The Impact of Network Noise at Large-Scale Communication Performance

Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine
Open Systems Laboratory
Indiana University
Bloomington IN 47405, USA
{*htor,timoschn,lums*}@*cs.indiana.edu*

Abstract

The impact of operating system noise on the performance of large-scale applications is a growing concern and ameliorating the effects of OS noise is a subject of active research. A related problem is that of network noise, which arises from shared use of an interconnection network by parallel processes. To characterize the impact of network noise on parallel applications we conducted a series of simulations and experiments using a newly-developed benchmark. Experiment results show a decrease in the communication performance of a parallel reduction operation by a factor of two on 246 nodes. In addition, simulations show that influence of network noise grows with the system size. Although network noise is not as well-studied as OS noise, our results clearly show that it is an important factor that must be considered when running large-scale applications.

1. Introduction

The influence of external effects to the performance of large-scale parallel application has attracted recent interest [1], [2], [3], [4], [5], [6]. Even though such effects usually impose a relatively small overhead to applications when run at smaller scales, they can become problematic at larger scale. For example a single context switch every second is very unlikely to cause a measurable perturbation to a small-scale application run. However, it was shown before that such small periodic events can significantly perturb large-scale applications, if they *resonate* with synchronization (often caused by communication). Such effects can be multiplied by global (collective) communication operations.

Most existing studies focus on perturbations on the host side, i.e., *operating system (OS) noise*. Such perturbations are caused by resource-sharing between the application process and entities that belong to the computing platform (e.g., OS, daemons, monitoring processes). Host-side delays are caused by such things as hardware interrupts, translation lookaside buffer (TLB) misses, context switches or cache misses—all of which result from time-sharing the main CPU among different processes. Due to the serial nature and obvious source of such overheads, there are some obvious approaches to minimize their influence, such as

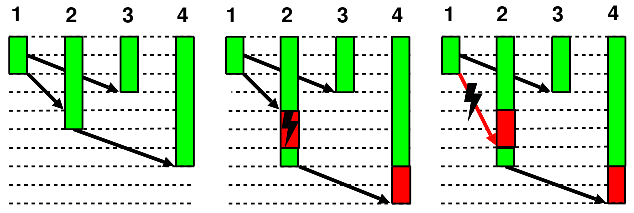

Figure 1. Different Sources of Noise

the use of low-noise operating systems specialized for high performance computing (HPC) (such as Catamount [7] or Blue Gene Linux [8]).

In addition to OS noise, *network noise*, can also affect parallel application performance. Just as processes sharing a computing resource can interfere with each other, parallel processes sharing an interconnection network can interfere with each other. The way that network noise manifests itself in an application, i.e., delay in effective message transmission, is also similar to OS noise. Figure 1 compares the influence of network noise and OS noise in a parallel application with four processes. The arrows represent communication, green areas show computation, and the red areas represent overhead caused due to OS or network noise. Time is indicated by horizontal dashed lines.

In the noiseless case every process does some computation for one time slice, then all processes exchange data in a tree based manner with rank 1 as the root. This process sends data to rank 2 and 3. All data transmissions have a latency of two time slices in our example. Between two consecutive transmissions there is always a gap of unit of time. This gap models the senders host overhead. In Figure 1(b) rank 2 can not progress and send data to rank 4 after it received the message from rank 1 because it is interrupted by OS noise for 2 time units. So both, rank 2 as well as rank 4 finish 2 slices later as in the noiseless case. Figure 1(c) shows an demonstrates network noise: the data transfer from rank 1 to rank 2 is slowed down by congestion in the network, because of this the latency doubles. As a result rank 2 and 4 receive their data two time slices later as in the noiseless case.

Other effects, such as the *resonance* mentioned may also amplify the effects network noise, but require complex interactions of different communications. Similar to OS noise, network noise can be absorbed in synchronization times or can accumulate in communication algorithms. Section 3 discusses absorption and accumulation of network noise in detail.

While OS noise can be modelled with statistical methods [1], [3] or signal processing methods [9], modelling network noise is much harder. Common sources of network noise are other applications that share the network, file I/O operations or monitoring activities. The network topology and the network routing plays also a very important role in this context and makes a precise prediction or modelling of the perturbation hard.

We designed a microbenchmark in order to assess the influence of other applications to specific communication patterns. Section 2 describes the benchmark and presents results for several systems. Section 3 describes a simulation methodology to assess the influence of network noise to existing large-scale networks. Section 4 describes a way to artificially generate future large-scale networks based on current established design principles and assess their properties with regards to network noise.

1.1. Related Work

Petrini et. al found in [6] that frequent short noise intervals together with a synchronizing collective operation such as Allreduce can cause a dramatic performance loss at large scale. In this particular case, the performance loss was caused by the resonance between the fine-grained OS noise and the synchronizing global communication.

Several studies, such as [1], [2], [3], [4], [5], [10] benchmark, model and quantify the impact of OS noise to parallel applications. However, those studies fully ignore the network as a source of additional noise.

Braccini et. al [11] examine the effect of network load on the bandwidth of point-to-point TCP/IP connections by inducing load on the data link layer in small Ethernet networks. In [12] demonstrated the sensibility of applications performance to bandwidth and latency at scale by changing the link speed of the used InfiniBand network.

Badia et. al [13] model the performance of parallel MPI applications based on MPI traces. The used network model considers a varying traffic function which influences the transmission speed. The model for collective operations assumes that all networking resources are busy during the whole transmission which does not apply to some collective algorithm, such as binary or binomial trees.

Sottile et. al [14] also model the performance of parallel applications based on MPI message traces. Sottile et al. recognize the effect of network noise and use a simple statistical model to assess the variation in latency and bandwidth. The applied model for collective communication, which assumes $log_2(P)$ message transmissions models the worst case. Our work extends this model to fully simulate collective algorithms and the influence of the interconnection topology of variable-sized networks. We did not choose the tracing approach in order to analyze communication algorithms at variable scale.

Mraz showed in [15] that operating system noise can cause delays for global communication patterns. He uses a ring pattern which represents the worst case in this spectrum. Mraz concludes that the observed delays result from either other processes or interrupt processing without considering variances in the network transmission.

In a previous work [16], we investigated the influence of static routing and congestion to large scale networks. This analysis only considered congestion in a given allocation without external influence or perturbations or dependencies between messages in collective operations. In this work, we extend the model to give a more accurate dependency-based prediction of the running time of collective operations. We also analyze the effects of perturbations from other applications in the network.

2. Design of a Microbenchmark

We describe a microbenchmark scheme to measure the influence of network noise to a parallel application. For this, we assume that other communications (either from other applications, I/O or monitoring) is the biggest source of network noise on parallel systems. The only way to prevent application noise on parallel systems is either to allocate the whole system or a separate partition which is not used by other jobs. While partitioning is simple for some network topologies such as three dimensional tori (allocate a partition in the network which is not crossed by other processes), it can be very tricky for topologies like Fat Trees that use a common “backplane”. Thus, we assume that it’s unlikely to get a separate, noise-less partition for our benchmark and that we have to accept the usual background network noise. The same applies to OS noise such that the benchmark has to be tolerant to OS noise too.

As an example, we investigate MPI collective operations which play a critical role for many large-scale applications. POP [17], CTH [18] and SAGE [19] rely on global allreductions which are, by definition, synchronization points. It has been shown [2] that those applications are heavily influenced by OS noise. Based on those results and the application study in [20], we investigate the collective operations MPI_Allreduce, MPI_Reduce and MPI_Bcast without the loss of generality. For our studies, we assume the communication of small data because large-scale applications commonly communicate very small number of elements. POP for example relies on a global eight-byte reduction operation.

Since we can not guarantee a noise-free system during the run, we choose a pro-active approach which generates noise on top of the "background noise". For this, we assume that the background noise follows a normal distribution during the run of the benchmark which enables us to consider the differences to our artificially generated noise (perturbation).

Our benchmark simulates an MPI application, represented only by its communication, and a background noise pattern. To do this, we split the full allocation (MPI_COMM_WORLD) into two independent communicators, the application communicator and the perturbation communicator. The ratio between the size of the application communicator and the perturbation communicator is called "perturbation ratio" in the following. The selection of MPI_COMM_WORLD ranks for each communicator is completely randomized. New communicators with random subsets of nodes from MPI_COMM_WORLD are generated for each benchmark.

The communicators are "warmed up" (cf. Gropp's comments on correct benchmarking [21]) before each use. The benchmark uses the collective benchmarking principles and the synchronization scheme proposed in [22]. Each benchmark run starts with a global synchronization, then all nodes in the perturbation communicator start to communicate with a random large-message perturbation pattern. Now, the independent application communicator benchmarks the time to perform a specific communication (in our case a collective MPI operation). Then, in a second step, the benchmark of the collective operation is repeated without the perturbing communication. This procedure is repeated multiple times with different communicators and perturbation ratios and analyzed statistically to show the influence of network noise. The benchmark is schematically shown in Figure 2.

2.1. Benchmark Results

We implemented our benchmark scheme in the open-source tool Netgauge [23] and ran several mid-scale tests on the CHiC cluster system. The CHiC system has 528 quad-core nodes connected with a full bisection bandwidth SDR InfiniBand Fat Tree network with 24 port switches. All benchmarks were done with Open MPI 1.2.8 and one process per node to keep the impact of OS noise minimal. We also performed runs with multiple processes per node, which are omitted due to space restrictions, that showed much higher perturbations. Figure 3 shows a detailed statistical analysis of the effect of a random perturbation communication to a variable size allreduction. We used a fixed allocation of 32 nodes for this benchmark and varied the number of nodes which took part in the allreduce operation from 2 to 32, as shown on the x axis. All the other nodes induced background noise in the network. The boxplot shows the distribution of single measurements for the perturbed and nonperturbed run. The sudden jump at 16

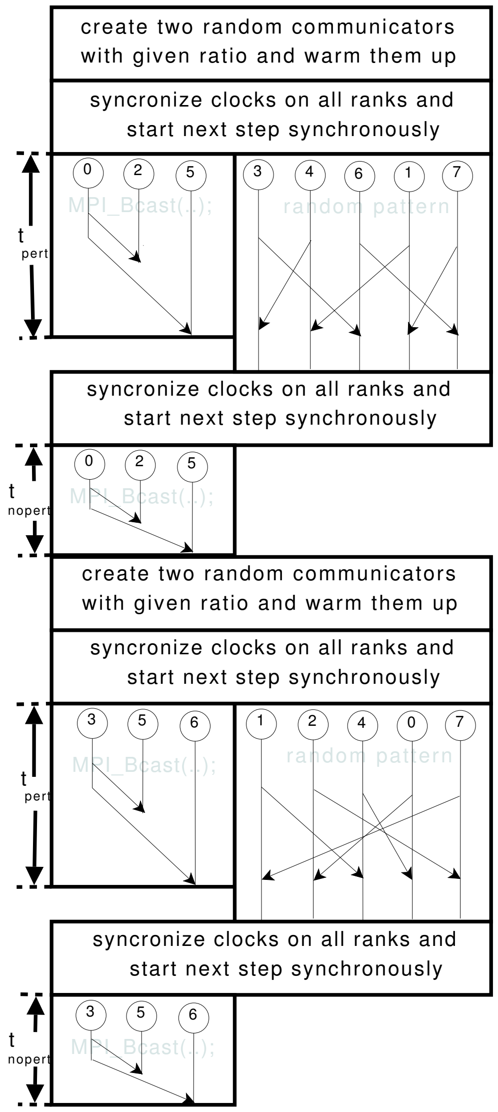
create two random communicators with given ratio and warm them up
syncronize clocks on all ranks and start next step synchronously
t pert
0 2 5
MPI_Bcast(...);
3 4 6 1 7
random pattern
syncronize clocks on all ranks and start next step synchronously
t nopert
0 2 5
MPI_Bcast(...);
create two random communicators with given ratio and warm them up
syncronize clocks on all ranks and start next step synchronously
t pert
3 5 6
MPI_Bcast(...);
1 2 4 0 7
random pattern
syncronize clocks on all ranks and start next step synchronously
t nopert
3 5 6
MPI_Bcast(...);

Figure 2. Benchmark Scheme

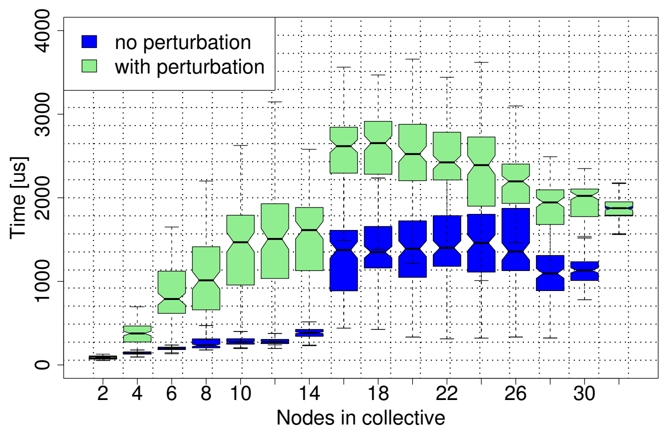

Figure 3. Network Noise in a 32 Node Allocation

nodes seems to indicate a protocol or algorithm change in the collective implementation. For this graph the benchmark shown in Figure 2 was run 128 times for each communicator size. The black bars in the middle of each box represent the sample median, while the upper and lower end of the box indicates the 25th and 75th percentile. The whiskers represent the total range of the sample. The notches in the middle of the boxes indicate if two samples show statistically significant difference—if the do not overlap the samples are different with a 95% confidence [24].

Figure 4 shows the relative slowdown caused by perturbation. The perturbation ratio is changed during the benchmark. We performed 128 measurements and used the average value to plot the data point. The points have a high variance due to other jobs on the system and the relatively small set of the random mapping. However, the results indicate a clear trend that is shown as the linear least squares fit plotted in the diagram. We see that the impact to

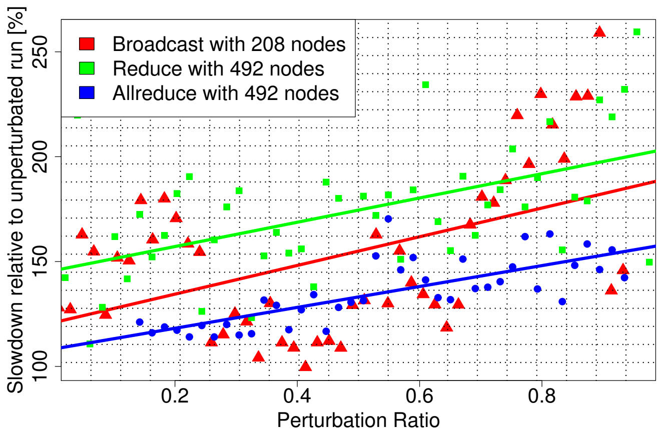

Figure 4. Network Noise with different Perturbation Ratios

the communication performance is significant even though the perturbation communication and the collective operation run in different non-overlapping communicators.

Figure 5 shows the scaling of MPI_Allreduce with a fixed perturbation ratio of 0.5. As expected, the impact of network noise grows with the number of communicating nodes.

The benchmarks show a perturbation of the communication even at a smaller scale. However, getting the necessary allocations to analyze large-scale networks is often not possible or too expensive. Thus, we designed a simulation scheme to model the effects of network noise to communication patterns at large-scale machines. We discuss our model and the simulation in the next section.

3. Simulating Network Noise

We propose a simple dependency-based simulation scheme to assess the influence of network noise to complex application communication patterns. Communication patterns usually consist of point-to-point messages and dependencies between them. For example a small-message broadcast operation is often implemented with a binomial tree. Such an algorithm usually consists of multiple rounds where each non-root and non-leaf node has to receive data and pass this data on to its children. A broadcast tree for 8 processes with 3 communication rounds is shown in Figure 6. This scheme allows the construction of a global

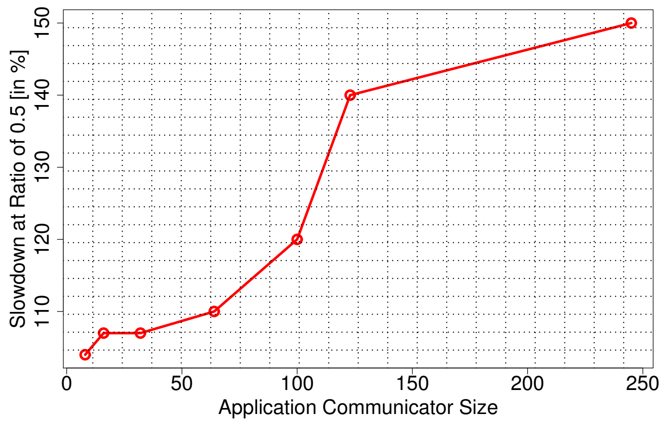

Figure 5. Slowdown with increasing Communicator Size and a fixed Perturbation Ratio of 0.5

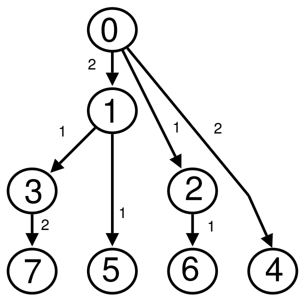

Figure 6. Binomial Tree Pattern

dependence graph in which the send operations of all nodes but the root process depend on a previous receive (from a previous communication round).

Our simulation considers an application communicator of size x and a perturbation communicator of size y. The communication in the application communicator is modeled as a collective operation that consists of multiple rounds. The perturbation communicator is modeled with random communication (each process picks a random peer to send data to, so that each process is receiving from exactly one peer) for each round of the application communicator. The simulation performs the routing of all messages (application and perturbation) through the network and counts the congestion of each physical link. Each edge in the dependence graph is then annotated with the maximum congestion along the corresponding logical link in the network simulation. Then, a breadth first search is started at every root node and the longest path in the dependence graph is reported as the time for this collective operation (we assume that the finishing time of the last process of the collective operation is significant). The simulation is performed twice and the result without perturbation is compared with the simulation result with perturbation. Figure 7 visualizes one run of the simulation of the third round in the eight-node binomial tree communication pattern on a 16 node fat tree built from 4x4 port crossbars. The rank-to-node mapping is $\{(0,3), (1,6), (2,5), (3,13), (4,7), (5,9), (6,0), (7,10)\}$ and all other hosts communicate randomly. The pattern itself is congestion-free

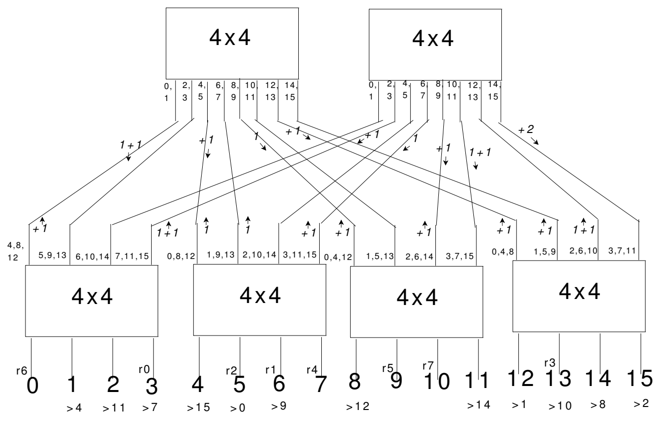

Figure 7. Simulation Example. The output ports of the switches are annotated with the target routes and the connecting cables are annotated with the number of virtual connections. The ranks in the tree are prefixed with "r". Perturbations are indicated with "+". Communication partners for each node are prefixed with ">".

for this particular mapping, i.e., congestion only happens due to perturbation. The tree in Figure 6 is already annotated with the congestion of all links in the tree rounds. The BFS search will find the longest path 0-1-3-7 with a total congestion of 5. The congestion of the unperturbed run (3 in this example) will be subtracted in the simulation.

We model the whole communication and all dependencies. Thus, our model also captures the effect of noise absorption and accumulation as described in [25], [26]. Both effects happen in our example, the congestion in round 3 between rank 0 and 4 is absorbed (in the BFS) while the congestion between rank 0 and 1 in round 1 and node 3 and 7 in round 3 accumulate.

3.1. Simulating Real-World Installations

In order to assess the influence of network noise to large-scale installations, we used InfiniBand network maps (generated from `ibnetdiscover` and `ibdiagnet` output) of large real-world systems. We investigate several several large-scale cluster systems. The "Thunderbird" (TBird) system is the largest installation with 4391 nodes. The second largest system, the "Ranger" system at the Texas Advanced Computing Center is connected with two 3456 port Sun Magnum switches and had a total of 3908 active endpoints during the query. The third largest system in our simulation, the Atlas system at the Lawrence Livermore National Lab, had 1142 nodes when the network structure was queried. The "CHiC" system at the Technical University of Chemnitz had 566 endpoints during the query and "Odin" at Indiana University has 128 nodes. All networks are based on Fat Tree topologies [27] with 24 port crossbars. Thunderbird is designed with half bisection bandwidth and all other systems should deliver full bisection bandwidth. We showed in [16] that the *effective bisection bandwidth* of the systems is significantly lower with 40.6%, 57.6% and 55.6% for Thunderbird, Ranger and Atlas. We used our network simulator [28] for those simulations, as well as for the simulations in this paper.

Figure 8 shows the average results of 1000 simulation runs of the tree pattern, which is typically used for small message reductions and broadcasts, for the analyzed systems. We see clearly that the effect of network noise increases

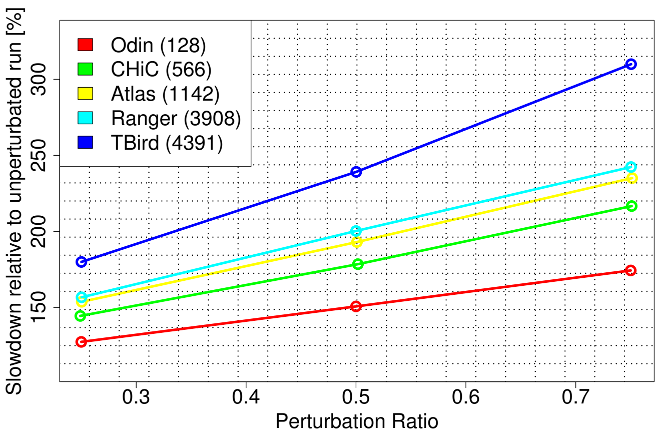

Figure 8. Simulated Networks with Tree Pattern

significantly with the network size and with the perturbation ratio. Our simulations also exhibit similar scaling behavior as the benchmark results in Section 2.1. However, different architectural properties of the real-world networks (e.g., CHiC is not a pure Fat Tree or TBird has only half bisection bandwidth) prohibits general statements about the scaling with network size. To be able to make such statements, we generate and analyze Fat Tree networks of different sizes and similar properties (full bisection bandwidth) in the next section.

4. Large-Scale Fat Tree Networks

In order to simulate future large-scale networks and evaluate the scaling with the system size, we decided to extend our study to large artificially generated Fat Tree networks. We used the recursive method described in [29] to generate extended generalized Fat Trees (XGFTs). An extended generalized fat tree $\mathrm{XGFT}(h, m_1, ..., m_h, w_1, ...w_h)$ of height h is generated in h recursion steps. In each step s, the $\mathrm{XGFT}(s, m_1, ..., m_h, w_1, ...w_h)$ is built by m_s copies of $\mathrm{XGFT}(s-1, m_1, ..., m_h, w_1, ...w_h)$ and $w_1 \cdots w_s$ additional new top-level nodes. Exact construction and wiring rules can be found in [29] and are omitted due to space restrictions.

All generated trees are built from 24-port switches and are designed to have full bisection bandwidth. The generated Fat Tree topologies are shown in Table 1. Figure 9 shows

Layout	# Endpoints	# Switches
xgft(2,12,6)	144	18
xgft(2,24,12)	288	36
xgft(3,12,8,12,4)	1152	240
xgft(3,12,16,12,8)	2304	480
xgft(3,12,24,12,12)	3456	720
xgft(4,12,12,6,12,12,3)	10368	3024
xgft(4,12,12,12,12,12,6)	20736	6048

Table 1. The simulated Fat Tree networks.

an $\mathrm{XGFT}(2, 12, 6)$ with 12 leaf switches and 144 ports. We used the Fat Tree optimized routing of OpenSM [30]

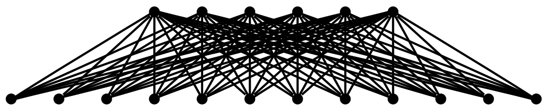

Figure 9. An $\mathrm{XGFT}(2, 12, 6)$ network

together with `ibsim` to generate realistic routes for the networks.

4.1. Simulation Results

We analyze the influence of network noise with growing network sizes and a fixed perturbation ratio of 0.5. Figure 10 shows the simulated influence of network noise to application runs on half of the nodes of Fat Tree networks while the other half communicates randomly. Our simulation results resemble the trend in the benchmark results from Section 2.1—the CHiC simulation in this graph predicts the same behaviour as the linear fitting of the measurement results for MPI_Reduce in Figure 4.

5. Conclusions and Future Work

We showed that network noise can have a significant impact on the performance of large-scale parallel applications.

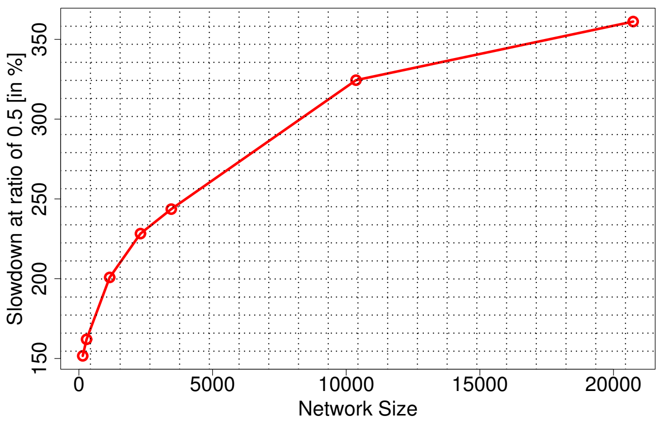

Figure 10. Slowdown with increasing Communicator Size and a fixed Perturbation Ratio of 0.5

Thus, network noise should be considered in addition to OS noise when analyzing large-scale application runs. The main factors that influence network noise are the network type and routing scheme and the node allocation policy. Our random allocation strategy resembles the average case for all mappings. Based on our results, we argue that batch systems for large-scale machines must be aware of these issues and allocate intelligently in order to minimize network noise.

Interesting future directions are the analysis of the effects of network noise on different topologies and mappings of real-world machines and applications. This directly leads to the idea to design noise-minimizing topology-dependent communication algorithms which could be used in collective operations. We also plan to improve the simulation accuracy, which currently bases in distinct communication rounds to a more precise (and more resource-intensive) LogGP-based model.

Acknowledgments

Thanks to Adam Moody and Ira Weiny (LLNL) who provided the Atlas system topology, Christopher Maestas (Sandia) who provided the input file for the Thunderbird cluster, and Len Wisniewski (Sun) and the TACC who provided the Ranger input. The authors also want to thank Frank Mietke (TUC) for supporting work with the CHiC cluster system. This work was supported by the Department of Energy project FASTOS II (LAB 07-23), a grant from the Lilly Endowment and a gift the Silicon Valley Community Foundation on behalf of the Cisco Collaborative Research Initiative.

References

[1] S. Agarwal, R. Garg, and N. Vishnoi, "The Impact of Noise on the Scaling of Collectives: A Theoretical Approach," in *12th*

Annual IEEE International Conference on High Performance Computing, Dec. 2005.

[2] K. B. Ferreira, P. Bridges, and R. Brightwell, "Characterizing application sensitivity to OS interference using kernel-level noise injection," in *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. Piscataway, NJ, USA: IEEE Press, 2008, pp. 1–12.

[3] R. Garg and P. De, "Impact of Noise on Scaling of Collectives: An Empirical Evaluation," in *HiPC 2006, 13th International Conference*, ser. LNCS, Y. Robert *et al.*, Eds., vol. 4297. Springer, 2006, pp. 460–471.

[4] K. Iskra, P. Beckman, K. Yoshii, and S. Coghlan, "The Influence of Operating Systems on the Performance of Collective Operations at Extreme Scale." in *Proceedings of Cluster Computing IEEE International Conference*, 2006.

[5] A. Nataraj *et al.*, "The ghost in the machine: observing the effects of kernel operation on parallel application performance," in *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*. New York, NY, USA: ACM, 2007, pp. 1–12.

[6] F. Petrini, D. J. 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 the ACM/IEEE SC2003 Conference*. ACM, 2003, p. 55.

[7] S. M. Kelly and R. Brightwell, "Software architecture of the light weight kernel, Catamount," in *Cray User Group Annual Technical Conference*, May 2005.

[8] J. E. Moreira *et al.*, "Blue Gene/L programming and operating environment." *IBM Journal of Research and Development*, vol. 49, no. 2-3, pp. 367–376, 2005.

[9] M. J. Sottile, "A measurement and simulation methodology for parallel computing performance studies," Ph.D. dissertation, Albuquerque, NM, USA, 2006.

[10] M. Sottile and R. Minnich, "Analysis of microbenchmarks for performance tuning of clusters," in *CLUSTER '04: Proceedings of the 2004 IEEE International Conference on Cluster Computing*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 371–377.

[11] A. Braccini, A. Del Bimbo, and E. Vicario, "Interprocess communication dependency on network load," *Software Engineering, IEEE Transactions on*, vol. 17, no. 4, pp. 357–369, 1991.

[12] D. J. Kerbyson, "A Look at Application Performance Sensitivity to the Bandwidth and Latency of Infiniband Networks," in *in Proc. of Workshop on Communication Architectures for Clusters (CAC), IEEE/ACM Int. Parallel and Distibuted Processing Symposium (IPDPS)*, Rhodes, Greece, March 2006.

[13] R. M. Badia, J. Labarta, and J. Gimenez, "DIMEMAS: Predicting MPI applications behavior in Grid environments." in *Workshop on Grid Applications and Programming Tools (GGF '03)*, 2003.

[14] M. J. Sottile, V. P. Chandu, and D. A. Bader, "Performance analysis of parallel programs via message-passing graph traversal," in *20th International Parallel and Distributed Processing Symposium, Proceedings, Rhodes Island, Greece*. IEEE, 2006.

[15] R. Mraz, "Reducing the variance of point to point transfers in the ibm 9076 parallel computer," in *Supercomputing '94: Proceedings of the 1994 ACM/IEEE conference on Supercomputing*. New York, NY, USA: ACM, 1994, pp. 620–629.

[16] T. Hoefler, T. Schneider, and A. Lumsdaine, "Multistage Switches are not Crossbars: Effects of Static Routing in High-Performance Networks," in *Proceedings of the IEEE International Conference on Cluster Computing*. IEEE Computer Society, Oct. 2008.

[17] K. Bryan, "A numerical method for the study of the circulation of the world ocean," *J. Comput. Phys.*, vol. 135, no. 2, pp. 154–169, 1997.

[18] E. S. Hertel *et al.*, "CTH: A Software Family for Multi-Dimensional Shock Physics Analysis," in *in Proceedings of the 19th International Symposium on Shock Waves, held at*, 1993, pp. 377–382.

[19] D. J. Kerbyson *et al.*, "Predictive performance and scalability modeling of a large-scale application," in *Supercomputing '01: Proceedings of the ACM/IEEE conference on Supercomputing*. New York, NY, USA: ACM, 2001, pp. 37–37.

[20] R. Rabenseifner, "Automatic MPI Counter Profiling," in *Proceedings of 42nd CUG Conference*, 2000.

[21] W. Gropp and E. L. Lusk, "Reproducible Measurements of MPI Performance Characteristics," in *Proceedings of the 6th European PVM/MPI Users' Group Meeting*. London, UK: Springer-Verlag, 1999, pp. 11–18.

[22] T. Hoefler, T. Schneider, and A. Lumsdaine, "Accurately Measuring Collective Operations at Massive Scale," in *Proceedings of the 22nd IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, Apr. 2008.

[23] T. Hoefler, T. Mehlan, A. Lumsdaine, and W. Rehm, "Netgauge: A Network Performance Measurement Framework," in *High Performance Computing and Communications, HPCC 2007, Houston, USA, Proceedings*, vol. 4782. Springer, Sep. 2007, pp. 659–671.

[24] J. Emerson and H. Strenio, "Box-plots and batch comparison. Understanding Robust and Exploratory Data Analysis," 1983.

[25] A. D. Malony and S. S. Shende, "Overhead compensation in performance profiling," in *In Proceedings of the European Conference on Parallel Processing (Euro-Par)*. Springer-Verlag, 2004, pp. 119–132.

[26] F. Wolf, A. Malony, S. Shende, and A. Morris, "Trace-Based Parallel Performance Overhead Compensation," in *In Proc. of the International Conference on High Performance Computing and Communications*, September 2005.

[27] C. E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE Trans. Comput.*, vol. 34, no. 10, pp. 892–901, 1985.

[28] T. Schneider and T. Hoefler, "ORCS: An Oblivious Routing Congestion Simulator," Indiana University, Computer Science Department, Tech. Rep., February 2009.

[29] S. R. Öhring, M. Ibel, S. K. Das, and M. J. Kumar, "On generalized fat trees," in *IPPS '95: Proceedings of the 9th International Symposium on Parallel Processing*. Washington, DC, USA: IEEE Computer Society, 1995, p. 37.

[30] E. Zahavi, G. Johnson, D. J. Kerbyson, and M. Lang, "Optimized InfiniBand Fat-tree Routing for Shift All-To-All Communication Patterns," in *Proceedings of the International Supercomputing Conference 2007 (ISC07)*, Dresden, Germany.