RI-05-003 Feb 24th,, 2005 Subject: Computer Science

IBM Research Report

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

Saurabh Agarwal
saurabh.agarwal@in.ibm.com

Rahul Garg
grahul@in.ibm.com

Nisheeth Vishnoi
nvishnoi@in.ibm.com

IBM Research Division,
IBM India Research Lab,
Hauz Khas, New Delhi - 110016. INDIA.
Phone: +91-11-2686-1100, Fax: +91-11-2686-1555

IBM Research Division
Almaden - Austin - Beijing - Delhi - Haifa - T.J. Watson - Tokyo - Zurich

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 (captured by the Pareto 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 of Bernoulli noise. 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 believe that this study will prove to be extremely useful in identifying and improving the 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.

1 Introduction

Motivation

It is well known that many parallel applications do not scale well on large high-performance computing systems [1, 2, 3]. The (per-node) performance degradation is more pronounced in systems more than 1K nodes, running a multi-tasking operating systems such as Unix. In order to build high-performance computing systems that are capable of very high sustained performance as required by emerging applications such as protein folding [4], it is important to understand the reasons for such performance degradation.

It is increasingly becoming evident one of the important causes of performance degradation is the *noise* in the system in the form of daemons and interrupts [1, 2]. For instance, Jones *et al.* [2] found that with only 0.31% daemon and OS activity on each CPU of a dedicated 4096 processor IBM SP system, *MPI_Allreduce* operation degraded more than 400%. The expected average time, using the fact that such reduction operations should scale as the logarithm of the number of processors [5, 6] was calculated to be $710\mu s$, while the observed average time was close to $3000\mu s$. Such collective operations, typical of a large class of parallel applications [6, 7], are most susceptible to a cascading effect, which results when one slow process impedes the progress of all other processes.

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. This also lead to a factor 2.21 improvement in the performance of the SAGE benchmark (see Table 3 in [3]) on 7680 processors. Similarly Kramer and Ryan [8] concluded that the performance variability in EP (of NAS parallel benchmarks) was due to 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 arises 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 (captured by the Pareto distribution) and the Bernoulli distribution. We show that the systems scale well in the presence of an exponential noise, but the performance goes down drastically in the presence of a heavy-tailed or a Bernoulli noise. Though our model is presently 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. In a subsequent paper, we will refine the model to capture more details arising in the real scenario. In there, we will also present more comprehensive studies of the noise arising in different clusters and compare the observed results with those predicted by our models.

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, 9, 10].

Another approach is to "synchronize" the noise across the nodes of the systems. This may be done by either periodically adjusting the scheduling priorities of the processes, or by changing the scheduler in the kernel. Two types of changes to the scheduling policies have been suggested in the literature: explicit and implicit. Explicit co-scheduling techniques, typically, implement a global gang scheduler [11, 12, 13], or tune the OS scheduling policy [14] to synchronize processes of a parallel job, so that, they are scheduled simultaneously across all the nodes. Implicit co-scheduling techniques use hints from the incoming messages to anticipate which job might be executing on the sending end, and raise the scheduling priority of the corresponding receiving process to synchronize with the sender [15, 16].

Though, all these techniques have resulted in 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 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, Pareto or Bernoulli distribution. The proofs of the theorems in this section are presented in Appendix A. In Section 4, we discuss our results by plugging-in some representative values for various parameters.

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. An example of such a program is presented in Figure 1. A footprint of such a program is typically present in many parallel applications, in particular in those which involve scientific computation.

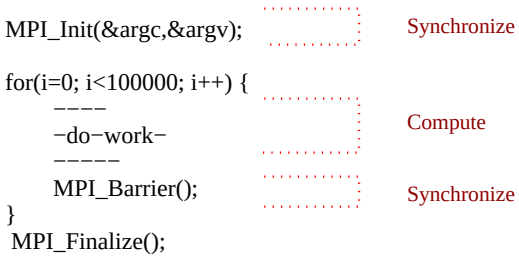

Figure 1: Typical MPI code in a scientific application

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 in Figure 1 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 τ, which is referred to as the *one-way latency*. Label the processors from $\{1, \ldots, N\}$. Denote by a_i the time taken by the message to reach node i in the post-barrier phase. This is 0 for the root node and $\tau(k-1) = \tau(\log(N+1)-1)$[1] for all the leaves. 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.

A Phase

The program consists of M such phases. Now we model various aspects of one such phase. Let t^s_{ij} represent the time instant when i-th thread begins the computation stage in the j-th phase. Let t^f_{ij} represent the time instant when 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

[1] From here on, *log* refers to log_2 and *ln* refers to log_e.

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 daemons and other asynchronous events. We capture this by a random variable δ_{ij}. More precisely,

$$t_{ij}^f - t_{ij}^s = w_{ij} + \delta_{ij},$$

where δ_{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 δ_{ij} also includes 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 δ_{ij} depends on the time taken to do the 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 is

$$t_{ij}^f - t_{ij}^s = w_{ij} \left(1 + \frac{f_{ij}}{1 - f_{ij}} \eta_{ij}\right),$$

where η_{ij} is a normalization of δ_{ij} such that $\mathrm{E}[\eta_{ij}] = 1$.

2.2 The Assumptions

In this section we state the assumptions we make about the model in the previous section. We justify and discuss them in more detail in the following section.

Balanced Load: $W_{ij} = W$ for all i, j.

Identical Processors: $w_{ij} = w$ for all i, j.

Stationary and Balanced Overheads: $f_{ij} = f$ for all i, j.

Identical Noise: Random variables η_{ij} are identically distributed for all i, j.

Spatial Independence: The random variables $\{\eta_{ij} : i \in [1 \ldots N]\}$ are independent for each j.

2.3 Justification of the Assumptions

The underlying principle in making the above set of 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. Our contribution is to show, in a formal manner, that even in this ideal setting, the nature of noise may impact the system performance considerably. Of course, one can make the model more and more *real* by removing some of these assumptions, but then the theoretical analysis of the model also becomes considerably difficult.

Balanced Load

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). In addition to this, we also assume that the load across every phase is equally divided. This assumption is made just to simplify the presentation. Our results do not change even if this additional assumption is dropped.

Identical Processors

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[2]. In addition to this, we make two more assumptions

- The application starts with all its threads in identical states.
- 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

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 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 $f_{ij} = f_{ij'}$, 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. Typical activities include kernel timer interrupts or a file system flush. The extrinsic processes run only on a subset of nodes and carry out activities that are external to the node. These activities include running various cluster management services or network monitoring daemons.

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. We ignore the effect of extrinsic processes mainly for the simplicity of analysis. 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

Again, due to homogeneity of nodes and the fact that we choose to ignore the effect of extrinsic

[2]We do not consider programs running on heterogeneous clusters that distribute the load across multiple nodes depending on the relative speed of the nodes.

processes, we may assume that the nature of noise associated to the intrinsic processes is the same across the nodes, and phases.

Spatial Independence
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. However, there may be reasons due to which the system behavior can be synchronized:

(i) Some HPC systems may deploy different scheduling policies to alleviate the daemon problems [14, 15, 16], as discussed earlier in Section 1.

(ii) Some unforeseen system-wide phenomenon (such as synchronization of periodic routing messages in communications networks [17]) may lead to synchronization of noise.

Our analysis is restricted to systems that do not employ a co-ordinated scheduling policy. Case (ii) is a possibility but we are unaware of any HPC system that reports this.

Note that we do not assume the random variables $\{\eta_{ij} : j \in [1 \ldots M]\}$ 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 η_{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

3.1 The Ideal Noiseless Case

Figure 2 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.

3.2 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, $t^f_{ij} - t^s_{ij}$ is randomly distributed. An example of this scenario is presented in Figure 3. The post-barrier phase of the communication is still the same as in the case of the ideal noiseless case.

Recall that a_i denotes 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 complete the j-th phase then is at-least $\max_{i=1}^{N}(a_i + t^f_{ij} - t^s_{ij} + b_i)$. 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 leave threads.

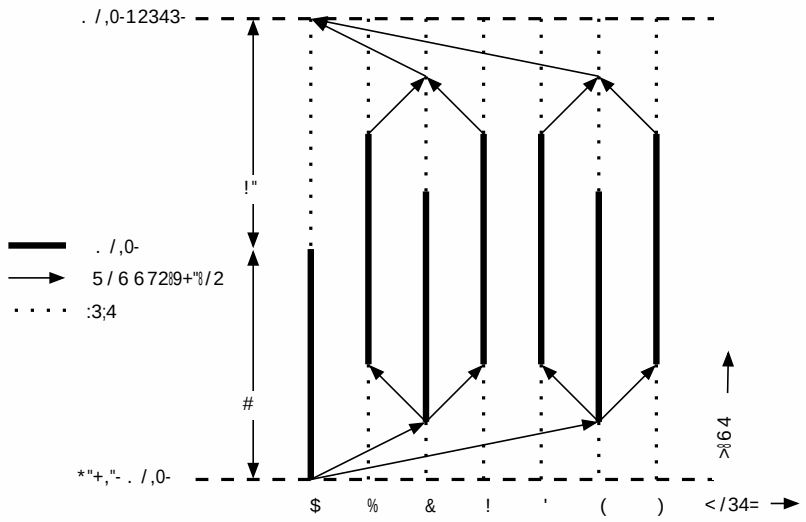

Figure 2: An ideal noiseless barrier computation cycle

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_η^r denote the maximum of r random variables which are independent and identically distributed according to η. Hence, the expectation of $t_{ij}^f - t_{ij}^s$ is at-least

$$2\tau(\log(N+1) - 2) + w\left(1 + \frac{f}{1-f}\mathrm{E}\left[Y_\eta^{\frac{N+1}{2}}\right]\right).$$

Therefore, we have the following theorem.

Theorem 3.1 *The expected time taken per phase is bounded by*

$$\begin{aligned} w\left(1 + fE\left[Y_\eta^{(N+1)/2}\right]\right) + 2\tau(\log(N+1) - 2) &\leq t_{ij+1}^s - t_{ij}^s \\ &\leq w\left(1 + \frac{f}{1-f}E\left[Y_\eta^N\right]\right) + 2\tau(\log(N+1) - 1) \end{aligned}$$

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 $wfE\left[Y_\eta^{(N+1)/2}\right]$ 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. In what follows, we just state the theorems, and the reader is referred to Appendix A for the proofs. We discuss the implications of the theorems in Section 4, where we plug in some observed values in the formulas.

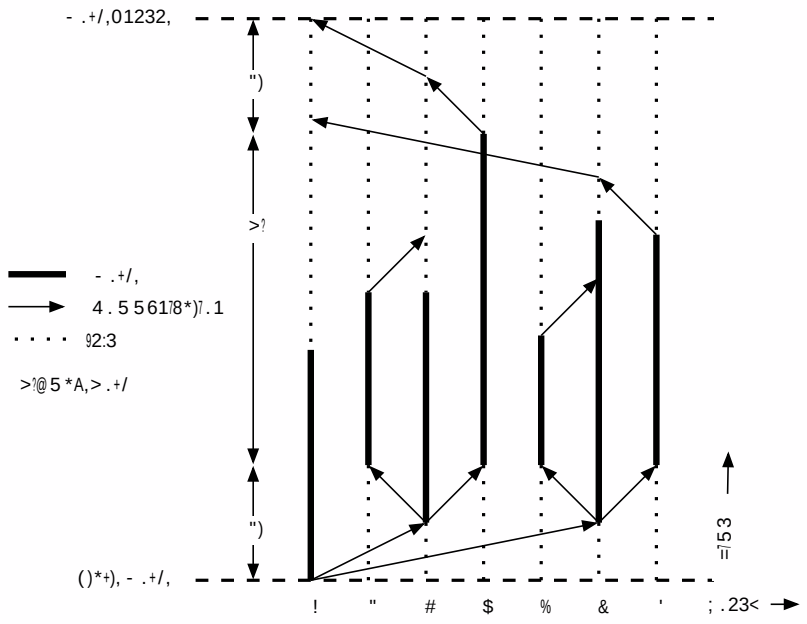

Figure 3: An ideal noisy barrier computation cycle

3.2.1 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 η_{ij}-s are also distributed according to the exponential distribution.
An exponential distribution $X_{\exp}$ with mean 1 has the following distribution:

$$\forall x \geq 0, \quad \Pr[X_{\exp} \leq x] = 1 - \exp(-x).$$

In this case, the following lower bound shows the impact of the noise being exponential.

Theorem 3.2 (Exponential Noise) *If the random variables $\{\eta_{ij} : i \in [1 \ldots N]\}$ are identically and independently distributed according to the exponential distribution, then the expected time taken per phase is lower bounded by*

$$w\left(1 + \frac{f}{1-f}\left(\ln(N+1) - \Theta(1)\right)\right) + 2\tau(\log(N+1) - 2).$$

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) 2τ, the performance is close to the ideal noiseless case. Hence, only when $\frac{wf}{(1-f)\log e}$ is large compared to 2τ, 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.

3.2.2 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. Here we describe the continuous version of it. The Pareto random variable X^a_{par} with parameter a has the following distribution:

$$\forall x \geq 1, \quad \Pr[X^a_{\text{par}} \leq x] = 1 - \frac{1}{x^a}.$$

The Pareto distribution has mean $\frac{a}{a-1}$. Hence, the mean is finite only when $a > 1$. Further, it has a finite second moment only when $a > 2$. To make this random variable with unit mean, we let η be $\frac{a-1}{a} X^a_{\text{par}}$.

It has been observed that massive networks such as the World Wide Web (WWW), the Internet, tele-communication networks and other social networks have various properties, such as degree sequences, eigenvalues, connected components, that follow the so called *Power Law*. For instance, it has been observed that for the WWW, the number of vertices having degree k is proportional to k^{-a}, see [18] for a discussion on this. The observed value of a is around 3. This exactly corresponds to the (discrete) Pareto distribution with parameter a. In that sense these massive social networks are unlike *random graphs* where the degree distribution is more closely approximated by an exponential distribution. Since we are also concerned with massively parallel machines, it seems useful to consider the Pareto distribution as another natural model for the noise.

The following 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.

Theorem 3.3 (Pareto(a) Noise) *If the random variables $\{\eta_{ij} : i \in [1 \ldots N]\}$ are identically and independently distributed according to the Pareto distribution with parameter a, then the expected time taken per phase is lower bounded by*

$$w\left(1 + \frac{f}{1-f}\left(\frac{N+1}{2}\right)^{1/a}\left(\frac{a-1}{a}\right)^{1-1/a}\right) + 2\tau(\log(N+1) - 2).$$

Hence, fixing w, τ, 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.

3.2.3 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 infrequent 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. The following theorem gives a lower bound on the expected time taken to complete a phase.

Theorem 3.4 (Bernoulli Noise) *If the random variables* $\{\eta_{ij} : i \in [1 \ldots N]\}$ *are identically and independently distributed according to the Bernoulli distribution, then the expected time taken by phase* j *is lower bounded by*

$$w + T(1 - (1-p)^{(N+1)/2}) + 2\tau(\log(N+1) - 2).$$

When $\frac{pN}{2}$ is small compared to 1, the first term in the above lower bound is essentially $w(1+\frac{pT}{2w}N) = w\left(1 + \frac{f}{1-f}N\right)$. 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, τ and f, which are typical to the HPC systems (see [3]). First, we explain the choice of these parameters, and then plot (lower bounds on) the expected time taken by a phase as a function of number of nodes.

Choice of parameters

- N, which denotes the number of processors in a cluster, is varied from 2^5 to 2^{17}.
- The values of w studied in [3] range from 1ms to 5ms. We also study the same range of values. In the case of the exponential distribution, we also present a plot for $w = 10\mu$s to demonstrate the impact of noise on applications which are communication intensive.
- τ is chosen to be a fixed value of 4μs, a representative average value for a unidirectional 16 byte message transfer in the Quadrics Interconnect [19], used in six out of top 10 supercomputers [20].
- In the ASCI Q system, f has been observed to be in the range $[0.002, 0.0125]$ [3]. We vary f from 0.001 to 0.1. This allows us to demonstrate the implication of our results in the extremely noisy cases.
- For the Pareto distribution, we vary a from 1.5 to 10. Thus, our range includes both the very heavy-tailed distributions and the relatively less heavy-tailed distributions.

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.

4.1 The Exponential Case

Figures 4 and 5 show the expected time needed for one phase of computation in our model when η is distributed according to the exponential distribution (see Theorem 3.2). These plots show scaling with different noise intensities ($0.001 < f < 0.1$) under different work quantas ($10\mu\text{s} < w < 5\text{ms}$). Note that the x-axis is on a logarithmic scale in these plots.

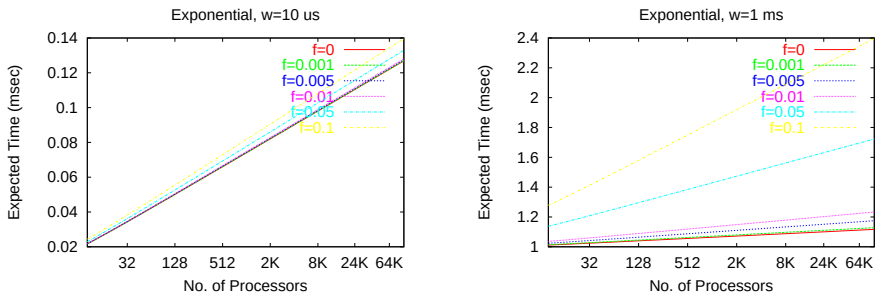

Figure 4: Expected time taken by a phase in the presence of exponentially distributed noise for different values of f with $w = 10\mu\text{s}$ and $w = 1\text{ms}$.

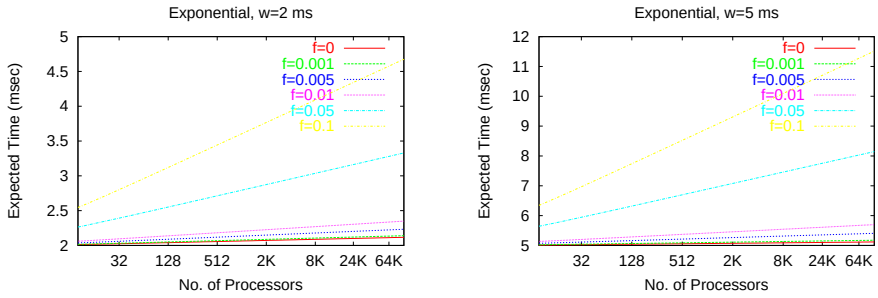

Figure 5: Expected time taken by a phase in the presence of exponentially distributed noise for different values of f with $w = 2\text{ms}$ and $w = 5\text{ms}$.

When the work quanta w is small ($10\mu\text{s}$), at $f = 0.01$ and $N = 32$, the expected time taken by a phase is $34.7\mu\text{s}$, of which $10\mu\text{s}$ is due to the work component, $24.3\mu\text{s}$ is due to the latency component ($2\tau(\log(N+1) - 2)$). In this case the noise intensity f has little impact on the performance. The same is true for all values of N. When wf becomes large ($10\mu\text{s}$ to $50\mu\text{s}$) as compared to the latency ($\tau = 4\mu\text{s}$), the noise component ($wf/(1-f)\ln(N+1)$) begins to dominate the latency component. For example, when $w = 2\text{ms}$, $f = 0.01$ and $N = 32$, the latency component is $24\mu\text{s}$ while the noise component is $70\mu\text{s}$.

In this case, $\mathcal{N}_{1/2}$ may be approximated as

$$\mathcal{N}_{1/2} \approx \exp\left(\frac{1}{f/(1-f) + 2\tau/(w\ln 2)}\right).$$

For the ideal noiseless case at $w = 2\text{ms}$, $\mathcal{N}_{1/2} \approx 1.8 \cdot 10^{75}$, while 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 the ideal noiseless case, the time taken by collectives may be dominated by the noise component as opposed to the latency component.

4.2 The Pareto Case

Figures 6 and 7 show the expected time needed for one phase of computation as a function of N when η is distributed according to the Pareto distribution (see Theorem 3.3). In Figure 6, different graphs represent different values of a, while the value of f is kept fixed at 0.005. In Figure 7 the graphs represent different values of f, while a is kept fixed at 3.

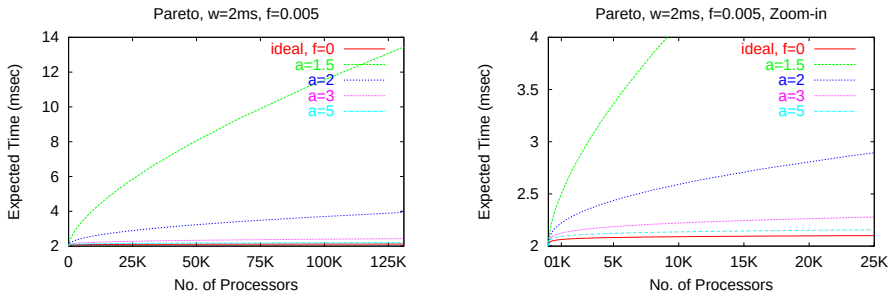

Figure 6: Expected time taken by a phase in the presence of Pareto noise for different values of a.

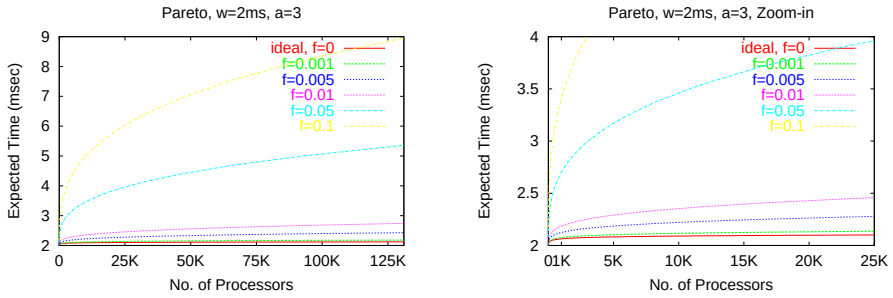

Figure 7: Expected time taken by a phase in the presence of Pareto noise ($a = 3$) for different values of f.

It may be seen from Figure 6 that for $a = 1.5$, the expected time taken by a phase is 5ms for $N = 17K$ and 9.92ms for $N = 75K$. The work component is this phase is only 2ms. This indicates poor scaling. For $a = 2$ and $N = 75K$, the expected time is 3.49ms, suggesting poor scaling in this range. For $a = 3$, the expected time to complete a phase is 2.37ms for $N = 75K$, indicating a better scaling behavior. Note that in this case, even if N is small (say 4000), the noise component (241μs) dominates the latency component (79μs).

The scaling behavior different for larger values of f. Figure 7 indicates that at $f = 0.1$, the expected time to complete a phase for $a = 3$ at $N = 10K$ is 4.99ms, indicating poor scaling.

In the Pareto case, $\mathcal{N}_{1/2}$ may be approximated as

$$\mathcal{N}_{1/2} \approx \min\left(2 \cdot \left[\frac{1-f}{f c_a}\right]^a, 2^{w/(2\tau)+2}\right),$$

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.

4.3 The Bernoulli Case

The Bernoulli distribution models the expected scaling behavior of collectives in the presence of infrequent and bursty noise, typical of cluster management software, journaling activity, file system cache flush, network monitoring or management softwares (page 8 of [3]). For Bernoulli noise, the time taken by the computation in a phase is $w + T$ with probability p, and w with probability $1 - p$. The daemon load in this case is given by $f = pT/(w + pT)$. For further discussion, we assume that $w = 2$ms.

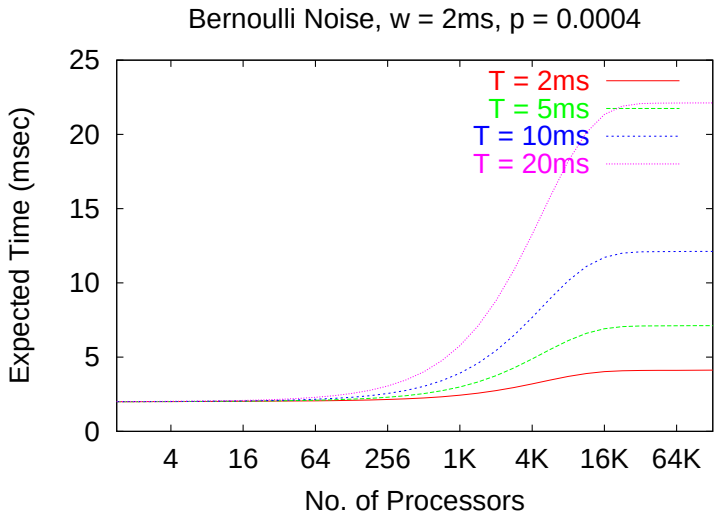

Figure 8: Expected time taken by a phase in the presence of Bernoulli noise for $p = 0.0004$ and different values of T

We first study the impact of the parameters p, T and f on performance of the system. Figure 8 shows the expected time taken by a phase as a function of number of nodes for different values of T. The value of p is fixed at $p = 0.0004$. Note that the x-axis is in logarithmic scale.

From Figure 8 it may be seen that the expected time to complete a phase is less than 2.29ms for all values of T, when N is less than 64. Out of this time, the compute step takes 2ms. For $N = 256$, the expected time is between 2.15ms and 3.05ms. For $N = 1024$, it is between 2.43ms and 5.77ms. This indicates that the impact of noise becomes noticeable only when $N \geq 64$, and is significant when $N \geq 1024$. Further, for $N \geq 16K$, the expected time is close to theoretical peak value of $(w + T + 2\tau(\log N - 2))$ms for $p = 0.1, 0.01$ and 0.001.

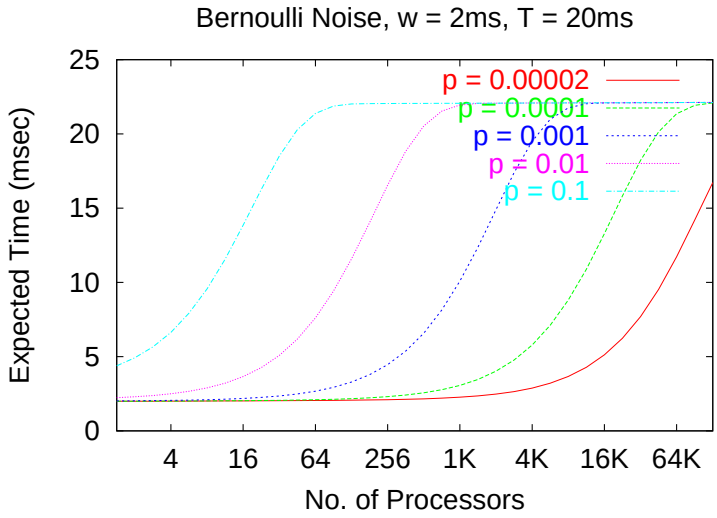

Figure 9: Expected time taken by a phase in the presence of Bernoulli noise with $T = 20ms$ and different values of p

Figure 9 shows the impact of p on the system performance. The value of T is kept fixed at 20ms. For large values of N, the expected time taken by a phase converges to 22.096ms. However, if p is large, the noise starts dominating even for small values of N. At $N \approx 1/(2p)$, the expected time taken becomes close to 5ms for all values of p.

We now analyze the impact on performance when the noise intensity $f \cong pT/w$ is kept the same, T is increased, and p is decreased proportionally. The parameter T is a measure of the *burstiness* of the noise. We increase T, thereby increasing the burstiness of the noise while keeping its intensity (f), the same. Figure 10 shows the expected time taken by a phase as a function of N. Note that the x-axis as well as the y-axis in this graph are on a logarithmic scale. The graphs indicate that the system performance with "bursty noise" is always worse than that with "smoother noise". For large values of N, the expected time taken by a phase converges to $w+T+2\tau(\log N-2)$ as expected.

The $\mathcal{N}_{1/2}$ in the presence of Bernoulli noise may be approximated as

$$\mathcal{N}_{1/2} \approx 2\frac{\log(1-w/T)}{\log(1-p)} = \frac{2}{f}\left[\frac{1+w/(2T)+w^2/(3T^2)+w^3/(4T^3)+\cdots}{1+p/2+p^2/3+p^3/4+\cdots}\right] \approx \frac{2}{f}$$

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.

We now discuss the impact of Bernoulli noise on systems with parameters similar to the ASCI Q system [3]. From [3] (Figure 13), we extracted three cases of noise and inferred the values of p and T as follows:

RMS Data A Quadrics resource management system (RMS) daemon collects cluster-wide data on 1 of 32 nodes in the cluster. This is a frequent but light-weight activity occurring every

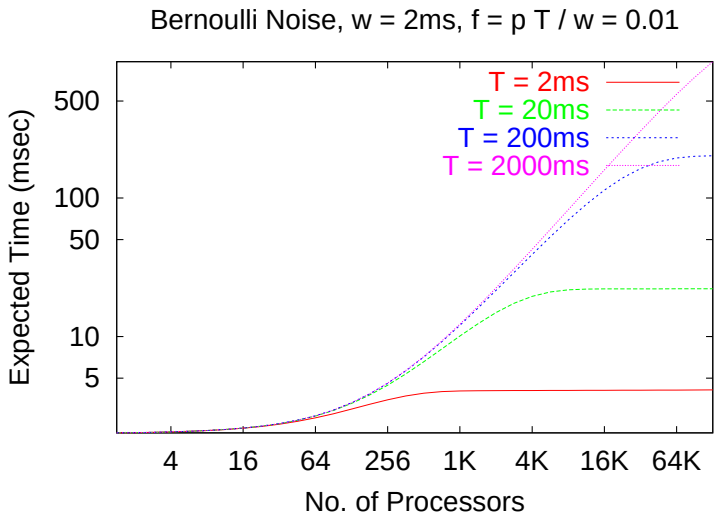

Figure 10: Expected time taken by a phase in the presence of Bernoulli noise with fixed noise intensity $f = 0.01$ and different values of T and p

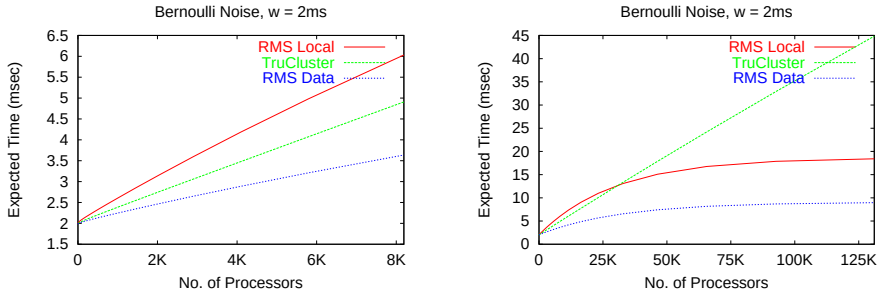

Figure 11: Expected impact of different types of Bernoulli noise on the ASCI Q system

1s on one node and takes about 7ms, giving $p = 2\text{ms}/1\text{s} = 0.002$, and $T = 7\text{ms}$. For this noise, $f = 0.007$ on 1 out of every 32 nodes.

RMS Local Two events generated by Quadrics resource management system (RMS), occur every 30s on each of the 32 nodes. These events take a time between 15ms and 18ms. This gives $p = 2\text{ms}/30\text{s} = 6.6 \cdot 10^{-5}$. T may be taken as the mean of the two values (16.5ms). Thus, for this type of noise, $f = 0.00055$.

TruCluster A TruCluster event (generated by the cluster management software) occurs every 60s on 2 of every 32 nodes and takes 335ms, giving $p = 3.3 \cdot 10^{-5}$ and $T = 335\text{ms}$. The corresponding noise intensity on these two nodes is given by $f = 0.0056$.

Figure 11 shows the impact of these three types of noises on the system performance. The graphs show the expected time taken by a phase as a function of the number of nodes in the system. Since the "TruCluster" daemon runs only on 2 out of 32 processors, and "RMS Data" on 1 out of 32 processors, the parameter N (in Theorem 3.4) was adjusted appropriately. The x-axis

in these graphs represents the total number of processors in the system (out of which only a subset has "TruCluster" and "RMS Data" noise).

From these graphs, it is evident that in the presence of only the "TruCluster" noise, the time taken by a phase (carrying out 2ms of useful computation) may become as large as 45ms when $N = 128K$. This is remarkable because the CPU load due to this noise is only 0.56% on just 2 out of 32 nodes!

When the number of processes is small, the "RMS Local" noise has the most impact followed by "TruCluster" and "RMS Data". For large values of N ($> 35K$), "TruCluster" noise has the most impact, followed by "RMS Local" and "RMS Data". The expected time taken by a phase varies, initially, linearly with the number of nodes, and then saturates to the value of T. The initial slope is equal to the product of frequency of the interruptions, the work quanta (w), the number of participating nodes and the time taken by the daemon (T) during each interruption. The latency component in these graphs is negligible.

5 Future Work

In this paper, we initiated the study of impact of noise on the scaling of parallel applications (specifically the collective operation) in a formal manner. The work can be extended in many ways. A simple extension would be to analyze a larger class of noise distributions. The scope of this study can be further broadened by including programs that carry extensive message passing in addition to periodic synchronization. An empirical study of the nature of noise will be useful in understanding real systems and refining our model. Finally, one needs to validate the model (especially the *independence of noise* assumption) by performing detailed experiments and comparing measured and predicted values on large HPC systems. We believe that this study, once matured, will prove to be extremely useful in identifying and improving bottlenecks in the scalability of HPC systems.

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 Scalibility 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] The BlueGene Team, "Blue Gene: A Vision for Protein Science using a Petaflop Supercomputer," *IBM Systems Journal*, vol. 40, no. 2, p. 310, 2001.

[5] A. Moody, J. Fernandez, F. Petrini, and D. K. Panda, "Scalable NIC-based Reduction on Large-scale Clusters," in *ACM/IEEE Conference on Supercomputing (SC'03)*, (Washington, DC, USA), p. 59, 2003.

[6] F. Petrini, J. Fernandez, E. Frachtenberg, and S. Coll, "Scalable Collective Communication on the ASCI Q Machine," in *11th Symposium on High Performance Interconnects*, (Stanford, California, USA), Aug. 2003.

[7] A. Gupta, A. Tucker, and S. Urushibara, "The Impact of Operating System Scheduling Policies and Synchronization Methods on the Performance of Parallel Applications," in *ACM SIGMETRICS Conference*, May 1991.

[8] W. T. C. Kramer and C. Ryan, "Performance Variability of Highly Parallel Architectures," in *International Conference on Computational Science (ICCS 2003)*, pp. 560–569, 2003.

[9] F. Petrini, D. Kerbyson, and A. Hoisie, "Further Improvements in ASCI Q Performance and Variability: an Application of the PAL Methodology for Identifying and Eliminating Performance Variability," Tech. Rep. LAUR 03-1021, Los Alamos National Laboratory, February 2003.

[10] A. Hoisie, D. Kerbyson, S. Pakin, F. Petrini, H. Wasserman, and J. F. Peinador, "Identifying and Eliminating the Performance Variability on the ASCI Q Machine," tech. rep., Los Alamos National Laboratory, January 2003.

[11] J. E. Moreira, H. Franke, W. Chan, L. L. Fong, M. A. Jette, and A. Yoo, "A Gang-Scheduling System for ASCI Blue-Pacific," in *7th International Conference on High-Performance Computing and Networking (HPCN'99)*, pp. 831–840, Springer-Verlag, 1999.

[12] A. Hori, H. Tezuka, and Y. Ishikawa, "Highly Efficient Gang Scheduling Implementation," in *ACM/IEEE conference on Supercomputing (SC'98)*, (San Jose, CA, USA), pp. 1–14, 1998.

[13] 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.

[14] T. Jones, S. Dawson, R. Neely, W. Tuel, L. Brenner, J. Fier, R. Blackmore, P. Caffrey, 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.

[15] 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.

[16] 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.

[17] S. Floyd and V. Jacobson, "The Synchronization of Periodic Routing Messages," *IEEE/ACM Transactions on Networking*, vol. 2, no. 2, pp. 122–136, 1994.

[18] L. A. Adamic and B. Huberman, "Power-Law Distribution of the World Wide Web," *Science Journal*, p. 287, 2000.

[19] J. Beecroft, D. Addison, F. Petrini, and M. McLaren, "Quadrics QsNet II: A Network for Supercomputing Applications," in *Hot Chips 15*, (Stanford University, Palo Alto, CA, USA), August 2003.

[20] "The Top 500 Supercomputers in the world." Refer http://top500.org.

Appendix A: Proofs of Theorems

In this section we present proofs of Theorems 3.4, 3.2 and 3.3.

The Bernoulli Distribution

The Bernoulli distribution with parameter $0 \leq p \leq 1$, takes on value 1 with probability p and 0 with probability $1-p$. The maximum of n identical and independent Bernoulli random variables is 0 with probability $(1-p)^n$ and 1 with probability $1-(1-p)^n$. Hence, the expectation is $1-(1-p)^n$.

Continuous random variables

For a continuous random variable X, we denote by $f(t)$ its probability density function (pdf), and by $F(x) := \Pr[X \leq x] = \int_{-\infty}^{x} f(t)\mathrm{d}t$ the cumulative density function (cdf). Let $X_1, \ldots, X_n$ be independent random variables, identically distributed to a continuous random variable X. Let $Y := \max_{i=1}^{n} X_i$. Then, $\Pr[Y \leq x] = \prod_{i=1}^{n} \Pr[X_i \leq x] = F(x)^n$.

The expectation of a continuous random variable X is $\mathrm{E}[X] := \int_{-\infty}^{\infty} tf(t)\ \mathrm{d}t$. If the random variable is non-negative, there is a particularly convenient formula for its expectation. Let X be a non-negative random variable with range $[b, \infty)$, for some $b \geq 0$. Then,

$$\int_b^\infty \Pr[X > x]\mathrm{d}x = \int_b^\infty \int_{t=x}^\infty f(t)\mathrm{d}t\mathrm{d}x = \int_b^\infty \int_{x=b}^t f(t)\mathrm{d}x\mathrm{d}t = \int_b^\infty tf(t)\mathrm{d}t - b.$$

Hence, we have proved the following:

Fact 5.1 *Let Y be the maximum of n identically and independently distributed non-negative random variables X, with range $[b, \infty)$. Then*

$$\mathrm{E}[Y] = \int_b^\infty 1 - F(x)^n \mathrm{d}x + b.$$

The Exponential Distribution

For the exponential distribution $X_{\exp}$ with mean 1,

$$\forall x \geq 0, \quad \Pr[X_{\exp} \leq x] = 1 - \exp(-x).$$

Let $Y_{\exp}^n$ denote the maximum of n independent copies $X_{\exp}$.

Proposition 5.2

$$\mathrm{E}\left[Y^n_{\exp}\right] = 1 + \frac{1}{2} + \cdots + \frac{1}{n} = \ln n + \Theta(1).$$

The proof follows from Fact 5.1 combined with the following integral.

Lemma 5.3

$$\int_0^\infty 1 - (1 - \exp(-x))^n \mathrm{d}x = 1 + \frac{1}{2} + \cdots + \frac{1}{n}.$$

Proof: Substitute $u := 1 - \exp(-x)$. $\mathrm{d}u = -\exp(-x)\mathrm{d}x$. Hence, $\mathrm{d}x = \frac{-1}{1-u}\mathrm{d}u$. Hence, the integral becomes

$$\int_0^1 \frac{1-u^n}{1-u}\mathrm{d}u = \int_0^1 \left(1 + u + u^2 + \cdots u^{n-1}\right)\,\mathrm{d}u = 1 + \frac{1}{2} + \cdots + \frac{1}{n}.$$

■

Theorem 3.2 follows as a corollary to this proposition.

The Pareto Distribution

For the Pareto random variable X^a_{par} with parameter a,

$$\forall x \geq 1, \quad \Pr[X^a_{\mathrm{par}} \leq x] = 1 - \frac{1}{x^a}.$$

The Pareto distribution has mean $\frac{a}{a-1}$. Hence, the mean is finite only when $a > 1$. Further, it has a finite second moment only when $a > 2$. We assume that $a > 1$. Let Y^n_{a} denote the maximum of n independent copies X^a_{par}.

Proposition 5.4

$$n^{1/a}\left(\frac{a}{a-1}\right) \geq \mathrm{E}\left[Y^n_a\right] \geq n^{1/a}\left(\frac{a\left(1+\frac{1}{n}\right)-\frac{1}{n}}{a-1}\right)^{1/a} \geq n^{1/a}\left(\frac{a}{a-1}\right)^{1/a}.$$

The proof of this follows by first evaluating the integral suggested by Fact 5.1. This is done in Lemma 5.5. We then give upper and lower bounds to the integral using Lemmas 5.7 and 5.8.

Lemma 5.5 *For $a > 1$,*

$$\int_1^\infty 1 - \left(1 - \frac{1}{x^a}\right)^n \mathrm{d}x = \frac{1}{\left(1-\frac{1}{a}\right)\left(1-\frac{1}{2a}\right)\cdots\left(1-\frac{1}{na}\right)} - 1.$$

Proof: The integral above can be written as

$$\begin{aligned}
\int_1^\infty 1 - \left(1 - \frac{1}{x^a}\right)^n \mathrm{d}x &= \int_1^\infty \left[1 - \sum_{k=0}^n \binom{n}{k}(-1)^k \frac{1}{x^{ak}}\right]\mathrm{d}x \\
= \sum_{k=1}^n \binom{n}{k}(-1)^k \int_1^\infty \frac{-1}{x^{ak}}\mathrm{d}x &= \sum_{k=1}^n \binom{n}{k}(-1)^k \left[\frac{1}{ak-1}\frac{1}{x^{ak-1}}\right]_1^\infty \\
= \sum_{k=1}^n \binom{n}{k}(-1)^k \frac{1}{1-ak}.
\end{aligned}$$

Hence, the following claim is sufficient to prove the lemma.

Claim:

$$\frac{1}{\left(1-\frac{1}{a}\right)\left(1-\frac{1}{2a}\right)\cdots\left(1-\frac{1}{na}\right)}-1=\sum_{k=1}^{n}\binom{n}{k}(-1)^k\frac{1}{1-ak}. \quad (1)$$

Proof of Claim: We prove this by induction on n. Let f_n denote the lhs of (1), and g_n the rhs of (1). For $n=1$, $g_1=\frac{1}{1-\frac{1}{a}}-1=\frac{1}{a-1}=f_1$. Assume that $f_k=g_k$ for all $1\le k<n$. We will show that under this assumption, $f_n=g_n$. Let $n>1$. By definition, $(g_n+1)(1-\frac{1}{na}=g_{n-1}+1$. Hence, $g_n=\frac{g_{n-1}+1}{1-\frac{1}{na}}-1$. By assumption $f_{n-1}=g_{n-1}$. Hence, $g_n=\frac{f_{n-1}+1}{1-\frac{1}{na}}-1=\frac{-Nam_{n-1}}{1-na}-\frac{1}{1-na}$. By definition,

$$f_{n-1}=\sum_{k=1}^{n-1}\binom{n-1}{k}(-1)^k\frac{1}{1-ak}.$$

Hence,

$$\begin{aligned}
g_n &= -na\sum_{k=1}^{n-1}\frac{\binom{n-1}{k}(-1)^k}{(1-na)(1-ak)}-\frac{1}{1-na}\\
&= \sum_{k=1}^{n-1}\frac{(-1)^k(na)(n-1)!}{k!(n-1-k)!}\left[\frac{1}{1-ak}-\frac{1}{1-na}\right]\left(\frac{-1}{(n-k)a}\right)-\frac{1}{1-na}\\
&= \sum_{k=1}^{n-1}\binom{n}{k}(-1)^k\left[\frac{1}{1-ak}-\frac{1}{1-na}\right]-\frac{1}{1-na}\\
&= \sum_{k=1}^{n-1}\frac{\binom{n}{k}(-1)^k}{1-ak}-\left[\sum_{k=1}^{n-1}\frac{\binom{n}{k}(-1)^k}{1-na}+\frac{1}{1-na}\right]\\
&= \sum_{k=1}^{n-1}\frac{\binom{n}{k}(-1)^k}{1-ak}-\frac{1}{1-na}\sum_{k=0}^{n-1}\binom{n}{k}(-1)^k.
\end{aligned}$$

Since $0=(1-1)^n=\sum_{k=0}^{n}\binom{n}{k}(-1)^k$, $\sum_{k=0}^{n-1}\binom{n}{k}(-1)^k+(-1)^n=0$. Hence, $\sum_{k=0}^{n-1}\binom{n}{k}(-1)^k=(-1)^{n+1}$. Substituting this in the equation above, we obtain that

$$g_n=\sum_{k=1}^{n-1}\frac{\binom{n}{k}(-1)^k}{1-ak}+\frac{(-1)^n}{1-na}=\sum_{k=1}^{n}\frac{\binom{n}{k}(-1)^k}{1-ak}=f_n.$$

This completes the proof of the claim. Hence, the lemma follows. ∎

Let $Q_n:=\left(1-\frac{1}{a}\right)\left(1-\frac{1}{2a}\right)\cdots\left(1-\frac{1}{na}\right)$.

Fact 5.6 *For any integers* $1\le j\le k$,

$$\frac{j}{j+1}\le\frac{k}{k+1}.$$

Lemma 5.7 *For all integers* $n,a\ge 1$, $Q_n^a\le\frac{a-1}{(n+1)a-1}$.

Proof:

$$\begin{aligned}
Q_n^a &= \left(\frac{a-1}{a}\right)^a\left(\frac{2a-1}{2a}\right)^a\cdots\left(\frac{na-1}{na}\right)^a\\
&\le \left[\frac{a-1}{a}\cdots\frac{2a-2}{2a-1}\right]\left[\frac{2a-1}{2a}\cdots\frac{3a-2}{3a-1}\right]\cdots\left[\frac{na-1}{na}\cdots\frac{(n+1)a-2}{(n+1)a-1}\right].
\end{aligned}$$

Here we have used the Fact 5.6. Hence, $Q_n^a\le\frac{a-1}{(n+1)a-1}$. This completes the proof of the lemma. ∎

Lemma 5.8 *For all integers* $n, a \geq 1$,

$$Q_n^a \geq \frac{1}{n}\left(\frac{a-1}{a}\right)^a \left(\frac{a}{a-\frac{1}{n+1}}\right)^a.$$

Proof:

$$\begin{aligned}\left(\frac{a}{a-1}Q_n\frac{(n+1)a-1}{(n+1)a}\right)^a &= \left(\frac{2a-1}{2a}\right)^a \cdots \left(\frac{na-1}{na}\right)^a \left(\frac{(n+1)a-1}{(n+1)a}\right)^a \\ &\geq \left[\frac{a}{a+1}\cdots\frac{2a-1}{2a}\right]\cdots\left[\frac{(n-1)a}{(n-1)a+1}\cdots\frac{na-1}{na}\right] \\ &= \frac{a}{na} = \frac{1}{n}.\end{aligned}$$

Hence, $Q_n^a \geq \frac{1}{n}\left(\frac{a-1}{a}\right)^a \left(\frac{(n+1)a}{(n+1)a-1}\right)^a$. This completes the proof of the lemma. ∎