The Impact of System Design Parameters on Application Noise Sensitivity

Kurt B. Ferreira*†, Patrick G. Bridges†, Ron Brightwell* and Kevin T. Pedretti*

* Scalable System Software Department
Sandia National Laboratories
Albuquerque, NM 87185–1319
{kbferre,rbbrigh,ktpedre}@sandia.gov

† Computer Science Department
The University of New Mexico
Albuquerque, NM 87131
{kurt,bridges}@cs.unm.edu

Abstract—**Operating system noise, or "jitter," is a key limiter of application scalability in high end computing systems. Several studies have attempted to quantify the sources and effects of system interference, though few of these studies show the influence that architectural and system characteristics have on the impact of OS noise at scale. In this paper, we examine the impact of three such system properties: platform balance, "noisy" node distribution, and non-blocking collective operations. Using a previously-developed noise injection tool, we explore how the impact of noise varies with these platform characteristics. We provide detailed performance results that indicate that a system with relatively less network bandwidth is able to absorb more noise than a system with more network bandwidth. Our results also show that application performance can be significantly degraded by only a subset of noisy nodes. Furthermore, the placement of the noisy nodes is also important, especially for applications that make substantial use of collective communication operations that are tree-based. Lastly, performance results indicate that non-blocking collective operations have the ability to greatly mitigate the impact of OS interference. Combined, these results show that the impact of OS noise is not solely a property of application communication behavior, but is also influenced by other properties of the system architecture and system software environment.**

I. INTRODUCTION

Research has shown that operating system (OS) interference is a key limiter of application performance in large-scale systems [7], [9], [17], with much of this work focusing on how different applications respond to different amounts and types of noise. However, the

measured impact of noise has varied widely between systems, with some platforms showing relatively little performance impact from noise [3] and others showing substantial performance impacts [7], [9], [17].

In this paper, we study how a number of important architectural and system software design features affect the noise sensitivity of High-End Computing (HEC) systems. In particular, we examine how the following important system features impact the sensitivity of applications to OS noise:

- The hardware *balance* of the system, the ratio of peak network bandwidth (bytes/second) to peak compute performance (floating point operations/second) [1], [16].
- The isolation of "noisy" nodes running full-featured operating systems to a subset of the nodes on the system.
- The use of collective communication mechanisms that are relatively insensitive to noise.

Our results provide important guidance to HEC hardware and system software designers by demonstrating that:

1) The impact of noise is dependent on machine parameters - specifically network performance and the resulting balance of the hardware platform;
2) Isolating noise to only a subset of system nodes is not sufficient to mitigate the impact of noise at scale on certain key HEC applications;
3) The placement of noisy nodes in the system matters, with noisy nodes close to the root of the system collective communication tree (rank 0) having less impact on application performance

than nodes further from rank 0; and,

4) Non-blocking collective reductions can substantially mitigate the impact of noise on applications running in HEC systems.

To our knowledge, this is the first such empirical study on the impact system design parameters have on an HEC applications sensitivity to OS noise.

The remainder of the paper is organized as follows. Section II provides background on OS noise, its effect on application performance in HEC systems, and the system features listed that we hypothesize affect the impact of noise on applications in HEC systems. Section III describes the hardware platform we use to test the impact of these system features on application noise sensitivity, how we vary these features on this hardware platform, and the applications we use to test how variations in system features impact application noise sensitivity. Section IV then presents and analyzes the results of these experiments. Section V follows with discussion of related work in this area, and Sections VI and VII present directions for future work and conclude.

II. BACKGROUND

A. OS Noise

The detrimental side effects of OS interference on massively parallel processing systems have been known and studied, primarily qualitatively, for nearly two decades [19]. Previous investigations have shown that the global performance cost of noise is, in many cases, due to the variance in the time for processes to participate in a collective operation, such as `MPI_Allreduce`, resulting in the accumulation of noise at scale [15]. These interruptions occur for a variety of reasons, from the periodic timer "tick" commonly used by many commodity operating systems to keep track of time, to the scheduling points used to replace the currently running process with another task or kernel daemon. In each of these cases, processor cycles are taken away for the duration of the noise event, which can typically vary from a few microseconds to a few milliseconds [3], [17]. A number of recent studies [7], [9], [17] have shown that even relatively minimal OS noise (e.g. 2.5% OS overhead) can reduce the performance of applications at scale by orders of magnitude.

B. System Features Affecting Noise

While noise has been recognized as a substantial factor in application scaling in HEC systems, different platforms have seen dramatic differences in how much impact noise has had on applications, and a number of techniques have been proposed for mitigating the impact

of system noise on applications [13]. For example, noise injection studies on two different systems, the Cray Red Storm system [7] and the IBM BlueGene/L system [3] measured dramatically different slowdowns in `MPI_AllReduce` performance in the presence of small amounts (e.g. 2.5%) of injected noise. Exactly what accounts for the different noise sensitivities of these two systems is not clear; system differences that could affect noise propagation include the different compute/communication balances of the two systems, BlueGene/L's isolation of operating systems with nontrivial amounts of noise to a subset of its nodes, or the noise-resistant collectives that BlueGene/L includes in the form of a hardware collective communication network.

1) System Balance: System balance, the ratio of peak network bandwidth (bytes/second) to peak compute performance (floating point operations/second) [1], [16] is one potential system hardware feature that we hypothesize could change the impact of OS noise on application performance. Previous work has shown that a bytes-to-flops ratio of one results in the best performance for a typical HEC workload [16]. However, given the ever-increasing compute performance available from multi-core processors and the inability of network performance to keep pace, well-balanced machines are becoming increasingly difficult to design and deploy. Our supposition is that OS noise is less likely to impact applications on systems that are less balanced, since OS noise is more likely to be absorbed in an environment where there are potentially excess compute cycles available.

2) OS Noise Isolation: Isolating OS services to a subset of system nodes has been a popular technique mitigating the impact of necessary OS noise on application performance. The ASCI Q system, for example, was changed to run most system services on dedicated processors and dedicated nodes in response to the well-known study of the impact of noise on the SAGE application [17]. Similarly, IBM BlueGene-series systems run a low-noise Compute Node Kernel (CNK) [14] on most compute nodes, and distribute Linux OS service nodes throughout the system to which system calls are forwarded from CNK-based nodes in an effort to isolate OS noise in the system. Finally, we note that a similar strategy was proposed for supporting full-featured operating system services on the original Sandia Intel Paragon system [10], but was never deployed to production. Our hypothesis is that this isolation of OS noise can substantially reduce the impact of noise to applications in the system, but that the placement of "noisy" nodes in the system may matter.

2

3) Noise-resistant Collectives: Finally, since collective communication primitives such as `MPI_AllReduce` have been shown to be a key factor in accumulating noise in HEC systems, a variety of noise-resistant collective communication primitives has been proposed as a possible means of mitigating this effect. Hardware-based collective communication primitives are one such technique, though they have had mixed success. Hardware-assisted barriers on the ASCI Q system, for example, had little impact on improving SAGE performance in the presence of noise [17], while the more general hardware-based collectives on IBM BlueGene-series systems [2] are generally regarded as an important source of that system's scalability.

Similarly, non-blocking collectives have been proposed to the recently reconvened MPI Forum for inclusion in MPI-3 [8] as another means of reducing the impact of OS noise on large-scale applications that require collective communication. Similar to non-blocking point-to-point operations, non-blocking collectives allow an application writer to hide the cost of the operation by overlapping the network communication with computation. The key to benefiting from a non-blocking operation is the size of the overlap portion of the computation. We hypothesize that the amount of overlap an application is actually able to achieve between computation and collective propagation is directly related to the ability of noise-resistant collective implementations to absorb noise.

III. APPROACH

In this section, we provide an overview of the hardware and software environment of the test system used to evaluate the impact of the system features described in Section II-B on noise sensitivity. This includes a description of the test platform, the changes to the platform we have implemented in system software on this platform, the three applications evaluated on this platform, and a benchmark we have developed to evaluate the impact of non-blocking collectives on noise propagation and accumulation.

A. Hardware Platform

We used the Red Storm system located at Sandia National Laboratories as a test platform. Red Storm is a Cray XT3/4 series machine consisting of nearly 13 thousand nodes. For our experiments, we used a 3000-node subset of the machine in dedicated mode. Each compute node in this subset contains a 2.4 GHz dual-core AMD Opteron processor and 4 GB of main memory. Additionally, each node contains a Cray SeaStar [4] network interface and high-speed router. The SeaStar

is connected to the Opteron via a HyperTransport link. The current-generation SeaStar is capable of sustaining a peak unidirectional injection bandwidth of more than 2 GB/s and a peak unidirectional link bandwidth of more than 3 GB/s. An important characteristic of the SeaStar network on the Cray XT is that it is interrupt driven. When a message arrives at the SeaStar, it interrupts the host processor, the host OS performs the necessary protocol processing, and then programs the SeaStar's network DMA engines directly to deliver the incoming message to the appropriate buffer in destination process' address space. Red Storm is an ideal platform on which to explore system balance, as it is one of the more balanced modern-day systems.

B. Noise Injection

For our experiments, we modified the system to run the Catamount lightweight kernel containing our noise injection framework described in [7] instead of the normal production version, along with additional modifications allowing us to control *which* nodes generated noise. The Catamount lightweight kernel is an ideal environment for noise studies due to its extremely low native noise signature and demonstrated record of scalability. All of our experiments were run using one process per node, thereby maximizing the overall balance of the system.

For this work, we used noise signatures that represents 2.5% net processor interference, focusing on a 10Hz 2500μsec noise profile that is representative of kernel daemon interference. We focused on this 2.5% profile due to both specific measurements made on commodity systems and results of previous research that demonstrated the importance of this noise level [7], [17]. It is important to note that unloaded systems (e.g. those doing no communication, I/O, or memory management activities) can have lower noise signatures with corresponding lower overheads. However, we believe these unloaded noise patterns are not realistic for characterizing the behavior of real-world HEC applications, and this view is supported by recent results [15] that show significant OS overhead from scheduling and ACPI interrupts on loaded HEC Linux systems.

C. System Balance Modification

To evaluate the impact of system balance on the noise sensitivity of HEC applications, we modified the balance of the Red Storm system by degrading network performance using an existing hardware mechanism. In addition to a Red Storm full-bandwidth configuration, we present results of 3/4, 1/2 and 1/4 bandwidth configurations. These configurations correspond to the
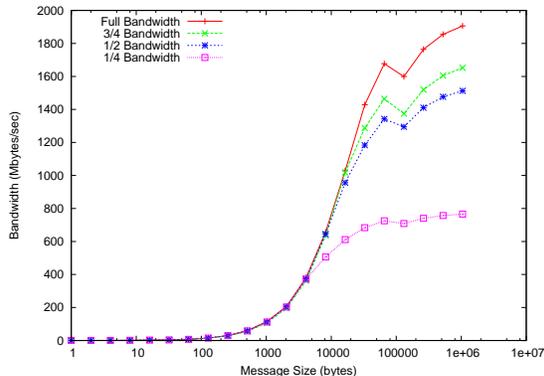
Fig. 1. MPI Ping-Pong link throughput for degraded bandwidths.

balance of the ASC Purple supercomputer located at Lawrence Livermore National Laboratories, a commodity single processor InfiniBand cluster, and a commodity dual-processor InfiniBand cluster similar to that of the Thunderbird cluster at Sandia National Laboratories, respectively. Note, that IBM's BlueGene/L&P series of machines are even further imbalanced towards excess computation than the configurations examined here. See [4] for a balance comparison of several recent large-scale computing systems.

Figure 1 shows the resulting MPI ping-pong bandwidth numbers. For message sizes less than 4 KB, the bandwidth of the three scenarios are nearly equal due to message overhead, but for message sizes greater than 4 KB, the bandwidth curves diverge, with 1 MB messages topping off at around 1900 MB/sec, 3/4 bandwidth at slightly more than 1600 MB/sec, 1/2 bandwidth at 1500 MB/sec and 1/4 bandwidth at just under 800 MB/sec. We also verified that network latency numbers are unaffected by these bandwidth changes.

D. Non-blocking Collectives

To test the impact of non-blocking collectives on noise propagation and accumulation, we implemented a non-blocking `MPI_Allreduce` collective operation. Allreduce was chosen due to the sensitivity of this collective operation shown in [7]. Although there are a number of non-blocking collective libraries currently in existence (most notably [8]), we chose to implement our own in order to take full advantage of the SeaStar interconnect on Red Storm.

Because using non-blocking collectives would require substantial changes to an application, we also implemented a bulk-synchronous microbenchmark that uses this non-blocking collective library. This bulk-synchronous microbenchmark allows us to specify the

length of time of the compute phase before all nodes join in the Allreduce-based synchronize step. We set the synchronization step of the microbenchmark to occur at a rate previously measured for the SAGE and POP applications, described below.

E. Test Applications

When appropriate, we used three applications that represent important HEC modeling and simulation workloads, CTH, SAGE, and POP, to evaluate the impact of hardware and system software changes on noise propagation and accumulation in applications. These applications represent a range of different computational techniques, all frequently run at very large scales (i.e. tens of thousands of nodes), and are key applications to both the United States Departments of Energy and Defense. We briefly describe these applications below.

CTH [6] is a multi-material, large deformation, strong shock wave, solid mechanics code developed by Sandia National Laboratories with models for multi-phase, elastic viscoplastic, porous, and explosive materials. CTH supports three-dimensional rectangular meshes; two-dimensional rectangular, and cylindrical meshes; and one-dimensional rectilinear, cylindrical, and spherical meshes, and uses second-order accurate numerical methods to reduce dispersion and dissipation and to produce accurate, efficient results. It is used for studying armor/anti-armor interactions, warhead design, high explosive initiation physics, and weapons safety issues.

SAGE, SAIC's Adaptive Grid Eulerian hydrocode, is a multi-dimensional, multi-material, Eulerian hydrodynamics code with adaptive mesh refinement that uses second-order accurate numerical techniques [11]. It represents a large class of production applications at Los Alamos National Laboratory. It is a large-scale parallel code written in Fortran 90 and uses MPI for interprocessor communications. SAGE routinely runs on thousands of processors for months at a time.

The Parallel Ocean Program (POP) [12] is an ocean circulation model developed at Los Alamos National Labs that is capable of ocean simulations as well as coupled atmosphere, ice, and land climate simulations. Time integration is split into two parts: baroclinic and barotropic. In the baroclinic stage, the three-dimensional vertically-varying tendencies are integrated using a leapfrog scheme. The baroclinic stage consists of a preconditioned conjugate gradient solver which is used to solve for the two-dimensional surface pressure.

4

IV. EXPERIMENTAL RESULTS

A. Changing System Balance

To measure the impact that changing system balance has on system noise sensitivity, we ran each of the three applications described in the previous section with different available network bandwidths and 2.5% injected noise ranging from low-frequency/high-duration profiles (10Hz/2500μsec) to high-frequency/low-duration profiles (1000Hz/25μsec). We then measured the amount of *excess* slowdown that each application experienced after subtracting out the 2.5% injected noise – that is, the amount of additional noise that *accumulated* during the application run. Each data point represents the average of three runs, and we ran each application at the largest system size that an available data set supported and for which we were able to get an allocation – 2500 nodes in the case of POP, 3360 nodes in the case of SAGE, and 2048 nodes in the case of CTH.

Figure 2 shows how noise accumulates for each of POP, SAGE, and CTH with varying network bandwidth and 2.5% noise profiles. Shifting the balance of the system in favor of computation by reducing the amount of network bandwidth available reduces the performance impact of noise on SAGE and POP, though POP in particular is still significantly impacted by low-frequency/high-duration noise similar to that caused by scheduling a kernel daemon. CTH also accumulates less noise on an unbalanced system, but because it accumulates little noise to begin with (approximately 12% in the worst case), this is less significant.

Noise accumulates under varying system balances for each of these applications because of differences in the amount of computation and types of communication they perform. Figure 3 shows breakdowns of how POP and SAGE divide their time between computation and different communication primitives. POP, for example, spends the majority of time at scale in small `MPI_Allreduce` operations and relatively little time in computation. As a result, it still incurs substantial slowdown due to noise accumulation (more than 1000% on 2500 nodes) even when the system balance is tilted heavily in favor of computation. SAGE, on the other hand, can leverage an imbalanced system more effectively to reduce noise impact because of its larger computational demands. Finally, while noise has the least impact on CTH in any case, changes in system balance do significantly affect its performance, as CTH is network bandwidth limited [16].

B. Isolating Noise to a Subset of Nodes

To measure the impact of isolating noise-generating actions onto a subset of system nodes, we varied the percentage and location of noise-injecting nodes in the system and measured the application noise accumulation. Nodes injecting noise were generally chosen randomly using a Fisher-Yates permutation [5] to shuffle the list of MPI ranks of the job. We then choose the first N elements of the list as the ranks of noisy nodes. Each data point in the following graphs corresponds to an average of at least 5 data points (maximum of 6) with error bars shown.

1) Varying percentage of nodes injecting noise: Figure 4 shows the impact on noise accumulation for the three applications of isolating noise generation to a varying percentage of system nodes. In the case of POP, reducing noise generation to just 5-10% of the system nodes still results in substantial application slowdown. In particular, we note the slowdown for POP is *not* related purely to the number of nodes injecting noise; for example, POP is slowed down by more than 500% on 2500 nodes when only 250 nodes are injecting noise, but slows down only nominally when 100% of the nodes are noisy in the 256 node case.

In contrast, noise isolation appears to be a very successful strategy for SAGE (as others have found [17]). For example, isolating noise to roughly 10% of the system nodes reduces noise accumulation in SAGE by a factor of 3 on 2048 nodes. Finally, noise isolation appears to be largely irrelevant to CTH. Also, in contrast to POP and SAGE, the slowdowns for CTH (though smaller than POP and SAGE) appear to be relatively constant, independent of the number of noisy nodes.

2) Placement of noisy nodes: To study how changing the location of noise injection affected application performance, we used two different policies to place 125-128 noisy nodes into a system run: random and sequential. Random uses the Fisher-Yates shuffle mentioned above, while sequential places noise-injection nodes as the first 125–128 nodes in the system starting at rank zero.

As seen in figure 5, placement of noisy nodes in the system can have a substantial effect. Noisy nodes close to rank 0 result in less noise accumulation. We believe this is due to the collective communication primitives on the Sandia Red Storm system using tree-based algorithms, with the root of the tree at rank 0. As a result, placement of noisy nodes near the root limits the accumulation of noise when performing collectives, while randomly placing noisy nodes, including potentially at leaves in the collective tree, allows more noise to accumulate on average. To illustrate this difference, we use a `MPI_Allreduce` microbenchmark on 128 nodes with eight noisy nodes either distributed randomly or around rank zero. From Figure 6 we see that, at this
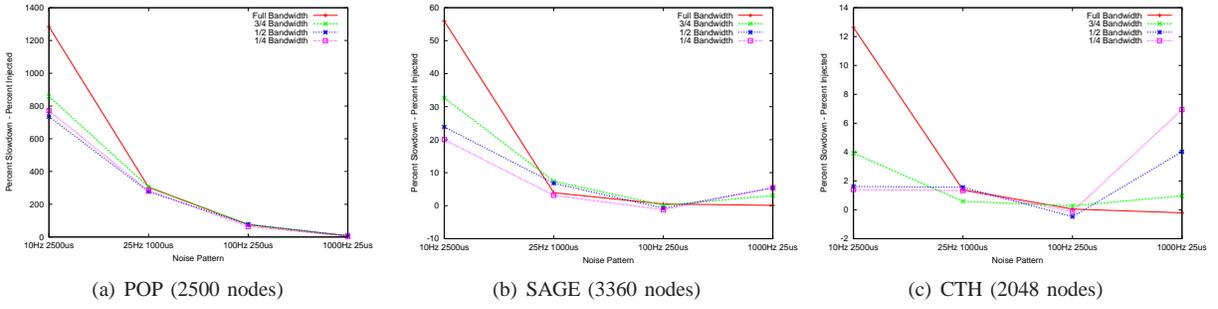
(a) POP (2500 nodes)　　　　(b) SAGE (3360 nodes)　　　　(c) CTH (2048 nodes)

Fig. 2.　Accumulation of noise in application runtime with varying network bandwidth and different 2.5% net injected CPU noise profiles.
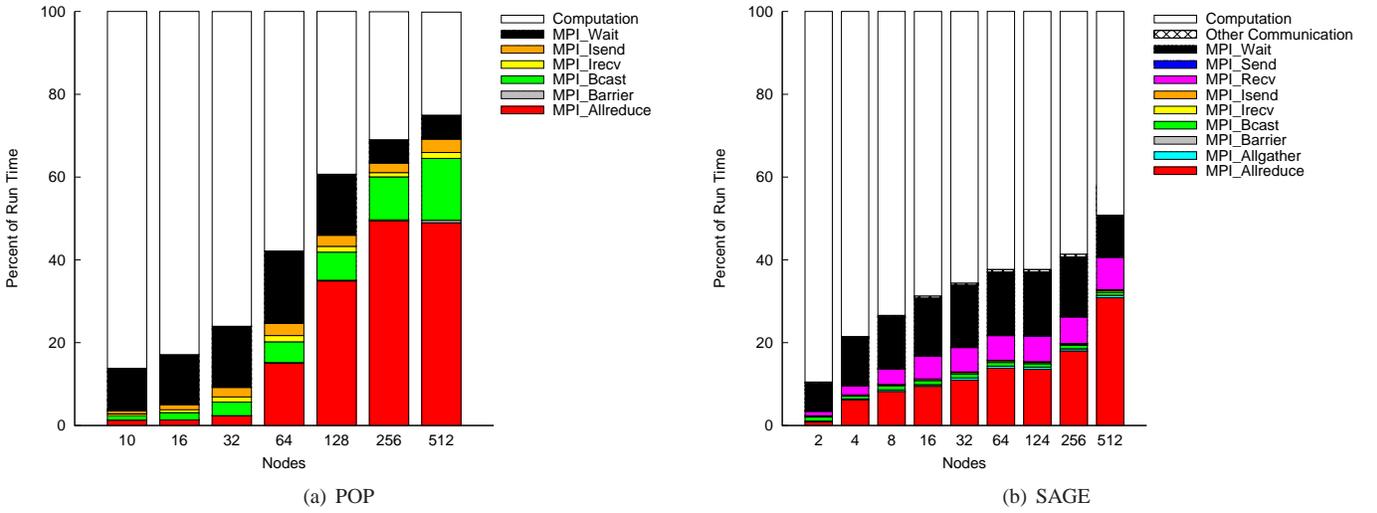

(a) POP　　　　(b) SAGE

Fig. 3.　Breakdown of application runtime between computation and various communication primitives with full network bandwidth and no injected noise.
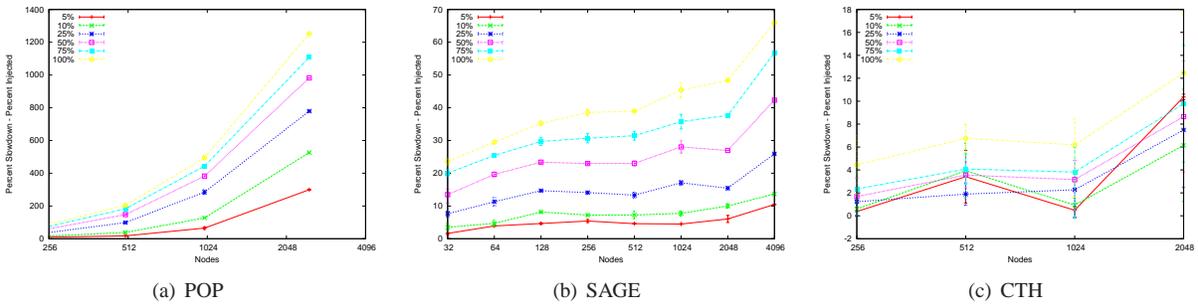

(a) POP　　　　(b) SAGE　　　　(c) CTH

Fig. 4.　Accumulation of noise in application runtime with varying percentages of nodes injecting 2.5% net CPU noise.
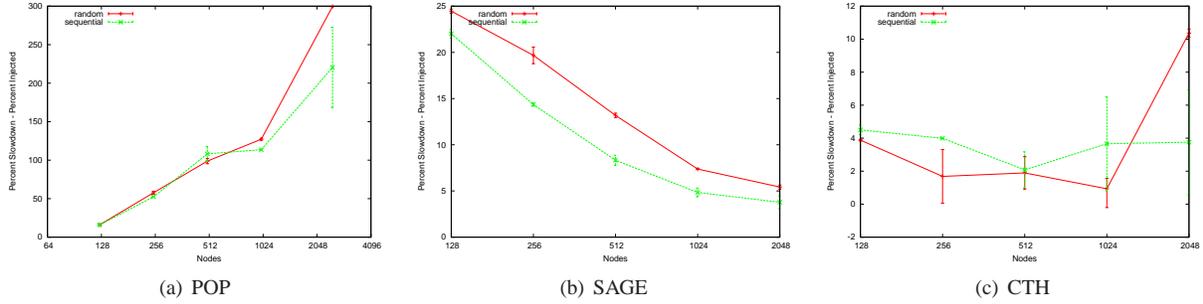
Fig. 5. Accumulation of noise in application runtime with nodes injecting noise arranged randomly or sequentially around rank 0. 125 nodes are injecting noise with POP, while 128 nodes are injecting noise with SAGE and CTH.
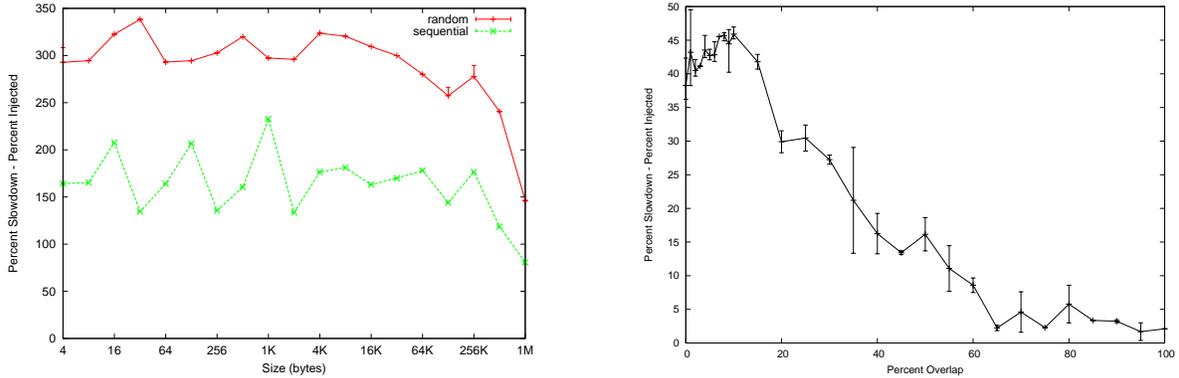

Fig. 6. Slowdown of `MPI_Allreduce` on 128 nodes with 6.25% of the nodes generating low-frequency, high-duration noise either distributed randomly throughout the application (random) or around and including rank zero (sequential)

Fig. 7. Slowdown of non-blocking Allreduce in bulk-synchronous microbenchmark as a function of overlap of communication with computation

scale, random distribution leads to a nearly factor of two slowdown over the noisy nodes distributed around rank zero.

C. Noise-Resistant Collectives

To examine the impact of noise-resistant collectives on system noise sensitivity, we studied how increasing the amount of overlap between application computation and collective communication affected the noise sensitivity of the synthetic bulk-synchronous benchmark described in Section III-D. We achieved this overlap using the non-blocking Allreduce collective also described above.

Figure 7 illustrates the slowdown of our non-blocking microbenchmark with a low-frequency, high duration noise signature, similar to that of a periodic kernel thread or daemon. For this test, we specified the bulk-synchronous interval to be that of what we measured for SAGE [7] and the slowdown is in comparison to a run with no noise. Each data point in the figure corresponds to an average of ten runs with error bars shown. From
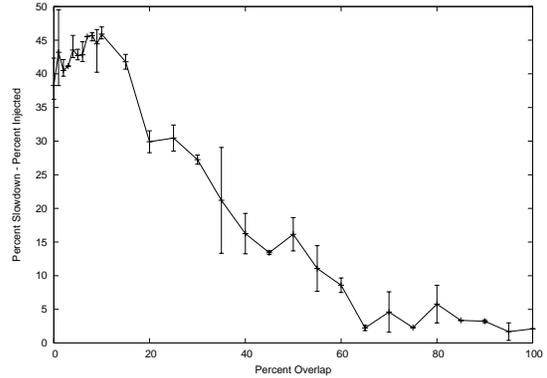
this figure, we see that in a noisy environment with sufficient overlap, the slowdown due to noise can be reduced to nearly zero. In this case in particular, a 2.5% noise signature could be completely absorbed with 62% overlap between application computation and collective communication.

V. RELATED WORK

Though the impact of OS noise on HEC systems has been studied for over 15 years [19], the seminal work of Petrini et al. [17] most recently raised the visibility of the impact of OS noise on application performance. This thorough study investigated performance issues from OS noise on a large-scale cluster built from commodity hardware components, running a commodity operating system, and running a cluster software environment designed for data center applications. While the findings of this paper from an OS perspective were largely well known, such as turning off unnecessary system daemons, the paper brought to light several important new findings relevant to OS noise. The authors developed a micro-

benchmark specifically for measuring OS noise on a parallel machine; such benchmarks were previously non-existent. Second, the paper demonstrates the inability of communication micro-benchmarks to accurately reflect and/or predict application performance. Also, the authors offered a conjecture that OS noise is most damaging when the application resonates with OS noise. Finally, this work showed the advantage of dedicating a portion of hardware resources to performing system tasks.

Beckman et al. [3] investigated the effect of user-level noise on an IBM BG/L system. This system runs a custom lightweight OS like Catamount that demonstrates very little noise. This system contains a number of hardware facilities which allow for collective operations to be performed in hardware and therefore not sensitive to CPU noise. In addition, this system has a balance shifted towards excess computation in comparison to other systems like the Cray XT series. In this paper the authors showed that a properly configured Linux kernel can have a noise signature similar to that of a lightweight kernel. Using a user-level injection mechanism built into the communication library and a series of micro-benchmarks, the authors showed that noise levels had to be very high in order to show any real impact. We believe this difference in noise impact with other studies is due to the difference in how noise is injected as well as the architectural differences of the BG/L system.

Nataraj et al. [15] used the KTAU toolkit to investigate the kernel activities of a general-purpose operating system. This toolkit instruments the Linux kernel to collect measurements from various kernel components including system calls, scheduling, interrupt handling, and network operations. The authors begin by showing the effectiveness of the KTAU toolkit for measuring the OS noise in Linux. In addition, they show how their toolkit can be used to track the accumulation and absorption of noise during the communication stages of an application. However, this work, unlike ours, only presents results from a 128-node development system that may or may not generalize to a massively parallel machine containing tens of thousands of nodes. More importantly, while their tool can be used to identify possible sources of noise, the authors do not relate the effects of noise to the performance of a large-scale application (e.g. the largest source of noise may not be the most harmful).

Recently, we examined the sensitivity of OS noise at scale for three real-world HEC applications using a kernel-level noise injection framework on a well balanced architecture [7]. In this paper we showed the importance of how noise is injected and the applica-

tion communication characteristics that impact noise absorption. For example, we show how the computation/communication ratios, collective communication sizes, and other characteristics of an application, relate to there ability to amplify or absorb noise. Finally, this paper discusses the implications of our findings on the design of new operating systems, middleware, and other system software laying out how system software tasks can be constructed as to minimize impact on HEC applications.

A number of studies have been conducted regarding the implementation and performance of non-blocking collective implementations. Hoefler et al. [8] describe the implementation of the non-blocking collective library currently being considered for inclusion of the MPI-3 standard. In this paper, the authors show that the performance benefit of non-blocking collectives is related to the ability of the system to overlap the communication cost of messages with computation of the application. In addition this work outlines the importance of intelligent network interfaces like the SeaStar on ensuring independent network progress for HEC systems.

Finally, Alam and Vetter [1] characterize the system balance requirements for GYRO, a Office of Science fusion simulation code, and the POP climate modeling code investigated here. In this work the authors measure the parallel efficiencies for these applications on a number of parallel systems: an SMP cluster, a shared-memory system, and a vector supercomputer. This work shows the sensitivity of POP to MPI latency and the bandwidth sensitivity of GYRO. Similarly, Pedretti et al. [16] investigate the sensitivity of HEC applications to link and injection bandwidth on the Cray Red Storm machine at Sandia National Laboratories. Using similar hardware methods employed in this paper, the authors show the sensitivity of CTH and PARTISN to link bandwidth and injection bandwidth (the bandwidth of the point-to-point HyperTransport link connecting a compute node's Opteron CPU to its SeaStar network interface) degradation.

VI. FUTURE WORK

There are several avenues of future work related to this study. First, we intend to analyze more applications in order to increase the understanding of application sensitivity to noise. While the set of applications in this study covers a range of important problems and scalable computational techniques, additional application experimentation would further increase the understanding of the relationship between OS noise and HEC hardware and system software design features. Obtaining access to large-scale applications, problem sets, appropriate

application scientist expertise, and dedicated system time to run these applications has proven challenging, but we believe that this approach is key to understanding the overall impact of OS noise, especially as future programming models may require additional system services with additional system demands.

We are also interested in analyzing how basic OS services, for example memory management, can influence the generation and impact of noise. We are exploring modifications to our noise framework that allow Catamount's memory management strategy to be more representative of a general-purpose OS like Linux. Specifically, we are implementing a non-contiguous memory page allocation scheme that better resembles the way Linux allocates and manages physical memory pages. There is evidence to suggest that these memory management strategies can significantly influence the scalability of certain HEC applications [18].

VII. Conclusion

In this paper, we showed how a number of important architectural and system software design features affect the impact of noise on HEC systems. Our results indicate that these system characteristics have a significant impact on the performance of applications at scale.

We used a previously-developed, kernel-based noise injection utility to explore several important aspects of OS interference. We showed that the impact of noise is not solely a property of the communication characteristics of an application. Using a hardware mechanism to degrade network bandwidth performance, we show that the relative peak compute-to-communication balance of the system is also important. This particular analysis helps to explain the disparity in observed results of the impact of noise on systems with disparate balance characteristics. Our results show that, in general, systems with excess compute cycles tend to be less sensitive to noise.

In addition, we explore whether isolating noise to only a subset of nodes can lessen performance degradation. We use our noise injection tool to impact only a subset of compute nodes rather than affecting all nodes equally. Results show that it takes a relatively small percentage of nodes – even as little as 5% – to have a significant impact on application performance. We also explore the distribution of the noisy nodes to determine whether placement makes any difference. Our results show that placement of noisy nodes can also have a substantial impact on application performance. If noise is generated on nodes whose MPI rank is closer to rank zero, the impact of noise is much less than if the noise is generated on a subset of ranks further from rank zero. We validate our hypothesis that this rank distribution effect is the result of tree-based collective operations where the ranks nearest the root are able to more easily absorb noise, while nodes furthest from the root are not.

Finally, we investigate the ability of non-blocking collective operations to mitigate the impact of OS noise. We implement a non-blocking Allreduce operation and a corresponding overlap benchmark. Combined with the noise injection utility, we are able to explore the amount of noise that a non-blocking collective operation could potentially absorb. Results show that, for a typical noise signature, a relatively modest amount of overlap between computation and communication is enough to nearly eliminate the impact of noise.

Together, these results increase our understanding of how and why OS noise impacts applications. Deeper knowledge about the important characteristics of noise-sensitive applications and key system or architectural features that can mitigate the negative impact of noise. This knowledge greatly enhances the ability to design future-generation platforms, system software, and applications.

References

[1] S. R. Alam and J. S. Vetter. An analysis of system balance requirements for scientific applications. In *ICPP '06: Proceedings of the 2006 International Conference on Parallel Processing*, pages 229–236, Washington, DC, USA, 2006. IEEE Computer Society.

[2] G. Almási, P. Heidelberger, C. J. Archer, X. Martorell, C. C. Erway, J. E. Moreira, B. Steinmacher-Burow, and Y. Zheng. Optimization of MPI collective communication on BlueGene/L systems. In *ICS '05: Proceedings of the 19th annual international conference on Supercomputing*, pages 253–262, New York, NY, USA, 2005. ACM.

[3] P. Beckman, K. Iskra, K. Yoshii, and S. Coghlan. The influence of operating systems on the performance of collective operations at extreme scale. In *IEEE Conference on Cluster Computing*, September 2006.

[4] R. Brightwell, T. Hudson, K. T. Pedretti, and K. D. Underwood. SeaStar Interconnect: Balanced bandwidth for scalable performance. *IEEE Micro*, 26(3):41–57, May/June 2006.

[5] R. Durstenfeld. Algorithm 235: Random permutation. *Commun. ACM*, 7(7):420, 1964.

[6] J. E.S. Hertel, R. Bell, M. Elrick, A. Farnsworth, G. Kerley, J. McGlaun, S. Petney, S. Silling, P. Taylor, and L. Yarrington. CTH: A Software Family for Multi-Dimensional Shock Physics Analysis. In *Proceedings of the 19th International Symposium on Shock Waves, held at Marseille, France*, pages 377–382, July 1993.

[7] K. B. Ferreira, R. Brightwell, and P. G. Bridges. Characterizing application sensitivity to OS interference using kernel-level noise injection. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing (Supercomputing'08)*, November 2008.

[8] T. Hoefler, A. Lumsdaine, and W. Rehm. Implementation and Performance Analysis of Non-Blocking Collective Operations for MPI. In *In proceedings of the 2007 International Conference on High Performance Computing, Networking, Storage and Analysis, SC07*. IEEE Computer Society/ACM, Nov. 2007.

[9] T. Jones, W. Tuel, L. Brenner, J. Fier, P. Caffrey, S. Dawson, R. Neely, R. Blackmore, B. Maskell, P. Tomlinson, and M. Roberts. Improving the scalability of parallel jobs by adding parallel awareness to the operating system. In *Proceedings of SC'03*, 2003.

[10] D. Katramatos, S. J. Chapin, P. Hillman, L. A. Fisk, and D. van Dresser. Cross-operating system process migration on a massively parallel processor. Technical Report CS-98-28, University of Virginia, 1998.

[11] D. J. Kerbyson, H. J. Alme, A. Hoisie, F. Petrini, H. J. Wasserman, and M. Gittings. Predictive performance and scalability modeling of a large-scale application. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing*, pages 37–48, Denver, CO, 2001. ACM Press.

[12] D. J. Kerbyson and P. W. Jones. A performance model of the parallel ocean program. *Int. J. High Perform. Comput. Appl.*, 19(3):261–276, 2005.

[13] P. D. V. Mann and U. Mittaly. Handling OS jitter on multicore multithreaded systems. In *IPDPS '09: Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing*, pages 1–12, Washington, DC, USA, 2009. IEEE Computer Society.

[14] J. Moreira, M. Brutman, J. Castanos, T. Gooding, T. Inglett, D. Lieber, P. McCarthy, M. Mundy, J. Parker, B. Wallenfelt, M. Giampapa, T. Engelsiepen, and R. Haskin. Designing a highly-scalable operating system: The Blue Gene/L story. In *Proceedings of the 2006 ACM/IEEE International Conference for High-Performance Computing, Networking, Storage, and Analysis (SC'06)*, Tampa, Florida, November 2006.

[15] A. Nataraj, A. Morris, A. D. Malony, M. Sottile, and P. Beckman. The ghost in the machine: Observing the effects of kernel operation on parallel application performance. In *Proceedings of SC'07*, 2007.

[16] K. T. Pedretti, C. Vaughan, K. S. Hemmert, and B. Barrett. Application sensitivity to link and injection bandwidth on a cray XT4 system. In *Proceedings of the 2008 Cray User Group Annual Technical Conference*, May 2008.

[17] F. Petrini, D. Kerbyson, and S. Pakin. The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of ASCI Q. In *Proceedings of SC'03*, Phoenix, AZ, 2003.

[18] B. V. Straalen, J. Shalf, T. Ligocki, N. Keen, and W.-S. Yan. Scalability challenges for massively parallel amr applications. In *Proceedings of the International Parallel and Distributed Processing Symposium*, May 2009.

[19] R. Zajcew, P. Roy, D. Black, C. Peak, P. Guedes, B. Kemp, J. LoVerso, M. Leibensperger, M. Barnett, F. Rabii, and D. Netterwala. An OSF/1 UNIX for Massively Parallel Multicomputers. In *Proceedings of the 1993 Winter USENIX Technical Conference*, pages 449–468, January 1993.