Collecting Unused Processing Capacity: An Analysis of Transient Distributed Systems

Leonard Kleinrock, *Fellow, IEEE,* and Willard Korfhage, *Member, IEEE*

Abstract—**Distributed systems frequently have large numbers of idle computers and workstations. If we could make use of these, then considerable computing power could be harnessed at low cost. We analyze such systems using Brownian motion with drift to model the execution of a program distributed over the idle computers in a network of idle and busy processors, determining how the use of these "transient" processors affects a program's execution time. We find the probability density of a programs finishing time on both single and multiple transient processors, explore these results for qualitative insight, and suggest some approximations for the finishing time probability density that may be useful.**

Index Terms— **Brownian motion, distributed processing, idle processors, performance analysis, transient processors.**

I. Introduction

DISTRIBUTED systems frequently have large numbers of idle computers and workstations. If we could use these, then considerable computing power could be harnessed at low cost. In this paper, we model program execution on a network of workstations, some idle and some not. Because we use only the idle time on the processors, they are not always available for use. Hence, we call these machines "transient" processors. Using a direct analysis and a cumulative alternating renewal process analysis both provide simple expressions for the probability density of the program finishing time on a single transient processor. We extend the cumulative alternating renewal process into a Brownian motion with drift model of the finishing time density on multiple processors. We then examine the properties of the finishing time probability density to achieve some qualitative insight into the results. Finally, we suggest several approximations for the finishing time probability density, and discuss under what conditions they can be used.

A. Background

Networks of computers are common in business and research environments throughout the world. Local area networks, which were originally introduced to ease data and device sharing, have grown in speed, sophistication, and size to the point that effective distributed processing can be performed on them. These networks vary in size from a handful of personal computers on a low-speed network, to networks consisting of thousands of workstations and a variety of larger machines on a high-speed, fiber-optic network. As a typical example, consider a network of workstations on a high-speed network in a research laboratory. Not only are there many machines, well connected by the network, but the users are likely to demand more and more computing power as the applications grow.

Networks of workstations have grown in spite of theoretical considerations that would discourage them. It is well known in queueing theory [2] that a single server of large capacity shared by many users provides better response time than many smaller servers with the same total capacity. Thus we might expect that a mainframe will provide faster service to its users than a network of workstations. Of course, one may argue that for the same amount of money one can buy much more workstation capacity than mainframe capacity, but even then we would like to make the best use possible of our computing resources. This implies that a network of workstations would have improved performance (overall) if the workstations were considered part of one processor pool, available for the execution of all programs. Some systems, such as Amoeba [3] provide a pool of processors strictly as compute servers. We, however, would like to retain the usual use of workstations on the network in addition to considering them part of the processor pool.

The solution to this is idle time. On these networks, we often have the situation that many of the personal computers and workstations are sitting idle, waiting for their users, and thus being wasted ([4], [5]). If we could recover this wasted time for useful processing, then we would have considerable computing power available to us at low cost. We refer to these processors, which are sometimes busy and sometimes not, as *transient* processors.

Whether this is technically feasible or not depends on a variety of factors, such as the properties of the communications medium, the properties of the computers, and the statistical characteristics of the user population.[1] In all systems of this type, one concern is that the "owner" of a machine should not see any degradation in performance because of the background programs. Any background computation should be aborted

Manuscript received July 1, 1990; revised July 29, 1991. This work was supported by the Defense Advanced Research Projects Agency under Contract MDA 903-82-C0064, Advanced Teleprocessing Systems, and Contract MDA 903-87-C0663, Parallel Systems Laboratory. W. Korfhage was also supported by an HP/AEA fellowship. This work was done while W. Korfhage was at U.C.L.A.

L. Kleinrock is with the Department of Computer Science, University of Calfiornia, Los Angeles.

W. Korfhage is with the Department of Computer Science, Polytechnic University, Brooklyn, NY 11201.

IEEE Log Number 9208484.

[1] There are also other important but nontechnological factors, such as people's resistance to the use of "their" machine, that would determine if and how a distributed system would be implemented. This paper does not examine such matters.

when user activity is detected, and not restarted until the system is sure that the machine is idle.

There has been much research on systems that make use of idle workstations through load balancing and process migration; we mention here a few such systems. Most of these provide remote program execution facilities (e.g., run a compiler at another machine rather than on the local workstation), rather than specifically supporting distributed computations. We make note of those systems that have been designed specifically for distributed computations.

Alonso and Cova [6] discuss a load balancing system for workstations in which a workstation tries to execute a job remotely if the local processor load is greater than some "High Mark," and accepts jobs from other workstations if the local processor load is less than some "Low Mark." This allows for a continuum of migration policies.

At U.C.L.A., the Benevolent Bandit Laboratory (BBL) [7] runs distributed computations under MS-DOS on a network of IBM PC-AT's. A special shell runs on each machine, and when a machine is at the operating system prompt level (as opposed to running a program), it is available for use. If someone starts to use a machine currently part of a background, distributed computation, the system can select and start a replacement machine from the pool of idle processors. Because the system was also intended for the investigation of distributed algorithms, special features, such as the ability to mimic any connection topology, and some distributed debugging facilities, have been built in.

Lyle and Lu [8] describe a simple remote program execution facility that operates at the shell level, like BBL, rather than the kernel level. This has the advantage of being simple to implement, yet still uses idle workstations.

Condor ([9], [10], [4]) is a very successful remote program execution facility running on workstations at the University of Wisconsin. The developers of the system have made a number of useful measurements of workstation behavior in [10].

The Butler system [11], running on Andrew workstations at Carnegie-Mellon also provides remote program execution facilities. The system uses this to run *gypsy servers*, which are network servers that run on idle workstations instead of on a fixed machine.

Stumm [12] discusses a remote program execution and task migration facility for the V kernel. His paper discusses various issues, such as the migration policy, and offers thoughts on using the system for distributed computations.

The *Worm* program [13] was developed at Xerox PARC as an experiment in distributed processing. Worms prowled the network, collecting idle workstations and using them to perform some action, typically displaying a message or running a diagnostic program.

There have also been ad-hoc attempts to use the idle time on processors. Dr. Tim Shimeall [14], during his dissertation research, wrote a program "polite" that ran a software analysis program on workstations when no one was logged in and suspended the program when the workstation was being used. He reports that he finished nearly 10 CPU years of work in about 6 months on 20 workstations using this program. But again, this was very much a simple remote job execution facility, put together out of need, and it was never analyzed.

B. Outline

Section II discusses our model of the network. Section III gives a simple analysis of the average time for a program to finish. Sections IV and V then develop three models of the network and analyze them to find the distribution of time to finish a fixed amount of work.

The first model, in Section IV-B, is a single processor model with general *available* and *nonavailable* times. We examine the number of nonavailable periods interrupting a program, and from this we find the Laplace transform of the distribution of the time to finish a program (response time), and then the mean and variance of the response time.

The second model, in Section IV-C, is also a single processor model, but it is analyzed as a cumulative, alternating renewal process. We find that the asymptotic distribution of the accumulated work (over a long period of time) is Gaussian, with simple expressions for the mean and variance.

The third model, in Section V-A, handles multiple processors and views the amount of work done over time as Brownian motion with drift. We scale the asymptotic mean and variance of the accumulated work from the second, single processor model to the case of M processors, and use this as the mean and variance of the Brownian motion with drift. From this we get the probability density of the time to finish a fixed amount of work on M processors. The mean and the variance agree very closely, for $M = 1$, with the first model.

Finally, Section VI contains the conclusions. These three models offer an approach to predicting performance of distributed programs on transient processors. By relaxing some of our assumptions, more sophisticated models could be derived from those described here.

II. The Model of the Network and the Workload

A. The Network

Assume that we have a network of M identical processors, each of which has a capacity to complete one minute of work per minute. A processor alternates between a *nonavailable state* (signified by *n* or *na*), when the owner is using it (e.g., typing at the keyboard), and an *available state* (signified by *a* or *av*), when it is sitting idle. The lengths of nonavailable periods are independent and identically distributed (i.i.d.) random variables from distribution $N(t)$, with mean t_n, variance σ_n^2, and corresponding density $n(t)$; we allow any general distribution for $N(t)$, unless otherwise specified. Likewise, available periods are i.i.d. random variables from a general distribution $A(t)$ with mean t_a, variance σ_a^2, and density $a(t)$. The available and nonavailable periods are mutually independent.

B. The Distributed Program Workload

We model a program as consisting of multiple stages of work, each of which must be completed before the start of the next, and each of which represents a deterministic amount

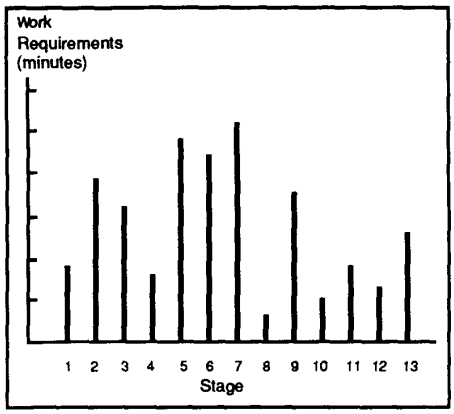


Fig. 1. Execution time profile of an algorithm.

of work (Fig. 1). The time to finish a program is the sum of the times to complete the individual stages. We assume that the time to finish a stage depends only on the amount of work in that stage, and is independent of the other stages. This means that the probability distribution of the *total* time to finish a program is the convolution of the distributions of the individual stage finishing times. Assuming that the network characteristics do not change during the execution of the program, then from the analysis of the time to finish a single stage requiring W minutes of work we can find the finishing time probability density function of all other stages, and from there, the finishing time probability density function of the program as a whole.

We make the simplifying assumption that the work in any stage is infinitely divisible—it can always be divided evenly among all available processors. Note, however, that this assumption does not always obscure program behavior. Some programs, such as the simulators used for the models of this paper, are, in fact, composed of very many independent tasks, and work is always available for any idle processor. In such cases, this assumption captures the program behavior and is not a simplification at all. In addition, in a system with multiple users on many machines, the aggregation of independent jobs, each with a number of tasks, from many users will yield an overall workload that tends to look like many independent tasks (or, at least, enough tasks to keep idle machines busy), and this would fit well with our assumption. We will explore this more in future work.

Another simplifying assumption we make is to ignore overhead that occurs in a real system (e.g., communication delays, processing delays), and thus our model provides an optimistic bound on system performance. Some techniques for removing this assumption are mentioned in [1].

C. What We Seek

The purpose of this paper is to find $f(t)$, the probability density function (pdf) of a program's finishing time (i.e., response time). Also of interest are its mean, $\bar{f}$, and its variance, σ_f^2.

In the process of finding these, we will need another function, namely, the pdf of the amount of work accumulated (i.e., completed) by a processor or network of processors over time. We denote this by $y(u \mid t)$, the probability density that after t minutes of time have elapsed, the processor (or processors) under consideration has accumulated u minutes of work. This function has mean $\mathrm{E}[y(u \mid t)]$ and variance $\mathrm{Var}[y(u \mid t)]$.

D. Notation

We use the abbreviations "PDF" to stand for "probability distribution function." Typically we use capital letters for a PDF and the corresponding lower case letters for a pdf. If $F(x)$ is a PDF, then its pdf is $f(x) = (\partial/\partial x)F(x)$.

E. Example Parameters

Mutka and Livny, [10], made actual measurements of a network of transient processors, and they developed models for the available and nonavailable period densities to fit these measurements. From their results, we derive two examples that we use throughout this paper.

The model they used for the available time PDF was a 3-stage hyperexponential distribution:

$$\begin{aligned} A(t) &= \mathrm{P}\ [\text{length of an available period} \le t] \\ &= 0.33(1 - e^{-(t/3)}) + 0.4(1 - e^{-(t/25)}) \\ &\quad + 0.27(1 - e^{-(t/300)}) \qquad t \ge 0 \end{aligned} \tag{1}$$

which has mean $t_a = 91$ min and variance $\sigma_a^2 = 40225$ min^2.

For the nonavailable time distribution, $N(t) = \mathrm{P}$[length of a nonavailable period $\le t$], they used a shifted 2-stage hyperexponential distribution:

$$N(t) = \begin{cases} 0.7(1 - e^{-(t/7)}) + 0.3(1 - e^{-(t/55)}) & \text{if } t \ge 7 \\ 0 & \text{if } 0 \le t < 7 \end{cases} \tag{2}$$

which has mean $t_n = 31.305$ min and variance $\sigma_n^2 = 2131.83$ min^2. The 7 min shift in the distribution arises because a processor was not declared idle until 7 idle minutes had elapsed.

We use Mutka and Livny's distributions wherever possible in our examples, but frequently we assume exponentially distributed available and nonavailable periods; at such times, we take the means of these exponential distributions to be the numbers given above. The use of exponential distributions instead of hyperexponential distributions will not affect any means that we derive, but any variances that we find will be lower than if we had used Mutka and Livny's distributions.

Regardless of which distributions we use, we take $W = 1000$ min and $M = 1$ for single processor examples, and, $W = 10000$ min (almost 7 days) and $M = 100$ for most multiple processor examples. The reason for the large values of W is explained below.

F. Related Work

One approach to analyzing a single processor system is to use queueing with vacation as a model. In such a system, the

queueing server is subject to randomly occurring stoppages lasting for random amounts of time. There are many varieties of such systems depending upon what restrictions we put on the vacations (see [15] for a survey). For our model, we require vacations to occur preemptively and at any time (as opposed to vacations that occur only when the processor is busy). The earliest analysis of such systems is in [16], and later in [17], but Gaver ([18]) derives the Laplace transform for the finishing time by assuming exponentially distributed available periods and generally distributed nonavailable periods. Federgruen and Green ([19]) extend the analysis to generally distributed available periods, but they find only the first two moments of the finishing time, and not its distribution.

For both single and multiple processor systems, performability analysis offers an alternative approach to ours. Performability analysis ([20], [21], [22]) combines dependability analysis with performance measures. A system is modeled as a Markov or semi-Markov process in which each state of the process represents a possible configuration of the system with respect to failed and working components. In a multiprocessor, for example, the state could be the number of working processors. This state represents the reliability aspect of performability analysis. Associated with each state is a reward representing either the performance measure of interest, or a quantity that may be used to calculate the performance measure. Applied to the models of this paper, the state of the system represents the number of available processors, and the reward for each state is the amount of available computing power (in operations per time unit) in that state. Our goal would be to find the distribution of time it takes for the accumulated reward (accumulated work) to reach a threshold representing the amount of work a program requires.

A number of researchers have examined the problem of finding the distribution of accumulated reward (also known as the performability distribution). Nonrepairable systems, in which a nonavailable processor cannot become available ("be repaired"), are easiest to analyze, but are not applicable to our problem. For repairable systems, some researchers have found methods to get the moments of the performability distribution ([23], [24], [25]), and other researchers have expressed the performability distribution as a double Laplace transform ([24], [26], [23], [27], [28], [29]). In the latter, typically the transform can be inverted analytically on one variable, then inverted numerically on the other variable, although [24] perform the inversion entirely numerically. In [30], de Souza e Silva and Gail apply randomization techniques to numerically find the distribution of performability over a finite time interval.

However, we do not want this distribution of accumulated reward itself, but we would like to find the distribution of time for the accumulated reward to reach a threshold. Because our reward represents accumulated work, this latter distribution is equivalent to the distribution of time to complete a job. In one of the early papers on performability, Beaudry [20] defines this quantity, but never derives it for systems of interest to us. Kulkarni, Nicola, Smith, and Trivedi [27] find a double transform of the job completion time distribution in terms of a system of equations, and provide an algorithm for numerical inversion of the transform. The difference between this previous work and the work contained in this paper is that our model takes a different view of the problem and involves simple, approximate analytical expressions, with no numerical techniques being necessary.

Finally, much of the work on performability concerns itself with transient analysis, because in a well-designed, fault-tolerant system, faults will be quite infrequent, and steady state analysis can be misleading. Our situation is the opposite, with "faults" (processor nonavailability) occurring frequently, and we expect many "faults" to occur before a program finishes execution. Iyer *et al.* [26] note that asymptotically, after a long enough time that every state of the system has been entered many times, the performability distribution is normally distributed, and the mean and standard deviation of this distribution can be found by solving sets of linear equations. In this paper, we come to the same conclusion about the normality of the asymptotic distribution by starting from the analysis of a single processor, as discussed in Sections IV-C and V-A, and in doing so we find simple expressions for the mean and variance of this distribution [(18) and (19)].

One appealing aspect of performability models is that by setting the rewards appropriately for each system state, the model could capture some of the inefficiencies that occur as a program executes on varying numbers of processors, examples of which are the additional communication overhead involved, or the program's inability to use all available processors. These rewards, however, would be specific to that particular program. Ammar and Islam [24] have done this using Generalized Stochastic Petri Nets to generate the reliability model for a specific architecture, and then trace-driven simulations of a specific algorithm to determine the reward for every state of the reliability model. The reward is the inverse of the total execution time of the computation, given the system is in a particular state. Because there may be many states in the reliability model, and hence many simulation runs required, their model is potentially quite time consuming. We are investigating methods by which we can capture the interaction of the algorithm with the architecture within our Brownian motion model.

III. Time to Finish a Program: Quick Means

With simple reasoning, we can find the mean cumulative work over time, $\mathrm{E}[y(u \mid t)]$, and the mean finishing time for a program, $\overline{f}$. Over a long period of time, a processor is available a fraction of the time $p_a = t_a/(t_a + t_n)$, and nonavailable the remaining fraction of the time, $p_n = t_n/(t_a + t_n)$. Over a period of t seconds, the amount of work a processor does is equal to the fraction of time it is available (assuming that there is a large amount of work to do, and that the processor never goes idle), and thus we have the equilibrium approximation

$$\mathrm{E}[y(u \mid t)] = \frac{t_a}{t_a + t_n}\, t. \tag{3}$$

Similarly, it takes $(t_a + t_n)/t_a$ seconds to accumulate one second of work, so the average finishing time for a program

on a single processor is

$$\overline{f} = \frac{t_a + t_n}{t_a} W. \tag{4}$$

In an M processor network, we accumulate work M times faster and thus finish in $(1/M)$th of the time. Therefore:

$$\mathrm{E}[y(u \mid t)] = \frac{t_a M}{(t_a + t_n)} t. \tag{5}$$

$$\overline{f} = \frac{t_a + t_n}{t_a M} W. \tag{6}$$

We will use these as a check on the other analyses.

IV. The Distribution of Finishing Time for One Processor

A. Introduction

We would now like to find the pdf of the finishing time for any particular algorithm or program. This is also known as the *first passage time*, the first time at which the accumulated work is greater than some particular amount (W minutes in our case). In this section, we analyze the behavior of a program on a single, transient processor using two methods. The direct method, in Section IV-B, yields $f(t)$ for general distributions, but unfortunately, it does not extend to multiple processors because the analysis depends upon the system being either fully available or fully nonavailable. In a multiprocessor system, we usually have partial availability: some of the machines are available and some are not. We do not derive the pdf of accumulated work, $y(u \mid t)$, using the direct analysis. However, by analyzing the problem as a cumulative, alternating, renewal process (Section IV-C), we do find the asymptotic probability density of the accumulated work as $t \to \infty$, and we use this in the Brownian motion analysis of the next section.

B. Direct Analysis of a Single Processor

We make a direct analysis of the single-processor problem by counting the number of nonavailable periods that interrupt our program before it completes. If our program starts at the beginning of an available period, as shown in the middle of Fig. 2, it will finish at time $W + T_a$, where T_a is the *additional* time the program spends in the system because of interrupting nonavailable periods.

Because we must finish the program in an available period, and because we assume work starts at the beginning of an available period, then none of the nonavailable periods are truncated, and it is relatively easy to analyze the total length of the nonavailable periods during the time $T_a + W$. By examining the arrival process for nonavailable periods, we find that the Laplace transform of the finishing time density is

$$F^*(s) = e^{-Ws} \sum_{k=0}^{\infty} [\mathrm{N}^*(s)]^k \, p(k \mid W)$$
$$= e^{-Ws} \mathrm{P}(\mathrm{N}^*(s)) \tag{7}$$

where $\mathrm{P}(z) = \sum_{k=0}^{\infty} p(k \mid W) z^k$ is the z-transform of $p(k \mid W)$, the probability that k zero-length nonavailable periods arrive in a W minute period starting at the beginning of an available period, and $\mathrm{N}^*(s)$ is the Laplace transform of the length of a nonavailable period. Note that we can also get the finishing time density directly using the same technique, but this usually results in an open expression; details are available in [1]. From the Laplace transform, we can derive the finishing time's mean:

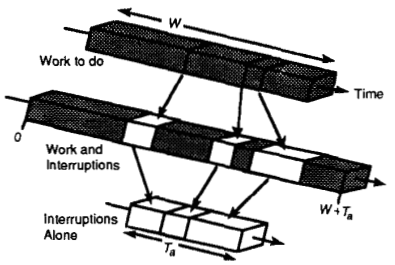


Fig. 2. Time for one node to finish W units of work.

$$\overline{f} = W \frac{t_a + t_n}{t_a}, \tag{8}$$

and variance:

$$\sigma_f^2 = \sigma_n^2 \overline{p} + t_n^2 \sigma_p^2 \tag{9}$$

where $\overline{p}$ and σ_p^2 are the mean and variance of $p(k \mid W)$.

The central limit theorem assures us that when we sum many independent random variables, the resulting distribution tends toward a normal distribution. We may note an important consequence of this: asymptotically, for large W compared to t_a and t_n, the finishing time density is normal with mean and variance given in (8) and (9).

1) Example: Exponential Distributions: If available and nonavailable periods are exponentially distributed, then the finishing time density is

$$f(t) = \begin{cases} e^{-W/t_a} & \text{if } t = W \\ \sum_{k=1}^{\infty} \left(\frac{(1/t_n)((t-W)/t_n)^{k-1}}{(k-1)!} e^{-(t-W)/t_n} \right) \left(\frac{(W/t_a)^k}{k!} e^{-W/t_a} \right) & \text{if } t > W \end{cases}. \tag{10}$$

This was derived using a direct analysis detailed in [1]. Fig. 3 illustrates this density using Mutka and Livny's parameters to select the means t_a = 91 min and t_n = 31.305 min of the exponential distributions. The mean finishing time is

$$\overline{f} = W \frac{t_a + t_n}{t_a} \tag{11}$$

and its variance is

$$\sigma_f^2 = \frac{2 t_n^2 W}{t_a}. \tag{12}$$

The finishing time density looks similar to a normal density, but it is asymmetrical. For t less than the mean first passage time, it rises sharply to a peak before the mean, then drops into a stretched-out tail for large t. This asymmetry is more apparent for small W, and as W grows, the density becomes

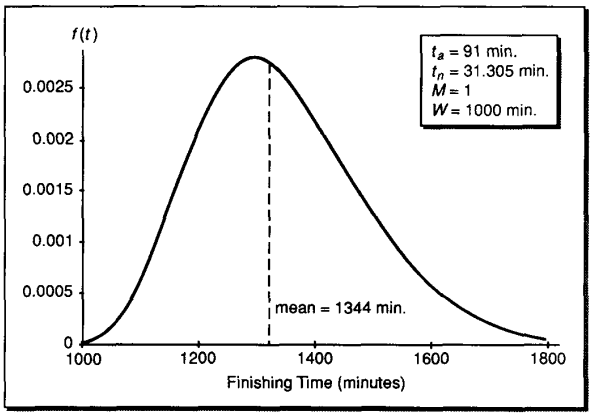


Fig. 3. Probability density of finishing time for direct analysis.

more similar to a normal density, as one would expect from the central limit theorem.

Unfortunately, the analysis of this section will not extend to multiple processors because it depends upon the system being fully available or fully nonavailable. With multiple processors the system is usually partially available. However, it is possible to get an approximation to the finishing time density for the case of multiple processors by studying a different process, namely the accumulated work. Thus, in the next section we use a cumulative, alternating renewal process to analyze the accumulated work on a single processor, then in Section V-A we apply this analysis to the multiple processor case.

C. Cumulative, Alternating Renewal Theoretic Analysis

Here we reexamine a single processor using a cumulative, alternating renewal process. Cox, in his book on renewal processes [31], discusses this type of process, and we make use of his analysis.

We can form a renewal process from the alternating states of a transient processor by letting a renewal period be a nonavailable period followed by an available period. In Fig. 4 the heavy dots indicate the beginning of each renewal period. The durations of the available periods are i.i.d. random variables from a general distribution, as are the the durations of the nonavailable periods, and the lengths of the available and nonavailable periods are mutually independent. Using Cox's results, we find that the distribution of accumulated available time has mean and variance:

$$\mathrm{E}[y(u \mid t)] \approx \frac{t_a}{t_a + t_n} t \tag{13}$$

$$\mathrm{Var}[y(u \mid t)] \approx \frac{\sigma_a^2 t_n^2 + t_a^2 \sigma_n^2}{(t_a + t_n)^3} t. \tag{14}$$

Cox derives these by ignoring the available time accumulated in the current available period if one is in progress at time t. However, as time goes to infinity, the asymptotic distribution of this approximation has the same properties as the true accumulated available time. Note that the approximation for $\mathrm{E}[y(u \mid t)]$ leads to a mean which corresponds exactly to the equilbrium approximation of (3) in Section III.

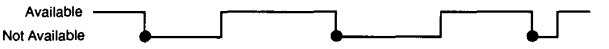


Fig. 4. Cumulative renewal process for a processor.

For exponentially distributed available and nonavailable periods, the mean remains the same and the variance may be rewritten as

$$\mathrm{Var}[y(u \mid t)] \approx \frac{2 t_a^2 t_n^2}{(t_a + t_n)^3} t. \tag{15}$$

We could use this approach to get a (possibly very complex) expression for the distribution of accumulated available time; this was also done in [1] based on the techniques of the previous section. Of more interest to us is the fact that the asymptotic pdf of the accumulated available time (for large t) is normal with mean and variance given by (13) and (14). This distribution is based on a sum of random variables (the available periods), and the Central-Limit Theorem [32] tells us that as the number of random variables in the sum approaches infinity, this pdf converges to a normal pdf. Thus the pdf of the accumulated work, $y(u \mid t)$, is well approximated by a normal pdf if many renewal periods have occurred, or equivalently, if $t \gg t_a + t_n$. This normal pdf will be the basis of the Brownian motion model in the next section, which will lead us to an approximation for the distribution of finishing time with multiple processors.

V. The Distribution of Finishing Time for M Processors

A. Brownian Motion Approximation

Brownian motion concerns the random movement of a particle through space. A stochastic process, $Q(t)$, that describes Brownian motion has two basic properties. The first is that $Q(t)$ has *independent increments*: $Q(t_1) - Q(t_0)$ and $Q(t_3) - Q(t_2)$ are independent for $0 \leq t_0 < t_1 < t_2 < t_3 < \infty$. Movement of the particle in one interval is independent of its movement in another interval. The second property is that each increment in the process, $Q(t_{i+1}) - Q(t_i)$ for all $i \geq 0$, is normally distributed with a mean and variance proportional to $t_{i+1} - t_i$. If the normal distribution has mean 0 and variance equal to $t_{i+1} - t_i$, then the process describes *standard Brownian motion*, which is also known as a *Wiener process*. Brownian motion with a nonzero mean is known as *Brownian motion with drift*.

In our model, we let the stochastic process $Q(t)$ represent the amount of work accumulated by a network of M transient processors up to time t. In Section IV-C, we found that over a long period of time (much longer than $t_a + t_n$), the amount of work done by one transient processor is asymptotically normal with mean and variance given in (13) and (14), respectively. If we have a network of M such processors, and all the processors are assumed to be independent and identical, then, asymptotically, the amount of work done by time t is the sum of M independent, (approximately) normally distributed random variables, and this is itself (approximately) normally

distributed. The mean amount of work done by time t is

$$\mu = \frac{t_a}{t_a + t_n} Mt = p_a Mt \tag{16}$$

and the variance of the amount of work done by time t is

$$\sigma^2 = \frac{\sigma_a^2 t_n^2 + \sigma_n^2 t_a^2}{(t_a + t_n)^3}\ Mt. \tag{17}$$

Thus, Brownian motion with drift is a natural model of our system. From μ and σ^2 above, we define $\bar{b}$ and σ_b^2 (b signifies Brownian), the mean and variance of the amount of work accumulated per unit time:

$$\bar{b} = \frac{t_a}{t_a + t_n} M = p_a M \tag{18}$$

$$\sigma_b^2 = \frac{\sigma_a^2 t_n^2 + \sigma_n^2 t_a^2}{(t_a + t_n)^3}\ M. \tag{19}$$

We use $\bar{b}$ and σ_b^2 later in this analysis. Note that $\bar{b}$ agrees with the average accumulated work that we found in Section III.

We must still assure ourselves that our stochastic process indeed has independent increments. On a short term scale, this is clearly not true. Two consecutive one-minute intervals are likely to have the same, or at least similar, numbers of available processors in both intervals, and hence similar amounts of work accumulated in those intervals. However, in two one-minute intervals separated by several hours, the number of available processors is quite unrelated (unless the network has some very unusual statistical properties), and the work accumulated in one interval is quite independent of the work accumulated in the other interval. Thus we conclude that the Brownian motion model is reasonable only over a long span of time, and we insure this by specifying that $t_a \ll W$ and $t_n \ll W$. Note, too, that we are using the asymptotic results of Section IV-C, and these are valid only for a long span of time, which also requires a large W relative to t_a and t_n.

The Brownian motion model does allow some behavior that seemingly cannot occur in a real network. For example, the process is allowed to move in the negative direction, implying that we can lose work that we have already done. This is an artifact of the model, and it is particularly apparent at small t, but it is negligible for the conditions under which the Brownian motion model is useful. Given $\bar{b}$, σ_b^2, and t, and using the fact that cumulative work is normally distributed, we can compute the probability of negative cumulative work as

$$\Phi\left(\frac{-\bar{b}t}{\sqrt{\sigma_b^2 t}}\right) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{-(\bar{b}t/\sqrt{\sigma_b^2 t})} e^{-\frac{1}{2}x^2} dx \tag{20}$$

where $\Phi(x)$ is the cumulative density function for a standard normal distribution. For t near 0, $\Phi(-\bar{b}t/\sqrt{\sigma_b^2 t}) \approx 0.5$, meaning that for very small t, our model says that almost 50% of the time the program has accumulated a negative amount of work. Clearly, Brownian motion is a poor model of networks of transient processors for very small t. If we manipulate the expression $-\bar{b}t/\sqrt{\sigma_b^2 t}$ we find it is equal to $-\sqrt{(Mt_n/2(1-p_a))t}$, and the coefficient of t under the radical is greater than 1 for any reasonable values of M, t_a, and t_n. Thus as time passes and t moves away from 0 and becomes large, $-\sqrt{(Mt_n/2(1-p_a))t}$ becomes quite negative and $\Phi((-\bar{b}t/\sqrt{\sigma_b^2 t})$ shrinks to near 0. In fact, for $t = 3\sqrt{\sigma_b^2/\bar{b}}$, the 3σ point, the probability that we have negative work is approximately 0.0023 and drops rapidly thereafter to negligible amounts as t grows. This is just further confirmation that our model is valid only for relatively large W that requires more than a short time to complete.

Using $\bar{b}$ and σ_b^2 as the parameters for our Brownian motion, and using results in Karlin and Taylor [33], we find the probability density of the time, t, that it takes for M processors to finish W minutes of work is

$$f(t) = \frac{W}{\sqrt{2\pi\sigma_b^2 t^3}}\ \exp\left[-\frac{(W - \bar{b}t)^2}{2\sigma_b^2 t}\right]. \tag{21}$$

This has mean

$$\overline{f} = \frac{W}{\bar{b}} = \frac{W}{M}\frac{(t_a + t_n)}{t_a} \tag{22}$$

and variance

$$\sigma_f^2 = \frac{W}{\bar{b}}\frac{\sigma_b^2}{\bar{b}^2}. \tag{23}$$

Note that for the case $M = 1$, the mean and variance agree with the direct analysis of Section IV-B. Of course the mean is consistent with that of Section III for all M.

B. Example: Exponential Distributions

If we assume that both available and nonavailable periods are exponentially distributed, then the mean and variance of the accumulated work per unit time are:

$$\bar{b} = \frac{t_a}{t_a + t_n} M = p_a M \tag{24}$$

$$\sigma_b^2 = \frac{2(t_a t_n)^2}{(t_a + t_n)^3} M = \frac{2p_a^2(1 - p_a)M}{t_n}. \tag{25}$$

Applying this to (22) and (23) yields the mean of the finishing time

$$\overline{f} = \frac{W}{\bar{b}} = \frac{W}{M}\frac{(t_a + t_n)}{t_a} \tag{26}$$

and variance

$$\sigma_f^2 = \frac{W}{\bar{b}}\frac{\sigma_b^2}{\bar{b}^2} = \frac{2W}{M^2}\frac{t_n^2}{t_a}. \tag{27}$$

Fig. 5 shows the finishing time density for both the direct and the Brownian motion analyses with $t_a = 3600, t_n = 300, M = 1$, and $W = 10^5$. We note the good concordance between the two analyses.

When we have $M = 100$ processors, Fig. 6 shows the pdf of finishing time for various t_n with $t_a = 91$ min and $W = 10^4$ min. Using $t_a = 91$ min and $t_n = 31.305$ min (the standard multiprocessor example), we have $\bar{b} = 74.4t$ and $\sigma_b^2 = 887.2t$, which leads to $\overline{f} = 0.0134W$ and $\sigma_f^2 = 0.00215W$. Our example job of 10^4 min would take about a week to run on a single, dedicated processor. When run on a network of 100 transient processors, it would take 134.06 min, or about 2.25 h. This particular finishing time pdf is in Fig. 7, which shows the density both enlarged and plotted on a full time axis (starting at 0).

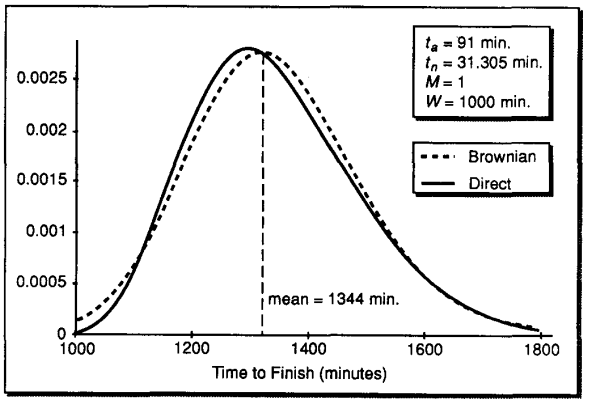


Fig. 5. Finishing time densities for direct and Brownian motion analyses.

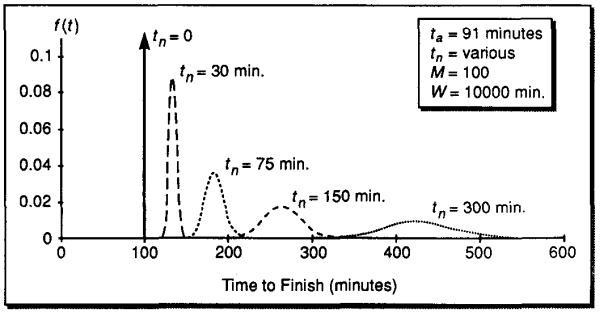


Fig. 6. Finishing time densities for Brownian motion model for various t_n.

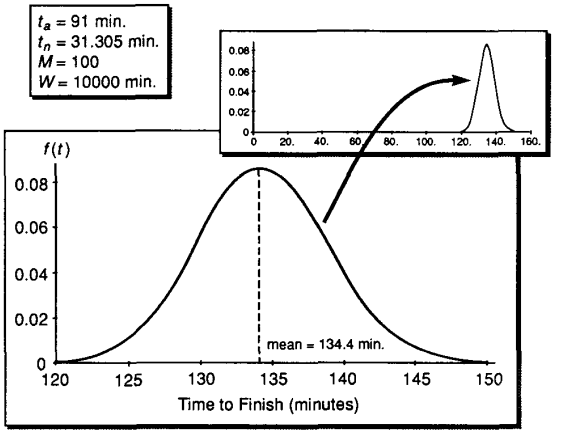


Fig. 7. Finishing time density for Brownian motion analysis.

C. Example: Mutka and Livny's Distributions

Let us use the distributions measured by Mutka and Livny. We have t_a = 91 min, σ_a^2 = 40225 min^2, t_n = 31.305 min, and σ_n^2 = 2131.83 min^2. Plugging these into (18) and (19), we find

$$\overline{b} = 74.4t \tag{28}$$

$$\sigma_b^2 = 6239t. \tag{29}$$

This leads to a finishing time mean and variance of

$$\overline{f} = 0.0134W \tag{30}$$

$$\sigma_f^2 = 0.0151W. \tag{31}$$

We note that the finishing time variance using Mutka and Livny's distributions is almost an order of magnitude more than for exponential distributions.

D. The Ratio $\sigma_f/\overline{f}$

It is instructive to examine the coefficient of variation of the finishing time, namely the ratio of σ_f to $\overline{f}$:

$$\frac{\sigma_f}{\overline{f}} = \frac{\sqrt{\sigma_b^2}}{\sqrt{\overline{b}W}}. \tag{32}$$

We note immediately that this ratio goes to zero as W increases. Consequently, for sufficiently large W, it may be accurate enough to consider the finishing time distribution as an impulse (i.e., the Dirac delta function) at the mean finishing time (in the spirit of the law of large numbers).

Assume that the available and nonavailable periods have exponential distributions. Then the ratio becomes

$$\frac{\sigma_f}{\overline{f}} = \sqrt{\frac{2t_n}{W}}\,\frac{\sqrt{t_n/t_a}}{1+t_n/t_a}. \tag{33}$$

Because we assumed $t_n \ll W$, this ratio tends to be less than 1. If we fix t_n/W and let t_n/t_a go to infinity (which implies $t_a \to 0$), the ratio goes to 0. We explain this by noting that for small t_a, it takes very many available–nonavailable cycles before the work is finished. The law of large numbers insures that the finishing time density, which is the sum of these many periods, will then be tight about its mean.

If, on the other hand, we let $t_a \to \infty$, the ratio of the standard deviation to the mean goes to zero once again. When t_a is large relative to t_n, the nonavailable periods become negligible, as if the processors are always available. Again, the finishing time density becomes very tight about its mean because nonavailable time periods add little variability to the finishing time, and under some circumstances we may consider the finishing time density as an impulse located at $t = W/M$. Using the standard multiprocessor example again ($M = 100$) with exponential distributions, we find $\sigma_f^2 = 21.53$, and approximating $f(t)$ as a normal density (discussed below), we find that 90% of the time, programs requiring 10^4 min of work will finish within 7.6 min of the 134.4 min mean finishing time, which is an interval 3% on either side of the mean. This is very narrow indeed. If we use Mutka and Livny's distributions, then 90% of the time programs finish in an interval 10 min on either side of the mean, which is still quite narrow.

We find the peak of (33) by taking the derivative with respect to t_a:

$$\frac{\partial}{\partial t_a}\frac{\sigma_f}{\overline{f}} = \frac{t_n(t_a+t_n) - 2t_a t_n}{\sqrt{Wt_a}(t_a+t_n)^2}. \tag{34}$$

Setting this equal to 0 yields $t_a = t_n$ as the peak of the ratio, at which point it takes on the value $\sigma_f/\overline{f} = \sqrt{t_n/2W}$. The ratio's value at the peak is small because of our assumption that $t_n \ll W$. Note that if we assume $t_a = t_n$, but not

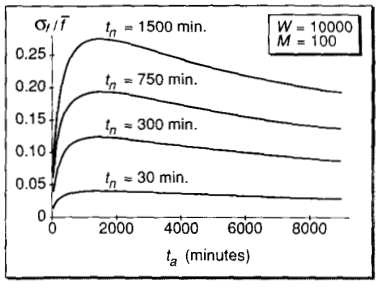


Fig. 8. $\sigma_f/\overline{f}$ with $W = 10^6$, varying t_a and t_n.

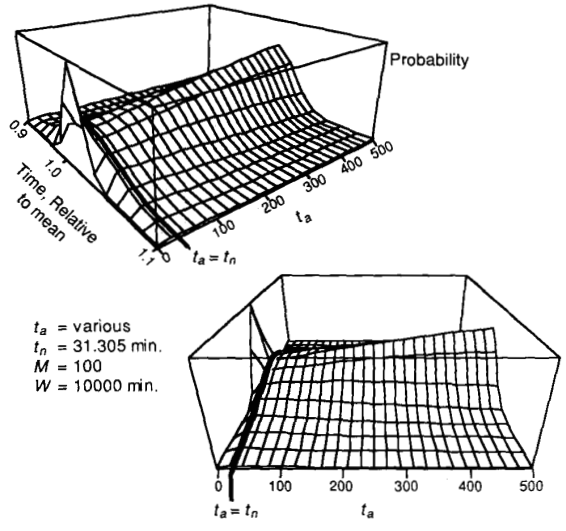


Fig. 9. Two views of the Brownian motion finishing time densities with varying t_a.

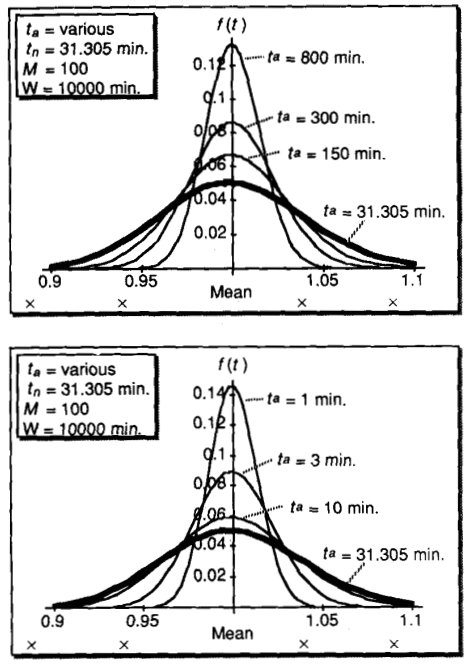


Fig. 10. Finishing time density narrowing as t_a grows (top figure) and as t_a shrinks (bottom figure).

$t_n \ll W$, then we can make the ratio as large as we want, simply by increasing t_n. If, for example, $t_a = t_n = 1$ year, then either the system is available immediately to do all our work, or else we will have to wait a very long time before it even starts. In such a case, the finishing time still has a reasonable mean but an enormous variance. Another fact to note is that M, the number of processors, does not affect the ratio $\sigma_f/\overline{f}$. Even if we have an infinite number of processors, we can still have great variance relative to the mean. Of course, both the mean and the standard deviation go to zero as M grows, but their ratio remains constant.

In Fig. 8 we plot $\sigma_f/\overline{f}$ versus t_a for $W = 10^4$, with t_n/W fixed for each curve. Fig. 9 shows finishing time densities for t_n = 31.305 min and various t_a. The x-axis (labeled "Time, Relative to Mean") is centered about the mean and plots the distance relative to the mean (varying from 0.9 times the mean to 1.1 times the mean). We note that the density is flattest and has the greatest spread for $t_a = t_n$; at this point $\sigma_f/\overline{f} = 0.035$, which is quite small. For comparison, if we use Mutka and Livny's distributions at the same point, the ratio is 0.092, which is still small. The narrowing of the density is also illustrated in Fig. 10. The parameters for both plots are: exponential distributions with varying t_a but fixed $t_n = 31.305$ min, $M = 100$, and $W = 10^4$ min. In the top plot, the density narrows as $t_a = 31.305$, 150, 300, and 800 min. In the bottom plot, we have $t_a = 31.305$, 10, 3, and 1 min as the density narrows.

E. Normal Approximation to the Finishing Time

The usual form of the central limit theorem states that the sum of n independent random variables tends to have a normal distribution as n gets large. Given this, we would expect the limiting distribution of $f(t)$ to be normal with mean $\overline{f}$ and variance σ_f^2. Let us denote this normal approximation by $\hat{f}(t)$:

$$\hat{f}(t) = \frac{1}{\sqrt{2\pi\sigma_f^2}}\, e^{-(t-\overline{f})^2/(2\sigma_f^2)}. \tag{35}$$

Substituting $t = \overline{f}$ shows that the finishing time and its normal approximation coincide at the mean:

$$f(\overline{f}) = \frac{W}{\sqrt{2\pi\sigma_b^2 W^3/\overline{b}^3}}\, e^0 = \frac{\overline{b}^{3/2}}{\sqrt{2\pi\sigma_b^2 W}}$$

$$\hat{f}(\overline{f}) = \frac{1}{\sqrt{2\pi\sigma_f^2}}\, e^0 = \frac{\overline{b}^{3/2}}{\sqrt{2\pi\sigma_b^2 W}}$$

$$= f(\overline{f}).$$

Observation shows that $f(t)$ and $\hat{f}(t)$ also coincide at two more points, but analytically these are not easily found because they are the solutions to a transcendental equation. Numer-

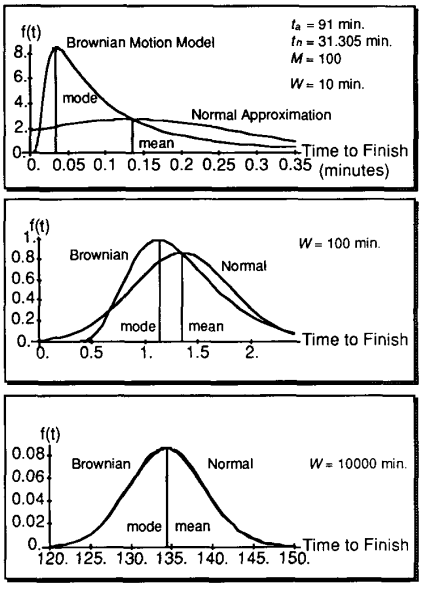


Fig. 11. Brownian motion finishing time pdf and its normal approximation.

ically, we find that these points appear to be separated by $\sqrt{12\sigma_f^2}$, and the distance from the lower point (smaller t) to the mean is very slightly less than half of the total distance between the two points (varying, but in the range of 49.5% of the total separation).

We need to know when $\hat{f}(t)$ is a good approximation for $f(t)$. Observation (see Fig. 11) shows that the approximation is good when the mode of the finishing time is close to (within a few percent of) its mean. We find the mode by taking the derivative of $f(t)$ with respect to t, setting it equal to 0, and solving for t. We end up with a quadratic equation that has a negative and a positive root. The positive root is the mode, namely

$$t_{\text{mode}} = \frac{1}{2}\sqrt{\frac{9(\sigma_b^2)^2}{\bar{b}^4} + \frac{4W^2}{\bar{b}^2}} - \frac{3\sigma_b^2}{2\bar{b}^2}. \tag{36}$$

By using the fact that $\sqrt{a+b} \le \sqrt{a} + \sqrt{b}$, we show that the mode is always less than or equal to the mean:

$$t_{\text{mode}} \le \frac{3\sigma_b^2}{2\bar{b}^2} + \frac{W}{\bar{b}} - \frac{3\sigma_b^2}{2\bar{b}^2} = \frac{W}{\bar{b}} = \bar{f}.$$

Furthermore, if we observe that $9(\sigma_b^2)^2/\bar{b}^4$ is usually much less than $4W^2/\bar{b}^2$, and we use the approximation $\sqrt{1+\epsilon} \approx 1 + \frac{\epsilon}{2}$ for $0 \le \epsilon \ll 1$, then

$$\begin{aligned} t_{\text{mode}} &\approx \frac{W}{\bar{b}} - \frac{3}{2}\frac{\sigma_b^2}{\bar{b}^2}\left(1 - \frac{3}{4}\frac{\sigma_b^2}{W\bar{b}}\right) \\ &\approx \bar{f} - \frac{3}{2}\frac{\sigma_b^2}{\bar{b}^2}. \end{aligned} \tag{37}$$

Under almost all circumstances, we may drop the negative term in the parenthesis, because when the Brownian motion approximation is valid we also know that $W \gg \sigma_b/\bar{b}$, and in general, $W \gg \sigma_b$. These would render the term $3\sigma_b^2/4W\bar{b}$ negligible. Only under very unusual circumstances would $W \not\gg \sigma_b^2$, and in such cases we could not drop the term. Excepting such circumstances, (37) is quite accurate for all conditions in which our Brownian motion model is operative. Using (37), we find that the percent difference between the mean and mode is approximately $3\sigma_b/2\bar{b}W$.

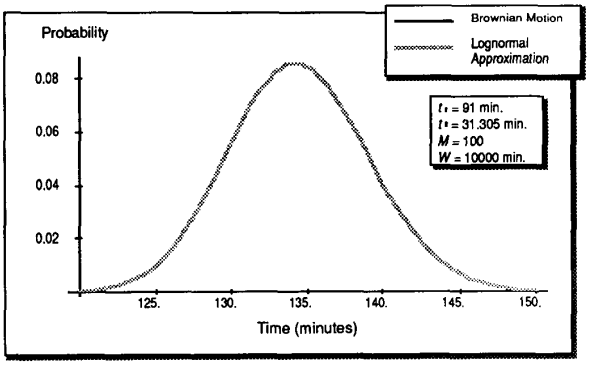


Fig. 12. Density of finishing time and its lognormal approximation.

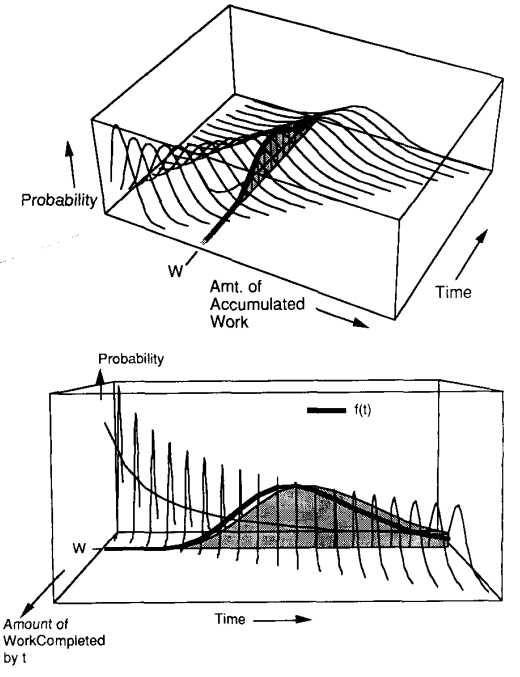


Fig. 13. Density of finishing time.

F. Lognormal Approximation to Finishing Time

A lognormal density provides a remarkably good approximation to (21). A lognormal distribution has two parameters, μ_l and σ_l^2. If we equate the mean and variance of the finishing time pdf from the Brownian motion model to that of a

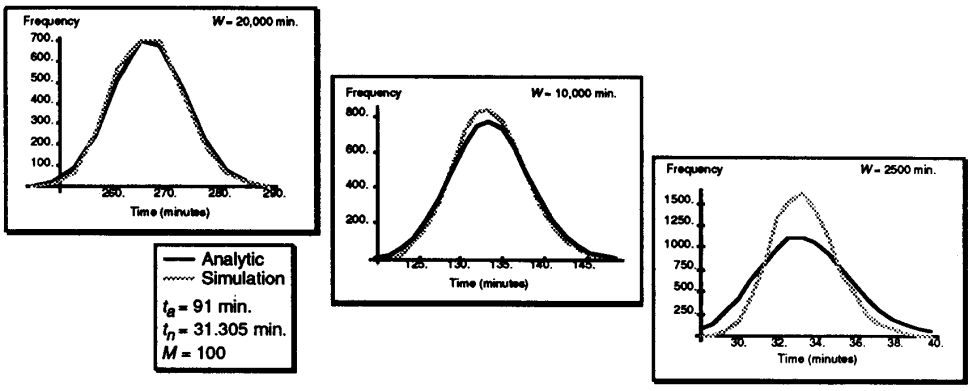


Fig. 14. Simulation results.

lognormal pdf, then we find that these parameters must be

$$\mu_l = \ln(\overline{f}) - \frac{1}{2}\sigma_l^2 = \ln\left(\frac{\overline{f}}{\sqrt{\sigma_f/\overline{f}^2 + 1}}\right) \tag{38}$$

$$\sigma_l^2 = \ln(\sigma_f^2/\overline{f}^2 + 1). \tag{39}$$

The lognormal pdf fit to (21) then becomes

$$l(t) = \frac{1}{t\sqrt{2\pi\sigma_l^2}} \exp\left[-\frac{(\ln(t) - \mu_l)^2}{2\sigma_l^2}\right]. \tag{40}$$

As shown in Fig. 12, when both the Brownian motion finishing time pdf and the lognormal approximation are plotted, the densities are extremely close, and the plotted curves appear to lie on top of each other. When the two do differ, it is under circumstances where the assumptions of the Brownian motion model do not hold (e.g. W small relative to all of M, t_a, and t_n).

G. The Finishing Time Density and its Derivation from a Normal pdf

We can rewrite the Brownian motion finishing time density, (21), in a form using a normal pdf. Let $\phi_{\mu,\sigma^2}(x)$ be the probability density that a random variable, normally distributed with mean μ and variance σ^2, takes on the value x. Using this, (21) becomes

$$f(t) = \frac{W}{t}\phi_{\overline{b}t,\sigma_b^2 t}(W). \tag{41}$$

The normal pdf term derives from the underlying Brownian motion; it is the probability that a total of W units of work have been accumulated by time t. As for the weighting factor of W/t, no intuitive explanation has yet been found for this. Fig. 13 illustrates the relationship between (21) and (41). In this figure, the normal pdf of the amount of work done by time x, $\phi_{\overline{b}t,\sigma_b^2 t}(x)$, is plotted with thin lines for various t. The shaded plane in the figure picks out those points on the normal pdf where $x = W$; the line arcing down (top left to lower right in the bottom view) within this plane represents W/t. The other line within the shaded plane is the finishing time density, i.e., the product of these last two curves. The curves have been scaled differently to make them fit into one plot, so relative heights, except within the group of normal curves, are meaningless.

H. Simulation Results

We ran simulations for the case of exponentially distributed available and nonavailable periods. Some results comparing the simulation to the Brownian motion model are shown in Fig. 14. The Brownian motion model and the simulation agree very well for large W, as we would expect, and they deviate as W becomes small.

VI. Conclusion

In this paper, we analyzed the distribution of the time to finish a distributed program running in a network of transient processors. We first made two analyses of a program running on a single transient processor. These results were then used as the basis for a Brownian motion, multiprocessor model, and from this we found four finishing time distributions: the actual Brownian motion finishing time distribution, and its normal, lognormal, and impulse approximations. The models in this paper offer an approach to predicting performance of distributed programs on transient processors. By relaxing some of our assumptions, as discussed below, more sophisticated models could be derived from those that have been described.

The first assumption that we would like to relax concerns the asymptotic nature of the results. The results we have given are valid only over a long period of time. If we have a relatively small amount of work to do (say, several hours), then our finishing time distributions are not valid. Their means are acceptable, but the variance is quite incorrect, and the distributions (except for the impulse approximation) show noticeable probability that the program will finish in less than the minimal time required (W/M). Judging from the simulation results, there may well be some simple way to heuristically modify the variance expression in our models so they provide acceptable results for small W.

A second assumption we would like to relax concerns our model of a program. Modeling an algorithm as sequential, independent stages is very simplistic. Many programs do not have clear stages, but instead have a more complex internal precedence structure among the tasks of the program which cause additional delays. The independent-stages model may provide a useful simplification, but testing this, and developing more complicated program models, remains for future work.

A third assumption of great importance is that our network model does not account for the realities of communications. Communication entails delay, and our model does not address this issue. Solutions to this are currently under investigation, and some possibilities are mentioned in [1].

The models in this paper do have many assumptions, yet their very simplicity makes them appealing. An alternative to our models is performability analysis, yet the complexities of performability models yield only numeric results or a Laplace transform, and not a direct analytic expression for the distribution of program completion time. Furthermore, the basic parameters of our models can be modified to remove some of the assumptions, and, at least for large W, the models already do capture the basic behavior of transient, distributed systems.

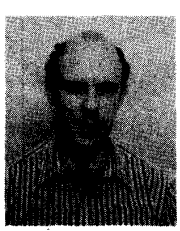
ACKNOWLEDGMENT

The authors would like to thank the referees for their very useful suggestions.

REFERENCES

[1] W. Korfhage, "Distributed systems and transient processors," Ph.D. dissertation, Univ. California, Los Angeles, Aug. 1989.
[2] L. Kleinrock, "Distributed systems," *Commun. ACM,* vol. 28, pp. 1200–1213, Nov. 1985.
[3] S. J. Mullender *et al.*, "Amoeba: A distributed operating system for the 1990s," *IEEE Comput. Mag.,* vol. 23, pp. 44–53, May 1990.
[4] M. W. Mutka and M. Livny, "Scheduling remote processing capacity in a workstation-processor bank network," in *Proc. 7th Int. Conf. Distributed Comput. Syst.,* Berlin, Germany, Sept. 1987.
[5] P. Kreuger and R. Chawla, "The stealth distributed scheduler," in *Proc. 11th Int. Conf. Distributed Comput. Syst.,* IEEE Computer Society, May 1991, pp. 336–343.
[6] R. Alonso and L. L. Cova, "Sharing jobs among independently owned processors," in *Proc. 8th Int. Conf. Distributed Comput. Syst.,* June 1988, pp. 282–288.
[7] R. Felderman, E. Schooler, and L. Kleinrock, "The benevolent bandit laboratory: A testbed for distributed algorithms," *IEEE J. Select. Areas Commun.,* vol. 7, pp. 303–311, Feb. 1989.
[8] J. R. Lyle and C. Lu, "Load balancing from a unix shell," in *Proc. 13th Conf. Local Comput. Networks,* Oct. 1988, pp. 181–183.
[9] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor - A hunter of idle workstations," in *Proc. 8th Conf. Distributed Comput. Syst.,* San Jose, CA, June 1988.
[10] M. W. Mutka and M. Livny, "Profiling workstation's available capacity for remote execution," Computer Sciences Tech. Rep. 697, CS Dept., Univ. Wisconsin, May 1987.
[11] D. A. Nichols, "Using idle workstations in a shared computing environment," in *Proc. Eleventh ACM Symp. Oper. Syst. Principles,* ACM, Nov. 1987, pp. 5–12.
[12] M. Stumm, "The design and implementation of a decentralized scheduling facility for a workstation cluster," in *Proc. 2nd IEEE Conf. Comput. Workstations,* Mar. 1988, pp. 12–22.
[13] J. F. Shoch and J. A. Hupp, "The "worm" programs — Early experience with a distributed computation," *Commun. ACM,* vol. 25, pp. 172–180, Mar. 1982.
[14] T. J. Shimeall, personal communication, 1989.
[15] B. T. Doshi, "Queueing systems with vacations — A survey," *Queueing Systems,* vol. 1, pp. 29–67, June 1986.
[16] H. White and L. S. Christie, "Queueing with preemptive priorities or with breakdown," *Oper. Res.,* vol. 6, pp. 79–95, Jan.–Feb. 1958.
[17] K. Thiruvengadam, "Queueing with breakdowns," *Oper. Res.,* vol. 11, pp. 62–71, Jan.–Feb. 1963.
[18] D. P. Gaver, Jr., "A waiting line with interrupted service, including priorities," *J. Roy. Statist. Soc.,* vol. B24, pp. 73–90, 1962.
[19] A. Federgruen and L. Green, "Queueing systems with service interruptions," *Oper. Res.,* vol. 34, pp. 752–768, Sept.–Oct. 1986.
[20] M. D. Beaudry, "Performance-related reliability measures for computing systems," *IEEE Trans. Comput.,* vol. C-27, pp. 540–547, June 1978.
[21] J. F. Meyer, "On evaluating the performability of degradable computing systems," *IEEE Trans. Comput.,* vol. C-29, pp. 720–731, Aug. 1980.
[22] J. F. Meyer, "Closed-form solutions of performability," *IEEE Trans. Comput.,* vol. C-31, pp. 648–657, July 1982.
[23] B. R. Iyer, "Recent results in performability analysis," in *Current Advances in Distributed Computing and Communications,* Computer Science Press, 1987, pp. 50–64.
[24] S. M. R. Islam and H. H. Ammar, "Performability of the hypercube (reliability)," *IEEE Trans. Reliability,* vol. 38, pp. 518–526, Dec. 1989.
[25] K. R. Pattipati and S. A. Shah, "On the computational aspects of performability models of fault-tolerant computer systems," *IEEE Trans. Comput.,* vol. 39, pp. 832–836, June 1990.
[26] B. R. Iyer, L. Donatiello, and P. Heidelberger, "Analysis of performability for stochastic models of fault-tolerant systems," *IEEE Trans. Comput.,* vol. C-35, pp. 902–907, Oct. 1986.
[27] V. G. Kulkarni, V. F. Nicola, R. M. Smith, and K. S. Trivedi, "Numerical evaluation of performability and job completion time in repairable fault-tolerant systems," in *Fault-Tolerant Computing Systems 16,* 1986, pp. 252–257.
[28] P. S. Puri, "A method for studying the integral functionals of stochastic processes with applications: I. The Markov chain case," *J. Appl. Probability,* vol. 8, pp. 331–343, 1971.
[29] R. M. Smith, K. S. Trivedi, and A. V. Ramesh, "Performability analysis: Measures, an algorithm, and a case study," *IEEE Trans. Comput.,* vol. 37, pp. 406–417, Apr. 1988.
[30] E. de Souza e Silva and H. R. Gail, "Calculating availability and performability measures of repairable computer systems using randomization," *J. ACM,* vol. 36, pp. 171–193, Jan. 1989.
[31] D. R. Cox, *Renewal Theory.* London: Methuen and Co., Ltd., science paperbacks ed., 1962.
[32] A. Mood, F. Graybill, and D. Boes, *Introduction to the Theory of Statistics.* New York: Series in Probability and Statistics, McGraw-Hill, 1974.
[33] S. Karlin and H. M. Taylor, *A First Course in Stochastic Processes,* second ed. New York: Academic, 1975.

Leonard Kleinrock (S'55–M'64–SM'71–F'73), for a photograph and biography, see the March 1993 issue of the TRANSACTIONS, p. 317.



Willard Korfhage (S'81–M'89) received the B.S. degree in electrical engineering and computer science from Princeton University in 1982, and the M.S. and Ph.D degrees in computer science from U.C.L.A in 1985 and 1989, respectively.

He is an assistant professor of Computer Science at Polytechnic University, Brooklyn, NY. He joined the faculty at Polytechnic in 1989, where he is a member of the Center for Advanced Technology in Telecommunication and runs Polytechnic's Distributed Systems Laboratory. His research interests are in distributed systems. He and his students are putting process migration and load balancing in the Hermes language, and creating Panorama, a distributed monitoring and visualization system.