wook Jin, and Dhabaleswar K. Panda. Performance Modeling of Subnet Management on Fat Tree InfiniBand Networks using OpenSM. In Workshop on System Management Tools on Large Scale Parallel Systems. In *In Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, 2005. 8, 19, 49

[28] X Xia, Yi Liu, Yunbin Wang, and Tengfei Mu. InfiniBand-Based Multi-path Mesh/Torus Interconnection Network for Massively Parallel Systems. In *Proceedings of the 2009 Fourth International Conference on Frontier of Computer Science and Technology*, FCST '09, pages 52–58, Washington, DC, USA, 2009. IEEE Computer Society. 4