Modeling of Workload in MPPs

Joefon Jann, Pratap Pattnaik, Hubertus Franke
IBM T. J. Watson Research Center, P. O. Box 218, Yorktown Heights, NY 10598

Fang Wang
Computer Science Department, Yale University, New Haven, CT 06520-8285

Joseph Skovira, Joseph Riordan
Cornell Theory Center, Cornell University, Ithaca, NY 14853-3801

email: joefon@watson.ibm.com

Abstract

In this paper we have characterized the inter-arrival time and service time distributions for jobs at a large MPP supercomputing center. Our findings show that the distributions are dispersive and complex enough that they require Hyper Erlang distributions to capture the first three moments of the observed workload. We also present the parameters from the characterization so that they can be easily used for both theoretical studies and the simulations of various scheduling algorithms.

1 Introduction

In recent years massively parallel processors (MPP) computers have made a significant presence. With this growth in MPPs, a number of researchers have developed and are continuing to develop various job scheduling subsystems for these MPPs [1, 2, 3, 4, 5, 6]. During the development of these schedulers and their related algorithms, it is important to have an accurate characterization of the workloads experienced by the MPPs. It is extremely advantageous to have these workloads characterized by a compact model that is representable by a few parameters, is suitable for theoretical queuing analysis of scheduling algorithms, and is reasonably straight-forward for the generation of synthetic workloads. In this paper we propose such a model, and demonstrate its efficacy by using it to fit the workload from the Cornell University Supercomputer.

The model we use for representing the inter-arrival time and the service time of jobs is a phase type distribution model, specifically, the Hyper Erlang Distribution of Common Order. Since we expect this model to be used by researchers with diverse background, we take a pedagogical approach in this paper, rather than simply refer the readers to the literature. In section 4, we describe the model parameter extraction procedure. In section 5, we present the parameters extracted from the workload of the SP2 at the Cornell Theory Center experienced during the period from June 25, 1996 to September 12, 1996. In section 6, we describe ways to generate synthetic workloads for simulation studies. In the appendix, we present a sample program to generate synthetic workloads.

2 Phase Type Distribution

The exponential distribution and the related Poisson process have been pervasively used in the stochastic modeling of computers and network workloads. The primary reason for the popularity of the exponential distribution is the ease with which it can be manipulated in theoretical studies, and not because of the presence of a large body of empirical data supporting it in a wide range of real life situations. The tractability of the exponential distribution in analytical work comes mainly from its memoryless property, leading to a simple form for the Laplace transform of the probability distribution function (pdf), namely,

$$f^*(s) = \left(\frac{\lambda}{s+\lambda}\right), \tag{1}$$

and to an underlying Markovian process. In this expression $1/\lambda$ is the first moment of the distribution and represents the average value of the random variable with the exponential distribution. In a Markovian process, the transition rate from a state to the next state depends only on the current state, and does not explicitly depend on past history. This property simplifies the steady state equations and enables one to represent all the needed information by a one dimensional vector of the current state, thus making the queuing analysis of the problem tractable. In 1947, Erlang generalized the exponential distribution to include more complex probability distributions, while preserving the analytic tractability. This generalized distribution is called the Erlang distribution and its Laplace transform is

$$f^*(s) = \left(\frac{\lambda}{s+\lambda}\right)^n. \tag{2}$$

Service time distributions that can be represented by the Erlang distribution can be thought of as originating from a system where the job goes through n phases or stages before completion. At each stage a job spends an exponentially distributed random amount of time, with the average time being $1/\lambda$. The distribution of the total service time of the jobs is then the convolution of n exponential distributions. The Laplace transform of this convolution is equation 2. The Erlang distribution, like the exponential distribution, has an underlying Markovian process, thus making it attractive for use in queuing theory. The Erlang distribution can represent more types of systems than the exponential distribution can. Finally in 1955, Cox, in a seminal paper [7], demonstrated that the key advantage of the Erlang and the exponential distributions for analytical work stems from the fact that their Laplace transforms are rational. He also developed a generalized distribution known as the Phase Type Distribution, which is capable of representing any stochastic process whose associated pdf's have Laplace transforms that are rational. Practically all the relevant systems one encounters in the stochastic modeling of workloads in computers can be modeled by the Phase Type Distribution.

In practice, one approximates only the first few moments of the probability distributions under study with Phase Type Distributions; and these Phase Type Distributions are characterized by a set of parameters m, p_0, p_i, n_i and $\lambda_{i,j}$ (where i,j=1,2,3,...), such that their Laplace transforms are of the form

$$f^*(s) = p_0 + \sum_{i=1}^{m} p_i \prod_{j=1}^{n_i} \left(\frac{\lambda_{i,j}}{s+\lambda_{i,j}}\right) \tag{3}$$

with

$$f^*(0) = 1. \tag{4}$$

Service time distributions representable by equation 3 can be thought of as coming from a system where jobs may reach completion using any of the m-possible paths. If a job goes through the i^{th} path, then it will traverse through n_i phases or stages before completion; and like the Erlang distribution, the job spends an exponentially distributed random amount of time at each stage. These exponential distributions are characterized by the parameters $\lambda_{i,j}$.

The Phase Type Distribution, in addition to its ability to conveniently represent a wide class of stochastic processes, provides an underlying Markovian process, a great advantage for queuing studies [8], for the reasons described earlier. Recently they have also been used successfully in analyzing gang-scheduling in MPPs [5].

3 Hyper Erlang Distribution of Common Order

Oftentimes, in analytical modeling of a stochastic process, only the first few moments of the random variables are considered. For most stochastic processes, the first few moments represent attributes that tend to be relatively sample invariant. Here we consider the first three moments of the random variables for our modeling. They carry the information about the mean, the variance, and the skewness of the random variables respectively.

In this paper, we choose the simplest distribution with an underlying Markovian process, that can fit the the first three moments of the observed data. As mentioned in the previous section, the underlying Markovian process makes the distribution tractable in theoretical studies. The distribution we have chosen is the Hyper Erlang Distribution of Common Order, which is a Phase Type Distribution that can exactly fit the first three moments of the observed random distribution. The Hyper Erlang Distribution of Common Order is a generalization of the exponential, the hyper exponential, and the Erlang distribution. Our fitting procedure automatically selects the simplest of these 4 distributions that is commensurate with the first three moments of the observed data.

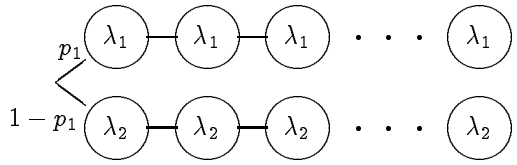
The Hyper Erlang Distribution of Common Order distribution has a Laplace transform of the form

$$f^*(s) = \sum_{i=1}^{2} p_i \Big(\frac{\lambda_i}{s + \lambda_i} \Big)^n \tag{5}$$

where n, a positive integer, is called the order of the distribution, and $0 \leq p_i \leq 1$ with $p_1 + p_2 = 1$. The Erlang distribution is a special case of the Hyper Erlang Distribution of Common Order with one of the p_is equal to 1, e.g. $p_1 = 1$. The hyper exponential distribution is a Hyper Erlang Distribution of Common Order with $n = 1$. The exponential distribution is also a special case with $p_1 = 1$ and $n = 1$.

An example of a server with service time distribution expressable by a Hyper Erlang Distribution of Common Order is a system where a job must pass through one and only one of two service paths to completion. In each path it has to pass through n stages (or phases), spending a random amount of service time at each of the n stages. The pdf

of service time at each stage of path 1 is an exponential distribution with mean time $1/\lambda_1$, and that of path 2 is an exponential distribution with mean time $1/\lambda_2$. Let p_1 be the probability of the job selecting path 1, and $(1 - p_1)$ be that of selecting path 2. Pictorially, the stages of this system can be depicted by



The k^{th} non-central moment of a distribution, for all integers $k \geq 1$, can be obtained from the Laplace transform of the distribution by,

$$\mu_k = E[t^k] = (-1)^k \Big[\frac{d^k f^*(s)}{ds^k}\Big]_{s=0} \tag{6}$$

which, for Hyper Erlang distribution of Common Order, is

$$\mu_k = \sum_{i=1}^{2} p_i \frac{n(n+1)...(n+k-1)}{\lambda_i^k} \tag{7}$$

The moments for the Erlang distribution are obtained by setting $p_1 = 1$ and $p_2 = 0$ in equation 7, yielding

$$\mu_k = \frac{n(n+1)...(n+k-1)}{\lambda^k}. \tag{8}$$

The moments for the hyper exponential distribution are obtained by setting $n = 1$ in equation 7, yielding

$$\mu_k = \sum_{i=1}^{2} p_i \frac{k!}{\lambda_i^k} \tag{9}$$

The moments for the exponential distribution is obtain by letting $n = 1$ in equation 8, giving

$$\mu_k = \frac{k!}{\lambda^k} \tag{10}$$

Examining the expressions for the first three moments of these distributions, and because physical situations imply non-negative p_is and λ_is, one finds a number of interrelationships among these moments. These interrelationships specify the constraints that must be satisfied by the moments of the observed data, for a particular model to represent the data. Without going into the proof (which is straight forward in most cases, but involves lengthy algebraic manipulations in the non-obvious cases), we state here the constraints on the first three moments of various distribution.

The constraints for the exponential distribution are:

$$\mu_2 = 2 * {\mu_1}^2 \tag{11}$$

and

$$\mu_3 = 6 * {\mu_1}^3 \tag{12}$$

Since the exponential distribution is a one parameter model, all higher moments are just functions of first moments. In the data we examined in this paper, none of the observed moments, either for inter-arrival time or the service time, fulfill this condition.

The constraints for the hyper exponential distribution are:

$$\mu_1\mu_3 > \frac{3}{2}\mu_2^2 \tag{13}$$

and

$$\mu_2 > 2\mu_1^2 \tag{14}$$

The constraints for the Erlang distribution are:

$$\mu_1\mu_3 = \frac{n+2}{n+1}\mu_2^2 \tag{15}$$

and

$$\mu_2 = \left(1+\frac{1}{n}\right)\mu_1^2 \tag{16}$$

The constraints for Hyper Erlang of Common Order are

$$\mu_1\mu_3 > \frac{n+2}{n+1}\mu_2^2 \tag{17}$$

and

$$\mu_2 > \frac{n+1}{n}\mu_1^2 \tag{18}$$

4 Our Modeling Procedure

Our procedure for modeling, selects the simplest model amongst exponential, hyper exponential, Erlang and Hyper Erlang Distribution of Common Order, as long as the first three moments of the data do not violate the constraints of the model under consideration. It also exactly matches the first three moments of the data to that of the model. In our procedure for modeling the data, we start with the three non-central moments of the data, namely the μs and fit them to the Hyper Erlang of Common Order distribution with the lowest value of n satisfying equations 17 and 18. The Hyper Erlang of Common Order distribution has four unknowns, namely λ_1, λ_2, n and p_1, p_2 where $p_1 + p_2 = 1$. For a given n, one can extract the remaining three parameters of this model by matching the expressions for $\mu_1, \mu_2,$ and μ_3 obtained from equation 7,to the first three observed non-central moments. This involves solving 3 simultaneous equations with 3 unknowns. The analytical expression for the three parameters has been derived by Johnson and Taaffe [9]. In general, an infinite number of n's can fit the data, while satisfying equations 17 and 18. We select the smallest such n. Furthermore, after solving for the p_is, if one of the p_is is very close to zero, we set it to zero yielding an Erlang or exponential distribution as a model for the data.

For example, the first three moments of the distribution of inter-arrival times for jobs requiring just one processor in our Cornell SP2 data are $\mu_1 = 1.05 \times 10^3$, $\mu_2 = 8.86 \times 10^6$ and $\mu_3 = 1.78 \times 10^{11}$ (table 7). The unit of time used in these moments is one second. An examination of the above moments for inter-arrival time will show that

they satisfy the constraints (equations 13, 14, 17 and 18) of the hyper exponential and the Hyper Erlang Distribution of Common Order, and do not satisfy those of the Erlang and the exponential distributions. Hence we choose the hyper exponential distribution (i.e. Hyper Erlang Distribution of Common Order, with $n = 1$, see line 2 of tables 1 and 2) to represent this data.

The moments for the service time distribution of these jobs requiring only one processor are: $\mu_1 = 1.54 \times 10^4$, $\mu_2 = 6.05 \times 10^8$ and $\mu_3 = 2.94 \times 10^{13}$. This set of data cannot be represented by exponential, hyper exponential or Erlang distributions, as the three moments do not satisfy the constraints of these distributions. A Hyper Erlang Distribution of Order 4 was needed to represent this data and this is reflected in the results section of this paper (see line 2 in tables 3, 4, 5 and 6, column n).

As the results section shows, none of the inter-arrival time and service time data examined in this paper is under-dispersive enough to satisfy equation 16, hence their second non-central moments are not representable by an Erlang or an exponential distribution. This over-dispersive data is sometimes referred to as longtail data.

Once the parameters of the model have been determined, it can be used either directly in theoretical studies, or in simulations by creating synthetic workloads. In the section following the results section, we outline a way to generate synthetic workloads from the model parameters presented in this paper. Also in the appendix section, we give a sample C program to illustrate the procedure for generating workloads.

5 Results

In our experimental workload, we have examined all the jobs for a 322 node SP2 at the Cornell Theory Center, for the period from June 25, 1996 to September 12, 1996. During this period, a total of 17440 jobs were serviced by the system. All these jobs required dedicated use of CPUs, and different jobs required different numbers of CPUs of the SP2. We have characterized the workload using Hyper Erlang Distribution of Common Order to model the inter-arrival time and the service time, the latter being the cumulative CPU time used by a job. For this, we have grouped the jobs into classes based on the number of CPUs they used. A job requesting p processors is assigned to a class, such that $p_{min} \leq p \leq p_{max}$, where p_{min} and p_{max} are the minimum and maximum number of processors of that class. The p_{min} and p_{max} values for classes considered in our study here are shown in tables 1 through 12. Tables 7 through 12 give the first three moments of the real workload, and these moments were used to extract the model parameters, shown in tables 1 through 6.

In order to provide models for different needs, we have done this classification in two different ways. In tables 1, 3 and 5, we have grouped jobs into classes by defining the upper boundary of the classes (p_{max}) at powers of two. In tables 2, 4 and 6, we have grouped jobs into classes by defining the upper class boundaries at multiples of five, except at the low end. Also when a class has less than 1% of the jobs, we have merged it with a neighboring class that has a smaller number of jobs; and we continue this process until the combined class contains at least 1% of the jobs. For completeness, we have included in the first row of each table information on the workload where we do not separate jobs into classes.

Columns 3 through 6 in tables 1, 2, 3, 4, 5 and 6, provide the parameters of the model. Column 7 gives E_4, which is the relative discrepancy (in percentage) between the non-central fourth moment of the data and that of the model. Since the model parameters are derived by fitting only the first three non-central moments, E_4 gives an estimate of the accuracy of the fit. The last column in these tables gives the percent of jobs in the workload that is modeled in this class.

A point to note is that the first three moments of the data and those of the model are identical, a consequence of our modeling procedure. Hence the numerical values of these moments, for any row of any of the tables in this paper, can be obtained by substituting the 4 parameter values (λ_1, λ_2, n, p_1) from the table into equation 7. For the convenience of the readers we have also included these first three moments from the workload data in tables 7 through 12. The rows in these tables are organized analogous to the rows of the corresponding table in tables 1 through 6.

Often in statistical modeling, one utilizes either the observed cumulative distribution function (cdf) or the moments of the random variables, as they do not depend on the quantization interval. To graphically illustrate the discrepancy between the observed cdf and the cdf obtained from the model, we show in Figure 1 the cdf's of the inter-arrival time of jobs requesting between 9 to 16 processors. Qualitatively the curves in this figure are typical in comparisons. Quantitatively this figure shows one of the workloads with high relative error between the fourth moment of the model and that of the observed data, namely 17% as shown in table 1, line 6, column E_4. We purposely chose this example so as to give an idea graphically of the accuracy of our modeling, even when the relative error in μ_4 is high. Even in this case, our model agrees quite well with the observed cdf for large values of inter-arrival time. This is a consequence of the ability of our model to handle long-tail (i.e. over-dispersive) distributions.

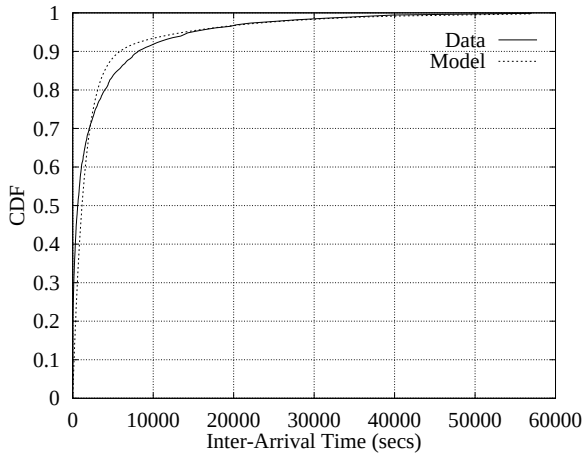


Fig. 1. Comparison of the observed cdf and that from the model for the inter-arrival time distribution for jobs requesting between 9 to 16 processors

Besides the inter-arrival time and CPU time, we have also characterized a derived quantity, which we call Scaled Wall-clock time. This is the amount of wall-clock time, in seconds, the job would need if it could use the maximum number of processors in its assigned class, while preserving its parallel efficiency. For example, if a job used 3 processors and 300 cumulative CPU seconds, i.e. running for 100 seconds concurrently on 3 processors, and we classify that job into a class having a maximum of 4 processors, then we define the Scaled Wall-clock time as 75 seconds, namely $100 * 3/4$ seconds.

Scaled Wall-clock time is often a theoretical quantity, as most of the time the parallel efficiencies are not invariant. We characterized this quantity because we feel that it is useful as an optimistic workload for various theoretical studies. The characterization of this Scaled Wall-clock time is presented in tables 5 and 6, in a format analogous to that in tables 3 and 4.

p_{min}	p_{max}	λ_1	λ_2	n	p_1	E_4	%of jobs
1	322	2.04e-04	3.80e-03	1	3.46e-02	11	100.00
1	1	1.43e-04	2.05e-03	1	8.56e-02	2	40.42
2	2	3.31e-05	9.94e-04	1	1.67e-01	5	7.17
3	4	5.58e-05	1.13e-03	1	1.55e-01	5	11.97
5	8	7.37e-05	1.88e-03	1	2.38e-01	9	11.64
9	16	6.87e-05	7.16e-04	1	1.30e-01	17	13.61
17	32	4.08e-05	4.98e-04	1	1.49e-01	12	7.89
33	64	5.67e-05	4.91e-03	1	4.79e-01	6	4.91
65	128	3.33e-05	6.07e-04	2	5.03e-01	3	1.32
129	256	4.48e-06	2.78e-04	1	2.78e-01	16	0.64
257	322	1.69e-06	1.99e-05	1	1.18e-01	17	0.37

Table 1. Hyper Erlang parameters for inter-arrival time. Unit of time is in seconds. The symbols are defined in Sections 3 and 5.

6 Synthetic Workload Generation

The first step in generating a synthetic workload from the models presented in this paper is to select the classes of jobs to be included in the synthetic workload, namely which rows or sets of p_{min} and p_{max} from table 1 or 2 are to be included. *In the program in the appendix, we denote the number of classes included in the workload generation by the variable* `nmodels`. Also we choose the size of the synthetic workload either in terms of the number of jobs to be generated in each of the chosen classes, or in terms of the total length of time for all arrivals to be generated. If one prefers to use the former criterion but only knows the total number of jobs for the synthetic workload, then the number of jobs in each of the chosen classes can be obtained by relative portioning of the total number of jobs amongst the chosen classes. These portions should be proportional to the % of jobs, i.e.the last column in the corresponding table. *In the sample program in*

p_{min}	p_{max}	λ_1	λ_2	n	p_1	E_4	%of jobs
1	322	2.04e-04	3.80e-03	1	3.46e-02	11	100.00
1	1	1.43e-04	2.05e-03	1	8.56e-02	2	40.42
2	2	3.31e-05	9.94e-04	1	1.67e-01	5	7.17
3	4	5.58e-05	1.13e-03	1	1.55e-01	5	11.97
5	10	8.48e-05	1.95e-03	1	2.06e-01	9	14.91
11	15	1.17e-05	1.58e-04	1	2.29e-01	2	1.71
16	20	6.57e-05	5.99e-04	1	1.81e-01	12	10.27
21	30	1.05e-05	5.72e-05	1	1.89e-01	5	1.30
31	35	3.78e-05	5.15e-04	1	2.00e-01	12	6.16
36	125	4.97e-05	1.23e-02	1	5.04e-01	8	4.13
126	322	3.43e-05	6.46e-04	2	3.43e-01	1	1.90

Table 2. Hyper Erlang parameters for inter-arrival time. Unit of time is in seconds. The symbols are defined in Sections 3 and 5.

p_{min}	p_{max}	λ_1	λ_2	n	p_1	E_4	%of jobs
1	322	3.40e-07	2.06e-05	1	1.48e-02	9	100.00
1	1	1.23e-04	4.72e-03	4	4.60e-01	7	40.43
2	2	8.04e-05	1.02e-01	7	3.94e-01	6	7.17
3	4	3.23e-05	1.61e-02	4	2.51e-01	9	11.97
5	8	1.17e-05	9.18e-03	1	3.76e-01	9	11.65
9	16	7.65e-06	6.40e-04	3	2.87e-01	9	13.62
17	32	2.80e-06	2.51e-04	1	3.75e-01	10	7.90
33	64	1.59e-06	7.99e-05	3	2.78e-01	9	4.92
65	128	1.19e-06	3.99e-04	6	2.07e-01	7	1.32
129	256	1.62e-05	3.15e-03	3	1.48e-01	9	0.65
257	322	5.18e-07	4.51e-04	6	1.65e-01	5	0.37

Table 3. Hyper Erlang parameters for CPU time used by the jobs. Unit of time is in seconds, and it is the cumulative CPU time used in all the processors on which the job is executing. The symbols are defined in Sections 3 and 5.

the appendix, we use the number of jobs to define the size of the workload, and comment on what needs to be modified to have the workload size based on total length of time for arrivals.

The model parameters for each of these classes are obtained from tables 1 through 6. With this information, for each job in each class to be included in the synthetic workload, one generates 3 random numbers: one for the inter-arrival time, one for the service time (i.e. CPU time used), and one for the number of CPUs needed for the job. The first of these 3 random numbers is from a Hyper Erlang distribution of Common Order (or a hyper exponential distribution, which is a special case of Hyper Erlang of Common Order), with appropriate model parameters for the particular class obtained from either

p_{min}	p_{max}	λ_1	λ_2	n	p_1	E_4	%of jobs
1	322	3.40e-07	2.06e-05	1	1.48e-02	9	100.00
1	1	1.23e-04	4.72e-03	4	4.60e-01	7	40.43
2	2	8.04e-05	1.02e-01	7	3.94e-01	6	7.17
3	4	3.23e-05	1.61e-02	4	2.51e-01	9	11.97
5	10	8.89e-06	3.21e-04	1	3.28e-01	8	14.91
11	15	1.34e-05	3.86e-03	5	1.68e-01	6	1.71
16	20	6.60e-06	1.29e-02	3	3.35e-01	10	10.28
21	30	3.66e-06	1.96e-04	2	2.10e-01	8	1.31
31	35	2.50e-06	6.90e-05	1	3.81e-01	11	6.17
36	125	1.45e-06	1.95e-04	3	2.86e-01	7	4.13
126	322	3.42e-07	1.99e-04	2	1.75e-01	3	1.91

Table 4. Hyper Erlang parameters for CPU time used by the jobs. Unit of time is in seconds, and it is the cumulative CPU time used in all the processors on which the job is executing. The symbols are defined in Sections 3 and 5.

p_{min}	p_{max}	λ_1	λ_2	n	p_1	E_4	%of jobs
1	322	6.80e-07	4.12e-05	1	1.48e-02	9	100.00
1	1	1.23e-04	4.72e-03	4	4.60e-01	7	40.43
2	2	1.61e-04	2.05e-01	7	3.94e-01	6	7.17
3	4	6.46e-05	3.23e-02	4	2.51e-01	9	11.97
5	8	2.33e-05	1.84e-02	1	3.76e-01	9	11.65
9	16	1.53e-05	1.28e-03	3	2.87e-01	9	13.62
17	32	5.60e-06	5.01e-04	1	3.75e-01	10	7.90
33	64	3.17e-06	1.60e-04	3	2.78e-01	9	4.92
65	128	2.39e-06	7.97e-04	6	2.07e-01	7	1.32
129	256	3.24e-05	6.29e-03	3	1.48e-01	9	0.65
257	322	1.04e-06	9.01e-04	6	1.65e-01	5	0.37

Table 5. Hyper Erlang parameters for Scaled Wall-clock time usedd by the jobs. Unit of time is in seconds. The symbols are defined in Sections 3 and 5.

table 1 or 2. Similarly, the second random number is also from a Hyper Erlang of Common Order (or hyper exponential) distribution, with appropriate model parameters for the particular class obtained from the corresponding table. The last random number, representing the number of CPUs needed, is a uniform random number that has been scaled to remain within the range of p_{min} and p_{max}. Once these 3 random numbers are obtained, one has the inter-arrival time, the service time, and the number of processors required by the job, which are commensurate with the workload model. This process continues for other jobs in the class until the desired amount of workload has been generated, and then the process is repeated for each of the other chosen classes in the synthetic workload.

p_{min}	p_{max}	λ_1	λ_2	n	p_1	E_4	%of jobs
1	322	6.80e-07	4.12e-05	1	1.48e-02	9	100.00
1	1	1.23e-04	4.72e-03	4	4.60e-01	7	40.43
2	2	1.61e-04	2.05e-01	7	3.94e-01	6	7.17
3	4	6.46e-05	3.23e-02	4	2.51e-01	9	11.97
5	10	1.78e-05	6.41e-04	1	3.28e-01	8	14.91
11	15	2.67e-05	7.71e-03	5	1.68e-01	6	1.71
16	20	1.32e-05	2.58e-02	3	3.35e-01	10	10.28
21	30	7.31e-06	3.92e-04	2	2.10e-01	8	1.31
31	35	5.00e-06	1.38e-04	1	3.81e-01	11	6.17
36	125	2.89e-06	3.91e-04	3	2.86e-01	7	4.13
126	322	6.84e-07	3.98e-04	2	1.75e-01	3	1.91

Table 6. Hyper Erlang parameters for Scaled Wall-clock time used by the jobs. Unit of time is in seconds. The symbols are defined in Sections 3 and 5.

p_{min}	p_{max}	μ_1	μ_2	μ_3	%of jobs
1	322	4.234e+02	1.794e+06	2.449e+10	99.99
1	1	1.046e+03	8.858e+06	1.778e+11	40.42
2	2	5.893e+03	3.070e+08	2.766e+13	7.17
3	4	3.529e+03	1.011e+08	5.371e+12	11.97
5	8	3.635e+03	8.808e+07	3.569e+12	11.64
9	16	3.107e+03	5.842e+07	2.416e+12	13.61
17	32	5.358e+03	1.858e+08	1.320e+13	7.89
33	64	8.544e+03	2.975e+08	1.573e+13	4.91
65	128	3.182e+04	2.727e+09	3.265e+14	1.32
129	256	6.471e+04	2.775e+10	1.857e+16	0.64
257	322	1.142e+05	8.698e+10	1.469e+17	0.37

Table 7. First three non-central moments of the of inter-arrival time distribution in the observed workload. Unit of time is in seconds. The symbols are defined in Sections 3 and 5. The model parameters extracted from these moments are given in table 1.

The only remaining algorithm that needs to be discussed here is a method to generate random numbers from a Hyper Erlang distribution of Common Order with appropriate model parameters for a particular class. A way to generate a sequence of random numbers, $\{x_j\}$, based on a non-uniform distribution, is to solve for x_j in

$$y_j = cdf(x_j) \tag{19}$$

where $\{y_j\}$, is a uniformly distributed sequence of random numbers, and $cdf(x_j)$ is the cumulative distribution function for the statistical model. For the Hyper Erlang distribution of Common Order, the cdf is obtained from equation 5 by inverse Laplace transform followed by a simple integral. After some straight-forward algebra, one ar-

p_{min}	p_{max}	μ_1	μ_2	μ_3	%of jobs
1	322	4.234e+02	1.794e+06	2.449e+10	99.99
1	1	1.046e+03	8.858e+06	1.778e+11	40.42
2	2	5.893e+03	3.070e+08	2.766e+13	7.17
3	4	3.529e+03	1.011e+08	5.371e+12	11.97
5	10	2.838e+03	5.774e+07	2.027e+12	14.91
11	15	2.447e+04	3.410e+09	8.593e+14	1.71
16	20	4.118e+03	8.825e+07	3.842e+12	10.27
21	30	3.207e+04	3.891e+09	9.927e+14	1.30
31	35	6.859e+03	2.869e+08	2.234e+13	6.16
36	125	1.019e+04	4.085e+08	2.467e+13	4.13
126	322	2.205e+04	1.759e+09	2.040e+14	1.90

Table 8. First three non-central moments of the of inter-arrival time distribution in the observed workload. Unit of time is in seconds. The symbols are defined in Sections 3 and 5. The model parameters extracted from these moments are given in table 2.

p_{min}	p_{max}	μ_1	μ_2	μ_3	%of jobs
1	322	9.126e+04	2.601e+11	2.255e+18	100.00
1	1	1.537e+04	6.051e+08	2.941e+13	40.43
2	2	3.432e+04	3.412e+09	3.821e+14	7.17
3	4	3.130e+04	4.814e+09	8.939e+14	11.97
5	8	3.230e+04	5.526e+09	1.421e+15	11.65
9	16	1.160e+05	5.895e+10	3.852e+16	13.62
17	32	1.365e+05	9.567e+10	1.024e+17	7.90
33	64	5.527e+05	1.328e+12	4.184e+18	4.92
65	128	1.052e+06	6.107e+12	4.097e+19	1.32
129	256	2.816e+04	6.754e+09	2.085e+15	0.65
257	322	1.928e+06	2.592e+13	4.005e+20	0.37

Table 9. First three non-central moments of the distribution of the CPU time used by the jobs in the observed workload. Unit of time is in seconds, and it is the cumulative CPU time used in all the processors on which the job is executing. The symbols are defined in Section 5. The model parameters extracted from these moments are given in table 3.

rives at an analytical expression for the cdf of the models used in this paper, namely

$$cdf(x_j) = 1 - \sum_{i=1}^{2} p_i e^{-\lambda_i x_j} \Big(\sum_{k=0}^{n-1} \frac{(x_j \lambda_i)^k}{k!} \Big) \tag{20}$$

From this expression of the cdf and a uniform random number generator, U[0,1], the random variables $\{x_j\}$ that are commensurate with equation 5 can be obtained by solving for x_j in equation 19. Since a $cdf(x)$ is a monotonic function that is bounded by 0

p_{min}	p_{max}	μ_1	μ_2	μ_3	%of jobs
1	322	9.126e+04	2.601e+11	2.255e+18	100.00
1	1	1.537e+04	6.051e+08	2.941e+13	40.43
2	2	3.432e+04	3.412e+09	3.821e+14	7.17
3	4	3.130e+04	4.814e+09	8.939e+14	11.97
5	10	3.898e+04	8.313e+09	2.802e+15	14.91
11	15	6.409e+04	2.829e+10	1.482e+16	1.71
16	20	1.526e+05	9.239e+10	6.998e+16	10.28
21	30	1.230e+05	9.442e+10	1.032e+17	1.31
31	35	1.613e+05	1.223e+11	1.465e+17	6.17
36	125	6.043e+05	1.640e+12	5.665e+18	4.13
126	322	1.032e+06	8.971e+12	1.049e+20	1.91

Table 10. First three non-central moments of the distribution of the CPU time used by the jobs in the observed workload. Unit of time is in seconds, and it is the cumulative CPU time used in all the processors on which the job is executing. The symbols are defined in Sections 3 and 5. The model parameters extracted from these moments are given in table 4.

p_{min}	p_{max}	μ_1	μ_2	μ_3	%of jobs
1	322	4.563e+04	6.504e+10	2.819e+17	100.00
1	1	1.537e+04	6.051e+08	2.941e+13	40.43
2	2	1.716e+04	8.530e+08	4.776e+13	7.17
3	4	1.565e+04	1.204e+09	1.117e+14	11.97
5	8	1.615e+04	1.382e+09	1.777e+14	11.65
9	16	5.801e+04	1.474e+10	4.815e+15	13.62
17	32	6.824e+04	2.392e+10	1.280e+16	7.90
33	64	2.763e+05	3.319e+11	5.230e+17	4.92
65	128	5.262e+05	1.527e+12	5.121e+18	1.32
129	256	1.408e+04	1.688e+09	2.606e+14	0.65
257	322	9.641e+05	6.480e+12	5.006e+19	0.37

Table 11. First three non-central moments of the distribution of the Scaled Wall-clock time used by the jobs in the observed workload. Unit of time is in seconds, and it is the cumulative CPU time used in all the processors on which the job is executing. The symbols are defined in Sections 3 and 5. The model parameters extracted from these moments are given in table 5.

and 1, equation 19 can easily be solved by using any of the simple root finding techniques such as interval halving, etc.

Among the functions shown in the appendix,

- function **H_Er_cdf()** computes the $cdf(x_j)$ based on equation 20,
- function **solve_bisec()** finds the root of an equation by interval halving, and
- function **ur_to_HER_r()** solves equation 19 for a given y.

p_{min}	p_{max}	μ_1	μ_2	μ_3	%of jobs
1	322	4.563e+04	6.504e+10	2.819e+17	100.00
1	1	1.537e+04	6.051e+08	2.941e+13	40.43
2	2	1.716e+04	8.530e+08	4.776e+13	7.17
3	4	1.565e+04	1.204e+09	1.117e+14	11.97
5	10	1.949e+04	2.078e+09	3.502e+14	14.91
11	15	3.205e+04	7.073e+09	1.852e+15	1.71
16	20	7.631e+04	2.310e+10	8.748e+15	10.28
21	30	6.148e+04	2.361e+10	1.290e+16	1.31
31	35	8.067e+04	3.056e+10	1.832e+16	6.17
36	125	3.021e+05	4.100e+11	7.081e+17	4.13
126	322	5.158e+05	2.243e+12	1.311e+19	1.91

Table 12. First three non-central moments of the distribution of the Scaled Wall-clock time used by the jobs in the observed workload. Unit of time is in seconds, and it is the cumulative CPU time used in all the processors on which the job is executing. The symbols are defined in Sections 3 and 5. The model parameters extracted from these moments are given in table 6.

7 Conclusions

In this paper we have characterized in a compact model the workload of a large supercomputing center. For this characterization we have chosen a phase type distribution, namely Hyper Erlang distribution of Common Order, that exactly fits the first three moments of the observed workload, and when appropriate, reduces to a simpler model such as exponential, hyper exponential or Erlang. The important findings of our study here are:

- The observed workload is quite dispersive, namely, the coefficient of variation is greater than one, and cannot be adequately represented by an Erlang or exponential distribution. A point to note is that, typically the number of users in these MPPs are relatively small, hence there is no a priori reason to expect an under-dispersive distribution to be able to represent the observed inter-arrival time or service time. Thus for theoretical studies and simulations dealing with scheduling of jobs in MPPs, it is more desirable, in our opinion, to use hyper exponential or Hyper Erlang distributions rather than simple exponential or Erlang distributions.

- Even though the inter-arrival time can often be modeled by hyper exponential distributions, the service time and the Scaled Wall-clock time often require Hyper Erlang distributions of order greater than one.

Another point to note is that, in light of the number of jobs we have sampled in each class, the relative errors in the fourth moment (shown as percentage in column 7 of the tables 1 through 6) are sometimes relatively high. However, as shown by Schassberger [10, 11, 12, 13], a number of steady state properties such as mean queue lengths, average waiting times, etc., depend only on the first few moments of the random distributions,

and since we want a model that is simple enough to be tractable in theoretical studies of scheduling algorithms and parameters, we feel that our model for characterization of the workloads in MPPs is quite adequate.

8 Acknowledgments

Joseph Riordan's portion of this research was funded by the Cornell Theory Center, which receives major funding from NSF (the National Science Foundation) and New York State, with additional support from the National Center for Research Resources at the NIH (National Institute of Health), IBM Corporation, and other members of the center's Corporate Partnership Program.

9 Appendix

The program shown in this section can be used to generate synthetic workloads for model parameters given in the results section of this paper. The overall algorithm used in this program is described in section 6. Here we point out a few specifics relevant to the accurate generation of workloads.

- This program needs a uniform random number generator generating random numbers in the interval of 0 to 1. This is assumed to be obtained from a library function called **drand48()**. If this function is not available in your computer, or if you have a different favorite one, replace the call to **drand48()** with an appropriate call. The use of a good uniform random number generator is essential to the quality of the generated workloads. In our simulations, instead of **drand48()**, we used a very high quality, but machine dependent random number generator.

- A point to note is that, for each job we need 3 random numbers, drawn from independent uniform random number pools. We achieve this from a high quality, single uniform random number generator by creating 3 pools or streams of uniform random numbers. This is done in the function **str_rand()**.

- The tolerance for solution parameter, **tol**, in function **solve_bisec()** should be tight.

- At the end of the program we also compute the non-central moments μ_k for each of the classes, from the jobs generated in the synthetic workload. These should be close to the corresponding numbers from tables 7 through 12.

This program generates a workload based on one class with $p_{min} = 65$ and $p_{max} = 128$, and the CPU time is used as the service time. The parameters correspond to row 9 in tables 1 and 3. The μ_k should be compared with row 9 of tables 7 and 9.

```
/* This program generates a stream of jobs based on the
 *   models described in the main text of this paper.
 * To keep this program simple, we have elimated all
```

```
 *   the checkings of the input.  If you use the
 *   program often, we strongly recommend you to
 *   add code to catch the error conditions. This
 *   program is meant for illustration purposes only.
 */
#include <stdio.h>
#include <math.h>

#define min(a,b) (((a) < (b)) ? (a) : (b))
#define RANSTREAM 3
#define RANPOOLSIZE 10000

typedef struct
{ double p1,lambda1,lambda2,ur;
        int    order;
}  MODEL_DATA;

/* Global Variables: */
double ran_pool[RANSTREAM][RANPOOLSIZE];
int    ran_remain[RANSTREAM];

/* --------- Function prototypes: ------------------ */
double ur_to_HER_r( double ur, MODEL_DATA model);

double H_Er_cdf(double x, void *model);

void solve_bisec( double xleft_0,
                  double xright_0,
                  double tol,
                  double *x,
                  double *y,
                  void *model_data,
                  double (*f)(double, void *),
                  int *rc);

double fn_to_solve(double x, void *);

int    fac(int k);

double str_rand(int str);

/* ================================================= */
main(int argc, char **argv)
{ MODEL_DATA inter_arrival_model, cpu_model;
  double inter_arrival_mu1, inter_arrival_mu2,
```

```
inter_arrival_mu3;
 double cpu_mu1, cpu_mu2, cpu_mu3;
 double simtime, inter_arrival_time, cpu_time_used;
 double dtemp;

 int njobs,  pmin,pmax,p_needed;
 int ijob,   nmodels, imodel;
 int i,j;

 /* One needs to change the value of nmodels here to
  * reflect  the number of models (namely number of
  * lines from Tables 1 through 6 in the Results
  * section of this paper) concurrently used for
  *  generating the job stream.
  */
 nmodels=1;

for(imodel=0; imodel<nmodels; imodel++)
{ simtime=0;

  /* The following section describes the parameters
   * of model imodel, and should be modified to
   * correspond to your chosen model.
   */
  pmin = 65;
  pmax = 128;

  inter_arrival_model.lambda1 = 3.33e-05;
  inter_arrival_model.lambda2 = 6.07e-04;
  inter_arrival_model.order=2;
  inter_arrival_model.p1=5.03e-01;

  cpu_model.lambda1 = 1.19e-06;
  cpu_model.lambda2 = 3.99e-04;
  cpu_model.order=6;
  cpu_model.p1=2.07e-01;

  njobs=10000;
  /* End of the Input section */

  inter_arrival_mu1=0;
  inter_arrival_mu2=0;
  inter_arrival_mu3=0;

  cpu_mu1=0;
```

```
cpu_mu2=0;
cpu_mu3=0;

ijob=0;

/* Here we are using the number of jobs (njobs) as the
 * termination criterion.  One may choose to use
 * simtime (i.e. total length of time represented
 * in this workload) as a termination criterion.
 * In that case, use simtime in the following while
 * statement.
 */

while( ijob < njobs)
{   ijob++;

    dtemp=str_rand(0);
    inter_arrival_time =
              ur_to_HER_r(dtemp,inter_arrival_model);
    simtime += inter_arrival_time;
    inter_arrival_mu1 +=inter_arrival_time;
    inter_arrival_mu2 +=(inter_arrival_time *
                         inter_arrival_time);
    inter_arrival_mu3 +=(inter_arrival_time *
                         inter_arrival_time *
                         inter_arrival_time);


    dtemp=str_rand(1);
    cpu_time_used = ur_to_HER_r(dtemp,cpu_model);
    cpu_mu1 +=  cpu_time_used;
    cpu_mu2 += (cpu_time_used  *
                cpu_time_used);
    cpu_mu3 += (cpu_time_used  *
                cpu_time_used  *
                cpu_time_used);


    dtemp=str_rand(2);
    dtemp *= (pmax-pmin+1);
    j=dtemp;
    if( ((double) j) < dtemp ) { p_needed = j + pmin; }
    else   { p_needed = j + pmin - 1; }

    /* One needs to replace the following section
```

```
* (enclosed by #if and #endif) with appropriate
* statements to  output the job stream information.
* If multiple models are concurrently used, it
* might be desirable to sort the final output by
* job arrival time.
*/

#if 0
    if(ijob < 50 )
    { printf( "Job used %d CPUs and %e cummulative CPUtime",
                p_needed,   cpu_time_used);
      printf( " arrived at time %e \n", simtime);
    }
#endif
    } /* end of the while loop */

      printf(" For Model %d :\n",imodel);
      inter_arrival_mu1 /= (double)ijob;
      inter_arrival_mu2 /= (double)ijob;
      inter_arrival_mu3 /= (double)ijob;
      printf("Inter-arrival Time Moments of the jobs are \n");
      printf( " %d  mu1 = %e  "
              "     mu2 = %e  "
              "     mu3 = %e \n",
            ijob, inter_arrival_mu1,
                  inter_arrival_mu2,
                  inter_arrival_mu3 );

      cpu_mu1 /= (double)ijob;
      cpu_mu2 /= (double)ijob;
      cpu_mu3 /= (double)ijob;
      printf("CPU Time Moments of the jobs are \n");
      printf(" %d mu1 = %e  "
                 "mu2 = %e "
                 "mu3 = %e\n",
             ijob, cpu_mu1, cpu_mu2, cpu_mu3 );

  } /* end of loop over models */
}   /* end of main program */

/* ================================================ */
double ur_to_HER_r(double ur, MODEL_DATA m)
{  double xleft, xright;
   double x,y;
   int rc;
```

```
   m.ur = ur;
   xleft = 1.e-20;   /* a small +ve number */
   xright = -log(1.0-ur) / min(m.lambda1,m.lambda2);
   xright *= 100;

   solve_bisec(xleft, xright,
               (double) 1.e-12,
               &x, &y, &m,
               fn_to_solve, &rc);
   if(rc !=0 )
   {  printf("Unable to find the Random Number for
                               ur = %e \n",m.ur);
   }
   return(x);
}
/* ================================================= */
double fn_to_solve(double x, void *model_data)
{  MODEL_DATA *m;

   m = (MODEL_DATA *) model_data;
   return( H_Er_cdf(x, model_data)  -  m->ur );
}
/* ================================================= */
double H_Er_cdf(double x, void *model_data)
{  MODEL_DATA *p;
   double cdf, t1,t2, dtemp;
   int k;
   t1=0;
   t2=0;
   p = (MODEL_DATA *) model_data;

   for(k=0;  k< p->order;  k++)
   {  dtemp=fac(k);
      t1 += (pow(x*p->lambda1,k)/dtemp);
      t2 += (pow(x*p->lambda2,k)/dtemp);
   }
   cdf = 1.0 -   p->p1 *exp(-(p->lambda1*x))*t1
             - (1-p->p1)*exp(-(p->lambda2*x))*t2;
   return(cdf);
 }
/* ================================================= */
void solve_bisec(double xleft_0,
                       xright_0,
                       double tol,
```

```
                        double *x,
                        double *y,
                        void *model_data,
                        double (*f)(double, void *),
                        int *rc)

{  double xleft, yleft,
          xright, yright,
          xnext,ynext;
   int   ic;

   xleft=xleft_0; xright=xright_0;
   yleft=(*f)(xleft, model_data);
   yright=(*f)(xright,model_data);
   *rc=1; ic=0;

   if( yleft*yright  > 0.0 ) { return; }
   /* No solution in this interval */

   while(ic < 100000 )
   {  ic++;
      xnext=(xleft+xright)/2.0;
      ynext=(*f)(xnext,model_data);
      if( ynext*yright >  0.0 )
      {   xright=xnext;
          yright=ynext;
      }
      else
      {   xleft=xnext;
          yleft=ynext;
      }
      if( fabs(ynext)  <  tol )
      {   *x=xnext; *y=ynext; *rc=0; return;}
   }
}
/* ================================================= */
int fac(int k)
{  int i,j;
   j=1;

   if(k==0) return (j);
   j=1;
   for(i=1;i<=k;i++) j*=i;
   return(j);
}
```

```
/* ================================================= */
double str_rand(int strm)
{  int i;
   if( ran_remain[strm] <=  0 )
   {  for(i=0;i<RANPOOLSIZE;i++)
          ran_pool[strm][i]=drand48();
          ran_remain[strm]=RANPOOLSIZE;
   }
   ran_remain[strm]--;
   return( ran_pool[strm] [RANPOOLSIZE-ran_remain[strm]]) ;
}
```

References

1. A. Tucker and A. Gupta, "Process control and scheduling issues for multiprogrammed shared-memory multiprocessors," in *Proceedings of the 12th. ACM Symposium on Operating Systems Principle*, pp. 159–166, 1989.
2. D. Feitelson and L. Rudolph, "Mapping and scheduling in a shared parallel environment using distributed hierarchical control," in *Intl. Conf. Parallel Processing*, pp. 1–8, 1990.
3. D. G. Feitelson, "Packing schemes for gang scheduling," in *Job Scheduling Strategies for Parallel Processing Springer-Verlag, LNCS Vol. 1162, Apr 1996*, pp. 89–110, 1996.
4. F. Wang, H. Franke, M. Papaefthymiou, P. Pattnaik, L. Rudolph, and M. S. Squillante, "A gang scheduling design for multiprogrammed parallel computing environments," in *Job Scheduling Strategies for Parallel Processing Springer-Verlag, LNCS Vol. 1162, Apr 1996*, pp. 111–125, 1996.
5. M. S. Squillante, F. Wang, and M. Papaefthymiou, "An analysis of gang scheduling for multiprogrammed parallel computing environments," in *Proceedings of the Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pp. 89–98, 1996.
6. H. Franke, P. Pattnaik, and L. Rudolph, "Gang scheduling for highly efficient distributed multiprocessor systems," in *Proceedings of the 6th Symposium on the Frontiers of Massively Parallel Computation, Annapolis, MD*, pp. 1–9, 1996.
7. D. R. Cox, "A use of complex probabilities in the theory of stochastic process," *Proc. Cambridge Philos. Soc*, 1955.
8. M. F. Neuts, *Matrix-Geometric Solutions in Stochastic Models*. The John Hopkins University Press, 1981.
9. M. A. Johnson and M. R. Taaffe, "Matching moments to phase distributions," *Commun. Stat. Stochastics Models*, vol. 5, pp. 711–743, 1989.
10. R. Schassberger, "Insensitivity of steady-state distributions of generalized semi-markov processes i," *Ann. Prob.*, pp. 87–99, 1977.
11. R. Schassberger, "Insensitivity of steady-state distributions of generalized semi-markov processes ii," *Ann. Prob.*, pp. 85–93, 1978.
12. R. Schassberger, "Insensitivity of steady-state distributions of generalized semi-markov processes with speeds," *Adv. Appl. Prob.*, pp. 836–851, 1978.
13. R. Schassberger, "The insensitivity of stationary probabilities in networks of queues," *Adv. Appl. Prob.*, pp. 906–912, 1978.

This article was processed using the LaTeX macro package with LLNCS style