Diagnosing Anomalous Network Performance with Confidence

Bradley W. Settlemyer, Stephen W. Hodson, Jeffery A. Kuehn, and Stephen W. Poole
Oak Ridge National Laboratory
One Bethel Valley Rd
PO Box 2008
Oak Ridge, TN 37831-6164
{settlemyerbw,hodsonsw,kuehn,spoole}@ornl.gov

***Abstract*—Variability in network performance is a major obstacle in effectively analyzing the throughput of modern high performance computer systems. High performance interconnection networks offer excellent best-case network latencies; however, highly parallel applications running on parallel machines typically require consistently high levels of performance to adequately leverage the massive amounts of available computing power. Performance analysts have usually quantified network performance using traditional summary statistics that assume the observational data is sampled from a normal distribution. In our examinations of network performance, we have found this method of analysis often provides too little data to understand anomalous network performance. Our tool, Confidence, instead uses an empirically derived probability distribution to characterize network performance. In this paper we describe several instances where the Confidence toolkit allowed us to understand and diagnose network performance anomalies that we could not adequately explore with the simple summary statistics provided by traditional measurement tools. In particular, we examine a multi-modal performance scenario encountered with an Infiniband interconnection network and we explore the performance repeatability on the custom Cray SeaStar2 interconnection network after a set of software and driver updates.**

I. Introduction

High speed interconnection networks, such as Infiniband and Cray's SeaStar2, offer a multitude of network transmission modes and features. Options such as zero-copy remote direct memory access (RDMA) and congestion avoidance routing make understanding an application's network behavior quite difficult. In addition to the network itself being a shared resource, other sources of performance variability also impact our ability to measure network performance in a repeatable manner. For example, computational noise [10] impacts not only the network measurements, but also the calls required to determine an accurate system timing mechanism. Additional factors, such as network throughput collapse [11] and transient effects such as temporary network and file system loads induced by other jobs running concurrently on time-shared systems combine to make high fidelity network performance measurement a difficult science.

In our field, high performance computing (HPC), scientific application developers typically rely heavily on networking and storage infrastructure to complete long-running parallel simulations and data analysis codes. The Message Passing Interface (MPI) has become the de facto standard for communicating data within the interconnection network. Although MPI offers a relatively lightweight application programming interface to users, the underlying software stack has become quite complicated in support of RDMA operations, rendezvous communication modes, and hardware offloading features. Complicated networking hardware combined with complicated software stacks can often lead to network performance that is difficult for the user to understand.

In this paper, we use Confidence [12], a novel toolkit we developed for analyzing network performance, to diagnose the causes of anomalous network behavior – and in one case reconfigure the networking stack to dramatically improve performance. In our experience, traditional summary statistics are not a good match for describing the performance of high speed interconnection networks that often have multiple performance modes for the same operation. Summary statistics assume the underlying data is sampled from a normal distribution; however, as we will demonstrate, the distribution of network performance observations tends to be one-sided with multiple modes present. Our tool, Confidence, differs from existing tools by sampling the network repeatedly to build an empirical probability distribution to describe the message passing latency.

Although Confidence was originally designed as a network performance characterization tool, we have also applied the Confidence toolkit to file system benchmarks, system call overheads, and multi-threaded data transfers. By describing benchmark performance as an empirical probability distribution rather than as a point-value, Confidence better describes the performance likely to be encountered over an entire application run.

A. Related Work

Network performance in HPC systems has been an active area of research for many years. Petrini, et al., described the performance of the Quadrics interconnection network using a suite of network tests [9], including: ping bandwidth, ping latency, offered load throughput in various traffic patterns, and an analysis of message size effects on network bandwidth. In general, the analysis focused on looking at average and best-case performance measurements on a quiesced system.

The HPC Challenge (HPCC) Benchmark Suite extended this approach to include tests on all processing an HPC system [6], both in a pair-wise fashion, and in process rings. In HPCC, the following results are reported for both short (8 Byte) and long (2,000,000 Byte) messages:

- minimum, maximum, and average ping pong network latency and bandwidth,
- average network latency and bandwidth on a random ring,
- average network latency and bandwidth on a naturally ordered ring.

The techniques described by Petrini and provided by the HPCC benchmarking suite have been used to describe the network (and overall system) performance of many emerging supercomputer installations, including the Cray XT4 and Blue Gene/P at Oak Ridge National Laboratory [1], [2], the Blue Gene/L systems at Argonne and Lawrence Livermore National Laboratory [5], and the Roadrunner system at Los Alamos National Laboratory [3]. The HPCC benchmark is a prototypical example of a benchmarking suite that reports results as summary statistics. In Confidence, we attempt to provide data beyond the typical summary statistics, and determine if that information can be used to examine machine performance meaningfully.

Bhatelé and Kalé examined the effects of contention in high performance interconnection networks [4]. A benchmark was constructed to have all pairs of processes send messages at the same time with the number of hops between each sender fixed. The results were averaged and reported for each factor of 4 message size between 4 Bytes and 1 MiB. The results indicated that for large message sizes and a large number of network hops, contention on the network could severely degrade communication performance. While the Confidence toolkit could be used to perform a similar study, our result reporting mechanisms include much more data than just the average latencies determined by the benchmark.

II. Confidence Configuration

Confidence seeks to present information about the performance of a benchmark or micro-benchmark over the entire range of observed results by presenting an empirically derived probability distribution that describes the performance of the system under study. It is important to note that the resultant probability distribution only describes the system performance at the time of the benchmark execution. Different systems will have different probability distributions, and the probability distributions may change over time depending on the degree and nature of resource sharing. On the other hand, if the benchmark execution adequately samples the entire network topology, we can expect that a high degree of measurement stability (e.g. stationarity and ergodicity) will exist. Here we present an abbreviated overview of the Confidence configuration used for our testing. A more detailed explanation of the Confidence toolkit is available in [12].

1) Collecting Data: Fundamentally, Confidence relies on *data binning* to gain insight into the behavior of benchmarks. Data binning is a data pre-processing technique that attempts to quantize observed values into a discrete number of buckets that adequately represent and emphasize the magnitude of the real observed values. By binning millions or billions of benchmark measurements, we can use the resulting frequency distribution to approximate the continuous stochastic process that describes the benchmark performance.

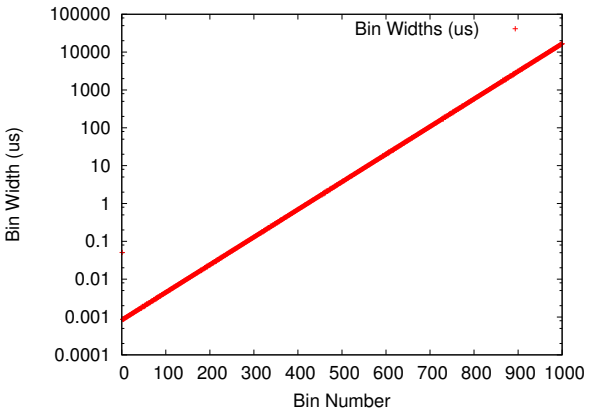

Fig. 1. Confidence logarithmic bin widths charted on a log scale.

Confidence recognizes that elapsed time measurements may differ by nanoseconds in terms of the returned values, but the timers used to observe those durations are not capable of providing measurements with such high degrees of precision. In recognizing that the precision of the timer used is an important component of how benchmark data is measured, Confidence includes a measurement abstraction layer called the Oak Ridge Benchmarking Timer, or *ORBTimer*, that both selects an appropriate timer for the underlying hardware (e.g. the Pentium/x86 cycle counter) and calibrates the timer in a manner consistent with the behavior of the actual timed kernel. The calibration is performed to determine a lower bound on the timing overhead (i.e. the absolute minimum timing value). During benchmark data collection we subtract the minimum observed timer value from the benchmark measurements to correct for timer related perturbation. By using the minimum timer value we recognize that we are only removing a fraction of the timing overhead in many cases, but more optimistic schemes would possibly contaminate the results. Additionally, during data collection, calls to the ORBtimer are timed themselves to ensure that the timer overhead does not change dramatically during a benchmarking trial and skew the results.

Once the fidelity of the timer has been determined, it is straightforward to determine an appropriate number of data bins and a bin width for the benchmark under study. Confidence provides both fixed width data bins and logarithmically scaled bin sizes. The fixed bin width, fw, is determined by dividing the maximum histogram time, T, by the number of requested bins, C:

$$fw_i = T/C \quad (0 \leq i < C). \tag{1}$$

Logarithmically scaled data bins are useful when the timing

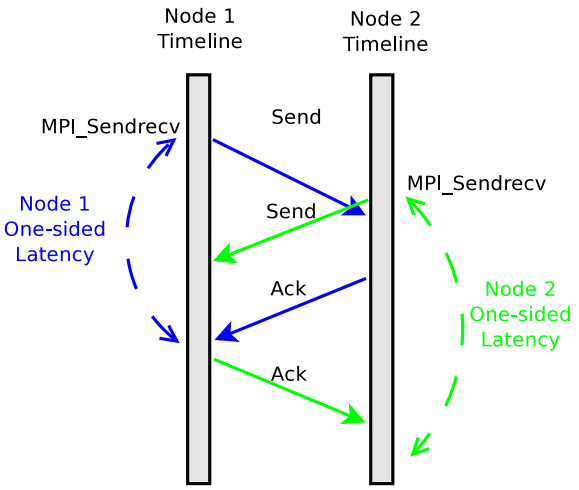

Fig. 2. Pair-wise communication pattern

data varies by several orders of magnitude or the amount of system memory for storing measurements is constrained. To use logarithmically scaled bins, the user must specify a bin size, S that is greater than 0. The logarithmic data bin width, lw, is described by the following function:

$$lw_0 = S \qquad (2)$$
$$lw_i = e^{S*i} - 1 \quad (0 < i < C).$$

Figure 1 displays the increasing bin widths for the first 1000 logarithmically sized bin widths with a size parameter of 50 nanoseconds (note the use of a logarithmic scale on the y-axis).

2) Latency Benchmark: For these tests we used *CommTest3*, a network benchmark included with Confidence. For each trial, *CommTest3* performed a pairwise MPI_Sendrecv between every MPI process running as part of the benchmark. MPI_Sendrecv was selected as the benchmarking kernel operation because it was well supported on all platforms and does not subdivide the communication over several user space calls (such as MPI_Wait), which would make it difficult to measure the constituent communication portions. For this test we did not use MPI collective operations to analyze network latencies. Fundamentally, extracting meaningful data from collective communication timings is difficult because collectives are an aggregation of sub-operations, and by timing the collective call we describe the average of all of the sub-operations, effectively smearing the observable detail during the measurement process.

Figure 2 illustrates the simple micro-benchmarking kernel used in these experiments. All of the processes cycle through each of their possible peers and performed a pair-wise MPI_Sendrecv operation of 1 byte of data. The results of these operations were reported in three different ways: the latency of the one-sided communication on node1, the latency of the one-sided communication on node2, and the pair-wise communication latency, which is the average of the two one-sided latencies. What we may prefer to catalog is the time between the first pair-wise send and the final pair-wise receipt acknowledgment; however, without a Global clock or time-stamp it is not possible to construct that quantity. Instead we use the pair-wise latency as an approximation of the desired quantity when the operations are not perfectly overlapped.

3) Data Analysis: Upon completing the benchmark and analysis routines locally at each node, all of the data is reduced to a single node by calling *measurement_aggregate*, and a final call to *measurement_analyze* performs a statistical analysis on the globally aggregated measurements. During the local and global analysis phases, Confidence produces the typical summary statistics (e.g. mean, mode, and standard deviation) as well as several higher-order statistics used to interpret the shape of the resultant data (skew and kurtosis). But the primary benefit of the Confidence analysis output is the capability to examine the empirically derived probability density functions (PDFs) and cumulative distribution functions (CDFs).

By binning many millions of timing samples, Confidence is able to construct an empirical approximation of the probability distribution of the timing data. The originating random process that generates the values is continuous but the individual measurements are discrete, so we must use a large number of discrete measurements to approximate the continuous probability distribution of the timing data. The resulting frequency distribution is used to mathematically construct the empirical PDFs, and empirical CDFs. Recall that a PDF, f, defines the probability for a random variable, X, to take a value in some range $[a:b]$, as:

$$P[a \leq X \leq b] = \int_a^b f(x)\,dx. \qquad (3)$$

Additionally, Confidence extracts the minimum observation for each benchmarking cycle (by default 100,000 trials), and bins that data and constructs distribution information and summary statistics for the observed minimums. Although the sample size of the observed minimums is small (1 observation per cycle per host by default), we will see in our network anomaly diagnoses that the observed minimum values can offer detailed insight into a benchmark's observed performance.

III. Diagnosing an Infiniband Performance Anomaly

While using the Confidence toolkit to examine the latency of various high performance machines at the National Center for Computational Science at Oak Ridge, we encountered the surprising network behavior evident in figures 3 and 4 on the Smoky Commodity Cluster, an Infiniband-based cluster. Smoky is an 80 node Linux test and development cluster available at Oak Ridge National Laboratory's National Center for Computational Science (NCCS). Each node contains four 2.0 GHz AMD Opteron processors, 32 GiB of main memory, an Intel Gigabit Ethernet NIC, and a Mellanox Infinihost III Lx DDR HCA. Five nodes act as dedicated routers onto the center-wide parallel file system, leaving 1200 cores available for user processes, and an aggregate system memory size of 24TiB. The Infiniband network is switched with a single

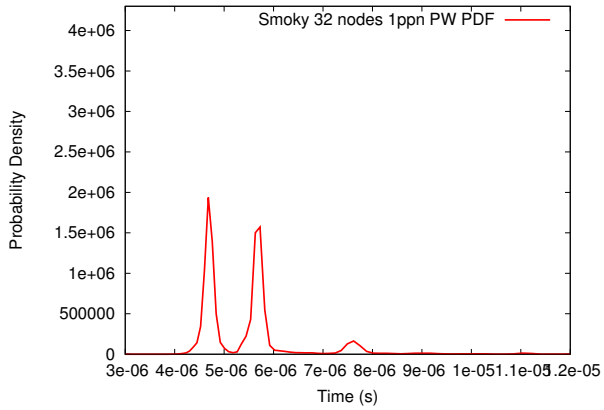

Fig. 3. PDF of pairwise communication latencies on Smoky, the NCCS Infiniband-based test and development cluster.

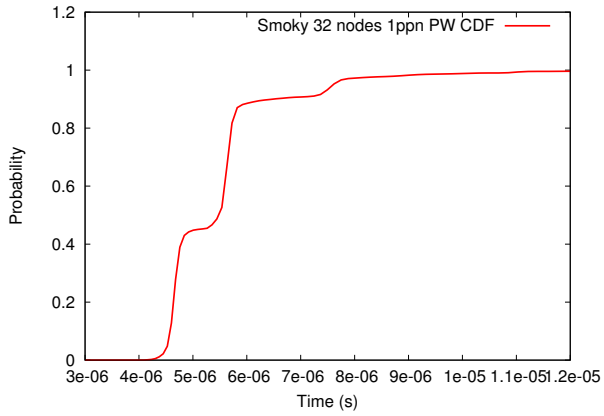

Fig. 4. CDF of pairwise communication latencies on Smoky, the NCCS Infiniband-based test and development cluster.

Voltaire DDR Infiniband Grid Director 2012 using four sLB-2024 24-port Infiniband Line cards. The switch provides 11.52 Tbps of bisection bandwidth with a reported port-to-port latency of 420 nanoseconds. The nodes run Scientific Linux SL release 5.0, a full Linux operating system based on the popular Red Hat Linux distribution. The benchmark was built using the Portland Group International compiler version 10.3.0 and OpenMPI version 1.2.6.

The pairwise message send latency PDF in Figure 3 clearly indicates the presence of three performance modes when 32 Smoky nodes are communicating simultaneously with only 1 communicating process per node. The first performance mode is centered at 4.6 μs, the second mode is centered at 5.7 μs, and the third mode is centered at 7.5 μs. With the aid of Confidence, our goal was to identify if any hardware issues contributed to the performance modes, and then would it be possible to "fix" the network performance such that send-receive latency was improved.

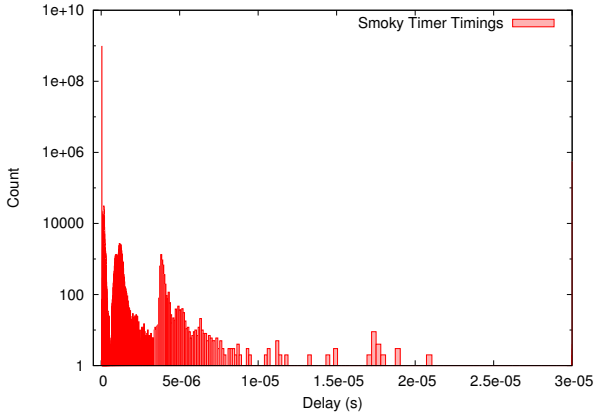

Fig. 5. Histogram of binned x86 timer values on Smoky.

A. *Ensuring Measurement Validity*

Before going any further with our analysis of the system performance, it is first critical to ensure that the system timer used for our measurements was returning appropriate values, and that the performance modes were not some artifact of our benchmarking techniques. Figure 5 shows a histogram of the observed timer overheads during our benchmarking run. Although the full version of Linux in use on Smoky results in somewhat noisy timing data, 99.8% of the timing overheads fall into the first histogram bucket, which spans 0 - 50 nanoseconds, indicating that the timer overhead was most often a negligible component of our measurements.

B. *Examining Hardware Performance*

Confidence decomposes benchmark execution iterations into cycles. For each benchmarking cycle, 200,000 messages are passed between each node-pair. In addition to recording the latencies for the 200,000 messages sent between each host pair, confidence also separately bins the absolute minimum observation for each host pair during a cycle. This best case minimum can be thought of as the actual hardware induced networking latency. We use the pairwise minimums rather than the one-sided minimums, because the one-sided minimum observation is typically only the time it takes to retrieve data from local memory after a successful RDMA put operation. Although the pairwise minimum is likely larger than the actual network hardware overheads, we believe that the pairwise timing acts as an accurate proxy for the actual hardware costs, and is closely correlated with the hardware costs. In order to increase the accuracy of our minimums distribution we increased the number of benchmarking cycles from the default, 10, to 100.

Figures 6 and 7 indicate that two of the performance modes are present in hardware only measurements. The third, much smaller, performance mode may be due to overhead in the network software stack, or it may simply be due to OS-based interrupts. Given the stark difference in latencies between the two hardware modes, we initially suspected that the performance issues may be due to some optimization

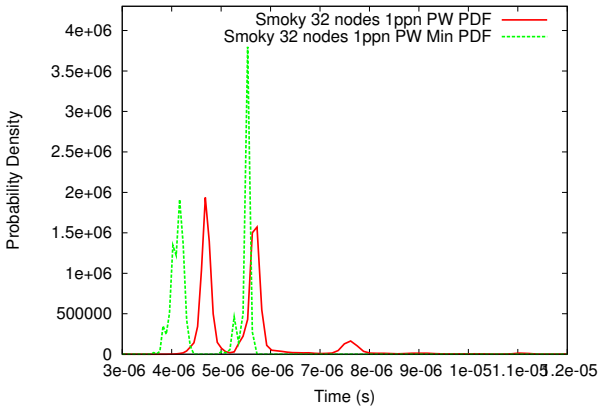

Fig. 6. All pairwise communications and minimum pairwise communications PDF for 32 nodes of Smoky with 1 process communicating per node.

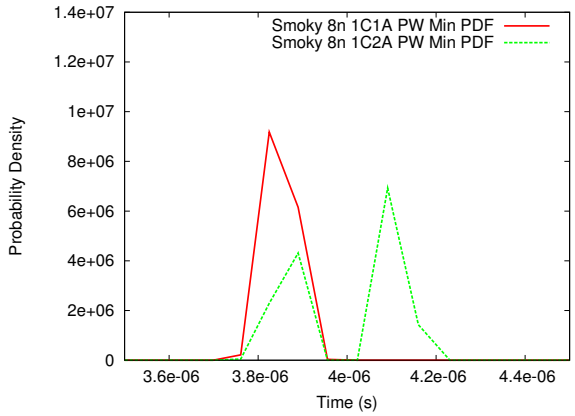

Fig. 8. Pairwise communication minimums for 8 nodes. The first plot engages only a single ASIC; the second plot spans both line card ASICS.

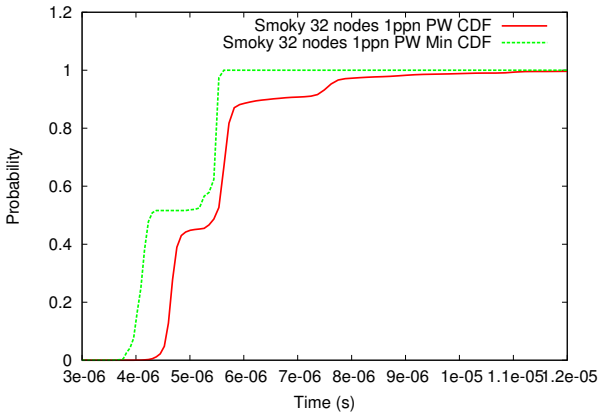

Fig. 7. All pairwise communications and minimum pairwise communications CDF for 32 nodes of Smoky with 1 process communicating per node.

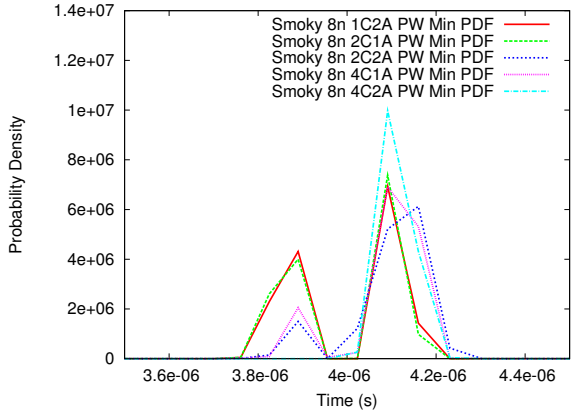

Fig. 9. Communication minimums for 8 nodes distributed in each balanced configuration across line cards and ASICs. The key indicates the number of line cards and ASICs per line card in use for each plotted PDF.

within the Infiniband switch that allowed lower send latencies (i.e. less hops) through some portions of the switch.

C. Analyzing Switching behavior

We contacted a representative for the switch vendor, who informed us that they too were puzzled by our hugely different performance modes, but that within a single line card, each application specific integrated circuit (ASIC) could communicate with every port within the ASIC without an additional network hop. However, there was no communication between the two ASICs that populated each line card. Realizing that this could indeed result in two performance modes, we immediately submitted a pair of benchmarking jobs that used only 8 smoky nodes (the maximum number of ports hosted by a single ASIC). The first job collected the pairwise communication minimums for 8 nodes spanning a single ASIC, the second job engaged only a single line card, but allocated 4 nodes to each ASIC.

Figure 8 clearly demonstrates that the vendor's information was accurate. Within the same ASIC, minimum pairwise latencies tended about 3.9 μs, whereas ASIC spanning communications required on the order of 4.1 μs. Although the observed ASIC spanning performance was bi-modal, we were concerned that both observed performance modes existed within the first original performance mode we demonstrated in figure 3.

We hypothesized that simply hopping across the switch backplane from one ASIC to the next within the switch may not require as many switch hops as crossing between both switch line cards and ASICs. Figure 9 shows the resulting latency PDF for our Confidence jobs that allocated 8 total nodes in each of the following configurations: on a single line card to a single ASIC, one line card using both ASICS, two line cards using a single ASIC per card, two line cards with both ASICs in use, four line cards with a single ASIC in use on each, and four line cards with two ASICs in use per line card. Clearly, the two resulting performance modes indicated that the switch backplane does not introduce any additional latency, and that the reduced performance detected in our original tests was not likely due to any anomalous switching behavior.

We needed to collect more data. Figure 10 demonstrates the resulting performance as we added nodes to our tests

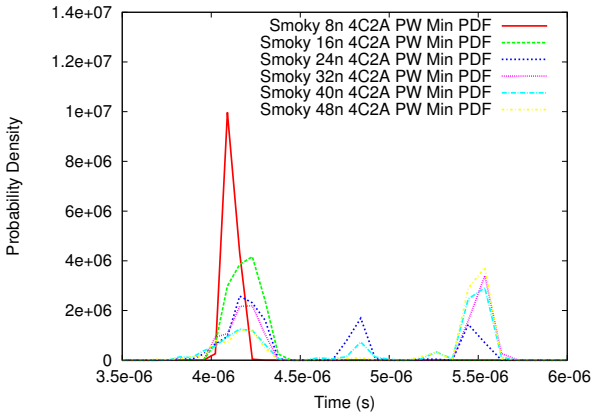

Fig. 10. Communication minimums for nodes distributed evenly across all of the switch resources (line cards and ASICs). Each plotted PDF shows the addition of exactly one node to each ASIC in the switch.

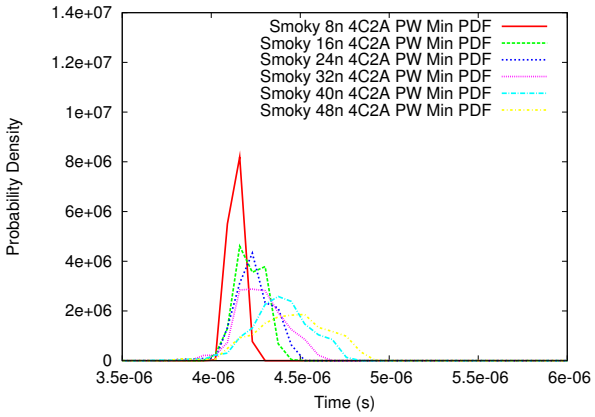

Fig. 11. Communication minimums for nodes distributed evenly across all of the switch resources (line cards and ASICs) with RDMA eager mode enabled for 80 hosts. Each plotted PDF shows the addition of exactly one node to each ASIC in the switch.

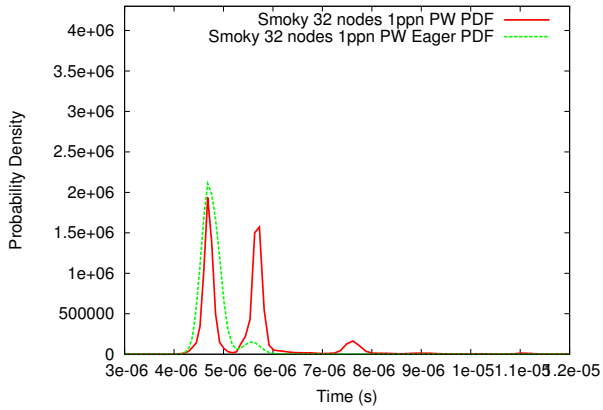

Fig. 12. Pairwise Communication costs at one process per node

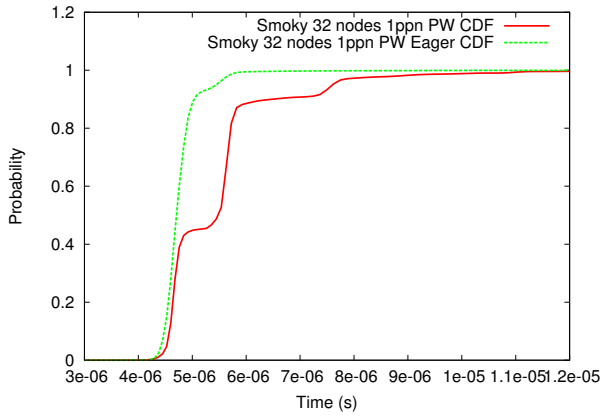

Fig. 13. Pairwise Communication costs at one process per node

while balancing the results across each switch line card and ASIC. That is each curve in the plot is the result of adding a single node for each ASIC, and then running the confidence benchmark with that configuration. At this point, the issue became clear, the performance modes were not due to switch hop counts, but instead due to an anomaly that appears at any time more than 16 switch ports were in use. Additional testing determined that the specific switch ports did not matter (though intra-ASIC communication makes that data hard to interpret), and at this point we realized that the switching hardware was not the root cause of the degraded performance.

Rather, we noted that the Infiniband driver compiled into OpenMPI used different message send protocols depending on the number of hosts configured in the system. The first 16 hosts an MPI process communicates with use an RDMA eager protocol; however, due to the relative lack of memory on the NIC, all subsequent MPI Sends resort to an eager send protocol that uses an operating system buffer (requiring context switches). Within OpenMPI we were able to adjust the eager RDMA limit to 80 hosts, with figure 11 showing the resultant performance PDFs.

Figures 12 and 13 show the resulting improvement in network latencies when scheduling 32 processes on Smoky with forced eager RDMA communications versus the default configuration. In addition, the removal of the operating system assisted send seemed to impact the smaller performance mode. This is likely due to the reduced likelihood of spurious context switches (computational noise) due to the smaller amount of time spent performing communications per benchmarking run.

IV. Assessing the Impacts of Network Upgrades

In our earlier study of the Jaguar network [12], we noted that network latencies were severely impacted by the number of node processes simultaneously sending. While re-performing a series of experiments to ensure the validity of our earlier work prior to a presentation, we learned that our benchmark results now varied greatly from our earlier observations. We noted that the default compilers and MPI platform had experienced revisions since our earlier tests, but we were skeptical that the software stack had caused the large degree of change

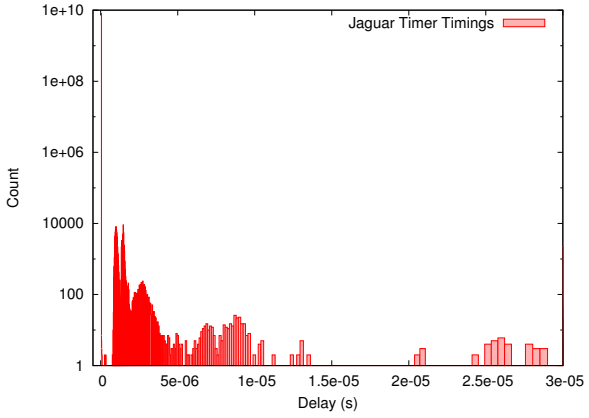

Fig. 14. Histogram of binned x86 timer values on Jaguar

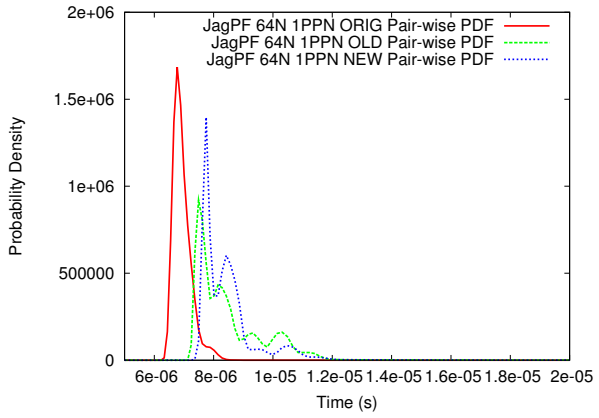

Fig. 15. Pairwise communication latency PDF for 64 Jaguar nodes with 1 communicating process per node. The first line is the data originally collected, the second line uses the old software stack running after the system updates, and the third line uses the new software stack running after the system update.

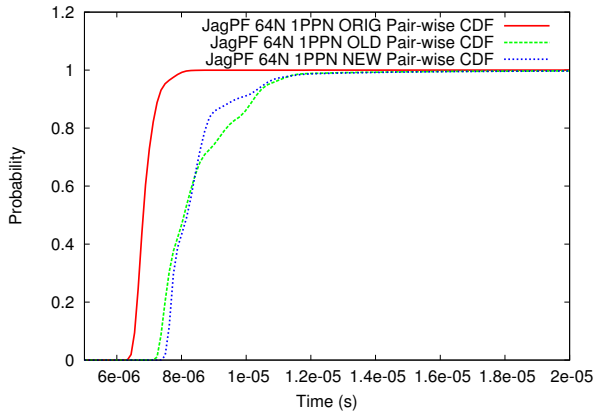

Fig. 16. Pairwise communication latency CDF for 64 Jaguar nodes with 1 communicating process per node. The first line is the data originally collected, the second line uses the old software stack running after the system updates, and the third line uses the new software stack running after the system update.

we observed in our Confidence-based benchmarking. Here we describe the process we used to understand how the behavior of the Jaguar interconnection network, SeaStar2, changed in 4 months time.

The Jaguar system at Oak Ridge National Laboratory's Leadership Computing Facility (LCF) includes both the fastest and 16th fastest machines in the Nov 2009 Top500 List [7]. For these tests we used the faster Jaguar system, the upgraded Cray XT5. Jaguar XT5 is composed of 18,688 dual socket compute nodes running Compute Node Linux, a lightweight Linux-based operating system. Each socket contains a hex-core AMD Opteron 2435 processor at 2.6 GHz for a total core count of 224,256, and each node includes 16GiB of DDR2-800 main memory for a total system memory of 299 TiB. Jaguar XT5 also include 256 service and I/O nodes running a full version of SuSE Linux that provide file system and external network access. Each node in Jaguar is connected using a SeaStar2 router capable of transmitting 76.8 Gbps in each direction on the 3-dimensional torus interconnection network (230.4 Gbps per node total). The aggregate network bandwidth is rated at 2.992 Tbps.

Our original benchmark code was built using the Cray XT5 compiler wrapper and MPI libraries, based on the Portland Group International compiler version 9.0.4 and XT Message Passing Toolkit 3.5.1. The more recent configuration relied on Portland Group International compiler version 10.3 and the XT Message Passing Toolkit 4.0.0. All benchmark runs were performed on 64 node allocations randomly selected by the scheduler. The allocations included no more than one shelf from each of 5 separate cabinets. By ensuring the allocation spanned multiple cabinets, we can ensure that all 3 dimensions of the torus network are actively utilized.

A. Ensuring Results Validity

As in the last study, we first begin with ensuring that our observed timer values are meaningful. In figure 14 we can see a histogram of the system timer measurements for Jaguar. Not surprisingly, the observed timer skews appear very similar to the OS noise described for the platform [8], and does not appear normally distributed. The largest number of timer entries fell into the first data bin, which spans 0 - 50 nanoseconds (the total number of timer observations was 8.064×10^9). Recall that these initial timer values (and all of the other gathered values) are reduced by exactly the amount of the *minimum* timer delay observed during the timer calibration phase. This is done to remove some of the cost of the timer overhead from our collected data in a safe manner that does not invalidate the measured results. In the case of Jaguar, the Cray XT5, this appears to be an adequate technique with well over 99.99% of the timer observations taking less than 50 nanoseconds.

B. System Upgrade Measurements

Figures 15 and 16 show the empirical probability density functions and cumulative distribution functions for a single communicating process per node using all three of our test configurations. The lines labeled "ORIG" are the original ob-

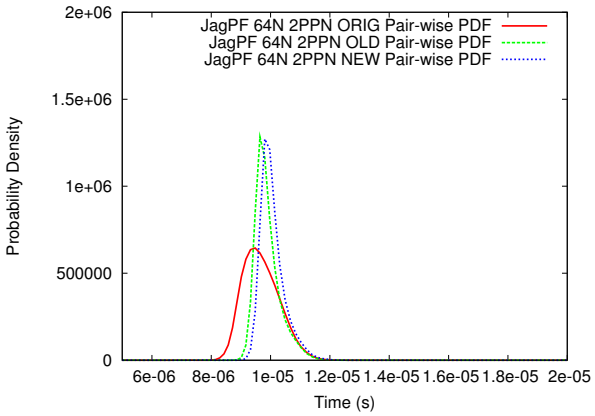

Fig. 17. Pairwise communication latency PDF for 64 Jaguar nodes with 2 communicating processes per node. The first line is the data originally collected, the second line uses the old software stack running after the system updates, and the third line uses the new software stack running after the system update.

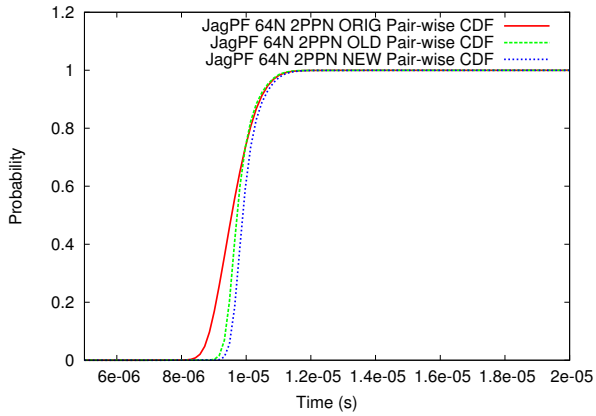

Fig. 18. Pairwise communication latency CDF for 64 Jaguar nodes with 2 communicating processes per node. The first line is the data originally collected, the second line uses the old software stack running after the system updates, and the third line uses the new software stack running after the system update.

servations from four months ago. The lines labeled "OLD" use the same compiler and parallel tool platforms as the original observations but were observed more recently, and the lines labeled "NEW" were recently observed and use the updated compiler and parallel tool platforms. Both the PDF and CDF clearly show that all of the performance changes are not due to incrementing the compiler and network middleware. In fact, although it appears that some performance changes are due to the change in software configuration, the most fundamental changes in the network latency appear to have originated outside of the software stack. In collaboration with the Jaguar system administration team we learned that a network driver upgrade had occurred in the intervening period, and that was likely the source of our observed performance differences. The driver upgrade nominally prevented a deadlock condition in the network; however, in this paper we are only concerned with evaluating the network performance modifications resulting from the system upgrade.

Our original study of the Jaguar network focused on determining the optimal number of independent message originating processes (e.g. MPI tasks) to use in pairing with the Cray XT hardware. In figures 17 and 18 we present the updated empirical probability distributions for two communicating processes per node and in figures 19 and 20 we present the empirical probabilities for four communicating processes per node. We again note that network latency sensitive applications may be well served to use a single MPI task for remote communications and employ a threading approach, such as OpenMP, to leverage the large number of processing cores with a Jaguar compute node. However, it does appear that the network driver update results in a more reproducible (i.e. "peakier") message latency, even if the performance is slightly degraded from the results of our original measurements

Figures 21 and 22 show the empirical probability distributions for network latency with twelve communicating processes per node. With all node processes sending and

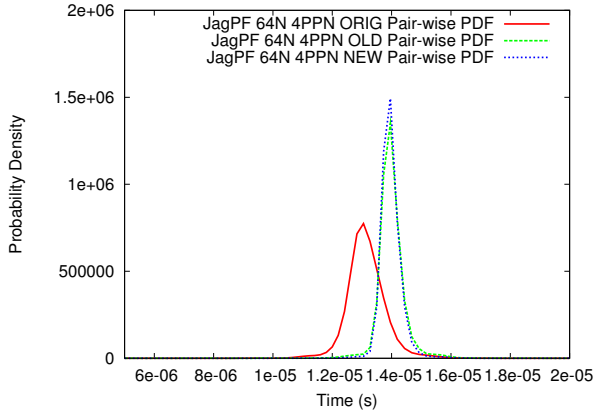

Fig. 19. Pairwise communication latency PDF for 64 Jaguar nodes with 4 communicating processes per node. The first line is the data originally collected, the second line uses the old software stack running after the system updates, and the third line uses the new software stack running after the system update.

receiving network messages it is apparent that the driver update has dramatically altered the measured network latency performance without significantly modifying the mean or median network latency. The CDF clearly demonstrates that in all configurations, observations will be evenly distributed about 38 microseconds. However, after the driver update observed network latencies are much more uniformly distributed over the sample space. The original network configuration appears to provide much more predictable network performance (i.e. greater peakiness) with basically identical average case performance. In particular, we expect that collective communication patterns that are performance limited by the slowest participating process will be negatively impacted by the updated performance distribution.

Although the SeaStar driver update clearly impacts the network message latency, it is less clear how exactly the update

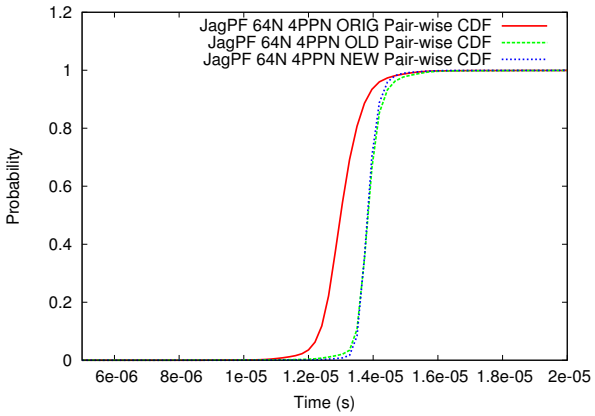

Fig. 20. Pairwise communication latency CDF for 64 Jaguar nodes with 4 communicating processes per node. The first line is the data originally collected, the second line uses the old software stack running after the system updates, and the third line uses the new software stack running after the system update.

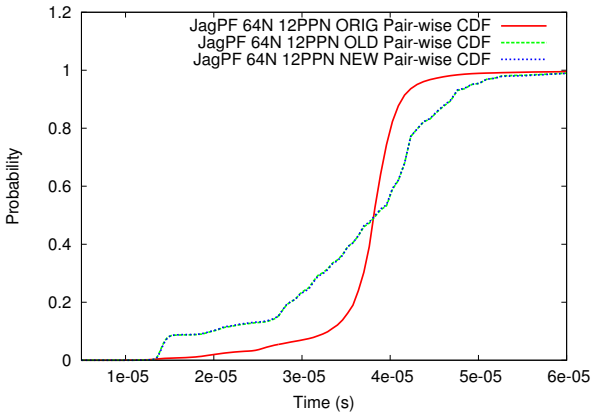

Fig. 22. Pairwise communication latency CDF for 64 Jaguar nodes with 12 communicating processes per node. The first line is the data originally collected, the second line uses the old software stack running after the system updates, and the third line uses the new software stack running after the system update.

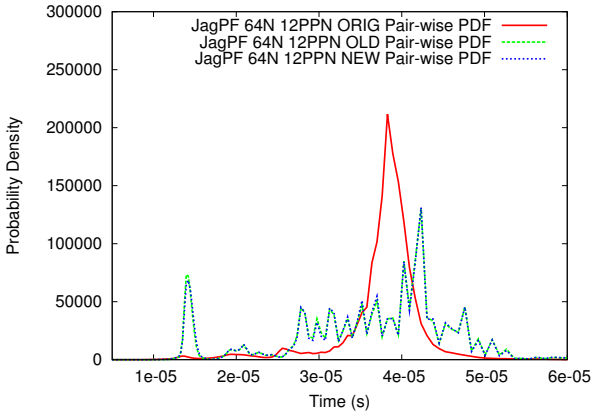

Fig. 21. Pairwise communication latency PDF for 64 Jaguar nodes with 12 communicating processes per node. The first line is the data originally collected, the second line uses the old software stack running after the system updates, and the third line uses the new software stack running after the system update.

will affect existing applications. A single communicating process per node will now likely achieve higher latencies and incur greater variability in its measured send-receive latency (though perhaps with less chance of encountering a system halting deadlock). It appears the Cray engineers have endeavored to improve the consistency of network operations when 2 or 4 processes are communicating per node at the cost of increasing the observed network latency. That is, network delays will be more repeatable, which may result in a simpler programming model, but that overall application performance and scalability are likely to be reduced with this driver update. In the case of 12 simultaneously communicating processes, it appears that updating the network driver has degraded the send-receive latencies, and has likely decreased the repeatability of network injections. Due to the degraded performance of this configuration we maintain our original suggestion: a hybrid programming model that uses four or less communicating processes per node with OpenMP or PThreads to utilize the remaining processing cores is likely to achieve better performance for latency-sensitive parallel applications running on the Jaguar system.

V. Conclusion

In this paper we utilized the Confidence toolkit to diagnose and analyze anomalous network behavior in HPC interconnection networks. On a commodity component Linux cluster we were able to determine the cause of a network performance degradation and on a Cray XT5 we were able to examine the impacts of network and software stack upgrades. Confidence is well suited for these type of detailed analysis tasks because it provides more than simple summary statistics. The empirical probability distributions generated by Confidence can aid in measuring performance, locating areas of degraded performance, and evaluating how system component performance changes over time.

The results presented in Section III demonstrate how to use the Confidence toolkit to thoroughly understand the symptoms of a network performance issue. Further, we demonstrated how we were able to refine our benchmarking configurations to demonstrate the performance modes for the switch in question, eventually ruling out the switch as a possible cause for the anomalous performance. From there, we were able to locate the problem as a poorly tuned interaction between the Infiniband host channel adapters and the MPI software stack. We then tuned the software stack to allow eager RDMA communications for all our small messages, resulting in significant performance improvements.

Secondly, in Section IV, we used Confidence to understand the network performance impacts of system software and network driver upgrades. Our tests demonstrated that upgrades to the compiler and MPI implementation result in relatively small changes in network latencies; however, the network driver

upgrades appeared to greatly change the network performance characteristics. In general, the SeaStar update improved the peakiness and repeatability of small message transmissions at the cost of slightly longer absolute network delays. Finally, the upgrade appeared to leave the average case performance of 12 communicating processes per node intact; but, the performance variability is greatly increased with the driver update. It is still our recommendation that application programmers typically employ very few MPI processes per node, and instead rely on techniques such as PThreads or OpenMP to leverage the processing capabilities of the majority of the node processing cores.

VI. Future Work

For our future work we are interested in continuing to explore the use of empirical probability distributions to study shared resource performance. We intend to study how the number of nodes allocated in our networking studies affects the empirical probability distributions. We are also beginning to experiment with fitting one-sided probability distributions (such as the F distribution and log-normal distribution) to benchmark results to provide for some further data characterization. In particular, we are interested in comparing the formulations for higher-order moments (i.e. skew and kurtosis) in one-sided distributions to our measured moments. Further, we are interested in looking at non-central moments of probability distributions. In HPC systems mean and median performance are not the desired performance levels, so calculating the distribution moments about the observed minimums may provide a higher quality performance summary than the typical central moments (mean, variance, skew, kurtosis).

One of the application areas we are most interested in analyzing with Confidence is the computational noise generated by the operating system. By instrumenting various micro-benchmarking kernels with the Confidence toolkit, we feel that it will be possible to perform a detailed examination of application induced noise, and determine the impacts of the various types of noise experienced on high performance computer systems. We are also exploring Confidence as a tool for analyzing the performance of intensive file system operations. We have instrumented both IOR and XDD with the confidence toolkit and we are in the process of analyzing "thread drift" during large data transfers, and the performance of various non-contiguous I/O access patterns.

Acknowledgments

This work was supported by the Department of Defense (DoD) and used resources at the Extreme Scale Systems Center, located at Oak Ridge National Laboratory (ORNL) and supported by DoD. This research also used resources at the National Center for Computational Sciences at ORNL, which is supported by the U.S. Department of Energy Office of Science under Contract No. DE-AC05-00OR22725. Special thanks to Pawel Shamis for explaining various details related to the OpenMPI BTL OpenIB driver.

References

[1] S. Alam, R. Barrett, M. Bast, M. R. Fahey, J. Kuehn, C. McCurdy, J. Rogers, P. Roth, R. Sankaran, J. S. Vetter, P. Worley, and W. Yu, *Early evaluation of IBM BlueGene/P*, SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing (Piscataway, NJ, USA), IEEE Press, 2008, pp. 1–12.

[2] Sadaf R. Alam, Jeffery A. Kuehn, Richard F. Barrett, Jeff M. Larkin, Mark R. Fahey, Ramanan Sankaran, and Patrick H. Worley, *Cray XT4: an early evaluation for petascale scientific simulation*, nov. 2007, pp. 1 –12.

[3] Kevin J. Barker, Kei Davis, Adolfy Hoisie, Darren J. Kerbyson, Mike Lang, Scott Pakin, and Jose C. Sancho, *Entering the petaflop era: the architecture and performance of Roadrunner*, SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing (Piscataway, NJ, USA), IEEE Press, 2008, pp. 1–11.

[4] Abhinav Bhatele and V. Laxmikant, *An evaluative study on the effect of contention on message latencies in large supercomputers*, IPDPS '09: Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing (Washington, DC, USA), IEEE Computer Society, 2009, pp. 1–8.

[5] Kei Davis, Adolfy Hoisie, Greg Johnson, Darren J. Kerbyson, Mike Lang, Scott Pakin, and Fabrizio Petrini, *A performance and scalability analysis of the BlueGene/L architecture*, SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing (Washington, DC, USA), IEEE Computer Society, 2004, p. 41.

[6] Piotr Luszczek, Jack J. Dongarra, David Koester, Rolf Rabenseifner, Bob Lucas, Jeremy Kepner, John Mccalpin, David Bailey, and Daisuke Takahashi, *Introduction to the hpc challenge benchmark suite*, Tech. report, 2005.

[7] H.W. Meuer, E. Strohmaier, H. Simon, and J.J. Dongarra, *TOP500 Supercomputer Sites, 34th edition*, The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '09), 2009.

[8] Sarp Oral, Feiyi Wang, David A. Dillow, Ross Miller, Galen M. Shipman, and Don Maxwell, *Reducing application runtime variability on Jaguar XT5*, CUG-2010 (Edinburgh, UK), Cray User's Group, 2010.

[9] Fabrizio Petrini, Eitan Frachtenberg, Adolfy Hoisie, and Salvador Coll, *Performance evaluation of the quadrics interconnection network*, Cluster Computing **6** (2003), no. 2, 125–142.

[10] Fabrizio Petrini, Darren J. Kerbyson, and Scott Pakin, *The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of ASCI Q*, SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing (Washington, DC, USA), IEEE Computer Society, 2003, p. 55.

[11] Amar Phanishayee, Elie Krevat, Vijay Vasudevan, David G. Andersen, Gregory R. Ganger, Garth A. Gibson, and Srinivasan Seshan, *Measurement and analysis of TCP throughput collapse in cluster-based storage systems*, FAST'08: Proceedings of the 6th USENIX Conference on File and Storage Technologies (Berkeley, CA, USA), USENIX Association, 2008, pp. 1–14.

[12] Bradley W. Settlemyer, Stephen W. Hodson, Jeffery A. Kuehn, and Stephen W. Poole, *Confidence: Analyzing performance with empirical probabilities*, Proceedings of 2010 Workshop on Application/Architecture Co-design for Extreme-scale Computing (AACEC), September 2010.