

# The Impact of Noise on the Scaling of Collectives: A Theoretical Approach

## [Extended Abstract]

Saurabh Agarwal<sup>1</sup>, Rahul Garg<sup>1</sup>, and Nisheeth K. Vishnoi<sup>1</sup>

IBM India Research Lab,  
Block-1, IIT Delhi, Hauz Khas, New Delhi, India 110016.  
{saurabh.agarwal,grahul,nvishnoi}@in.ibm.com

**Abstract.** The performance of parallel applications running on large clusters is known to degrade due to the interference of kernel and daemon activities on individual nodes, often referred to as *noise*. In this paper, we focus on an important class of parallel applications, which repeatedly perform computation, followed by a collective operation such as a barrier. We model this theoretically and demonstrate, in a rigorous way, the effect of noise on the scalability of such applications. We study three natural and important classes of noise distributions: The exponential distribution, the heavy-tailed distribution, and the Bernoulli distribution. We show that the systems scale well in the presence of exponential noise, but the performance goes down drastically in the presence of heavy-tailed or Bernoulli noise.

## 1 Introduction

**Motivation.** It is well known that many parallel applications do not scale well on large high-performance computing systems [1–3]. The per-node performance degradation is more pronounced in systems with more than 1K nodes, running a multi-tasking operating system such as Unix. In order to build high-performance computing systems that are capable of very high and sustained performance, it is important to understand the reasons for such performance degradation.

It is increasingly becoming evident that one of the main causes of performance degradation is the *noise* in the system; in the form of daemons and interrupts, see [1, 2].

A detailed study of the *noise* and its impact on performance was done by Petrini *et al.* [3] on the 8192 processor ASCI Q machine. It was observed that the overheads due to noise were mostly in the range 0.5% to 2.5% (see Figure 9 in [3]). However, this noise had a large impact on the system performance. By reducing the *intensity* of noise in the system it was possible to get a factor of 13 improvement in the performance of a micro-benchmark that repeatedly calls *barrier* with no intervening computation. Similarly, Kramer and Ryan [4] concluded that the performance variability in EP (of NAS parallel benchmarks) was due to the noise in systems.

**Our Contribution.** The main contribution of this paper is to initiate the study of the impact of noise on the scaling of parallel applications in a *formal manner*. We focus on a particularly important class of parallel applications which often arise in scientific computations. Here, typically, each node in the cluster is repetitively involved in a computation stage, followed by a collective operation; such as a barrier computation. We model this theoretically and demonstrate the effect of noise on the performance of such parallel applications. We study three natural and important classes of noise distributions: The exponential distribution, the heavy-tailed distribution, and the Bernoulli distribution. We show that the systems scale well in the presence of an exponential noise, but their performance goes down drastically in the presence of a heavy-tailed or a Bernoulli noise. Though our model is very simple, it is powerful enough to predict the effect of noise on scaling. We believe that this study will also be extremely useful in identifying and improving bottlenecks in the scalability of systems in a more systematic way, for instance, by designing scheduling policies, which take into account the nature of the noise, to improve the overall system performance. To the best of our knowledge, this is the *first attempt* to explain the impact of noise with a mathematical model.

**Related Work.** One way to reduce the impact of noise on scalability is to reduce the intensity of noise itself. This can be done by removing several system daemons, dedicating a spare processor to absorb the noise, and decreasing the frequency of daemons [3].

Another approach is to *synchronize* the noise across the nodes of the system. This may be done by either periodically adjusting the scheduling priorities of the processes, or by changing the scheduler in the kernel, see [5–9].

Though these methods have resulted in a reduced impact of noise on the performance of the respective systems, a general solution is more desirable both with regards to scalability and applicability. Our work provides a structured approach to understand the impact of noise on the overall system performance. Using the insights from our results, it might be possible to further enhance all of the above approaches, thereby advancing the frontier of scalability and yielding better resource utilization in the present high-performance computing systems.

**Organization.** Section 2 presents the theoretical model of a typical scientific parallel application with noise. Therein, we also justify the assumptions about the model. In Section 3, we analyze the proposed model and present the results obtained when the noise is distributed according to the exponential, heavy-tailed or Bernoulli distribution. Due to space limitations, the proofs of the theorems in Section 3 only appear in the full version of this paper [10]. Therein, we also discuss our results in detail.

## 2 The Model

In this section we describe a general model that captures the case of a compute intensive program with periodic synchronization. We assume that the program has

perfectly balanced load and it carries out minimal I/O and message exchanges. However, it carries out periodic synchronization using a collective operation. A footprint of such a program is typically present in many parallel applications, in particular in those which involve scientific computations.

## 2.1 Modeling the Computation

Consider a parallel program with  $N$  threads running on a system which has  $N$  processors. We assume, for simplicity of analysis, that  $N = 2^k - 1$  for some positive integer  $k$ .

**The communication and the three stages.** We assume that the barrier is implemented using message passing along a complete binary tree. A thread is associated to each node of the binary tree. There is a special node called the *root* which initiates the *post-barrier* stage and the *pre-barrier* stage ends at it. In the post-barrier stage, the root thread starts by sending a message to both its children to start with the *compute* stage. Whenever this message reaches a thread, it forwards the message to both its children in the tree (unless the node is a leaf) and starts the computation assigned to it. After finishing its computation, a leaf node sends a message to its parent indicating the end of its computation stage. This starts the pre-barrier stage. The parent, after finishing its computation and receiving the message from both its children, sends the message to its parent indicating the end of computation at every node in its subtree. This stage ends when the root finishes its computation and receives a message from both its children indicating the same. An iteration of the loop would, thus, consist of a compute stage, followed by a pre-barrier and a post barrier stage. For simplicity, we assume that each message transmission between a parent and a child node takes time  $\tau$ . Again, for simplicity of analysis, we consider a *phase* which consists of a sequence of a post-barrier, a compute and a pre-barrier stage. The program consists of  $M$  such phases. Now, we model various aspects of one such phase.

**A Phase.** Let  $t_{ij}^s$  represent the time instant when the  $i$ -th thread begins the computation stage in the  $j$ -th phase. Let  $t_{ij}^f$  represent the time instant when the  $i$ -th process ends the computation stage in the  $j$ -th phase. Let  $W_{ij}$  represent the amount of work (say the number of operations) carried out by thread  $i$  in the compute stage of the  $j$ -th phase. If the system is noiseless, the time required by processor  $i$  to finish work  $W_{ij}$  in its  $j$ -th phase will be a constant, say  $w_{ij}$ , which typically depends on the characteristics of the processor, such as clock frequency, architectural parameters, and the state of the node (such as cache contents) just before the  $j$ -th phase is entered. Therefore,  $t_{ij}^f - t_{ij}^s = w_{ij}$ . Due to the presence of system level daemons that get scheduled arbitrarily, the wall-clock time taken by processor  $i$  to finish the work  $W_{ij}$  is typically not a constant. There will be a variable component that represents the time consumed to service the daemons and other asynchronous events. We capture this by a

random variable  $\delta_{ij}$ . More precisely,  $t_{ij}^f - t_{ij}^s = w_{ij} + \delta_{ij}$ , where  $\delta_{ij}$  is a random variable that captures the overheads incurred by processor  $i$  in servicing the daemons and other asynchronous events during the  $j$ -th phase. Note that  $\delta_{ij}$  also includes the context switching overheads, as well as, the time required to handle additional cache or TLB misses that arise due to cache pollution by background processes. The mean value of  $\delta_{ij}$  depends on the time taken to do work  $W_{ij}$  and the system load on processor  $i$  during the  $j$ -th phase. Let  $f_{ij} \in [0, 1]$  be the fraction representing the system overhead for the processor. We may write the wall-clock time taken by processor  $i$  for the compute stage of the  $j$ -th phase as  $t_{ij}^f - t_{ij}^s = w_{ij} \left(1 + \frac{f_{ij}}{1-f_{ij}} \eta_{ij}\right)$ , where  $\eta_{ij}$  is the normalization of  $\delta_{ij}$  such that  $E[\eta_{ij}] = 1$ .

## 2.2 The Assumptions and Justifications

In this section we state and justify the assumptions we make about the model. The underlying principle in making the assumptions is to present an *ideal* model which captures the impact of the noise on typical parallel programs for scientific applications, and which is at the same time, susceptible to a rigorous theoretical analysis. We show, in a formal manner, that even in this ideal setting, the nature of noise may impact the system performance considerably.

**Balanced Load:**  $W_{ij} = W$  for all  $i, j$ . Application programs try to divide the load equally among its threads. Best performance is obtained when the load across every thread in a compute phase is equal (i.e.  $W_{ij} = W_j$  for all  $i$ ).

**Identical Processors:**  $w_{ij} = w$  for all  $i, j$ . If the processors are heterogeneous, the performance of the parallel application will be dictated by the performance of the slowest processor in the system. Best performance is obtained (with perfectly balanced load) when the processors are identical<sup>1</sup>. In addition to this, we make two more assumptions: (1) The application starts with all its threads in identical states. (2) The time taken by a computation does not depend on the input data. Together, these assumptions imply that the time taken by a compute phase is same across all the processors. The second assumption will not be true in general, because, due to cache effects, the time taken to carry out a set of operations also depends on the order in which the operations are carried out. However, it can be verified that this is the most optimistic assumption that will give the best program performance.

**Stationary and Balanced Overheads:**  $f_{ij} = f$  for all  $i, j$ . In typical HPC systems, the processors are allocated to an application for the lifetime of the application. Running any other application on the node is avoided. Thus, all the interference is due to the background processes or daemons. The amount of daemon activity is not expected to change over time. Thus we may assume

---

<sup>1</sup> We do not consider programs running on heterogeneous clusters that distribute the load across multiple nodes depending on the relative speed of the nodes.

$f_{ij} = f_{i'j}$ , for all  $i, j$  and  $j'$ . The daemons and overheads may be classified into *intrinsic* and *extrinsic* processes. The intrinsic processes run on every node and carry out book-keeping activities for the node. The system overhead due to intrinsic processes is expected to remain the same across all the nodes. However, the overheads of extrinsic processes are expected to vary across nodes. A detailed analysis may be carried out along the same lines while taking into account the activities of the extrinsic processes as well. Therefore, we assume  $f_{ij} = f_{i'j}$ , for all  $i, i'$  and  $j$ .

**Identical Noise:**  $\eta_{ij} \sim \eta$  for all  $i, j$ . Due to homogeneity of nodes and the fact that we choose to ignore the effect of extrinsic processes, we may assume that the nature of noise associated to the intrinsic processes is the same across the nodes and phases.

**Spatial Independence:**  $\{\eta_{ij} : i \in [1 \dots N]\}$  are independent for each  $j$ .

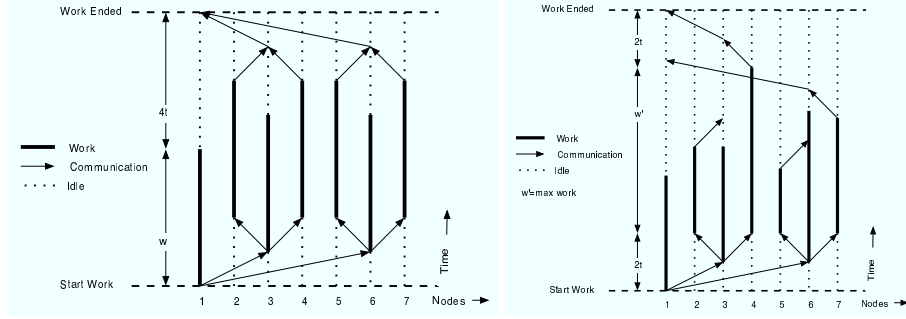
This assumption is the key to all of our results. In a typical cluster environment, there is no co-ordinated scheduling policy to synchronize processes across different nodes. Some HPC systems may deploy different scheduling policies to alleviate the daemon problems [7–9], as discussed earlier in Section 1. However, our analysis is restricted to systems that do not employ a co-ordinated scheduling policy. Note that we do not assume the random variables  $\{\eta_{ij} : j \in [1 \dots M]\}$  to be independent. In fact, many of the daemons are periodic, and we do expect complex correlation pattern between these random variables. In general, the nature of noise  $\eta_{ij}$  may depend on the quantum of work  $w_{ij}$  carried out in the phase. To analyze this, the compute phases may be grouped into quanta of work  $w_j$  and the same analysis may be independently carried for each quantum (with its associated noise). Due to linearity of expectation, the expected run time for each quantum of work can be added up to give the expected running time of the application.

### 3 Analysis

**The Ideal Noiseless Case.** Figure 1 illustrates the sequence of events in the ideal noiseless case. It is clear that in this case the time taken by each thread in a phase is  $w + 2\tau(k - 1)$ , where  $N = 2^k - 1$ . (The figure is for the case when  $k = 3$ .) In terms of  $N$ , this is  $w + 2\tau(\log(N + 1) - 1)$ . This will be the benchmark performance we will use for comparison with the noisy case.

**The Ideal Noisy Case.** Now, we no longer assume that  $f_{ij}$  are 0. We refer to this as the *ideal noisy* case. In this case,  $t_{ij}^f - t_{ij}^s$  is randomly distributed. An example of this scenario is presented in Figure 1. The post-barrier phase of the communication is still the same as in the case of the ideal noiseless case.

Let  $a_i$  denote the time it takes the message to reach thread  $i$  from the root in the post-barrier stage. Further, let  $b_i$  denote the time it takes the message from thread  $i$  to reach the root in the pre-barrier stage. The time taken to



**Fig. 1.** An ideal noiseless (left) and noisy (right) barrier computation cycle

complete the  $j$ -th phase then is at-least  $\max_{i=1}^N (a_i + t_{ij}^f - t_{ij}^s + b_i)$ . Notice that since the pre and post-barrier stages are done via communicating through a binary tree, for the leaves of this binary tree, which are  $2^{k-1} = \frac{N+1}{2}$  in number,  $a_i = b_i = \tau(k-1) = \tau(\log(N+1) - 2)$ . Let us just restrict our attention to these leaf threads. Since the noise is independent across the threads, the maximum of  $a_i + b_i + w_{ij}$ , for  $i$  restricted to these threads is a lower bound to the time taken to complete the  $j$ -th phase. Let  $Y_\eta^r$  denote the maximum of  $r$  random variables which are independent and identically distributed according to  $\eta$ . Hence, the expectation of  $t_{ij}^f - t_{ij}^s$  is at-least  $2\tau(\log(N+1) - 2) + w \left(1 + \frac{f}{1-f} \mathbb{E} \left[ Y_\eta^{\frac{N+1}{2}} \right] \right)$ . Therefore, we have the following theorem.

**Theorem 1.** *The expected time taken per phase is bounded by*

$$\begin{aligned} \text{LowerBound} &: w \left( 1 + \frac{f}{1-f} \mathbb{E} \left[ Y_\eta^{(N+1)/2} \right] \right) + 2\tau(\log(N+1) - 2) \\ \text{UpperBound} &: w \left( 1 + \frac{f}{1-f} \mathbb{E} \left[ Y_\eta^N \right] \right) + 2\tau(\log(N+1) - 1) \end{aligned}$$

We call the term  $2\tau(\log(N+1) - 2)$  in the above expression as the *latency component* which is an indication of time spent in barrier due to the communication latency. The term  $w \frac{f}{1-f} \mathbb{E} \left[ Y_\eta^{(N+1)/2} \right]$  is called the *noise component* as it represents the slow-down due to the presence of asynchronous daemons. The expected time taken by a phase can be decomposed into the *work component*  $w$ , the *latency component* and the *noise component*. The daemons start playing a significant role as soon as the noise component becomes comparable to the latency component. Now, we examine different types of noise distributions and prove a lower bound for the expected time taken to complete a phase.

**The Exponential Case.** This distribution arises as the continuous limit of the discrete geometric random distribution and occurs very often in practice as a description of the time elapsing between unpredictable events, such as,

telephone calls, radioactive emission, arrivals of buses. Being one of the most natural and important distribution to model such events, in this section we consider the case when the noise  $\eta_{ij}$ -s are also distributed according to the exponential distribution. An exponential distribution  $X_{\text{exp}}$  with mean 1 has the following distribution:  $\forall x \geq 0, \Pr[X_{\text{exp}} \leq x] = 1 - \exp(-x)$ . In this case, the following lower bound shows the impact of the noise being exponential.

**Theorem 2 (Exponential Noise).** *If  $\{\eta_{ij} : i \in [1 \dots N]\}$  are independently and identically distributed according to  $X_{\text{exp}}$ , then the expected time taken per phase is at-least  $w \left(1 + \frac{f}{1-f} (\ln(N+1) - \Theta(1))\right) + 2\tau(\log(N+1) - 2)$ .*

The lower bound has the form  $c \log N + d$ , where  $c = \frac{wf}{(1-f)\log e} + 2\tau$ . This is a linear function of  $\log N$ , similar to the ideal noiseless case. When  $\frac{wf}{(1-f)\log e}$  is comparable to (or less than)  $2\tau$ , the performance is close to the ideal noiseless case. Hence, only when  $\frac{wf}{(1-f)\log e}$  is large compared to  $2\tau$ , this model of noise impacts the performance by a constant factor of  $\frac{wf}{2\tau(1-f)\log e}$  compared to the ideal noiseless case.

**The Heavy-Tailed Case: The Pareto Distributions.** In this section we consider the case when the noise has a heavy tail. This is unlike the exponential case and the noise looks more like the uniform distribution. A natural and very popular way to model data which has heavy tail is the so-called Pareto distribution. The Pareto random variable  $X_{\text{par}}^a$  with parameter  $a$  has the following distribution:  $\forall x \geq 1, \Pr[X_{\text{par}}^a \leq x] = 1 - \frac{1}{x^a}$ . The Pareto distribution has mean  $\frac{a}{a-1}$ . To make this random variable with unit mean, we let  $\eta$  be  $\frac{a-1}{a} X_{\text{par}}^a$ .

**Theorem 3 (Pareto( $a$ ) Noise).** *If  $\{\eta_{ij} : i \in [1 \dots N]\}$  are identically and independently distributed according  $\frac{a-1}{a} X_{\text{par}}^a$ , then the expected time taken per phase is at-least  $w \left(1 + \frac{f}{1-f} \left(\frac{N+1}{2}\right)^{1/a} \left(\frac{a-1}{a}\right)^{1-1/a}\right) + 2\tau(\log(N+1) - 2)$ .*

The theorem shows that, in this case, the scalability of the parallel systems suffers far more than in the exponential case or the ideal noiseless case. The scaling becomes worse as the value of  $a$  goes lower. Hence, fixing  $w, \tau, f$  and  $a$ , and letting  $N$  increase, the term that will dominate here is  $N^{1/a}$ . We refer to this as *polynomial* scaling, and such a scenario is extremely undesirable, especially for small values of  $a$ .

**The Bernoulli Case.** This is parameterized by a probability  $p$  and a time  $T$ . In this setting, each thread takes time  $w+T$  with probability  $p$ , and time  $w$  with probability  $1-p$ . The Bernoulli distribution models the expected scaling behavior of collectives in the presence of in-frequent and bursty noise. This model can also be thought of as a first order and discrete approximation of a heavy-tailed noise, where the size of the tail can be controlled by varying  $pT$ .

**Theorem 4 (Bernoulli Noise).** *If  $\{\eta_{ij} : i \in [1 \dots N]\}$  are identically and independently distributed according to the Bernoulli distribution, then the expected time taken by a phase  $j$  is at-least  $w+T(1 - (1-p)^{(N+1)/2}) + 2\tau(\log(N+1) - 2)$ .*

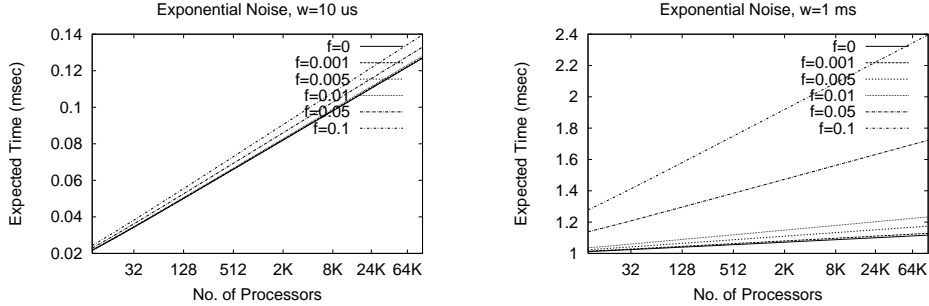
When  $\frac{pN}{2}$  is small compared to 1, the first term in the above lower bound is essentially  $w(1 + \frac{pT}{2w}N) = w(1 + \frac{f}{1-f}N)$ . Hence, in this range, the system is expected to show *linear* scaling. For very large values of  $N$ , the maximum slow-down in the performance is approximately a factor of  $1 + T/w$ .

## 4 Discussion

In this section, we discuss the implications of our results by plugging in the values for  $w, \tau$  and  $f$ , which are typical to the HPC systems (see [3]).

We use the weak scaling model to measure the scalability of the system in presence of different noise distributions. In the weak scaling model, the work per processor is kept fixed and the performance is studied as the number of processors is increased. Define  $\mathcal{N}_{1/2}$  as the minimum number of processors with which the program takes twice as much time as with one processor. For the ideal noiseless case, this happens when  $w \approx 2\tau \log(N + 1)$ , or  $N + 1 \approx 2^{w/(2\tau)}$ . This parameter gives an indication of how well a program scales in the presence of noise. Subsequently, we also discuss the values of  $\mathcal{N}_{1/2}$  for different noise distributions.

**The Exponential Case.** Figure 2 shows the expected time needed for one phase of computation in our model when  $\eta$  is distributed according to the exponential distribution (see Theorem 2). When  $w = 10\mu s$ , the noise intensity  $f$  has little impact on the performance, whereas when  $w = 1$  ms,  $f$  has significant impact.



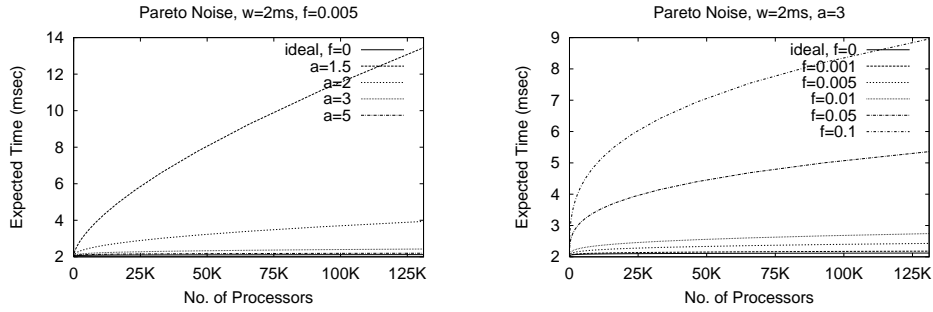
**Fig. 2.** Expected time taken by a phase in the presence of exponential noise

In this case,  $\mathcal{N}_{1/2}$  may be approximated as  $\mathcal{N}_{1/2} \approx \exp\left(\frac{1}{f/(1-f) + 2\tau/(w \ln 2)}\right)$ . With 1% exponentially distributed noise,  $\mathcal{N}_{1/2} \approx 2.3 \cdot 10^{27}$  and with 10% noise,  $\mathcal{N}_{1/2} \approx 5196$ . This shows that in the presence of an exponentially distributed noise, the programs are expected to scale well. However, unlike the ideal



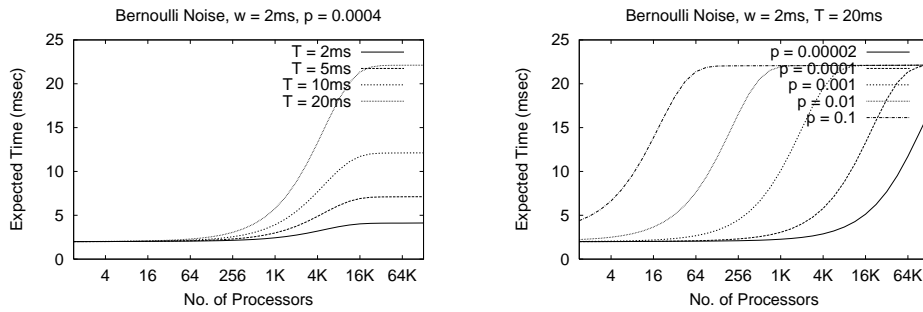
noiseless case, the time taken by collectives may be dominated by the noise component (when  $wf > \tau$ ) as opposed to the latency component.

**The Pareto Case.** Figure 3 shows the expected time needed for one phase of computation as a function of  $N$  when  $\eta$  is distributed according to the Pareto distribution (see Theorem 3). In the first plot different lines represent different values of  $a$ , while the value of  $f$  is kept fixed at 0.005. In the second plot different lines represent different values of  $f$ , while  $a$  is kept fixed at 3.



**Fig. 3.** Expected time taken by a phase in the presence of Pareto noise

In this case,  $\mathcal{N}_{1/2}$  may be approximated as  $\mathcal{N}_{1/2} \approx \min \left( 2 \cdot \left[ \frac{1-f}{fc_a} \right]^a, 2^{w/(2\tau)+2} \right)$ , where  $c_a = \left( \frac{a-1}{a} \right)^{1-1/a}$ . If  $f$  is kept fixed at 0.005, then  $\mathcal{N}_{1/2} \approx 35.4 \cdot 10^6$  for  $a = 3$ ,  $\mathcal{N}_{1/2} \approx 158,404$  for  $a = 2$ , and  $\mathcal{N}_{1/2} \approx 9724$  for  $a = 1.5$ . Similarly, for  $a = 2$ ,  $\mathcal{N}_{1/2} \approx 39,203$  when  $f = 0.01$ , and  $\mathcal{N}_{1/2} \approx 9604$  when  $f = 0.02$ . This shows that scaling behavior is sensitive to the Pareto parameter  $a$ , as well as the noise intensity  $f$ .



**Fig. 4.** Expected time taken by a phase in the presence of Bernoulli noise

**The Bernoulli Case.** Figure 4 shows the expected time taken by a phase as a function of number of nodes for different values of  $p$  and  $T$  (with  $w = 2\text{ms}$ ). Note that the x-axis is in logarithmic scale. For small values of  $N$ , the total time varies linearly with  $N$  (with a slope of  $pT/(2w)$ ). For large values of  $N$ , it saturates to  $w + T$ .

The  $\mathcal{N}_{1/2}$  in the presence of Bernoulli noise may be approximated as  $\mathcal{N}_{1/2} \approx 2/f$ . For  $f = 0.01$ ,  $\mathcal{N}_{1/2} \approx 200$ . This indicates that systems with Bernoulli noise are expected to have very poor scaling properties.

## References

1. R. Gioiosa, F. Petrini, K. Davis, and F. Lebaillif-Delamare, "Analysis of System Overhead on Parallel Computers," in *The 4th IEEE International Symposium on Signal Processing and Information Technology (ISSPIT 2004)*, (Rome, Italy), Dec. 2004.
2. T. R. Jones, L. B. Brenner, and J. M. Fier, "Impacts of Operating Systems on the Scalability of Parallel Applications," Tech. Rep. UCRL-MI-202629, Lawrence Livermore National Laboratory, Mar. 2003.
3. 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 *ACM/IEEE Conference on Supercomputing (SC'03)*, (Phoenix, Arizona, USA), Nov. 2003.
4. W. T. C. Kramer and C. Ryan, "Performance Variability of Highly Parallel Architectures," in *International Conference on Computational Science (ICCS 2003)*, (Melbourne, Australia), Jun. 2003.
5. E. Frachtenberg, F. Petrini, J. Fernandez, S. Pakin, and S. Coll, "STORM: Lightning-Fast Resource Management," in *ACM/IEEE Conference on Supercomputing (SC'02)*, (Baltimore, Maryland, USA), Nov. 2002.
6. A. Hori, H. Tezuka, and Y. Ishikawa, "Highly Efficient Gang Scheduling Implementation," in *ACM/IEEE Conference on Supercomputing (SC'98)*, (Orlando, FL, USA), Nov. 1998.
7. T. Jones, S. Dawson, R. Neely, W. Tuel, L. Brenner, J. Fier, R. Blackmore, P. Cafrey, B. Maskell, P. Tomlinson, and M. Roberts, "Improving the Scalability of Parallel Jobs by adding Parallel Awareness to the Operating System," in *ACM/IEEE Conference on Supercomputing (SC'03)*, (Phoenix, Arizona, USA), Nov. 2003.
8. E. Frachtenberg, D. Feitelson, F. Petrini, and J. Fernández, "Flexible Coscheduling: Mitigating Load Imbalance and Improving Utilization of Heterogeneous Resources," in *International Parallel and Distributed Processing Symposium 2003 (IPDPS03)*, (Nice, France), Apr. 2003.
9. S. Agarwal, G. S. Choi, C. R. Das, A. B. Yoo, and S. Nagar, "Co-ordinated Coscheduling in Time-Sharing Clusters through a Generic Framework," in *IEEE International Conference on Cluster Computing (CLUSTER'03)*, (Hong Kong), Dec. 2003.
10. S. Agarwal, R. Garg and N. Vishnoi, "The Impact of Noise on the Scaling of Collectives: A Theoretical Approach", Tech. Rep. RI-05003, *IBM Research Report*, Feb. 2005.