January 2015

Identifying Data Exchange Congestion Through Real-Time Monitoring Of Beowulf Cluster Infiniband Networks

Michael James Aguilar

IDENTIFYING DATA EXCHANGE CONGESTION THROUGH REAL-TIME MONITORING OF BEOWULF CLUSTER INFINIBAND NETWORKS

by

Michael J. Aguilar
Bachelor of Science, Iowa State University, 1991

A Thesis

Submitted to the Graduate Faculty of the

University of North Dakota

in partial fulfillment of the requirements

for the degree of

Master of Science

Grand Forks, North Dakota
December
2015

This thesis, submitted by Michael J. Aguilar in partial fulfillment of the requirements for the Degree of Master of Science from the University of North Dakota, has been read by the Faculty Advisory Committee under whom the work has been done and is hereby approved.

____
Ronald Marsh, Chairperson

____
Hassan Reza

____
Travis Desell

This thesis is being submitted by the appointed advisory committee as having met all of the requirements of the School of Graduate Studies of the University of North Dakota and is hereby approved.

____
Wayne E. Swisher
Dean of the School of Graduate Studies

____
Date

PERMISSION

Title:	Identifying Data Exchange Congestion Through Real-Time Monitoring of Beowulf Cluster Infiniband Networks
Department:	Computer Science
Degree:	Master of Science

Michael J. Aguilar
November 19, 2015

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

This thesis would not have been possible without the help and support of many different people. To begin with, I would like to thank a group of System Administrators who inspired me to do the research that went into this thesis.

Secondly, I would like to thank my advisor, Dr. Ronald Marsh for providing guidance throughout the creation of the thesis. I would not have been able to organize my thoughts and move forward with the research without his guidance.

I feel very fortunate to have Dr. Hassan Reza and Dr. Travis Desell for members of my committee. Both of these committee members helped keep me motivated in class, and to do the research that was involved with this thesis. I feel I would not have made it through the rigors of the Master's Degree program without their support.

Finally, I would like to thank my wife, Heidi, for putting up with all of the hours of study and for giving up her kitchen counter to my laptop computers.

ABSTRACT

The ability to gather data from many types of new information sources has grown quickly using new technologies. The ability to store and retrieve large quantities of data from these new sources has created a need for computing platforms that are able to process the data for information. High Performance Computing Cluster systems have been developed to fulfill a role required for fast processing of large amounts of data for many difficult types of computing applications.

Beowulf Clusters use many separate compute nodes to create a tightly coupled parallel HPCC system. The ability for a Beowulf Cluster HPCC system to process data depends on the ability of the compute nodes within the HPCC system to be able to retrieve data, share data, and store data with as little delay as possible. With many compute nodes competing to exchange data over limited network connections, network congestion can occur that can negatively impact the speed of computations.

With concerns about network performance optimization, and uneven distribution of computational capacity, it is important for Beowulf HPCC System Administrators to be able to evaluate real-time data transfer metrics for congestion within a particular HPCC system. In this thesis, Heat-Maps will be created to identify potential issues with Infiniband network congestion due to simultaneous data exchanges between compute nodes.

CHAPTER I

INTRODUCTION

Relevance and Problem Definition

High Performance Computing Cluster (HPCC) systems are designed to process large amounts of data as quickly as possible (Dongarra, Sterling, Simon, Strohmaier, 2005). HPCC systems are an aggregation of processing components that can produce higher floating-point operations per second (FLOPS) processing then normal computing systems. While there can be many different types of designs for HPCC systems, the most common format of HPCC system is called a Beowulf Cluster (Gropp, Lusk, & Sterling, 2003).

Beowulf HPCC systems are constructed of many separate compute nodes that run asynchronous threads simultaneously. Applications written for Beowulf HPCC systems, unlike other types of computing platforms, are created with 'tight coupling' and 'low cohesion' (Mishra, & Mohanty, 2011). Thus, parallel threads running on compute nodes within a Beowulf HPCC system are expected to rely on each other and trade information frequently.

Networks within Beowulf HPCC systems must be fast and efficient enough so that there is not a slow exchange of data between threads. Slow data exchanges between compute nodes can affect overall FLOPS processing speed of an application. The ability of Beowulf Cluster communication networks to aggregate its compute nodes and provide data for tightly coupled threads determines how quickly an application running on a Beowulf HPCC system can process data for information.

Beowulf HPCC systems are flexible in design. Beowulf systems can be designed to be faster at processing fixed applications by providing better data exchange between specific threads processed on individual compute nodes.

While hardware design is important to FLOPS performance of an HPCC system, what is often overlooked is how an application can be written to distribute its threads across the compute nodes and how an application is written in regards to thread coupling. Applications expecting high FLOPS need to be distributed in a way that allows efficient and fast computation taking into account the design of the Beowulf HPCC system and its networks.

Most Beowulf HPCC systems exchange data using networks that are constructed from Infiniband network interconnects. Infiniband provides a network interconnection resource that is capable of quickly moving large amounts of data from compute node to compute node. The method for movement of data on Infiniband is Remote Direct Memory Access. Since RDMA does not require data exchange transfers to be processed through a CPU, no memory transfers require CPU time slices.

However, when working with RDMA there are tradeoffs that must be considered. These tradeoffs include the fact that all data moves are done as a block of data. So, application threads doing exchanges from compute node to compute node must wait for entire blocks of data to the transferred before an IO wait can be lifted.

Secondly, pages of main memory are locked out of Virtual memory paging. While transfers are performed using a page of memory, the paged memory used cannot be swapped out in favor of another waiting process thread.

Finally, all RDMA memory transfers are performed on a First In/First Out basis that forces the data from other compute nodes to wait in line for transfers. So, if multiple compute nodes are performing data exchanges over the same Infiniband network links, each set of compute nodes can be forced into a queue to wait its turn.

When there are RDMA transfer congestion issues with Infiniband, the effective FLOPS performance of a Beowulf HPCC system can be lowered. Application Programmers and Researchers can be left asking why expected processing times are increased. So, it is important to be able to identify when Infiniband network congestion is occurring what the precise effects are to the FLOPS performance of computing an application. Using this information, HPCC System Administrators and Application Programmers can work to improve the processing times of applications running on Beowulf HPCC systems.

Objectives

Infiniband is a common interconnection product used within Beowulf HPCC systems. In an effort to improve the FLOPS performance of a Beowulf HPCC system, it is helpful for System Administrators and Application Programmers to be able to gather metrics that show the run-time state of Infiniband. These metrics can help identify at what network traffic thresholds the shortcomings of Infiniband become a performance degradation issue. The metrics that are helpful with regards to Infiniband performance are the amount of network traffic congestion on specific Infiniband links and the effects of the network congestion.

Research in this thesis will be done to identify potential effects of Infiniband network congestion on the tightly coupled threads running on compute nodes. Once those potential effects are found, a run-time monitoring system will be created that provides an easy to read display format to show both System Administrators and Application Programmers areas of concern.

Finally, the new monitoring system will be tested on a production Beowulf HPCC system against random running applications.

Motivation

This thesis was inspired by my experience as a System Administrator with the University of North Dakota High Performance Computing Clusters and through collaborative discussions with other System Administrators working in the same field. A goal for HPCC System Administrators is to provide fast computing platform with consistent processing times for Application Programmers and Researchers.

While performing the normal functions of a System Administrator, it was discovered that computational speed of application programs seemed to vary from day to day. Unfortunately, the monitoring systems in use on the HPCC systems were not adequate to allows answers to why performance would not be uniform.

A High Performance Computing System is managed to perform difficult computations and research as quickly as possible. Not only do HPCC systems need to be reliable and have a good ‘up time’ for users, but the computational performance of the application programs performing research on an HPCC system should meet the time expectations of the Application Programmers and Researchers.

To gain better insight into the performance variations of the HPCC system, there was motivation to do research into various aspects of Beowulf HPCC designs that could cause variable computing performance.

Significance

In previous products for monitoring Beowulf HPCC systems, real-time analysis of performance metrics consisted of levels of activity of components and whether or not the components were operational. In some cases, the monitoring system added greatly to the CPU load on the compute nodes. It was important to create a metric gathering system that did not significantly add CPU load to the compute nodes and added in a way to gather and display the missing metrics regarding Infiniband congestion.

This thesis will create a monitoring system that lightly adds in the metrics needed to identify network congestion issues with a Beowulf HPCC system, and areas of uneven distribution of computations. The monitoring system will be a method to help improve consistency in application run-times and overall computational speed of applications.

Another point of the research in this thesis is to gain a deeper understanding of the interaction of very different components within an HPCC system. The research is designed in a manner that System Administrators and Application Programmers can be able to identify and highlight real issues with the interaction of compute node CPUs, memory, and networks in ways that haven’t been considered before.

Thesis Roadmap

This thesis is written is and organized into six chapters. Chapter II provides the background information on both Beowulf HPCC topologies and potential computational issues with Infiniband congestion. Chapter III explores efforts and shortcomings related to current Beowulf HPCC monitoring systems. Chapter IV explores what metrics will be targeted for the new monitoring system and the implementation. Chapter V will contain a detailed explanation of the algorithm used to generate the Heat Maps. Chapter VI demonstrates a Case-Study using a production HPCC system. Chapter 7 provides a final discussion and an evaluation of the results.

CHAPTER II

BACKGROUND

This chapter provides the reader of this document with a basic understanding of the concepts and definitions used as they relate to the particular research performed for this thesis in the field of High Performance Computing. The objective of this chapter is to provide the reader with both background knowledge in Beowulf High Performance Computing Clusters and performance issues relating to the networking strategies used within most Beowulf HPCC systems. This background knowledge will help prepare the reader to understand the benefits of this research towards improving the performance of running applications on a Beowulf HPCC system.

Analysis of Large Data Sets

The advent of digital computing and large storage systems has brought about the ability to assemble and analyze information in much greater detail than has been attainable before. With the ability to process increasing quantities of data, large data set applications are gathered together to improve human lives. For example, large data sets have applications in science, finance, sales, and anthropology. With increased quality and quantity of information that large data sets contain, new theories can be formulated or tested, and new understandings and discoveries can be made of the domain that created the data (Bell, 2013; Dubitzky, Kurowski, & Schott, 2012). To process large data sets, newer computing platforms are continually being modified and developed. Yet, computing with very large data sets can be a challenge. Large storage needs and robust data networks are required to assemble, exchange, and organize enormous amounts of data, plus when necessary, to process the data for information. Also, computing systems that are designed to analyze these data sets must be able to process the data in a consistent and reasonable amount of time.

Beowulf High Performance Computing Clusters

High Performance Computation Cluster (HPCC) systems are one type of computing platform used to do analysis on large data sets. HPCC system designs use tightly coupled parallel threads to process large data sets for applications that would be cumbersome on non-parallel computing platforms. For example, HPCC computation problems on very large sets of data might involve non-polynomial computation problems or large simulations. While there have been a number of designs of HPCC systems, the Beowulf Cluster design has become the dominant class of HPCC system in use today. A Beowulf High Performance Computing Cluster is a computing system that is aggregated out of multiple asynchronous components that work in unison together, orchestrated by a head node.

Large data set applications are submitted as batch jobs to a Beowulf HPCC system. A Beowulf HPCC system works by subdividing the work of computing a large data set application and making use of multiple threads to cooperate on computing a solution. Threads spawned from an application job on a Beowulf HPCC system are designed to run concurrently, whenever possible. The low cohesion parallel processes and threads from the batch jobs are distributed across a Beowulf HPCC system in dedicated compute nodes. The compute nodes separate CPUs and memories from each other. Because memories and CPUs are physically separate from each other, there is no need for job related threads to compete for CPU scheduler time intervals and main memory.

A typical Beowulf HPCC system is constructed of mass produced, off-the-shelf, commodity servers that are pieced together into a computationally complete system. The fact that commodity servers are used in the design and assembly of a Beowulf HPCC system, reduces the per-unit cost of the overall computing system (Dongarra et al., 2005; Gropp et al., 2003). A secondary reason for the popularity of Beowulf Cluster systems is due to the flexibility in design that is built into the system model. Flexibility in design comes with the fact that a complete computational system can be built and tasked to perform dedicated research with specific types of data and applications. When a research team is done processing computational tasks of certain set of applications, components of a Beowulf HPCC system can be reconfigured and reprogrammed to perform an entirely different domain of research in another area. In this manner, Beowulf HPCC systems can be designed to insure maximum performance for a specific group of researchers then modified to maximize performance for a completely different group of researchers.

As an example, one type of Beowulf HPCC system might be better suited to solving sparse matrices. Sparse matrix computations require trading of information with small amounts of processing in each compute node. Within a Beowulf HPCC system, a design could be created that worked by increasing the number of network connections and reducing compute node memory (Pothen & Fan, 1990). A Beowulf HPCC design might then be modified when requirements dictate that there is less data to be exchanged between compute nodes like a Beowulf HPCC system designed to compute nearly embarrassingly parallel computations. A Beowulf HPCC system designed to fulfill the requirements of computing nearly embarrassingly parallel computations would typically be created with less network interconnectivity but increased local memory within the compute nodes.

With the many advantages of the Beowulf HPCC system, there are some inherent performance limitations to the design arrangement. Even with embarrassingly parallel computations, a Beowulf HPCC system will simply not operate without some exchange of data between compute nodes and across the Beowulf HPCC system. Data exchanges occur to allow the head node to communicate the distribution of batch jobs to compute nodes, notifications from compute nodes that a sequence of a computation thread is completed, and data exchanges from compute node to compute node to fulfill the needs of threads that are tightly coupled with low cohesion.

The realization that data availability is important to parallel computations can be expanded from Gene Amdahl's discussions related to the availability of data to computational processes (Dubitzky et al., 2012). One of the laws, the Amdahl number law simply states that a processor will need some I/O per second when there is a number of instructions performed per second. In another law, the Amdahl's IOPS Ratio law, programs are expected to need to do I/O every 50,000 CPU instructions. Considering that I/O is expected to occur, and that concurrent running threads are tightly coupled, it becomes clear that Beowulf HPCC parallel systems will have network links that are aggregating I/O data exchanges from large numbers of compute nodes. These network links can be expected to carry a large amount of I/O and that Beowulf HPCC systems will perform better with higher bandwidth hardware/firmware network links.

Popular Beowulf HPCC Network Topologies

When designing a Beowulf HPCC system to compute a certain set of applications, a network topology is often chosen to match the application. This is done because certain threads within an application might have a more cohesive relationship with each other, as opposed to other threads. To attempt to provide a better way to provide data exchanges between these cohesive threads using the flexible design features of a Beowulf HPCC system, there are several different types of network and cluster topologies used in the layout and design of a Beowulf HPCC cluster (Deploying HPC Cluster with Mellanox Infiniband Interconnect Solutions, 2014). Three popular topologies used in Beowulf HPCC designs are a 'Flat Network' design where every node is connected to a common switch (Figure 1), a Fat-Tree network where tiers of switches are used for distribution of data (Figure 2), and a Torus network (Figure 3) design in which nodes are directly connected to one another. In a Flat network interconnection scheme, every compute node is equal in hierarchy to one another. For access from one compute node to another, information must traverse two network connections and one switch. This is a simple network schema that many smaller designs for Beowulf HPCC systems use.

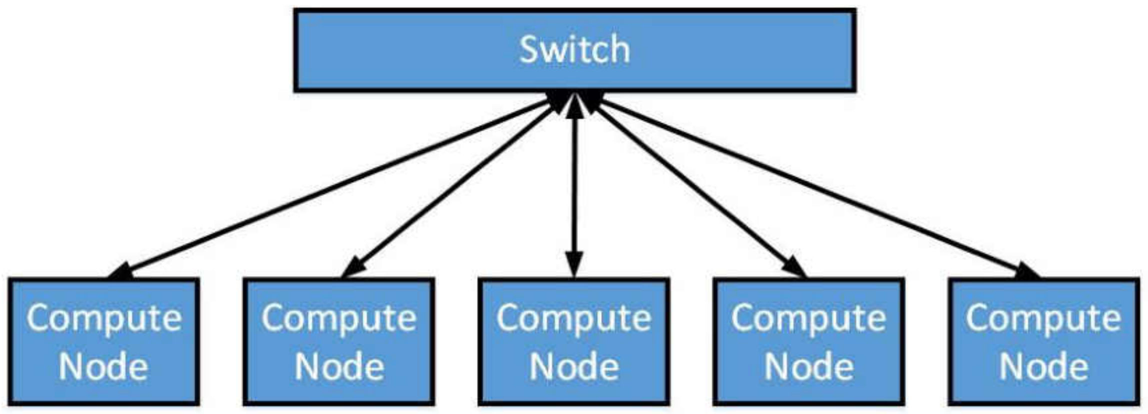

Figure. 1. A Beowulf Cluster Flat Network Design.

When large Beowulf HPCC systems are designed, network connections are designed with redundancy and data locality in mind. In a Fat-Tree network layout, as shown in Figure 2, the interconnections are all redundant and there is a hierarchical structure to the network system. Neighboring compute nodes are coupled to each other by using a local switch. Using a tiered group of switches, data locality for cohesive threads is created for compute nodes linked to the lower level switches. Thus, faster localized data exchanges can occur between compute nodes as they have shorter paths to traverse for data that remains local to that region of the HPCC system.

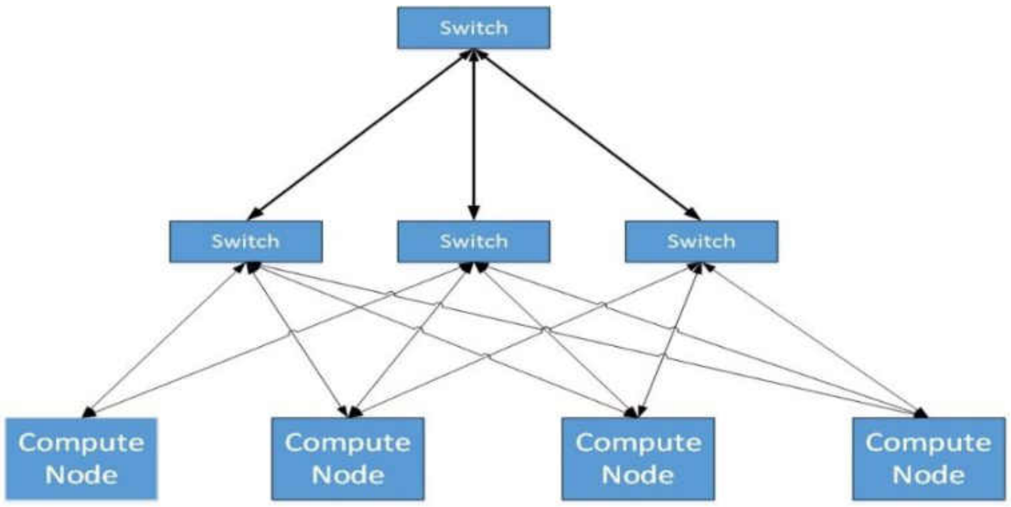

Figure. 2. Fat Tree Network.

Finally, the 3-D Torus layout, shown in Figure 3, creates node locality by tiling compute nodes together in the manner of a doughnut shape. Multiple paths exist for data exchanges. However, data exchanges must traverse compute nodes. This means that compute nodes that are in the path of a data exchange must be participants in the data exchange. As shown later, participating in data exchanges can reduce the computational performance of every compute node in the communication path. The advantage to the

Torus design is that many compute nodes are local to each other and the network connections are fully redundant.

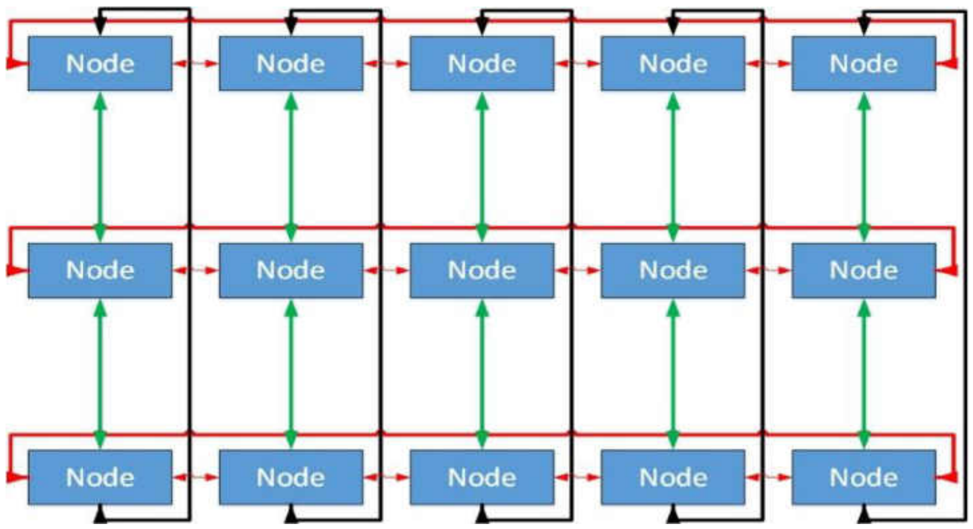

Figure 3. Beowulf HPCC System Design as a 3-Dimensional Torus Network.

Programming for Cohesive Threads

While data availability choices can be made in the hardware layout of a Beowulf HPCC system, there are other choices made programmatically when applications are created. Program choices can be made about placement of batch jobs on compute nodes and how often compute nodes exchange data. These are choices that are made that can create dynamic changes in connection availability and bandwidth. The ability to visualize the dynamic changes in activity within a Beowulf HPCC system network can aid a System Administrator and an Application programmer in creating better software. The ability to visualize both effects and side effects of network congestion can potentially lead to better distribution of compute node jobs and network traffic balancing. Even though it is not common for System Administrators and Application Programmers to concern themselves with how network activity affects their applications, software that is written for Beowulf HPCC systems do suffer computational performance issues when there is network congestion. The effects of network congestion may be more easily understood when an explanation is given of certain network and memory usage limitations that exist within most Beowulf HPCC cluster designs. These Beowulf HPCC systems use a common Beowulf HPCC system interconnection product called Infiniband.

Infiniband

Infiniband is a commonly used within Beowulf HPCC systems due to the ability of Infiniband to move data with lower latency than other types of network connections. Infiniband data rates vary by connection type (Infiniband Fact Sheet, 2015). The latest generation of Infiniband transmission links provide Quad Data Rate (QDR) at 10 Gb/s and Fourteen Data Rate (FDR) at 14 Gb/s. With an Infiniband data exchange, network communication is performed as a Remote Direct Memory Access (RDMA) data move. To prepare for an RDMA data exchange, Beowulf HPCC systems with most versions of Infiniband Device Drivers (Mellanox mxl4 and earlier) will lock up Virtual Memory to be able to transfer their blocks of memory without interference from possible changes to the main memory data integrity (Mellanox Infiniband Product Overview, 2015; Dreier, 2015; Liu, Wiu, & Panda, 2003). The memory is ‘pinned’. The Infiniband Device Driver will generate a Virtual Memory lock as soon as an Infiniband link becomes available for RDMA transmission or reception on the compute nodes involved in the data exchange. The memory lock consists of a full page size or more of memory in the Main memory.

In some cases, part or all of the Virtual Memory can be tied up while a process thread has to wait its turn to transmit or receive data over the network. Figure 4 shows how an example of how Virtual memory locks work while Infiniband data exchanges are performed. In the example, when ‘node A’ is performing a copy to ‘node B’, the sections of main memory being used for the transfer in both compute nodes is unavailable for other threads to use while the data exchange occurs. In another case, ‘node D’ must wait for a data exchange until ‘node C’ receives a data exchange from ‘node B’. The fact that Virtual Memory is under lock becomes an issue if another pending thread requires the use of the main memory under lock. Whether or not a process thread itself is in a blocked or unblocked state, the Virtual Memory is unavailable for swap out when another process thread receives a page fault (Remote Direct Memory Access, 2015). This means that pending threads that would benefit from attention while a job thread is paged out for an Infiniband I/O block cannot access memory since it is locked. This can lead to lowered CPU time resource utilization on compute nodes. One other issue is that the Virtual Memory being exchanged must fit completely within the Main Memory. This issue is created because the data exchanges are performed as a Direct Memory Address data transfer. Once the data exchange is complete and the transferred data is placed into main memory at the compute node destination, the Virtual memory lock is released and main memory within both compute nodes is readied for other threads to use.

For compatibility to Ethernet networks, Infiniband network connections have the ability to transmit and receive simulated Internet Protocol (IP) transmissions. The simulated IP transmissions are called Internet Protocol over Infiniband (IPoIB). IPoIB transmissions bypass the normal IP stack functions in the Linux kernel. In fact, simulated IP transmissions bypass the normal CPU IP controls and features because Infiniband

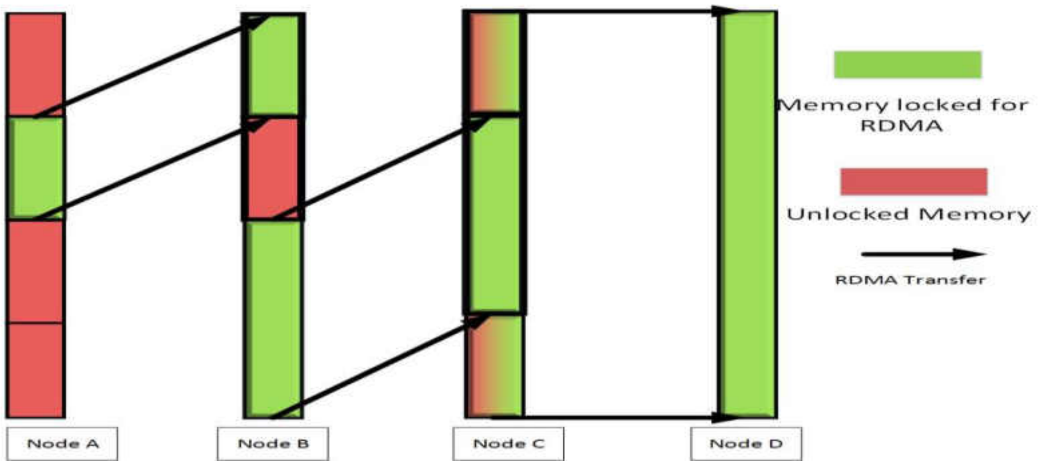

Figure 4. Virtual Memory Locked During RDMA Transmission.

IPoIB transmissions are RDMA data transfers (Rosen, 2015). IPoIB is also commonly used in normal Beowulf HPCC systems for functional tasks such as storing data on network connected data storage with the NFS protocol (Sandberg, 2013). IPoIB NFS storage information interchange requires the ability to perform remote procedure server and client calls, to pass on Access Control List information, to propagate file read and write locks, and to provide file system quota information.

Because data transfers using IPoIB are performed with RDMA, the actual network transmissions are single-sided. The receiving server is unaware that a transmission has taken place. This means that no regular IP 'handshaking' between a compute node that is transmitting data and compute node that is receiving data will take place, unlike the regular IP stack. This leads to some important considerations and issues when handling threads for processes that are involved in an IPoIB data transmission.

Using Cluster Metrics to Improve Beowulf HPCC System Performance

Improving computational performance with a Beowulf HPCC system involves making sure that as much of an application is programmed to run in parallel, as possible, that exchanged data is available when computations on each compute node need them, and that data exchanges occur as rapidly as possible to free up main memory within each compute node.

When applications require large quantities of data exchanges, Infiniband network congestion can slow down entire computation applications. For example, some parallel computations performed on Beowulf HPCC systems can consist of N-body computations, matrix computations for differential equations, simulations, and other types of computations (Ford, 2015). Each compute node will contain a thread that processes a subset of the full matrix calculation, programmed as a running batch job. Both rows and columns in a matrix computation often need to be traded multiple times before an application program has generated a result. With a matrix computation or an N-body simulation, a full computation often requires that each step of the computation performed receive a row update of data or even a state update of data from peer compute nodes. Performance that is less than optimal in computing a single parallel thread, on a single compute node, can degrade the performance of an entire computation because of the tight coupling between neighboring threads. Typically, as updates to computations occur, necessary the data and information that are interchanged between computational threads can transit one or many different compute nodes. Figure 5 shows an example of how matrix rows between nodes can be shared in an application. In this example, an input for a thread calculating a matrix on the current compute node might take the row of data from an adjacent compute node.

Network congestion can limit performance on batch jobs running on a Beowulf HPCC system. Uneven network traffic within Beowulf HPCC networks can mean that compute nodes are in contention to send data across the same network connections, while other connections may actually be available to exchange data. When a thread is running on a compute node, the exchange of data between compute nodes can cause memory locks and I/O waits for threads requesting a data exchange. It is important to keep network traffic from being concentrated on specific network links on the internal Infiniband network.

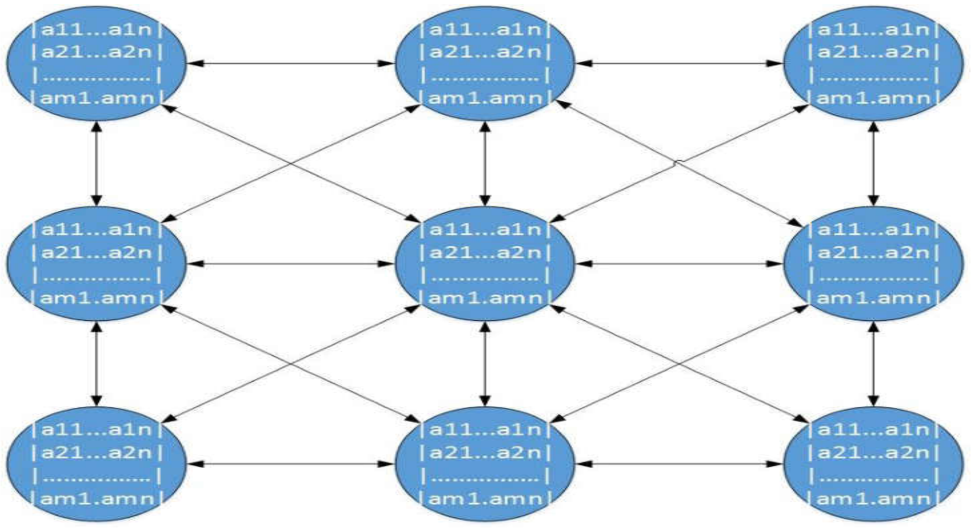

Figure 5. Matrix Computation Distributed Across Compute Nodes.

To verify that an application is balancing network traffic in a manner that prevents network congestion, a Beowulf HPCC System Administrator and an associated Application Programmer can gather the health and performance metrics of networks and components that make up a Beowulf HPCC system. The method for gathering metric performance data on Beowulf HPCC systems is through a monitoring system. Unfortunately, while monitoring systems exist for both real-time health and performance analysis of Beowulf HPCC systems, no monitoring systems exist to measure issues related to Infiniband network congestion. Typical Beowulf HPCC monitoring systems measure just the computational load and simple network metrics for each compute node and throughout the HPCC system. It is important to be able to measure Infiniband network congestion because of the potential for the congestion to degrade the performance of an HPCC computing application.

CHAPTER III

RELATED WORK

The goal of the research performed in this thesis is to create a monitoring system that will help System Administrators and Application Programmers understand the impact of network congestion created by programs running on real production Beowulf HPCC systems. In order to provide an analysis of Beowulf HPCC system internal networks Application Programmers need to be able to understand how often Infiniband congestion occurs and how to reduce the network congestion through improving the distribution of tightly coupled threads generated within their applications. The work in this thesis is related both to Beowulf HPCC system code optimization and previous research and development of Beowulf HPCC monitoring systems.

Beowulf HPCC Code Optimization

Optimization of application code for computational performance is an area of active research within the Beowulf HPCC community. For instance, at Argonne National Laboratories, 'optimized' code is created often while a Beowulf HPCC system is still in development (Messina, 2015). In some cases, enhanced programming code is developed for a specific Beowulf HPCC system without the code actually being run and inspected on a specific HPCC system. To save development time, often the strategy at Argonne National Labs is to create better application code before installation in an HPCC system. This is done by identifying portions of application code that can be run in parallel and threads that are highly cohesive during development and compilation of the software. Unfortunately, in these cases, once the application code is generated, no further analysis and review is done to the application while it is running on the cluster. There is no effort to determine if Infiniband network congestion might be degrading computing performance of the application.

To optimize HPCC application code it is desirable to gain information on deficiencies with how a particular HPCC system might be processing the code. In the paper, "Optimizing Latency in Beowulf Clusters", the authors discuss the effects of networks in computational performance of Beowulf HPCC systems (Garabato, More, & Rosales, 2012). The goal of the authors of the paper was to create recommendations for improved network performance with applications. Actual monitoring of network performance metrics was performed when various applications were run on Beowulf HPCC systems. However, the authors limit the performance tuning of a Beowulf HPCC system to the TCP/IP stack. While a cursory mention of Infiniband is given, the TCP/IP modifications are not effective with RDMA data exchanges. Also, the changes the authors make are general-purpose changes. There is no mention of analyzing applications for cases where application code might create Infiniband congestion on Beowulf HPCC systems.

Finally, in the paper, "Combining Congested-Flow Isolation and Injection Throttling in HPC Interconnection Networks", the authors do address network congestion on HPC computing performance (Escuer-Sahuquillo et al., 2011). The authors of this paper deduced that there would be effects of network congestion on Infiniband and TCP/IP networks and by reducing the congestion they could improve computational performance of applications on Beowulf HPCC systems. In the paper, the authors attempted to reduce network congestion by using active methods of re-routing of network traffic, queues, and injection throttling. However, there was no effort to monitor real-time Infiniband network activity created by running applications. Such monitoring of Infiniband traffic could have aided Application Programmers and System Administrators in addressing possible code changes that might have also helped to reduce Infiniband network congestion issues, as well. Code analysis of an application running on a Beowulf HPCC system, in such an effort, would need to reflect actual network activity changes over a portion of time to be useful.

Beowulf HPCC Monitoring

Time dependent monitoring of performance issues within Infiniband networks can help present to both Application Programmers and System Administrators periods of time when network congestion occurs on Infiniband networks during an application run on a Beowulf HPCC system. It is important to this thesis, to understand why other popular monitoring systems used in Beowulf HPCC system designs are not able to provide good information on Infiniband network congestion. Monitoring systems are an important element of Beowulf HPCC systems designed to provide both cluster health information and performance statistics to both System Administrators and Application Programmers. Because Beowulf HPCC systems are flexible in design and may contain several different network topologies, monitoring systems for Beowulf HPCC systems must be elastic enough to be able to gather metrics from each component of the cluster. Since there are a large number of compute nodes and network interconnects assembled into a Beowulf HPCC system, it is important for monitoring systems to be able to aggregate metrics from all of these component sources into an easy to read display. There are currently many different options for monitoring systems that are used for health and performance metrics for computing systems. The importance of gaining meaningful metrics on the quality of an HPCC system, has been an inducement for many System Administrators, organizations, and companies to develop their own monitoring systems. Yet, because System Administrators and Application Programmers have differing views on what constitutes a meaningful metric, each monitoring system provides very different methods for aggregating and displaying those metrics.

One example of a common monitoring system used in Beowulf HPCC systems is Nagios (Kocian, 2014). Nagios is a well-developed monitoring system for distributed computing systems. Nagios gathers metrics from individual components within a Beowulf HPCC system through communication with a daemon service via a remote procedure call or through the use of a secure shell connection**.** Nagios contains commonly used monitoring metrics and also allows a System Administrator to create extra metrics through the use of custom programming. While Nagios is a popular monitoring system for Beowulf HPCC systems, when working with Nagios, it becomes apparent that the response time for each metric gathering session is not conducive to displaying real-time metrics from within an HPCC system. Also, increased metric processing on each compute node and large network data traffic are required by Nagios before it can analyze and display the metrics. Lastly, another issue with Nagios is the fact that it can be a challenge to quickly view individual metrics for a specific compute node due to the fact that the interface requires scrolling down a web interface to find a specific metric.

In an effort to improve metric readability for dynamic web cache and database queries, the Claspin monitoring system was created (Lynch, 2012). Claspin was developed by a Facebook team to improve metric display issues that are prevalent in monitoring systems like Nagios. Claspin displays it metrics via the use of Heat Maps so that System Administrators can quickly look over the operational state of a system of clustered components. However, while Claspin provides Heat Maps for an easy to read interface for System Administrators, the metrics gathered do not provide System Administrators real-time metrics for how applications are performing on their clustered systems.

To improve data gathering performance over the entire cluster, to allow a monitoring system to aggregate real-time metrics for processing, and to reduce the amount of metric processing load within the compute nodes, the monitoring system Supermon was created (J. Wang et al., 2011). Supermon concentrates on performance load levels for both the CPU in each compute node and the network connections using kernel modules. However, Supermon cannot isolate metrics related to degradation of performance of Infiniband networks. Also, metrics gathered by Supermon must be processed and displayed by an external program to generate a display of metrics.

Ganglia, Cacti, and Carbon are a group of well-developed monitoring systems for distributed computing systems (Massie et al., 2013; Kundu & Lavlu, 2009; Dixon, 2015). Ganglia, Cacti, and Carbon gather real-time statistics and provide graph functionality to display the information. Activity graphs do show System Administrators activity levels on Beowulf HPCC networks, but the information is limited to the amount of traffic on the network and how much CPU activity is on a particular compute node. However, Ganglia, Cacti, and Carbon do not provide information on data exchange performance on the Infiniband networks. Finally, the metrics that are gathered are available for analysis for a relatively short period.

Finally, another lightweight monitoring system, LDMS was created by Sandia Laboratories to provide raw metric data that can be processed and aggregated by a monitor daemon (Brandt, Devine, Gentile, & Pedretti, 2014; (Brandt, Gentile, Marzouk, & Pebay, 2005). Like Supermon, the metrics gathered by LDMS are simple and unprocessed raw metrics from each compute node. Thus, LDMS impacts the system as little as possible when gathering statistics. Unlike Supermon, however, specific user-level library modules can retrieve specific metrics that are important to System Administrators in a time consistent manner (Agelastos et al., 2015) and the daemons running on each compute node are fully programmable. Also, metric thresholds can be observed and checked to determine whether a compute node is out of range of its normal operation. LDMS is designed to provide real-time system-wide metrics for analysis of application resource utilization metrics and can give System Administrators a good picture of how a normal running HPCC system should look. It must be noted, however, that the authors do not specifically target Infiniband network links for analysis, links that can potentially cause a significant degradation of computational performance for an application. Further, using LDMS metric data from compute nodes in a different way could help garner HPCC System Administrators an understanding of how well applications utilize resources within a Beowulf HPCC system.

While other Beowulf HPCC monitoring systems have benefits with regard to verifying the current operational state of a Beowulf HPCC system, these same monitoring systems are not providing Application Programmers and System Administrators with information about why certain application programs might be slow or vary in run times. The monitoring system provided in this thesis is designed to specifically target Infiniband performance metrics. As has been discussed before, Infiniband network congestion has potential to produce uneven computational times for applications. Heat Maps created for this new monitoring system are easy to read so System Administrators and Application Programmers can easily pinpoint links where Infiniband network congestion is occurring.

CHAPTER IV

APPLICATION

This chapter contains a complete explanation of the design and creation of a new monitoring system to measure real-time Infiniband network congestion. The Infiniband monitoring system created for this thesis is designed to create an apparatus to show both Infiniband network congestion as it occurs and repercussions from the congestion that can cause slower and uneven computation times for applications. System Administrators and Application Administrators are provided with easy to read Heat Maps so that problem areas can readily be found and evaluated.

Requirements

To create a new monitoring system for network link congestion and its effects, it was necessary to be able to periodically gather both the amount of data in the process of being exchanged on each Infiniband network link and the amount of data that was being effectively blocked due to FIFO queuing. To gather the necessary Infiniband information in an unprocessed and granular form, service scripts were created to take advantage of the underlying metric gathering and aggregation software available from specific components within the Lightweight Distributed Metric System (LDMS). While typical usage of the LDMS monitoring system is to find compute nodes that are not functioning properly in an HPCC system (Ovis, 2015; The ins and outs of HPC, 2010), it was determined that raw statistics from LDMS metric gathering could also be used gather

Infiniband network statistics that this thesis was interested in. Once statistics were gathered, the Infiniband network information could then be processed and displayed using Heat Maps.

Infiniband statistics were acquired from each compute node Infiniband card port connected to an Infiniband network link. Each Infiniband network card connection port is called a Host Channel Adapter (HCA) (Infiniband Fact Sheet, 2011; (Mellanox Infiniband Product Overview, 2015). An HCA keeps running statistics of its operation and the statistical data that it collects can then be polled to gather granular metric information about Infiniband network link activity. Each data metric that was acquired and used for the new monitoring system matched a specific function of Infiniband data exchange that could be affected by network congestion. The metrics gathered are shown in the annotated class diagram in Figure 6. Metrics gathered from the HCAs included the rate of transmission of data, the rate of reception of data, and the rate of transmission waits. This was done so either a System Administrator or an Application Administrator would be able to match the rate of transmission waits, a performance degradation metric, to the level of Infiniband congestion on the network link.

As can be seen in Figure 6, the volume of data packets that can be transmitted, via IPoIB, is completely dependent upon the current value of the "rate_transmit_wait_bytes" metric value of an HCA. The rate of packet transmission and reception would be effectively reduced in performance as a result of the HCA spin-waits. Further, since NFS data exchanges are transmitted using IPoIB, NFS data exchanges depend upon the bandwidth of data that can be transmitted via IPoIB. Both IPoIB metrics and NFS metrics were recorded and displayed within the new Infiniband monitoring system as a way to visually understand part of the unintended consequences of having Infiniband network congestion.

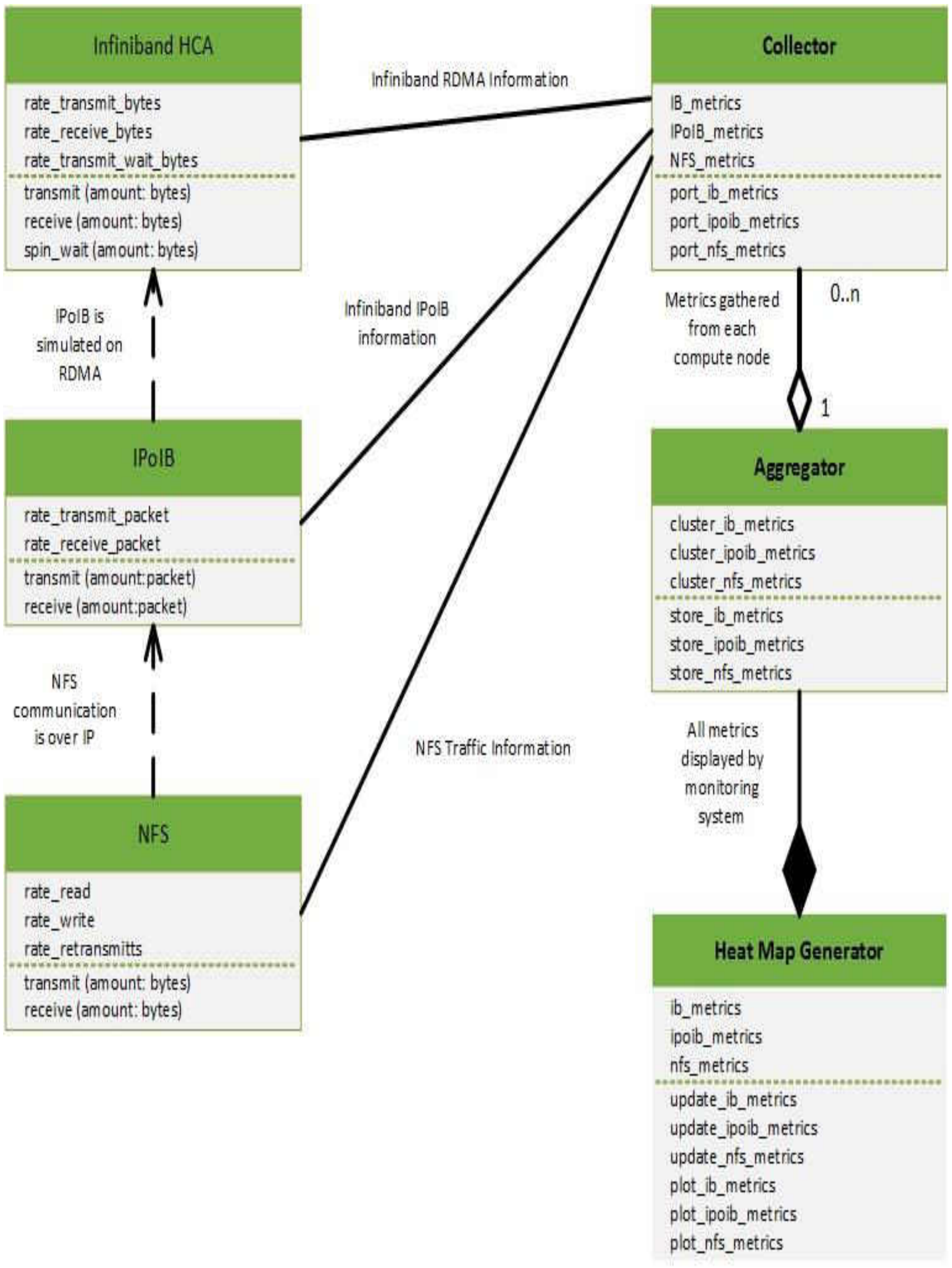

Figure 6. Annotated Class Diagram Showing Monitored Infiniband Metrics.

Development Platform

The Heat Maps developed for this thesis required Infiniband network activity from an operational Beowulf HPCC system for development. The University of North Dakota Hodor Beowulf HPCC system, consisting of 32 compute nodes and 256 cores, was chosen for development and testing. The Hodor Beowulf HPCC system is used by many distinct research applications such as Gromacs, Quantum Espresso, and other custom software. Local disk storage is used for both the compute node OS and for swap space eliminating the need for Virtual Memory page swaps to a shared 'diskless' storage server through NFS. On the Hodor system, the /home directory is mounted from a common NFS storage using IPoIB for each compute node and a master node. Each compute node contains either an NVidia GPU or an Intel MIC co-processor and these cards communicate through IPoIB and RDMA, as well.

The Hodor HPCC system is designed with a Flat Infiniband network topology. Each compute node in Hodor can reach another compute node by simply traversing a network link to the Infiniband switch, then down another link. Each HCA on a Hodor compute node is a Mellanox version Connect-X3 installed with Mellanox mlx4 device drivers utilizing Fourteen Data Rate (FDR) communications. The mlx4 device drivers used in the Hodor compute nodes are written to use Virtual memory 'pinning' to lock main memory and prepare for RDMA data exchanges.

Infiniband Metric Gathering at the Compute Nodes

The objective of the new monitoring system created for this thesis was to acquire real-time network traffic and congestion information from each Infiniband network link for analysis. It was important to have a portion of the monitoring software be operative from within each compute node so that every Infiniband network connection in the HPCC system could be analyzed. Metric information gathered at each compute node consisted of Infiniband HCA statistics and both IPoIB and NFS statistics from the OS Kernel. Table 1 shows a Use Case Description of the necessary features required of the compute node metric gathering service for the Infiniband network links. Each compute node in Hodor contained a single active HCA, a single IPoIB network link called IB0, and NFS statistics that would be read on one second intervals.

Table 1. Use Case Description of the Compute Node Metric Gathering Service.

Use Case Name	Gather Metrics
Actors	Infiniband HCA, OS Kernel, compute node metric service
Descriptions	Gather information from HCA, kernel IP metrics, kernel NFS metrics on each node.
Normal Flow	• Start running background thread for reading metrics on each compute node. • Every second read HCA information, IP information for IB0, NFS information for IB0. • Place metrics in memory for assembly by Aggregator.
Alternative Flow 1	• Background thread cannot start because service lock in place. • Stop
Alternative Flow 2	• Start running background thread for reading metrics on each compute node. • Every second read HCA information, IP information for IB0, NFS information for IB0. • Missing report on either HCA information, IP information for IB0, NFS information for IB0. • Skip missing metric and gather metrics on other categories. • Place metrics in memory for assembly by Aggregator.

In the event that the underlying LDMS metric gathering software failed to gather a set of IB statistics, or IPoIB statistics, or NFS statistics from a compute node, the LDMS metric gathering software would continue to gather and report the other statistics. Also, it was desired that the metric gathering software on each compute node would automatically start and run as a background service whenever the compute node was operational. A custom Red Hat Linux Operating System service program was created to control the operation of the LDMS metric gathering software. When this custom OS service program was started, a lock file was created in the OS process lock directory /var/lock/subsys to insure that multiple instances of LDMS metric gathering could not be run at the same time. When the metric gathering service was called to stop operation, a process kill signal function would halt the underlying LDMS metric gathering and the lock file would be removed from the process lock directory. A status call would list the LDMS metric gathering processes in operation and would provide the Linux OS process identifiers as an output.

The LDMS statistics were transmitted through the HPCC system's slower 1 GB Ethernet monitoring network so as to not interfere with Infiniband RDMA traffic. For metric data transfer, a POSIX socket directory was created using a shortened version of the hostname to allow later identification and matching of the metrics to a compute node. Each type of metric, whether HCA IB data, NFS data, or IPoIB data, was given a designated TCP/IP port and a run socket in the POSIX socket directory to be used by a master node collection service. When assigning TCP sockets for each type of metric, start information and errors were reported to the main Red Hat Operating System log,

/var/log/messages. The HCA IB metrics were transmitted through TCP port 60000. TCP port 60001 provided IPoIB information and TCP port 60002 provided NFS metrics.

The underlying LDMS software called from the OS service script was compiled separately into a designated compute node directory, /opt/ovis. Libevent-2.0 was installed to provide the method by which discrete sampling periods could be programmed into the LDMS metric gathering services. LDMS support libraries containing the necessary Infiniband metric gathering modules, the NFS modules, and the IPoIB modules were compiled and placed into the directory /opt/ovis/ovis.lib/. To aid LDMS in finding the libraries used in the metric gathering, the compute node BASH library path was modified to include the LDMS library installation path.

Metric Aggregation for Infiniband Monitoring System

In order to be able to create detailed Heat Maps of current Infiniband network activity, it was important for the HCA statistics from each of the compute nodes to be gathered together and stored for later data analysis. Another custom Red Hat Operating System service was created to manage an underlying LDMS program used for metric aggregation, then to sort, and store the data. Table 2 shows a Use Case Description of the aggregation service for the new Infiniband monitoring system. Again, it was desirable to have the metric aggregation start and stop automatically when the master node was turned on or off. The service script began by creating a 'lock file' to be used to prevent more than one instance of the service script. This 'lock file' was placed into the process lock directory /var/lock/subsys/.

Table 2. Use Case Description of the Aggregator and Store Service.

Use Case Name	Aggregator and Store
Actors	compute node metric service, head node aggregator
Description	• Assemble metrics from each compute node. • Organize the metrics into categories of IB traffic, IPoIB traffic, and NFS traffic. • Store each category for later retrieval and analysis.
Normal Flow	• Clock detects 1 second interval. • Open loop. Starting with first compute node. • Open port and read IB metrics from compute node. Close port. • Open port and read IPoIB metrics from compute node. Close port. • Open port and read NFS metrics from compute node. Close port. • If not last compute node, repeat. • Update files for IB metrics, IPoIB metrics, NFS metrics. • Repeat.
Alternative Flow	• Clock detects 1 second interval. • Open loop. Starting with first compute node. • Open port to read IB metrics from compute node. Compute node not reporting. Skip compute node. • Open port and read IB metrics from compute node. Close port. • Open port and read IPoIB metrics from compute node. Close port. • Open port and read NFS metrics from compute node. Close port. • If not last compute node, repeat. • Update files for IB metrics, IPoIB metrics, NFS metrics. • Repeat.

Important to the correct timing of LDMS storage of data to files was the amount of metric data collected before a ‘storage dump’ occurred. By analyzing and counting the number of metrics gathered from each compute node, using the ldms_ls function, and multiplying that number by the number of nodes (32), the total quantity of metric data before a storage dump was found to be 1952. In order to store data in a manner that would be usable by the Infiniband Heat-Maps, the underlying LDMS aggregation software was programmed to record the quantity of change in the metric data from each compute node. Metric data was stored in files labeled by the epoch time and by the TCP port that the metric data was retrieved from. For instance, the Infiniband metrics from TCP port 60000, were stored in a file with ‘sysclassib’ in the name field. The IPoIB statistics were stored in a file with ‘procnetdev’ in the title field from TCP port 60001. NFS file metrics were stored in a file with ‘procnfs’ in the title from TCP port 60002. To ensure that ownership of the files were not assigned to a user account that might reach a maximum account allotment, ownership of the data files was kept under the ownership of the Hodor local root user.

Heat-Map Creation

One of the design goals for the new Infiniband Monitoring System was to create a high quality display that would allow System Administrators and Application Programmers to be able to quickly identify areas of Infiniband network congestion and the resulting degradation of computational performance during the run of an application on a Beowulf HPCC system. Heat Maps provide a method of quantity measurement that is easy to read. A Heat-Map is a 3-Dimensional representation of data where z=f(x,y) is a color representation of the amplitude or quality of a measurement. Using Infiniband

Heat-Maps created for the new Infiniband monitoring system, the metrics retrieved from the measurement of congestion on Infiniband network link, the amount of back-up of data transmissions on each link, and the resulting performance degradation effects could be quickly identified.

Gnuplot was chosen to generate the Infiniband Heat-Maps. Each plot of data was based upon sampling of a compute node at a certain point in time. In the case of each Heat-Map, the X axis was chosen to be the sample time of the data. The Y axis was chosen to be the compute node measured. The metric changes to be displayed became the Z axis temperature display. Correlation of program performance could be done by checking both the current Heat-Map values and by checking the Heat-Map for previous periods of time. Gnuplot was set to display the resulting Heat-Maps as JPEG pictures.

To generate the Heat-Maps, a program was created to determine the names of the current metric files, synchronize the files to a local location for plotting, and then to drive each separate Gnuplot plotting program with the appropriate metric information. Eight different Heat-Maps were generated on a periodic basis to provide information on Infiniband read and write activity, Infiniband data backup, IPoIB packet transmit and receive rates, NFS read rates, NFS write rates, and NFS re-transmits. Table 3 provides a Use Case Description of the design of the Heat Map Display function of the new Infiniband Monitoring System. In order to ascertain the name of the current metric file, a secure shell call was made to list the metric storage directory contents. It was noticed that Linux would not complete the creation date until a file access was closed by an application, in this case, the LDMS aggregator. In the case that a previous day of metric files was to be examined the Heat Map software accepted the epoch name of the file set

Table 3. Use Case Description for Heat Map Display Function.

Use Case Name	Display Heat Maps
Actors	Local monitor node, head node.
Description	Update files with Infiniband metric information. Draw Heat Maps for IB congestion, IPoIB retransmits, NFS backup.
Normal Flow	• Find out what files have the latest run-time metrics in them. • Synchronize local IB, IPoIB, and NFS files with the latest metrics. • Translate epoch date from latest files to calendar date • Plot IB transmit traffic Heat Map. • Plot IB receive traffic Heat Map • Plot IB port transmit wait Heat Map • Plot NFS re-transmit Heat Map. • Plot NFS receive volume Heat Map. • Plot NFS write volume Heat Map. • Plot IPoIB receive volume Heat Map. • Plot IPoIB write volume Heat Map.
Alternative Flow	• User inputs epoch to get information on. • Synchronize local IB, IPoIB, and NFS files with the metrics from that epoch. • Translate epoch date from the requested files to calendar date • Plot IB transmit traffic Heat Map • Plot IB receive traffic Heat Map. • Plot IB port transmit wait Heat Map • Plot NFS re-transmit Heat Map. • Plot NFS receive volume Heat Map. • Plot NFS write volume Heat Map. • Plot IPoIB receive volume Heat Map

to be examined. The Linux function Rsync was used to synchronize file data at the monitoring node with the LDMS aggregator storage directory contained in Hodor.

It was determined by experiment that presetting the range of color bar outputs for the Heat-Maps to a fixed value would not work because each application provided a different quantity of congestion. Some applications generated more Infiniband network traffic in data exchanges than others. Experimentally it was found that summing both the mean and the mean absolute deviation of the plotted data provided an appropriate dynamic range to Heat Maps. Also by experiment, palette values were chosen that would give the Heat-Maps a wide color range from black, to blue, to green, to red for readability.

CHAPTER V

ALGORITHM

The Infiniband monitoring system created for this thesis was designed to answer questions about how often Infiniband spin-waits occur on specific HPCC systems and the conditions that can cause spin-waits. Another goal for the Infiniband monitoring system was to aid Application Programmers and System Administrators in reducing the amount of time spent computing an application by reducing the effects of Infiniband congestion and spin-waits to running applications.

Granular Metric Data Gathering

In order to provide analysis of Infiniband network links while an application job was being processed, the Infiniband monitoring system was required to be able to gather real-time metric data. It was necessary be able to gather consistent and detailed metric data while creating a small network and computational load for the Beowulf HPCC system. Both the Metric Data Gathering and Metric Data Aggregation components were designed and written to run as background services for the Red Hat and CentOS Operating Systems that are commonly used in Beowulf HPCC systems. The Metric Gathering Service (Table 1) was written to use LDMS metric gathering components but then acquired only the metrics that were important to research in this thesis. The Infiniband monitoring system information for each compute node was gathered in three portions, Infiniband HCA metrics, IPoIB metrics for IB0, and NFS metrics.

Choices were made to avoid using the Infiniband network links that were being studied to provide the gathered metrics. A second choice was made to provide the gathered metric information using POSIX sockets. It was noted that nearly every Beowulf HPCC system contains a dedicated 1Gb Ethernet monitoring network. The slower 1Gb network was adequate to provide the unprocessed data that would be provided by each compute node Metric Gathering Service. In order to use POSIX TCP sockets, it was noticed that some higher and unblocked TCP port numbers were available to convey the data across the 1GB network. Each compute node service was assigned port 60000 for Infiniband HCA metric information, port 60001 for IPoIB metric information, and 60002 for NFS metric information. The POSIX port numbers chosen were kept consistent across the Beowulf HPCC system, at each compute node. Information analysis of the Infiniband network links began by having the Metric Gathering Services started on each compute node.

A choice was made to gather metric data in one second intervals. In doing so, the network data congestion metrics could later be displayed as traffic level changes per second. Figure 7 shows a portion of the program code that was used to assign ports and gather at one second intervals. In the first portion of the displayed code, a POSIX socket has been previously created that contains both the compute node name and the metric type. An LDMS metric daemon has been requested to be put into service for each type of metric gathering required and matched with the appropriate metric socket. In the event of an error, the error is reported to the main Operating System log file. In the second portion of the displayed code in Figure 7, the metric gathering daemons are turned on with a request that data be gathered at a pace of 1,000,000 microseconds, or 1 second intervals.

```
ldmsd -x sock:60000 -S /var/run/ldmsd/$nodename/infiniband -l
    /var/log/messages > /dev/null 2>&1
ldmsd -x sock:60001 -S /var/run/ldmsd/$nodename/net -l
   /var/log/messages > /dev/null 2>&1
ldmsd -x sock:60002 -S /var/run/ldmsd/$nodename/nfs -l
   /var/log/messages > /dev/null 2>&1
                                               …
echo "load name=sysclassib" | ldmsctl -S
   /var/run/ldmsd/$nodename/infiniband
                                               …
echo "config name=sysclassib component_id="$component_id"
   set=$nodename/sysclassib" | ldmsctl -S
   /var/run/ldmsd/$nodename/infiniband
                                               …
echo "start name=sysclassib interval=1000000 offset=0" | ldmsctl
-S
   /var/run/ldmsd/$nodename/infiniband
```

Figure 7. Socket Definitions and Sampling Interval From Metric Gathering Service.

An Aggregator and Store service written to match the Use-Case description in Table 2 to assemble and store the metric data from each compute node using Hodor's 1Gb Ethernet monitoring network. The aggregation function portion of the Aggregate and Store service, shown in Figure 8, gathered the metrics from each of the compute nodes and stored them inside the underlying LDMS service. A loop was created to associate TCP sockets from each compute node to an LDMS service that was configured to aggregate and organize the metrics for later storage. Again, only the metrics necessary to perform analysis related to the Infiniband network links was used. Each compute node was given an identification that represented the compute node that the data was received from. To allow for transmission time for each compute node metric, it was found experimentally, that 1/10 of a second offset delay should be used. To match the metric gathering rate, the sampling interval was set at 1,000,000 microseconds, or 1 second.

```
Aggregate() {
node=$1
echo 'add host='$node' type=active interval=1000000 offset=100000
xprt=sock port=60000 sets='$node'/sysclassib' | ldmsctl -S
/var/run/ldmsd/infiniband
echo 'add host='$node' type=active interval=1000000 offset=100000
xprt=sock port=60001 sets='$node'/procnetdev' | ldmsctl -S
/var/run/ldmsd/infiniband
echo 'add host='$node' type=active interval=1000000 offset=100000
xprt=sock port=60002 sets='$node'/procnfs' | ldmsctl -S
/var/run/ldmsd/infiniband
}
```
Figure 8. Aggregation Function From Aggregate and Store Service.

Once the metrics were stored in the LDMS service, a metric ‘data dump’ was performed and the latest metric data changes were placed into storage, as is shown in Figure 9. Again, the goal of the real-time data gathering for the Infiniband network links and the associated IPoIB and NFS metrics was to show congestion on a per second basis. Therefore, only delta changes were stored in the data files. This requirement was met by assigning LDMS to only store the changes between the previous sample and the current sample. The compute node name was placed in the data file, for correct identification, in the first data metric position. After 1,000,000 entries, to keep the file sizes manageable, the file was requested to roll over into a new file. These metric files were matched to the same date and time as POSIX Epoch Time for later plotting.

```
ldmsctl -S /var/run/ldmsd/infiniband
echo "config name=store_derived_csv alt_header=1 id_pos=1
rolltype=3 rollover=1000000 derivedconf=/root/ldms_store.config
path="$storage_path | ldmsctl -S /var/run/ldmsd/infiniband
echo "store name=store_derived_csv comp_type=node set=sysclassib
container=sysclassib" | ldmsctl -S /var/run/ldmsd/infiniband
echo "store name=store_derived_csv comp_type=node set=procnetdev
container=procnetdev" | ldmsctl -S /var/run/ldmsd/infiniband
echo "store name=store_derived_csv comp_type=node set=procnfs
container=procnfs" | ldmsctl -S /var/run/ldmsd/infiniband
```

Figure 9. Store Data Function Using Metric Data Dump From Aggregator and Store Service.

To plot the metrics for later analysis, a complete set of programmed batch runs were performed on the Display Heat Maps portion of the Infiniband monitoring system to plot the metric data samples taken over several months on a separate monitoring server. Figure 10 shows the batch script used to plot all of the Epoch date files that were gathered. In order to plot the files, an Epoch time listing of all of the current files within the storage directory were gathered. Once a file list was created, the Epoch time parameter was passed to the heatmap_plot program written to fulfill the Display Heat Maps (Chapter 4, Table 3) portion of the Infiniband monitoring system. Using the new Infiniband monitoring system in this manner, sample metric data was gathered from Hodor between the months of July 2015 to October 2015 and plotted.

```
    #Retrieve the Epoch time list from the Infiniband metric files. #
    ssh root@hodor "ls /home/undmaguilar/ldms_stored_data/node | grep sysclassib | sed
's/[a-z][.]*//g'" > heatmap_file_list
    # Batch run the plotting of all of the Epoch files             #
    while read epoch ; do
        echo $epoch;./heatmap_plot $epoch
    done < heatmap_file_list
```

Figure 10. Batch Script Used To Plot All Stored Unix Epoch Date Files.

Figure 11 shows the necessary program code used to create on-demand real-time plots for the Heat Map Display function. Using the real-time qualities of the new Infiniband monitoring system, on demand plotting began with an Application Programmer present to start and test an application. While the Infiniband monitoring system background services continued to gather data, the Display Heat Maps portion of the monitoring system was run without a filename selected. The heatmap_plot application then chose the latest real-time data to perform the plotting functions. This can be seen by the use of a regular expression to check the contents of a ‘passed parameter’ to the program code. In the code, upon seeing that the contents of the parameter are empty, a listing request for the directory was parsed to determine files that were currently being used for Aggregator and Store data dumps. Once an open file was identified, the Epoch time parameters was passed on to each Heat Map plotting program.

```
if [ "$#" -eq 0 ]
then
	epoch=`ssh root@hodor 'ls -l
/home/undmaguilar/ldms_stored_data/node/' | grep 2015 | awk
'/sysclassib/ {print $9}' | sed 's/[A-Za-z.]*//g'`
	echo $epoch
else
	epoch=$1
fi
```

Figure 11. Epoch Time Determination Portion of Heat Map Display Function.

The Appendix contains complete programs that were used to create the Infiniband monitoring system. The monitoring system services will work on Red Hat and CentOS Operating Systems containing System V init tools. In order to prepare a system to receive metrics from each compute node and then store the metrics, the Metric Gathering service should be installed in a compute node in the /etc/init.d directory. To activate the Metric Gathering service, at the command prompt, type /sbin/chkconfig add /etc/init.d/gather_metrics . This command should be executed to cause the service to start when the compute node is operational. The Aggregator and Store service, aggr_and_store, should be stored in the Beowulf HPCC system head node.at /etc/init.d . Again, the service is activated at the command prompt by typing /sbin/ckconfig add /etc/init.d/aggr_store.

CHAPTER VI

CASE STUDIES

Using the new Infiniband monitoring system, research was performed on the UND Hodor Beowulf HPCC system to provide information that could be used to improve computational performance. In addition, research on the Hodor Beowulf cluster provided an example HPCC system to assess whether the Infiniband monitoring system Heat Map displays could provide detailed information that could useful in other HPCC systems.

Prevalence of Spin Waits and Performance Degradation on Hodor

The Hodor computing cluster is the principal Beowulf HPCC system for performing research at the University of North Dakota. To answer questions of how often spin-waits could be expected to happen and how much data was backed up by spin-waits on Hodor, the data retrieved from Hodor reflected runs of random applications created by HPCC system programmers. To plot the metrics for later analysis, a complete set of programmed batch runs were performed on the Display Heat Maps portion of the Infiniband monitoring system to plot the metric data samples taken over several months on a separate monitoring server.

After analysis of the plotted data, it was found that during the period of time that the samples were taken, Infiniband spin-waits would occur quite frequently on the Hodor HPCC system. In many cases, there were identifiable reasons for why the spin-waits occurred. After sampling metrics on Infiniband network links for several months, one of the most prevalent causes for Infiniband spin-waits appeared to be a high amount of run-time NFS data storage accesses. Each compute node in Hodor shares a /home directory through an NFS file share. The NFS storage file system provided for Hodor HPCC system storage is linked to a flat network design through a switch with one Infiniband connection.

In was noticed from the data gathered from Hodor that in several cases, applications were written to exchange data, write data, and retrieve data during batch job runs from the /home directory and thus through the NFS storage share. As an example, figure 12 shows an application run that is backing up a large quantity of data in spin-waits. Figure 12 contains Heat Maps to display the quantity of bytes per second of spin-waits, the quantity per second of bytes written and read for NFS transmissions, and the quantity of bytes transmitted over the Infiniband HCA port as RDMA communications. Since each compute node in Hodor is connected to exactly one Infiniband network connection through an HCA, the Y-axis contains the range of compute node Infiniband network links associated with a quantity of network traffic. On the right side of the Spin-Wait Heat Map, the color bar ranges from a dark red at 45,000 bytes per second to dark blue at 0 bytes per second. At time mark 1:00, the Heat Map is mostly dark red from compute nodes 5 to 30. In the example, at time mark 1:00, an application is continuing to read information from the NFS file share represented in light blue to dark red to be between 4 and 9 bytes per second, and also has begun writing information back into the NFS file share, represented in light blue, yellow, and dark red to be between 4 and 24 bytes per second. The relatively high cost of NFS data transmissions is shown by the amount of bytes necessary to communicate NFS over Infiniband RDMA, at 450,000 to

600,000 bytes per second, show on the bar at the right of the plot as the yellow range. Overall, around 10% of the data being transferred to the NFS share is being delayed in spin-waits.

In another example, Figure 13 shows spin-waits from another application run being generated at the time mark 3:30 from NFS data writes. A high quantity of spin-waits is being generated from compute nodes 0 to 32 (yellow representing 3,500 bytes per second) with a higher concentration from nodes 15 to 25 (orange representing 4,000 bytes per second). Some of the IPoIB data traffic is being generated on the compute node Infiniband network links in exchanges of data between compute nodes, the light blue hue represents close to 750000 bytes per second of RDMA traffic. However, it can be seen that at times when data is exchanged with the NFS share (yellow representing 13 bytes per second), there is a noticeable increase in spin-waits on the Infiniband network links. In both examples, it can be seen that despite relatively low quantities of data being transferred to the NFS share, Infiniband network link congestion has caused spin-waits and degradation of application computation performance. It must be remembered, while examining the plots in Figure 12 and Figure 13, that as NFS file calls retrieve and store data from the relatively slow secondary storage, the FIFO nature of Infiniband data transfers require that each network link remain locked until a full block of memory data is completely exchanged. In addition, from the Heat Maps in both Figure 12 and Figure 13, it can be seen that NFS data transmissions require a high cost in RDMA bytes exchanged over Infiniband networks.

Further, while applications were accessing the NFS storage during a batch run, a compute node CPU could spend a lower percentage of its time processing application threads because of Virtual Memory locks. Finally, other process threads that were waiting to exchange data with each other were not available for Virtual Memory page swaps due to memory locking requirements written into the Mellanox mlx4 device driver.

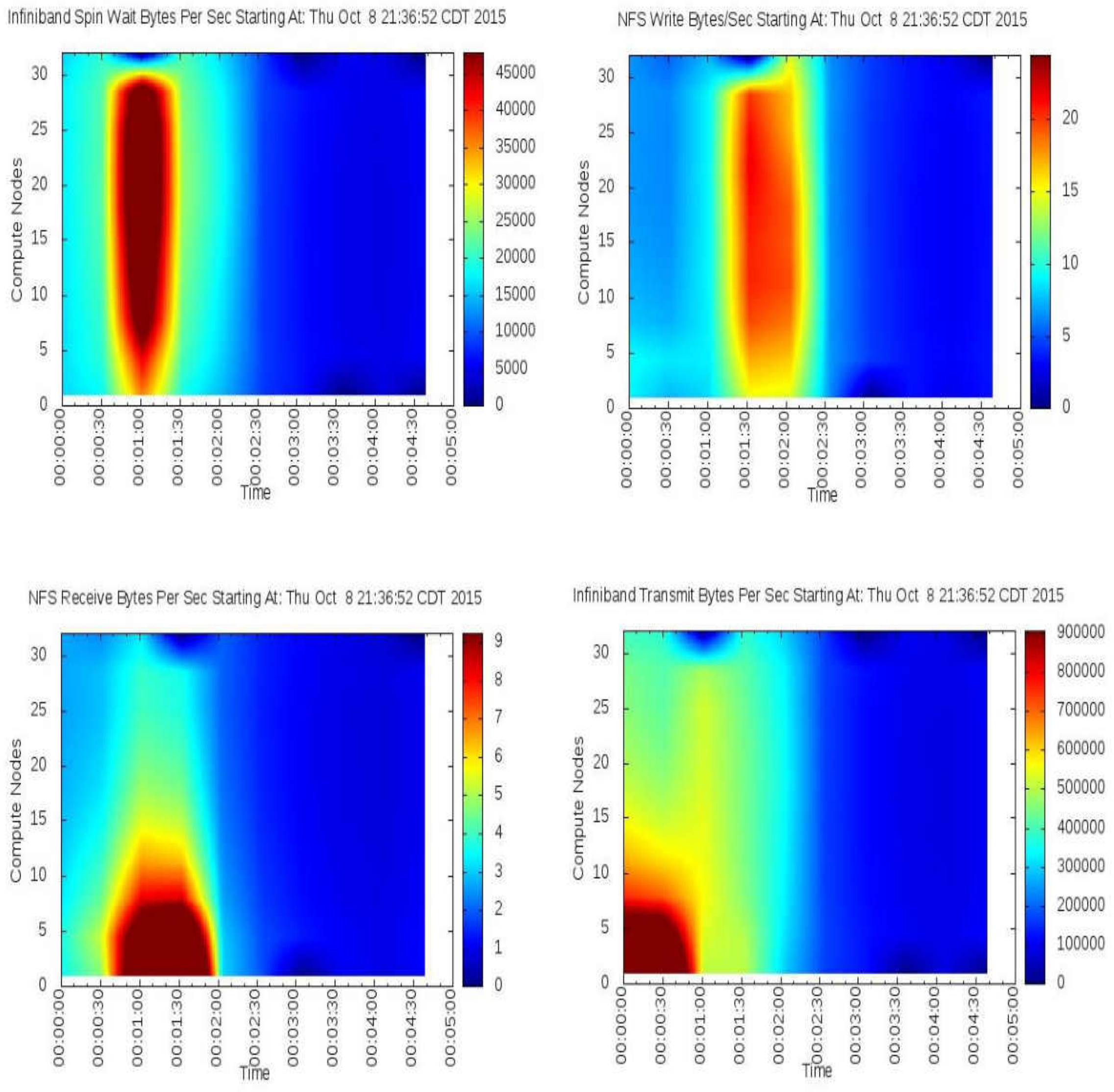

Figure 12. First Example of Spin-Waits Cause d By Accessing NFS Storage During Application Run.

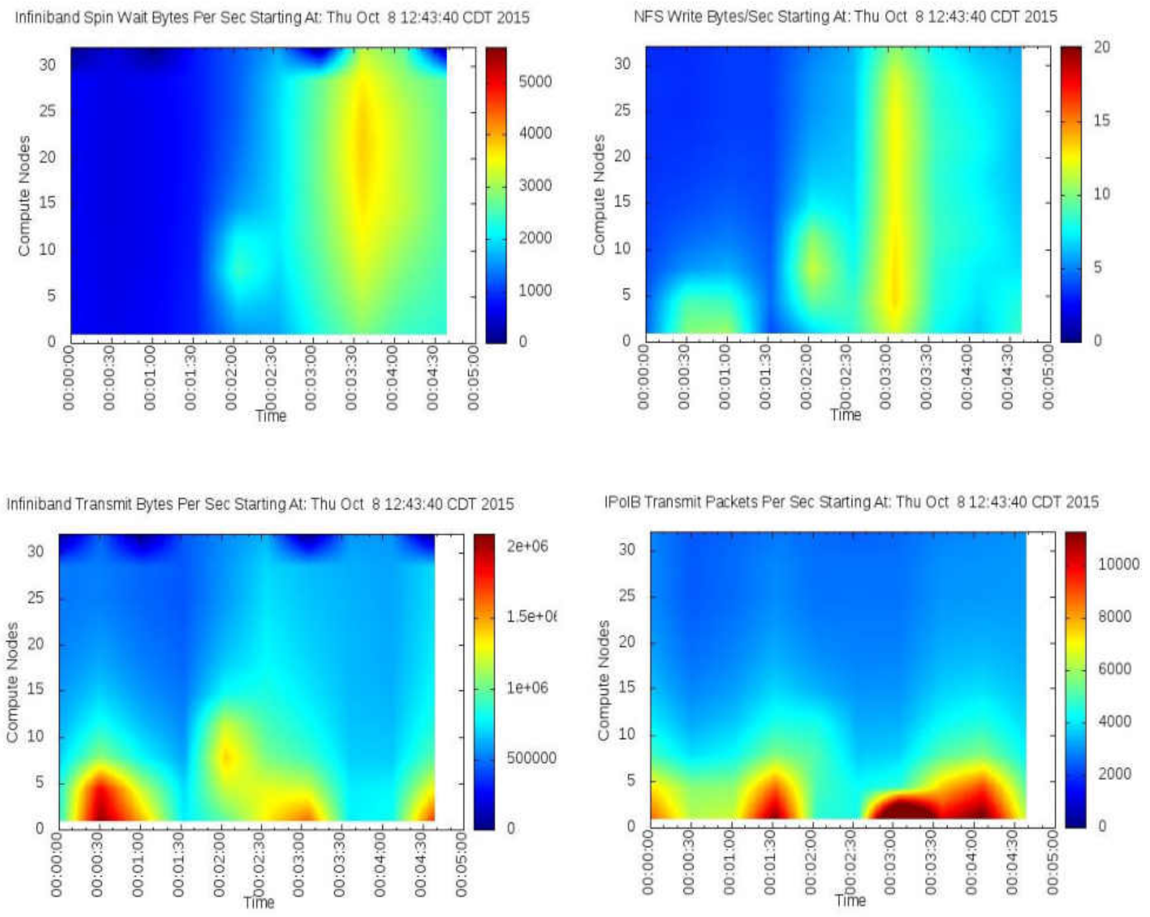

Figure 13. Second Example of Spin-Waits Caused By Accessing NFS storage During Application Run.

Message passing is a method for exchanging data from compute node to compute node and compute nodes and an HPCC system head node. Hodor has an installation of Message Passing Interface called OpenMPI that is used by many different applications to trade messages from compute node to compute node through library calls. The version of OpenMPI installed on the Hodor HPCC system performs its message passing using IPoIB. It becomes clear from Infiniband Monitoring System Heat Maps from Hodor that even a low data exchange rate for Message Passing IPoIB communications can lead to a higher percentage of data spin-waits on the Infiniband network links between the compute nodes. In Figure 14, an application that is performing OpenMPI data exchanges has been started at time mark 4:45. The color bar for the Spin-Wait Heat Map ranges from 0 (dark blue hue) to 250 bytes per second (dark red). The Heat Map for Infiniband transmissions shows 2,800 bytes per second from compute nodes 0 to 20. The Heat Map for Infiniband receive bytes shows 400 bytes per second on nodes 0 to 20. This represents the small amount of IPoIB traffic that shows 9 packets per second received and 25 packets per second transmitted on the same nodes, during the same period. At the same time, corresponding spin-waits representing around 10% of the application data being transferred have also begun. It is clear from the Heat Map plots that despite low packet rates of IPoIB on each active compute node, high RDMA traffic overhead and FIFO ordering of data exchanges has caused a relatively high percentage of data spin-waits for each compute node being used in processing the application.

Finally, assessing the data, gathered using the batch runs of the Display Heat Maps portion of the Infiniband monitoring system from Hodor, the superiority of data computations using RDMA to perform message passing and data transfers becomes clear. In Figure 15, it can be seen that the percentage of data in spin-wait each Infiniband link is relatively low. At the time mark 2:30, spin-waits of an order of magnitude much lower relative to the Infiniband data exchanges are being created. Dark red on the Heat Map displaying Infiniband spin-waits represents the maximum quantity of 900 bytes per second of data held in spin-waits. No IPoIB transmission data was recorded during this period of time on the Hodor HPCC system. However, the quantity of data being transmitted and received by RDMA is several orders of magnitude higher as can be seen

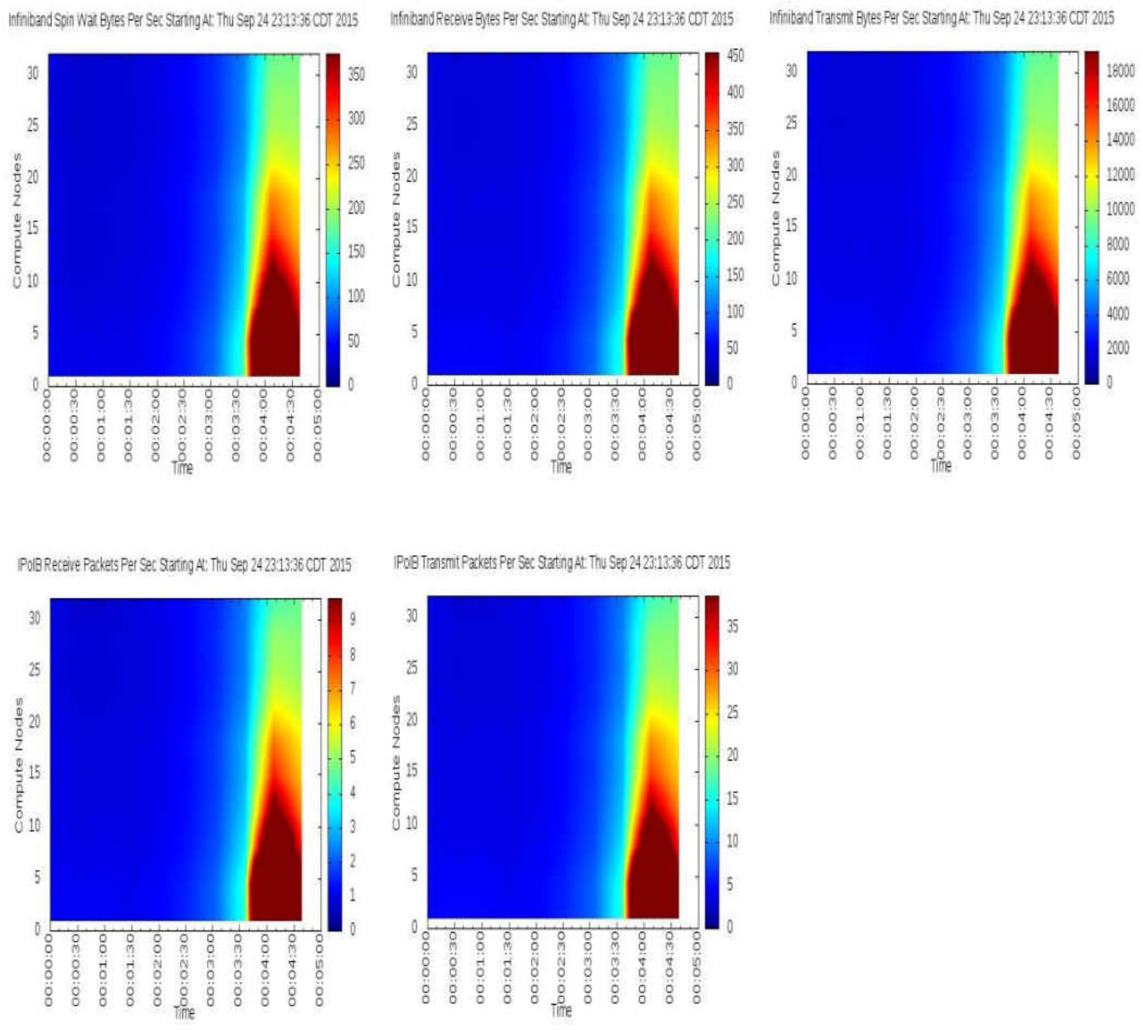

Figure 14. Spin-Waits Generated During IPoIB Data Exchanges.

by the color bars to the left of the plots. Because no IPoIB transmissions were recorded it can be quickly deduced that the data exchanges between compute nodes are strictly being performed as RDMA exchanges. The application run in this example appears to be using either the MVAPICH message passing interface libraries with RDMA data exchanges, or lower level RDMA exchanges using the Infiniband Verbs libraries. RDMA memory exchanges occur more quickly then IPoIB data exchanges and then the Infiniband network links are released for other process thread data exchanges.

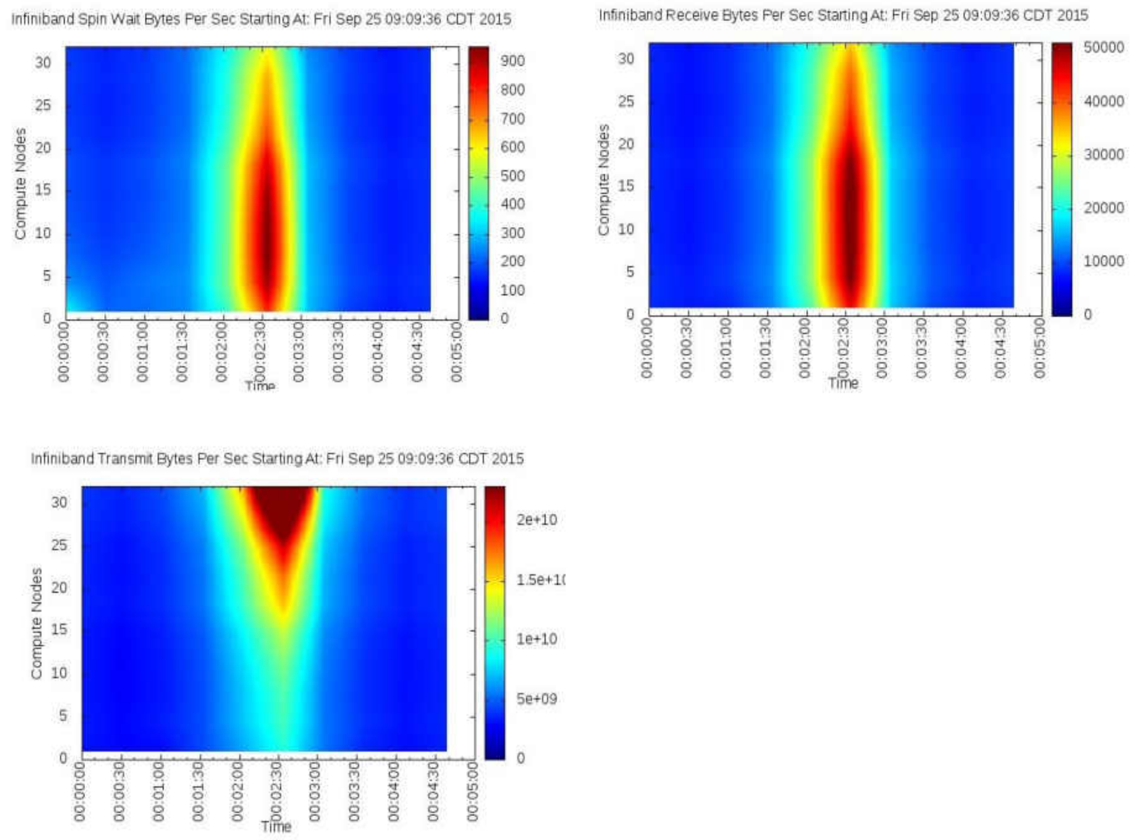

Figure 15. Example of Spin-Waits Created During RDMA Data Exchanges.

Review of a Known Application Using the Infiniband Monitoring System

While it was beneficial to gather information on random applications to determine how often congestion and spin-waits were occurring on the Hodor HPCC system, the Infiniband Monitoring System Heat Maps provide an easy to read method to analyze programs created by Application Programmers for congestion and spin-waits. In this manner Application Programmers can be made aware of ways to improve computational time for their applications.

In Figure 16, an example FFT application was started on Hodor and real-time Infiniband network performance measurements were gathered and displayed. Analysis of the application run started at time mark 1:45 indicates that there is a wide dispersion of computations spread across the HPCC system. It also is clear that RDMA is being used to perform exchanges of data between compute nodes and NFS data exchanges are not occurring while the application is being batch processed. Thus, the percentage of data in spin-waits is very low (200 bytes per second) compared to the amount of data being exchanged (90,000 bytes per second for both receive and transmit). It can be seen on the Heat Maps that most of the data congestion is occurring between node 14 and node 30, the dark red hue. To reduce congestion on the Infiniband network links and further improve the computational speed of this application would require that the process module computations being performed heavily between node 14 and node 30 be more evenly spread out. Better balancing of the process modules between compute nodes could then reduce the amount of Infiniband network congestion on the network links to these compute nodes. In this manner the percentage of data in spin-waits could be lowered and the time required to process the application could be lowered, as well.

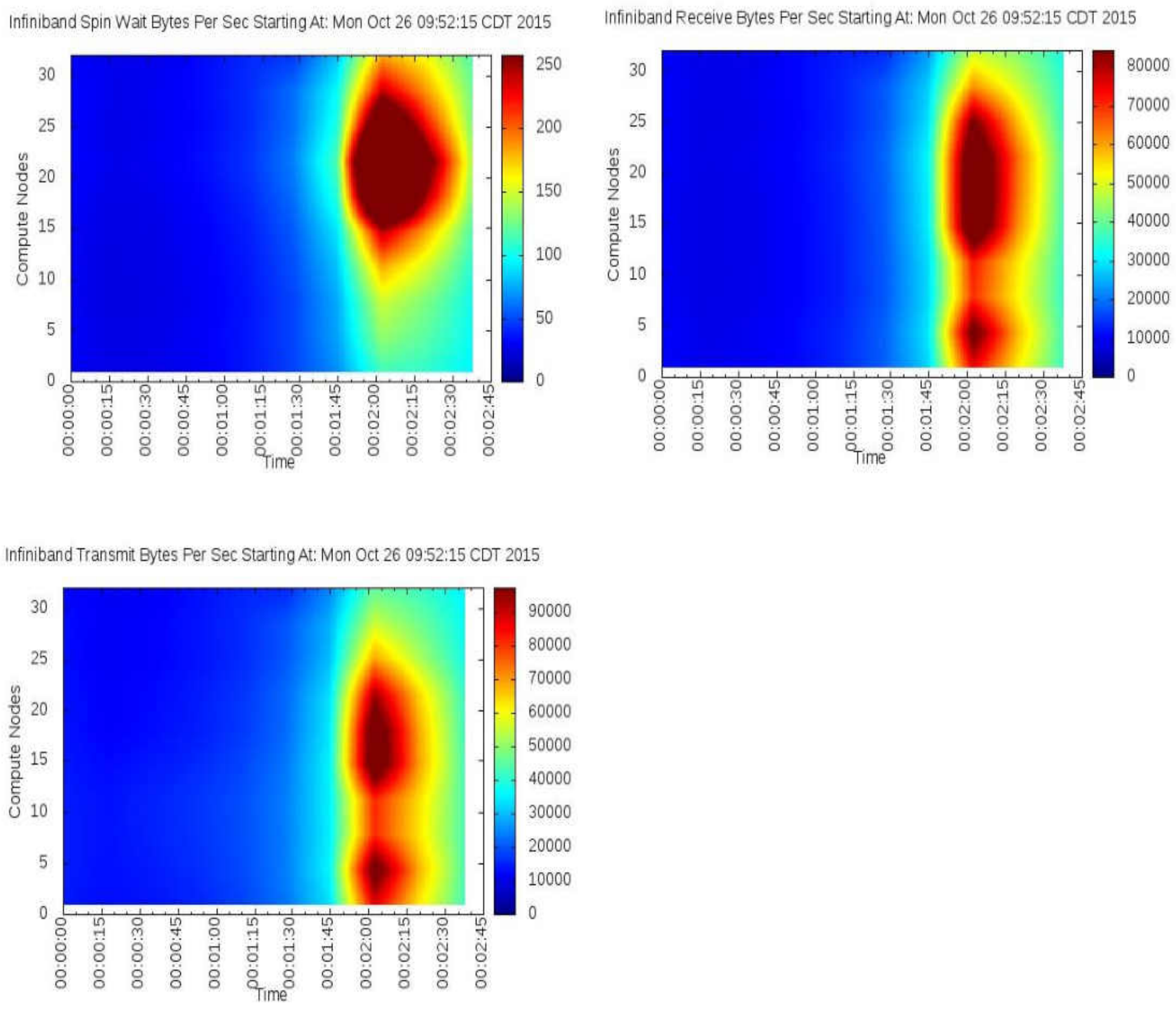

Figure 16. Example of Analysis of a Batch Run for an Application Programmer Using Hodor.

CHAPTER VII

CONCLUSION AND FUTURE WORK

This thesis chronicles research efforts into an area research that can be used to reduce fluctuations in application run-times and better utilize available parallel cores to create higher performance for a Beowulf HPCC system. After performing an analysis of the design of a Beowulf HPCC system, Infiniband network link congestion was identified as a possible source for application program run-time variations and a restriction on using available Beowulf HPCC system cores. A Beowulf HPCC system computational performance is maximized when tightly coupled parallel application process threads are freely able to exchange data and information. When there is less network congestion, the effective computational power of a Beowulf HPCC system is increased.

It is through Infiniband network links that several essential methods of data storage and internode communications for computations are conveyed in a typical Beowulf HPCC system. When an Infiniband network link becomes congested because of exchange traffic from a computation thread, the Infiniband driver responsible for the network link creates spin-waits for activity from other threads. This spin-wait behavior due to congestion on Infiniband links will affect HPCC computational performance for applications. Unfortunately, it was discovered that no available real-time monitoring system existed that was able to gather spin-wait metric data that could show how often Infiniband network link congestion was constricting data and information exchanges.

The work performed in this thesis contributed a new Infiniband monitoring system that allowed both System Administrators and Application Programmers to be able to observe real-time Infiniband network traffic as program batch jobs were run. A goal was to be able to observe how often Infiniband network congestion occurred on a Beowulf HPCC system and the specific times and the effects of the congestion on RDMA communications, IPoIB communications, and NFS storage transfers.

The new monitoring system was installed into the University of North Dakota Hodor Beowulf HPCC system. Samples of random running applications were taken for several months from Hodor. From these samples, it was found that Infiniband congestion would occur frequently on the Hodor HPCC system. From the samples taken, it could be seen that congestion was caused not only by high quantities of data transmissions but also from high quantities of slower data transmissions. The information gathered from Hodor demonstrated that slower IPoIB data exchanges would generate higher quantities of spin-waits and were responsible for much of the slow computational processing and much of the variations in run-time computational performance on the Hodor HPCC system. In fact it was found that IPoIB congestion would occur as a higher percentage of the transmission data with both frequent OpenMPI data exchanges between compute nodes and with frequent NFS storage data exchanges.

The fact that IPoIB data exchange rates are lower than data exchange rates for RDMA data transfers exacerbated the congestion problems with the Infiniband network links. When IPoIB transmissions were occurring other data traffic was required to wait its turn due to the fact that all Infiniband data traffic occurs in FIFO order. These Infiniband network link data transmissions queued and backed up appeared as Infiniband spin-waits. Curiously, the quantity of data backed up in spin-waits during IPoIB data exchanges stayed around 10% of the data being transferred.

While it could be concluded from the data gathered from Hodor that RDMA and MVAPICH definitively proved to be a better way to perform data exchanges, there remained many instances where small quantities of spin-waits still occurred. The plots showed that time periods when high concentration of computations occurred on a few nodes, network congestion to the compute nodes would often lead to Infiniband spin-waits. From the samples, it can be concluded that a wider dispersal of the computational load, in these time periods, across many compute nodes would lead to faster application run-times, as well.

Although the information gathered from sampling Hodor network traffic proved valuable to understanding how congestion occurred during an application program run, more research would be valuable. Future use of the Hodor Infiniband Monitoring System can be to provide Application Programmers with information on their running applications so that modifications can be made to the applications to improve run-times. These changes would be made to attempt to reduce the amount of IPoIB traffic and to balance the computational load across the compute nodes. Actual calculation of the application run-times could then be used to verify more consistent and faster run-time performance for each application program. These calculations would be made with the aid of the HPCC Application Programmers. Another area of research involving Hodor could be done to determine why the amount of data in Infiniband spin-wait queues represents 10% of the transmission levels of the data.

Much as the research into the Hodor HPCC system would benefit both Beowulf HPCC applications and system design, the ‘flat design’ of Hodor is a small subset of the available topologies that can be employed in construction of an HPCC system. Other research should be done with Beowulf HPCC systems using other topologies such as Fat-Tree networks, separate data storage networks that allow connections between compute nodes to be dedicated to data exchanges, and parallel file systems for faster data retrieval. The data gathered could then be used to learn what Beowulf HPCC cluster topologies are optimum and the actual computational time improvement that could be made to the application run-times.

Finally, with new HPCC system topologies, sample measurements could then be taken to insure that better parallel programming practices are employed and that a consistent wide distribution of processes are assured within the Beowulf HPCC systems.

APPENDIX

gather_metrics

```
#!/bin/bash
##############################################################################
#                                   Gather Metrics Service
#
##############################################################################
#
#
#  /etc/rc.d/init.d/gather_metrics
#
#
#
#  For Redhat and CentOS with System V init
#
#
#
# Metric collector startup/shutdown script for compute nodes.
#
#
#
# chkconfig: 2345 83 25
#
# description: Gather Metrics uses underlying LDMS services to gather targeted
# Infiniband
# and NFS  metrics.
#
# processname: gather_metric
#
#############################################################################

processname='ldmsd'
lockfile=/var/lock/subsys/$processname          # create a lockfile for system check
nodename=`hostname -s`                          # current short nodename

component_id=`echo $nodename | sed 's/[^0-9]//g' | sed 's/^0*//g'`
                                                # strip out letters for node component id
if  [ -f $lockfile ]; then              # verify lockfile isn't in place
                echo "LDMS collector already running?  process: "`pgrep ldmsd`
                exit
        fi

        if [ -f /opt/ovis/ldms.usr/sbin/ldmsd ]; then  # check for ldms
```

```
touch $lockfile  #install a file lock so we know that the LDMS daemon should be running

 # Log messages are stored in /var/log/messages #

    ldmsd -x sock:60000 -S /var/run/ldmsd/$nodename/infiniband -l /var/log/messages \
            > /dev/null 2>&1
    ldmsd -x sock:60001 -S /var/run/ldmsd/$nodename/net -l /var/log/messages \
            > /dev/null 2>&1
    ldmsd -x sock:60002 -S /var/run/ldmsd/$nodename/nfs -l /var/log/messages \
            > /dev/null 2>&1

     echo "load name=sysclassib" | ldmsctl -S /var/run/ldmsd/$nodename/infiniband
     echo "config name=sysclassib component_id="$component_id" \
             set=$nodename/sysclassib" | ldmsctl -S /var/run/ldmsd/$nodename/infiniband
     echo "start name=sysclassib interval=1000000 offset=0" | ldmsctl –S \
             /var/run/ldmsd/$nodename/infiniband
     echo "LDMS Infiniband collection is started with port 60000" >> /var/log/messages

     echo "load name=procnetdev" | ldmsctl -S /var/run/ldmsd/$nodename/net
     echo "config name=procnetdev component_id="$component_id" \
         set=$nodename/procnetdev ifaces=ib0" | ldmsctl -S /var/run/ldmsd/$nodename/net
     echo "start name=procnetdev interval=1000000 offset=0" | ldmsctl –S  \
        /var/run/ldmsd/$nodename/net
     echo "LDMS Network collection is started with port 60001" >> /var/log/messages

     echo "load name=procnfs" | ldmsctl -S /var/run/ldmsd/$nodename/nfs
     echo "config name=procnfs component_id="$component_id" \
          set=$nodename/procnfs" | ldmsctl -S /var/run/ldmsd/$nodename/nfs
     echo "start name=procnfs interval=1000000 offset=0" | ldmsctl –S \
          /var/run/ldmsd/$nodename/nfs
     echo "LDMS NFS collection is started with port 60002" >> /var/log/messages

     echo "The LDMS services are running as pid: "`pgrep ldmsd`

  else
       echo `date` "Error: Service gather_metrics is reporting that Lightweight Distributed \
              Metric System is not installed." >> /var/log/messages
  fi

}
################ End Start
###########################################################################
```

```
###############################################################################
# Stop all of the LDMS daemons currently running
#
##############################################################################

Stop () {

        echo "Stopping LDMS......."
        pkill ldmsd
        rm -f $lockfile  # Remove our lockfile

}
################# End Stop
###############################################################################

case "$1" in

'start')

        Start # start up the LDMS daemons for our chosen metrics sysclassib, \
              # procnetdev, and procnfs
        ;;

'stop')

        Stop # stop all of the ldms daemons, at once
        ;;

'status')
        if pgrep ldmsd > /dev/null 2>&1
        then
                echo "LDMS is running as pid: "`pgrep ldmsd`
        else
                echo "LDMS is not running."
        fi
        ;;

'clean') # In case of a lock file and socket file currently in place, that shouldn't be

if pgrep ldmsd > /dev/null 2>&1
        then
                echo "LDMS is runnning, so the file locks must stay in"
                exit 1;
        else
                echo "Removing the lock file and socket file"
```

```
		rm -f $lockfile			# drop the lock file
		rm -f /var/run/ldmsd/$nodename/infiniband  # drop the socket file
		exit 1;
	fi
	;;

'restart')

	Stop    # stop allu currently running LDMS daemons

	sleep 1

	Start  # restart our metric checks

	;;

*)
	echo "Usage: $0 { start | stop | status | restart | clean }"
	exit 1
	;;

esac
exit 0
```

aggr_store

```
#!/bin/bash
###########################################################################
#                                        Aggregate and Store Service
#
###########################################################################
#
#
#  /etc/rc.d/init.d/aggr_store
#
#
#
#  For Redhat and CentOS  with System V init
#
#
#
# Aggregator startup/shutdown script for head node.
#
#
#
# chkconfig: 2345 83 25
#
# description: Aggregator and Store assembles metrics from each compute node and \
# stores  the
# data for later processing and display.
#
# processname: aggr_store
#
###########################################################################

processname='ldms'
lockfile=/var/lock/subsys/$processname
nodename='hodor'
node_quantity=32 #There are 32 nodes in service right now in the Hodor cluster
metrics=71 # sysclassib has 22 lines of values + 16 (procnetdev) + 23 (procnfs)
        # metric quantity comes from a compute node using #>ldms_ls -h <node_name> \
        # -p <port> -v
total_metrics=node_quantity*metrics #Metrics to dump by Aggregator on a pass
storage_path=/home/undmaguilar/ldms_stored_data #LDMS stored data path


# Aggregate metrics we want across HPC cluster
Aggregate() {
        node=$1

          echo 'add host='$node' type=active interval=1000000 offset=100000 xprt=sock \
                   port=60000 sets='$node'/sysclassib' | ldmsctl -S /var/run/ldmsd/infiniband
          echo 'add host='$node' type=active interval=1000000 offset=100000 xprt=sock \
                   port=60001 sets='$node'/procnetdev' | ldmsctl -S /var/run/ldmsd/infiniband
          echo 'add host='$node' type=active interval=1000000 offset=100000 xprt=sock \
                   port=60002 sets='$node'/procnfs' | ldmsctl -S /var/run/ldmsd/infiniband
}


############### Load our metrics from across the HPC cluster
######################
# Storage will be done with id_pos printing out the component_id from the node metrics
as  #
# an identifying label.
#
##############################################################################

Start() {

        echo "Starting Aggregate and Store Services." >> /var/log/messages

        if [ -f $lockfile ]; then
           echo "LDMS collector already running?  process: "`pgrep ldmsd`
           exit
        fi

        if [ -f /opt/ovis/ldms.usr/sbin/ldmsd ]; then
           mkdir -p /var/run/ldmsd/ #make sure that run directory is installed
           # Start ldms daemon that will aggregate from samplers
           # -D says flush data to output file whenever total metric values outstanding
           ldmsd -x sock:60010 -S /var/run/ldmsd/infiniband -D $metrics –l \
                  /var/log/messages > /dev/null 2>&1

           # Add host and metric set combinations to aggregate from
           # Collection interval is 1 second and offset is 0.1 second

################ Broken into 2 parts for node ranges #####################

           for i in {1..9}
           do
              current_node=`echo "node00"$i`
              Aggregate $current_node
           done
          for i in {10..32}
          do
             current_node=`echo "node0"$i`
             Aggregate $current_node
          done

############################## Store data ###########################
# Load and configure csv store plugin
echo "load name=store_derived_csv" | ldmsctl -S /var/run/ldmsd/infiniband
echo "config name=store_derived_csv alt_header=1 id_pos=1 rolltype=3 \
          rollover=1000000 derivedconf=/root/ldms_store.config path="$storage_path | \
          ldmsctl -S /var/run/ldmsd/infiniband
echo "store name=store_derived_csv comp_type=node set=sysclassib \
          container=sysclassib" | ldmsctl -S /var/run/ldmsd/infiniband
echo "store name=store_derived_csv comp_type=node set=procnetdev \
          container=procnetdev" | ldmsctl -S /var/run/ldmsd/infiniband
echo "store name=store_derived_csv comp_type=node set=procnfs container=procnfs" | \
          ldmsctl -S /var/run/ldmsd/infiniband

          touch $lockfile  # install a file lock so we know that the
                           # LDMS daemon should be running
          echo "The LDMS service is running as pid: "`pgrep ldmsd`

       else
       echo `date` "Error:  Service start_ldms is reporting that \
            Lightweight Distributed Metric System is not installed." >> /var/log/messages
   fi

}

############################## End Start ####################################

########################## Kill Aggregate and Store #######################
Stop() {

   echo "Stopping LDMS......."
   pkill ldmsd
   rm -f $lockfile  # Remove our lockfile

}

############################################################################

case "$1" in
```

```
'start')

   Start # Start the LDMS daemon and begin aggregation of our values

   ;;

'stop')

   Stop # Kill LDMS Aggregate daemon
   ;;

'status')
   if pgrep ldmsd > /dev/null 2>&1
   then
      echo "LDMS is running as pid: "`pgrep ldmsd`
   else
      echo "LDMS is not running."
   fi
   ;;

########### Clean up in case of an unclean shutdown of LDMS Aggregate ##########

'clean')
   if pgrep ldmsd > /dev/null 2>&1
   then
      echo "LDMS is runnning, so the file locks must stay in"
      exit 1;
   else
      echo "Removing the lock file"
      rm -f $lockfile
      rm -f /var/run/ldmsd/infiniband # remove the socket file
      exit 1;
   fi
   ;;

'restart')

   Stop
   sleep 2
   Start
   ;;

*)
   echo "Usage: $0 { start | stop | status | restart | clean }"
   exit 1
```

```
    ;;

esac
exit 0
```

ldms_store_config

```
ib.port_xmit_data#mlx4_0.1,1,1
ib.port_rcv_data#mlx4_0.1,1,1
ib.port_xmit_packets#mlx4_0.1,1,1
ib.port_rcv_packets#mlx4_0.1,1,1
ib.port_xmit_wait#mlx4_0.1,1,1

numcalls,1,1
retransmitts,1,1
getattr,1,1
setattr,1,1
lookup,1,1
access,1,1
readlink,1,1
read,1,1
write,1,1
readdir,1,1
readdirplus,1,1

rx_bytes#ib0,1,1
rx_packets#ib0,1,1
tx_bytes#ib0,1,1
tx_packets#ib0,1,1
```

heatmap_plot

```
#!/bin/bash
################################################################################
####
# Heat Map Plotter Program for Creating Plots for System Admins and Application
#
# Programmers.
#
################################################################################
####

################ Based on user input is this a real-time plot or a batched plot?
#
#  If it is a batched plot, there with be an Epoch Time entered.
#

if [ "$#" -eq 0 ]
then
        epoch=`ssh root@hodor 'ls -l /home/undmaguilar/ldms_stored_data/node/' | \
                        grep 2015 | awk '/sysclassib/ {print $9}' | sed 's/[A-Za-z.]*//g'`
        echo $epoch
else
        epoch=$1
fi

############## Create User readable time and a folder for the plots #########

output_period=`date -d @$epoch | sed 's/ /_/g'`
period=`date -d @$epoch`
echo $period
mkdir -p ~/hodor_performance/$output_period

# Lets gather the plot data from source and temporarily store it in the local /tmp directory
#

name="/tmp/sysclassib."$epoch

# Up to date data and appended to any current data for real-time use

rsync -avz root@hodor:/home/undmaguilar/ldms_stored_data/node/sysclassib.$epoch
/tmp/sysclassib.$epoch
rsync -avz root@hodor:/home/undmaguilar/ldms_stored_data/node/procnfs.$epoch
/tmp/procnfs.$epoch
```

```
rsync -avz root@hodor:/home/undmaguilar/ldms_stored_data/node/procnetdev.$epoch
/tmp/procnetdev.$epoch

##############################################################################
#
#                                    Plotting Begins!
#
##############################################################################
#

################### Hodor IB Run-Time Statistics
##############################

####### IB Stats Filename #########

name="/tmp/sysclassib."$epoch

##########  IB Transmit #############

type="Infiniband Transmit Bytes Per Sec Starting At: "$period
outfile='~/hodor_performance/'$output_period'/infiniband_transmit_bytes.jpg'
#########Auto Range the Color Bar   ################
mean=`awk '{ar[NR]=$6; sum+=$6;} END{m=sum/NR; printf "%.4f", m}' $name`
mabdev=`awk '{ar[NR]=$6; sum+=$6;} END{m=sum/NR; for (i in ar) {mcol+= \
              sqrt((ar[i]-m)^2)}; mabdev=(mcol/NR); printf "%.4f",mabdev}' $name`
mabdev=`echo "( $mabdev ) +$mean" | bc`

if [ $( echo "$mabdev < 1" | bc ) -eq 1 ]
then
         mabdev=1
fi
####### Call ib_transmit Gnuplot program written for this plot #################
gnuplot -e "filename='$name'" -e "date='$period'" -e "type='$type'" -e "outfile='$outfile'"
\
         -e "sdev='$mabdev'" ib_transmit

##########  IB Receive ##############

type="Infiniband Receive Bytes Per Sec Starting At: "$period
outfile='~/hodor_performance/'$output_period'/infiniband_receive_bytes.jpg'
mean=`awk '{ar[NR]=$7; sum+=$7;} END{m=sum/NR; printf "%.4f", m}' $name`
mabdev=`awk '{ar[NR]=$7; sum+=$7;} END{m=sum/NR; for (i in ar) \
            {mcol+= sqrt((ar[i]-m)^2)}; mabdev=(mcol/NR); printf "%.4f",mabdev}'
$name`  mabdev=`echo "( $mabdev ) +$mean" | bc`
```

```
if [ $( echo "$mabdev < 1" | bc ) -eq 1 ]
then
        mabdev=1
fi

gnuplot -e "filename='$name'" -e "date='$period'" -e "type='$type'" -e "outfile='$outfile'"
\
          -e "sdev='$mabdev'" ib_receive

###########  IB Spin Waits ############

type="Infiniband Spin Wait Bytes Per Sec Starting At: "$period
outfile='~/hodor_performance/'$output_period'/infiniband_spin_wait_bytes.jpg'
mean=`awk '{ar[NR]=$10; sum+=$10;} END{m=sum/NR; printf "%.4f", m}' $name`
mabdev=`awk '{ar[NR]=$10; sum+=$10;} END{m=sum/NR; for (i in ar) \
            {mcol+= sqrt((ar[i]-m)^2)}; mabdev=(mcol/NR); printf "%.4f",mabdev}'
$name`
mabdev=`echo "( $mabdev ) +$mean" | bc`

if [ $( echo "$mabdev < 1" | bc ) -eq 1 ]
then
        mabdev=1
fi

gnuplot -e "filename='$name'" -e "date='$period'" -e "type='$type'" -e "outfile='$outfile'"
-e "sdev='$mabdev'" ib_port_xmit_wait

############################### NFS Statistics
##################################

##########  NFS Stats Filename ########

name="/tmp/procnfs."$epoch

######## NFS Retransmits ##############

mean=`awk '{ar[NR]=$7; sum+=$7;} END{m=sum/NR; printf "%.4f", m}' $name`
mabdev=`awk '{ar[NR]=$7; sum+=$7;} END{m=sum/NR; for (i in ar) \
            {mcol+= sqrt((ar[i]-m)^2)}; mabdev=(mcol/NR); printf "%.4f",mabdev}'
$name` mabdev=`echo "( $mabdev ) +$mean" | bc`

if [ $( echo "$mabdev < 1" | bc ) -eq 1 ]
then
        mabdev=1
fi
```

```
type="NFS Retransmit Bytes Per Sec Starting At: "$period
outfile='~/hodor_performance/'$output_period'/nfs_retransmit.jpg'
gnuplot -e "filename='$name'" -e "date='$period'" -e "type='$type'" -e \
              "outfile='$outfile'" -e "sdev='$mabdev'" nfs_retx

########### NFS Bytes Received #######

mean=`awk '{ar[NR]=$13; sum+=$13;} END{m=sum/NR; printf "%.4f", m}' $name`
mabdev=`awk '{ar[NR]=$13; sum+=$13;} END{m=sum/NR; \
            for (i in ar) {mcol+= sqrt((ar[i]-m)^2)}; mabdev=(mcol/NR); \
            printf "%.4f",mabdev}' $name`
mabdev=`echo "( $mabdev ) +$mean" | bc`

if [ $( echo "$mabdev < 1" | bc ) -eq 1 ]
then
        mabdev=1
fi

type="NFS Receive Bytes Per Sec Starting At: "$period
outfile='~/hodor_performance/'$output_period'/nfs_receive.jpg'
gnuplot -e "filename='$name'" -e "date='$period'" -e "type='$type'" -e \
           "outfile='$outfile'" -e "sdev='$mabdev'" nfs_rec

########### NFS Bytes Written #########

mean=`awk '{ar[NR]=$14; sum+=$14;} END{m=sum/NR; printf "%.4f", m}' $name`
mabdev=`awk '{ar[NR]=$14; sum+=$14;} END{m=sum/NR; \
      for (i in ar) {mcol+= sqrt((ar[i]-m)^2)}; \
      mabdev=(mcol/NR); printf "%.4f",mabdev}' $name`
mabdev=`echo "( $mabdev ) +$mean" | bc`

if [ $( echo "$mabdev < 1" | bc ) -eq 1 ]
then
        mabdev=1
fi

type="NFS Write Bytes/Sec Starting At: "$period
outfile='~/hodor_performance/'$output_period'/nfs_write.jpg'
gnuplot -e "filename='$name'" -e "date='$period'" -e "type='$type'" \
           -e "outfile='$outfile'" -e "sdev='$mabdev'" nfs_wri

########################### IPoIB Statistics ###############################

###### IPoIB Stats Filename ##########
```

```
name="/tmp/procnetdev."$epoch

####### IPoIB Packets received #######

mean=`awk '{ar[NR]=$7; sum+=$7;} END{m=sum/NR; printf "%.4f", m}' $name`
mabdev=`awk '{ar[NR]=$7; sum+=$7;} END{m=sum/NR; for (i in ar) \
            {mcol+= sqrt((ar[i]-m)^2)}; mabdev=(mcol/NR); printf "%.4f",mabdev}'
$name`
mabdev=`echo "( $mabdev ) +$mean" | bc`

if [ $( echo "$mabdev < 1" | bc ) -eq 1 ]
then
        mabdev=1
fi

type="IPoIB Receive Packets Per Sec Starting At: "$period
outfile='~/hodor_performance/'$output_period'/rec_ipoib.jpg'
gnuplot -e "filename='$name'" -e "date='$period'" -e "type='$type'" \
           -e "outfile='$outfile'" -e "sdev='$mabdev'" rc_ipoib

####### IPoIB Packets transmitted ####

mean=`awk '{ar[NR]=$9; sum+=$9;} END{m=sum/NR; printf "%.4f", m}' $name`
mabdev=`awk '{ar[NR]=$9; sum+=$9;} END{m=sum/NR; for (i in ar) \
           {mcol+= sqrt((ar[i]-m)^2)}; mabdev=(mcol/NR); printf "%.4f",mabdev}'
$name`
mabdev=`echo "( $mabdev ) +$mean" | bc`

if [ $( echo "$mabdev < 1" | bc ) -eq 1 ]
then
        mabdev=1
fi

type="IPoIB Transmit Packets Per Sec Starting At: "$period
outfile='~/hodor_performance/'$output_period'/trx_ipoib.jpg'
gnuplot -e "filename='$name'" -e "date='$period'" -e "type='$type'" \
          -e "outfile='$outfile'" -e "sdev='$mabdev'" tx_ipoib
```

ib_port_xmit_wait

```
#!/usr/bin/gnuplot

set terminal jpeg
set output outfile
set view map
set title type
set xlabel "Time" offset 0,-1
set ylabel "Compute Nodes"
set xtics rotate offset 0,-.4
set dgrid3d
set yrange[0:32]
set cbrange[0:sdev]
set pm3d interpolate 20,20
set xdata time
set timefmt "%s"
set format x "%H:%M:%S"
set dgrid3d
set palette defined (0 0 0 0.5, 1 0 0 1, 2 0 0.5 1, 3 0 1 1, 4 0.5 1 0.5, 5 1 1 0, 6 1 0.5 0, 7 1
0 0, 8 0.5 0 0)

splot filename using ($0/3600):5:10 with pm3d
```

ib_receive

```
#!/usr/bin/gnuplot

set terminal jpeg
set output outfile
set view map
set title type
set xlabel "Time" offset 0,-1
set ylabel "Compute Nodes"
set xtics rotate offset 0,-.4
set dgrid3d
set yrange[0:32]
set cbrange[0:sdev]
set pm3d interpolate 20,20
set xdata time
set timefmt "%s"
set format x "%H:%M:%S"
set dgrid3d
set palette defined (0 0 0 0.5, 1 0 0 1, 2 0 0.5 1, 3 0 1 1, 4 0.5 1 0.5, 5 1 1 0, 6 1 0.5 0, 7 1
0 0, 8 0.5 0 0)

splot filename using ($0/3600):5:7 with pm3d
```

ib_transmit

```
#!/usr/bin/gnuplot

set terminal jpeg
set output outfile
set view map
set title type
set xlabel "Time" offset 0,-1
set ylabel "Compute Nodes"
set xtics rotate offset 0,-.4
set dgrid3d
set yrange[0:32]
set cbrange[0:sdev]
set pm3d interpolate 20,20
set xdata time
set timefmt "%s"
set format x "%H:%M:%S"
set dgrid3d
set palette defined (0 0 0 0.5, 1 0 0 1, 2 0 0.5 1, 3 0 1 1, 4 0.5 1 0.5, 5 1 1 0, 6 1 0.5 0, 7 1
0 0, 8 0.5 0 0)

splot filename using ($0/3600):5:6 with pm3d
```

nfs_rec

```
#!/usr/bin/gnuplot

set terminal jpeg
set output outfile
set view map
set title type
set xlabel "Time" offset 0,-1
set ylabel "Compute Nodes"
set xtics rotate offset 0,-.4
set dgrid3d
set yrange[0:32]
set cbrange[0:sdev]
set pm3d interpolate 20,20
set xdata time
set timefmt "%s"
set format x "%H:%M:%S"
set dgrid3d
set palette defined (0 0 0 0.5, 1 0 0 1, 2 0 0.5 1, 3 0 1 1, 4 0.5 1 0.5, 5 1 1 0, 6 1 0.5 0, 7 1
0 0, 8 0.5 0 0)

splot filename using ($0/3600):5:13 with pm3d
```

nfs_retx

```
#!/usr/bin/gnuplot

set terminal jpeg
set output outfile
set view map
set title type
set xlabel "Time" offset 0,-1
set ylabel "Compute Nodes"
set xtics rotate offset 0,-.4
set dgrid3d
set yrange[0:32]
set cbrange[0:sdev]
set pm3d interpolate 20,20
set xdata time
set timefmt "%s"
set format x "%H:%M:%S"
set dgrid3d
set palette defined (0 0 0 0.5, 1 0 0 1, 2 0 0.5 1, 3 0 1 1, 4 0.5 1 0.5, 5 1 1 0, 6 1 0.5 0, 7 1
0 0, 8 0.5 0 0)

splot filename using ($0/3600):5:7 with pm3d
```

nfs_wri

```
#!/usr/bin/gnuplot

set terminal jpeg
set output outfile
set view map
set title type
set xlabel "Time" offset 0,-1
set ylabel "Compute Nodes"
set xtics rotate offset 0,-.4
set dgrid3d
set yrange[0:32]
set cbrange[0:sdev]
set pm3d interpolate 20,20
set xdata time
set timefmt "%s"
set format x "%H:%M:%S"
set dgrid3d
set palette defined (0 0 0 0.5, 1 0 0 1, 2 0 0.5 1, 3 0 1 1, 4 0.5 1 0.5, 5 1 1 0, 6 1 0.5 0, 7 1
0 0, 8 0.5 0 0)

    splot filename using ($0/3600):5:14 with pm3d
```

rc_ipoib

```
#!/usr/bin/gnuplot

set terminal jpeg
set output outfile
set view map
set title type
set xlabel "Time" offset 0,-1
set ylabel "Compute Nodes"
set xtics rotate offset 0,-.4
set dgrid3d
set yrange[0:32]
set cbrange[0:sdev]
set pm3d interpolate 20,20
set xdata time
set timefmt "%s"
set format x "%H:%M:%S"
set dgrid3d
set palette defined (0 0 0 0.5, 1 0 0 1, 2 0 0.5 1, 3 0 1 1, 4 0.5 1 0.5, 5 1 1 0, 6 1 0.5 0, 7 1
0 0, 8 0.5 0 0)

splot filename using ($0/3600):5:7 with pm3d
```

tx_ipoib

```
#!/usr/bin/gnuplot

set terminal jpeg
set output outfile
set view map
set title type
set xlabel "Time" offset 0,-1
set ylabel "Compute Nodes"
set xtics rotate offset 0,-.4
set dgrid3d
set yrange[0:32]
set cbrange[0:sdev]
set pm3d interpolate 20,20
set xdata time
set timefmt "%s"
set format x "%H:%M:%S"
set dgrid3d
set palette defined (0 0 0 0.5, 1 0 0 1, 2 0 0.5 1, 3 0 1 1, 4 0.5 1 0.5, 5 1 1 0, 6 1 0.5 0, 7 1
0 0, 8 0.5 0 0)

    splot filename using ($0/3600):5:9 with pm3d
```

read_heatmap_list

```
#Retrieve the Epoch time list from the Infiniband metric files. #
ssh root@hodor "ls /home/undmaguilar/ldms_stored_data/node | grep sysclassib | \
       sed 's/[a-z][.]*//g'" > heatmap_file_list
# Batch run the plotting of all of the Epoch files              #
while read epoch ; do
      echo $epoch;./heatmap_plot $epoch
done < heatmap_file_list
```

REFERENCES

Agelastos, A., Allan, B., Brandt, J., Gentile, A., Lefatzi, S., Monk, S., Ogden, J., Ogden, J., Rajan, M. & Stevenson, J. (2015). *Toward Rapid Understanding of Production HPC Applications and Systems*, IEEE International Conference on Cluster Computing, pp. 464-473.

Bell, G., [Intel], (2013, November 18). *The Secret Life of Big Data* [Video file], Retrieved from https://www.youtube.com/watch?v=CNoi-XqwJnA

Brandt, J., Devine, K., Gentile, A., & Pedretti, K. (2014). *Demonstrating Improved Application Performance Using Dynamic Monitoring and Task Mapping,* IEEE International Conference on Cluster Computing, pp. 408-415.

Brandt, J., Gentile, A., Marzouk, Y., Pebay, P. (July, 2005). *Meaningful Automated Statistical Analysis of Computational Clusters*, IEEE International Conference on Cluster Computing, pp. 1-2.

Deploying HPC Cluster with Mellanox Infiniband Interconnect Solutions, (June, 2014). Retrieved from*:* http://www.mellanox.com/related-docs/solutions/deploying-hpc-cluster-with-mellanox-infiniband-interconnect-solutions.pdf

Dixon, J. (2015). In, *Monitoring with Graphite*, O'Reilly Media, Inc.

Dongarra, J., Sterling, T. Simon, H., & Strohmeier, E. (March, 2005). High-Performance Computing: Clusters, Constellations, MPPs, and Future Directions, IEEE *Computing in Science & Engineering, 7*(2), pp. 51-59.

Dreier, R. (2015), *device.c*, (2015). Retrieved from: lxr.free-electrons.com/source/drivers/infiniband/core/device.c

Dubitzky, W., Kurowski, K., & Schott, B. (2012). In, *Large Scale Computing Techniques for Complex System Simulations* (pp. 133), J. Wiley & Sons.

Escuder-Sahuquilio, J., Gran, E., Garcia, P., Flich, J., Skeie, T., Lysne, O., Qules, F., & Duato, J. (Sept 2011). *Combining Congested-Flow and Injection Throttling in HPC Interconnection Networks*, IEEE International Conference on Parallel Processing, pp. 662-672.

Ford, W. (2015) In P. Vassilevski (Ed.), *Numerical Linear Algebra with Applications*, (pp. 157-166), Academic Press

Garabato, R., More, A., & Rosales, V. (2012). Optimizing Latency in Beowulf Clusters, *CLEI Electronic Journal 15*(3). Retrieved From: http://www.scielo.edu.uy/scielo.php?pid=S0717500020120003000004&script=sci_arttext

Gropp, W., Lusk, E., & Sterling, T. (2003). In J. Kowalik (Ed.), *Beowulf Cluster Computing with Linux* (pp.12), MIT Press

Infiniband Fact Sheet, (2011). Retrieved from: https://cw.infinibandta.org/document/dl/7260

Kocian, W. (2014). In A. Albuquerque & N. Chinnari (Eds.), *Learning Nagios 4*, Packt Publishing Ltd.

Kundu, D., & Lavlu, S. (2009). In R. Phadnis (Ed.), *Cacti 0.8 Network Monitoring*, Packt Publishing Ltd.

Liu, J., Wiu, J., & Panda, D. (June, 2004). High Performance RDMA-Based MPI Implementation over Infiniband, International Journal of Parallel Programming – Special issue 1: The 17th annual international conference on supercomputing (ICS'03) 32(3), pp.167-198.

Lynch, S. (2012). Monitoring Cache with Claspin, Retrieved From: https://www.facebook.com/notes/facebook-engineering/monitoring-cache-with-claspin/10151076705703920

Massie, M., Li, B. Nicholes, B., Vuksan, V., Alexander, R., Buchbinder, J., Costa, F., Dean, A., Josehsen, D., Phasi, P., & Pocock, D. (2013) In M. Loukides & M. Blanchette (Eds.), *Monitoring with Ganglia*, O'Reilly Media, Inc.

Mellanox Infiniband Product Overview, (2015). Retrieved from: http://www.mellanox.com/page/infiniband_cards_overview

Messina, P. (2015). Paul Messina on the Code Optimization Path to Exascale, Retrieved From: http://www.hpcwire.com/2015/09/14/alcfs-paul-messina-on-the-code-optimization-path-to-exascale

Mishra, J. (Ed.), & Mohanty (2011), *Software Engineering*, Pearson India, 2011.

Pothen, A. & C. Fan (December, 1990). Computing the Block Triangular Form of a Sparse Matrix, *ACM Transactions on Mathematical Software, 16*(4), pp. 301-324.

Remote Direct Memory Access (2015), Retrieved from: http://en.wikipedia.org/wiki/Remote_direct_memory_access

Rosen, R. (2014). In M. Lowman (Ed.), *Linux Kernel Networking*, (pp. 373-404), Apress

Sandberg, M. (2013). *The Sun Network Filesystem: Design, Implementation, and Experience*, Retrieved From: http://www.cse.buffalo.edu/faculty/tkosar/cse710_spring13/papers/nfs.pdf

Wang, L., Ranjan, R., Chen. J., & Bentallah, B. (2011). In L. Wang (Ed.), *Cloud Computing: Methodology, Systems, and Applications,* CRC Press.