Exploiting Process Lifetime Distributions for Dynamic Load Balancing

Mor Harchol-Balter
and
Allen B. Downey
University of California, Berkeley

We consider policies for CPU load balancing in networks of workstations. We address the question whether preemptive migration (migrating active processes) is necessary, or whether remote execution (migrating processes only at the time of birth) is sufficient for load balancing. We show that resolving this isssue is strongly tied to understanding the process lifetime distribution. Our measurements indicate that the distribution of lifetimes for UNIX process is Pareto (heavy-tailed), with a consistent functional form over a variety of workloads. We show how to apply this distribution to derive a preemptive migration policy that requires no hand-tuned parameters. We use a trace-driven simulation to show that our preemptive migration strategy is far more effective than remote execution, even when the memory transfer cost is high.

Categories and Subject Descriptors: unknown [**unknown**]: unknown—*unknown*

General Terms: unknown

Additional Key Words and Phrases: Load balancing, load sharing, migration, remote execution, workload modeling, trace-driven simulation, network of workstations, heavy-tailed, Pareto distribution

1. INTRODUCTION

Most systems that perform load balancing use remote execution (i.e. non-preemptive migration) based on a priori knowledge of process behavior, often in the form of a list of process names eligible for migration. Although some systems are capable of migrating active processes, most do so only for reasons other than load balancing, such as preserving autonomy. A previous analytic study by Eager et al. discourages

Mor Harchol-Balter supported by National Physical Science Consortium (NPSC) Fellowship and NSF grant number CCR-9201092. Allen Downey partially supported by NSF (DARA) grant DMW-8919074.
An earlier version of this paper appeared in the *Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems* (May 23-26,1996) pp. 13-24.
Address: {harchol,downey}@cs.berkeley.edu, University of California, Berkeley, CA 94720.

implementing preemptive migration for load balancing, showing that the additional performance benefit of preemptive migration is small compared with the benefit of simple non-preemptive migration schemes [Eager et al. 1988]. But simulation studies, which can use more realistic workload descriptions, and implemented systems have shown greater benefits for preemptive migration [Krueger and Livny 1988] [Barak et al. 1993]. This paper uses a measured distribution of process lifetimes and a trace-driven simulation to investigate these conflicting results.

1.1 Load balancing taxonomy

On a network of shared processors, *CPU load balancing* is the idea of migrating processes across the network from hosts with high loads to hosts with lower loads. The motivation for load balancing is to reduce the average completion time of processes and improve the utilization of the processors. Analytic models and simulation studies have demonstrated the performance benefits of load balancing, and these results have been confirmed in existing distributed systems (see Section 1.4).

An important part of the load balancing strategy is the *migration policy*, which determines when migrations occur and which processes are migrated. This is the question we address in this paper. The other half of a load balancing strategy is the location policy — the selection a new host for the migrated process. Previous work has suggested that simply choosing the target host with the shortest CPU run queue is both simple and effective [Zhou 1987] [Kunz 1991]. Our work confirms the relative unimportance of location policy.

Process migration for purposes of load balancing comes in two forms: *remote execution*, also called *non-preemptive* migration, in which some new processes are (possibly automatically) executed on remote hosts, and *preemptive* migration, in which running processes may be suspended, moved to a remote host, and restarted. In non-preemptive migration only newborn processes are migrated.

Load balancing may be done explicitly (by the user) or implicitly (by the system). Implicit migration policies may or may not use a priori information about the function of processes, how long they will run, etc.

Since the cost of remote execution is usually significant relative to the average lifetime of processes, implicit non-preemptive policies require some a priori information about job lifetimes. This information is often implemented as an eligibility list that specifies by process name which processes are worth migrating [Svensson 1990] [Zhou et al. 1993].

In contrast, most preemptive migration policies do not use a priori information, since this is often difficult to maintain and preemptive strategies can perform well without it. These systems use only system-visible data like the current age of each process or its memory size.

This paper examines the performance benefits of **preemptive**, **implicit** load balancing strategies that assume **no a priori information** about processes.

We answer the following two questions:

(1) Is preemptive migration worthwhile, given the additional cost (CPU and latency) associated with migrating an active process?

(2) Which active processes, if any, are worth migrating?

1.2 Process model

In our model, processes use two resources: CPU and memory (we do not consider I/O). Thus, we use "age" to mean CPU age (the CPU time a process has used thus far) and "lifetime" to mean CPU lifetime (the total CPU time from start to completion). We assume that processors implement time-sharing with round-robin scheduling; in Section 7 we discuss the effect of other local scheduling policies. Since processes may be delayed while on the run queue or while migrating, the slowdown imposed on a process is

$$\text{Slowdown of process } p = \frac{\textit{wall-time}(p)}{\textit{CPU-time}(p)}$$

where wall time is the total time a process spends running, waiting in queue, or migrating.

1.3 Outline

The effectiveness of load balancing — either by remote execution or preemptive migration — depends strongly on the nature of the workload, including the distribution of process lifetimes and the arrival process. This paper presents empirical observations about the workload on a network of UNIX workstations, and uses a trace-driven simulation to evaluate the impact of this workload on proposed load balancing strategies.

Section 2 presents a study of the distribution of process lifetimes for a variety of workloads in an academic environment, including instructional machines, research machines, and machines used for system administration. We find that the distribution is predictable with goodness of fit greater than 99% and consistent across a variety of machines and workloads. As a rule of thumb, the probability that a process with CPU age of one second uses more than T seconds of total CPU time is $1/T$ (see Figure 1).

Our measurements are consistent with those of Leland and Ott [Leland and Ott 1986], but this prior work has been incorporated in few subsequent analytic and simulator studies of load balancing. This omission is unfortunate, since the results of these are sensitive to the lifetime model (see Section 2.2).

Our observations of lifetime distributions have the following consequences for load balancing:

—They suggest that it is preferable to migrate older processes because these processes have a higher probability of living long enough (eventually using enough CPU) to amortize their migration cost.

—A functional model of the distribution provides an analytic tool for deriving the eligibility of a process for migration as a function of its current age, migration cost, and the loads at its source and target host.

In Section 3 we derive a migration eligibility criterion that guarantees that the slowdown imposed on a migrant process is lower in expectation than it would be without migration. According to this criterion, a process is eligible for migration only if its

$$\text{CPU age} > \frac{1}{n-m} \cdot \text{migration cost}$$

where n (respectively m) is the number of processes at the source (target) host.

In Section 5 we use a trace-driven simulation to compare our preemptive migration policy with a non-preemptive policy based on name-lists. The simulator uses start times and durations from traces of a real system, and migration costs chosen from a measured distribution.

We use the simulator to run three experiments: first we evaluate the effect of migration cost on the relative performance of the two strategies. Not surprisingly, we find that as the cost of preemptive migration increases, it becomes less effective. Nevertheless, preemptive migration performs better than non-preemptive migration even with surprisingly large migration costs, despite several conservative assumptions that give non-preemptive migration an unfair advantage.

Next we choose a specific model of preemptive and non-preemptive migration costs based on real systems (see Section 4), and use this model to compare the two migration strategies in more detail. We find that preemptive migration reduces the mean delay (queueing and migration) by 35–50%, compared to non-preemptive migration. We also propose several alternative metrics intended to measure users' perception of system performance. By these metrics, the additional benefits of preemptive migration compared to non-preemptive migration appear even more significant.

In Section 5.4 we discuss in detail why a simple preemptive migration policy is more effective than even a well-tuned non-preemptive migration policy. In Section 5.5 we use the simulator to compare our preemptive migration strategy with previously proposed preemptive strategies.

We finish with a criticism of our model in Section 6, a discussion of future work in Section 7 and conclusions in Section 8.

1.4 Related work

1.4.1 *Systems.* Although several systems have the mechanism to migrate active jobs, few have implemented implicit load balancing *policies*. Most systems only allow for *explicit* load balancing. That is, there is no load balancing policy; the user decides which processes to migrate, and when. Examples include Accent [Zayas 1987], Locus [Thiel 1991], Utopia [Zhou et al. 1993], DEMOS/MP [Powell and Miller 1983], V [Theimer et al. 1985], NEST [Agrawal and Ezzet 1987], RHODOS [De Paoli and Goscinski 1995], and MIST [Casas et al. 1995].

A few systems have *implicit* load balancing policies, however they are strictly non-preemptive policies (active processes are only migrated for purposes other than load balancing, such as preserving workstation autonomy). Examples include Amoeba [Tanenbaum et al. 1990], Charlotte [Artsy and Finkel 1989], Sprite [Douglis and Ousterhout 1991], Condor [Litzkow et al. 1988], and Mach [Milojicic 1993]. In general, non-preemptive load balancing policies depend on *a priori* information about processes; e.g., explicit knowledge about the runtimes of processes or user-provided lists of migratable processes [Agrawal and Ezzet 1987] [Litzkow and Livny 1990] [Douglis and Ousterhout 1991] [Zhou et al. 1993].

One existing system that has implemented implemented automated preemptive load balancing is MOSIX [Barak et al. 1993]. Our results support the MOSIX claim that their scheme is effective and robust.

1.4.2 *Studies.* Although few systems incorporate migration policies, there have been many simulation and analytical studies of various migration policies. Most of these studies have focused on load balancing by remote execution [Livny and Melman 1982] [Wang and Morris 1985] [Casavant and Kuhl 1987] [Zhou 1987] [Pulidas et al. 1988] [Kunz 1991] [Bonomi and Kumar 1990] [Evans and Butt 1993] [Lin and Raghavendra 1993] [Mirchandaney et al. 1990] [Zhang et al. 1995] [Zhou and Ferrari 1987] [Hać and Jin 1990] [Eager et al. 1986].

Only a few studies address preemptive migration policies [Leland and Ott 1986] [Krueger and Livny 1988]. The Leland and Ott migration policy is also age based, but doesn't take migration cost into account.

Eager et. al.,[Eager et al. 1988], conclude that the additional performance benefit of preemptive migration is too small compared with the benefit of non-preemptive migration to make preemptive migration worthwhile. This result has been widely cited, and in several cases used to justify the decision not to implement preemptive migration, as in the Utopia system, [Zhou et al. 1993]. Our work differs from [Eager et al. 1988] in both system model and workload description. [Eager et al. 1988] model a server farm in which incoming jobs have no affinity for a particular processor, and thus the cost of initial placement (remote execution) is free. This is different from our model, a network of workstations, in which incoming jobs arrive at a particular host and the cost of moving them away, even by remote execution, is significant compared to most process lifetimes. Also, [Eager et al. 1988] use a degenerate hyperexponential distribution of lifetimes that includes few jobs with non-zero lifetimes. When the coefficient of variation of this distribution matches the distributions we observed, fewer than 4% of the simulated processes have non-zero lifetimes. With so few jobs (and balanced initial placement) there is seldom any load imbalance in the system, and thus little benefit for preemptive migration. Furthermore, the [Eager et al. 1988] process lifetime distribution is exponential for jobs with non-zero lifetimes, the consequences of which we discuss in Section 2.2. For a more detailed explanation of this distribution and its effect on the study, see [Downey and Harchol-Balter 1995].

Krueger and Livny investigate the benefits of supplementing non-preemptive migration with preemptive migration and find that preemptive migration is worthwhile. They use a hyperexponential lifetime distribution that approximates closely the distribution we observed; as a result, their findings are largely in accord with ours. One difference between their work and ours is that they used a synthetic workload with Poisson arrivals. The workload we observed, and used in our trace-driven simulations, exhibits serial correlation; i.e. it is more bursty than a Poisson process. Another difference is that their migration policy requires several hand-tuned parameters. In Section 3.1 we show how to use the distribution of lifetimes to eliminate these parameters.

Like us, Bryant and Finkel discuss the distribution of process lifetimes and its effect on preemptive migration policy, but their hypothetical distributions are not based on system measurements [Bryant and Finkel 1981]. Also like us, they choose migrant processes on the basis of expected slowdown on the source and target hosts, but their estimation of those slowdowns is very different from ours. In particular, they use the distribution of process lifetimes to predict a host's future load as a function of its current load and the ages of the processes running there. We have examined this issue and found (1) that this model fails to predict future loads because it ignores future arrivals, and (2) that current load is the best predictor of future load (see Section 3.1). Thus, in our estimates of slowdown, we assume that the future load on a host is equal to the current load.

2. DISTRIBUTION OF LIFETIMES

The general shape of the distribution of process lifetimes in an academic environment has been known for a long time [Rosin 1965]: there are many short jobs and a few long jobs, and the variance of the distribution is greater than that of an exponential distribution.

In 1986, Cabrera measured UNIX processes and found that over 40% doubled their current age [Cabrera 1986]. That same year, Leland and Ott proposed a functional form for the process lifetime distribution, based on measurements of the lifetimes of 9.5 million UNIX processes between 1984 and 1985 [Leland and Ott 1986]. They conclude that process lifetimes have a UBNE (used-better-than-new-in-expectation) type of distribution. That is, the greater the current CPU age of a process, the greater its expected remaining CPU lifetime. Specifically, they find that for $T > 3$ seconds, the probability of a process's lifetime exceeding T seconds is rT^k, where $-1.25 < k < -1.05$ and r normalizes the distribution.

In contrast, Rommel [Rommel 1991] claims that his measurements show that "long processes have exponential service times." Many subsequent studies assume an exponential lifetime distribution.

Because of the importance of the process lifetime distribution for load balancing policies, we performed an independent study of this distribution, and found that the functional form proposed by Leland and Ott fits the observed distributions well, for processes with lifetimes greater than 1 second. This functional form is consistent across a variety of machines and workloads, and although the parameter, k, varies from -1.3 to -.8, it is generally near -1. Thus, as a *rule of thumb*,

—The probability that a process with age 1 second uses at least T seconds of total CPU time is about $1/T$.

—The probability that a process with age T seconds uses at least an additional T seconds of CPU time is about $1/2$. Thus, the median remaining lifetime of a process is equal to its current age.

Section 2.1 describes our measurements and the distribution of lifetimes we observed. Section 2.2 discusses other models for the distribution of lifetimes, and argues that the particular shape of this distribution is critical for evaluating migration policies.

2.1 Lifetime distribution when lifetime $> 1s$

To determine the distribution of lifetimes for UNIX processes, we measured the lifetimes of over one million processes, generated from a variety of academic workloads, including instructional machines, research machines, and machines used for system administration. We obtained our data using the UNIX command `lastcomm`, which outputs the CPU time used by each completed process.

Figure 1 shows the distribution of lifetimes from one of the machines. The plot shows only processes whose lifetimes exceed one second. The dotted (heavy) line

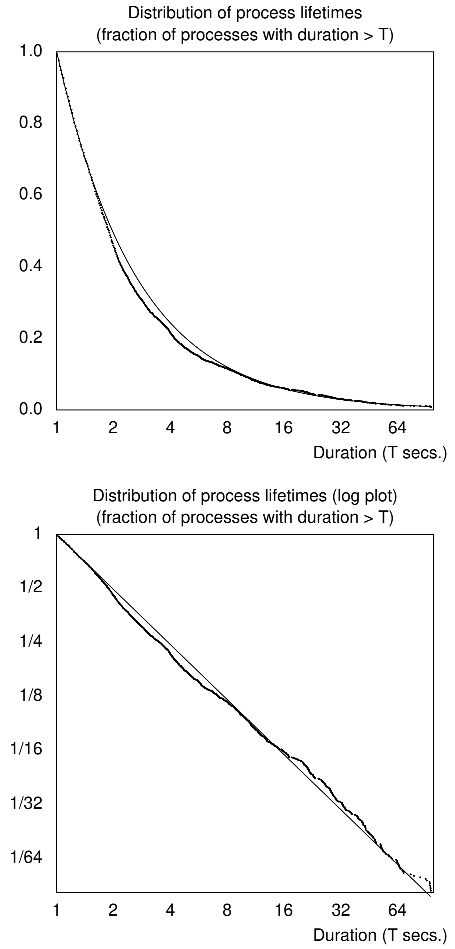

Fig. 1. (a) Distribution of lifetimes for processes with lifetimes greater than 1 second, observed on machine `po` mid-semester. The dotted (thicker) line shows the measured distribution; the solid (thinner) line shows the least squares curve fit. (b) The same distribution on a log-log scale. The straight line in log-log space indicates that the distribution can be modeled by T^k, where k is the slope of the line.

shows the measured distribution; the solid (thinner) line shows the least-squares fit to the data using the proposed functional form $Pr\{Lifetime > T\} = T^k$.

By visual inspection, it is clear that the proposed model fits the observed data well. In contrast, Figure 2 shows that it is impossible to find an exponential curve that fits the distribution of lifetimes we observed.

For all the machines we studied, the distribution of process lifetimes fits a curve of the form T^k, with k varying from -1.3 to -0.8 for different machines. Table 1 shows the value of the parameter for each machine we studied, estimated by an iteratively weighted least-squares fit (with no intercept, in accordance with the functional model). We calculated these estimates with the BLSS command `robust` [Abrahams and Rizzardi 1988].

The standard error associated with each estimate gives a confidence interval for that parameter (all of these parameters are statistically significant at a high degree of certainty). The R^2 value indicates the goodness of fit of the model — the values shown here indicate that the fitted curve accounts for greater than 99% of the variation of the observed values. Thus, the goodness of fit of these models is high (for an explanation of R^2 values, see [Larsen and Marx 1986]).

Table 2 shows the cumulative distribution function, probability density function, and conditional distribution function for process lifetimes. The second column shows these functions when $k = -1$, which we will assume for our analysis in Section 3.

The functional form we are proposing (the fitted distribution) has the property that its moments (mean, variance, etc.) are infinite. Of course, since the observed distributions have finite sample size, they have finite mean (0.4 seconds) and coefficient of variation (5–7). One must be cautious when summarizing long-tailed distributions, though, because calculated moments tend to be dominated by a few outliers. In our analyses we use more robust summary statistics (order statistics like the median, or the estimated parameter k) to summarize distributions, rather than moments.

2.1.1 *Process with lifetime < 1 second.* For processes with lifetimes less than 1 second, we did not find a consistent functional form; however, for the machines we studied these processes had an even lower hazard rate than those of age > 1 second. That is, the probability that a process of age $T < 1$ seconds lives another T seconds is always greater than 1/2. Thus for jobs with lifetimes less than 1 second, the median remaining lifetime is greater than the current age.

2.2 Why the distribution is critical

Many prior studies of process migration assume an exponential distribution of process lifetimes, both in analytical papers [Lin and Raghavendra 1993] [Mirchandaney et al. 1990] [Eager et al. 1986] [Ahmad et al. 1991] and in simulation studies [Kunz 1991] [Pulidas et al. 1988] [Wang and Morris 1985] [Evans and Butt 1993] [Livny and Melman 1982] [Zhang et al. 1995] [Chowdhury 1990]. The reasons for this assumption include: (1) analytic tractability, and (2) the belief that even if the actual lifetime distribution is in fact not exponential, assuming an exponential distribution will not affect the results of load balancing studies.

Regarding the first point, although the functional form that we and Leland and

Name of Host	Number of Procs	Number Procs > 1 sec	Estimated Distrib.	Std Error	R^2
po1	77440	4107	$T^{-0.97}$	.016	0.997
po2	154368	11468	$T^{-1.22}$	.012	0.999
po3	111997	7524	$T^{-1.27}$	.021	0.997
cory	182523	14253	$T^{-0.88}$	.030	0.982
pors	141950	10402	$T^{-0.94}$	.015	0.997
bugs	83600	4940	$T^{-0.82}$	.007	0.999
faith	76507	3328	$T^{-0.78}$	.045	0.964

Table 1. The estimated lifetime distribution for each machine measured, and the associated goodness of fit statistics. Description of machines: po is a heavily-used DECserver5000/240, used primarily for undergraduate coursework. Po1, po2, and po3 refer to measurements made on po mid-semester, late-semester, and end-semester. Cory is a heavily-used machine, used for coursework and research. Porsche is a less frequently-used machine, used primarily for research on scientific computing. Bugs is a heavily-used machine, used primarily for multimedia research. Faith is an infrequently-used machine, used both for video applications and system administration.

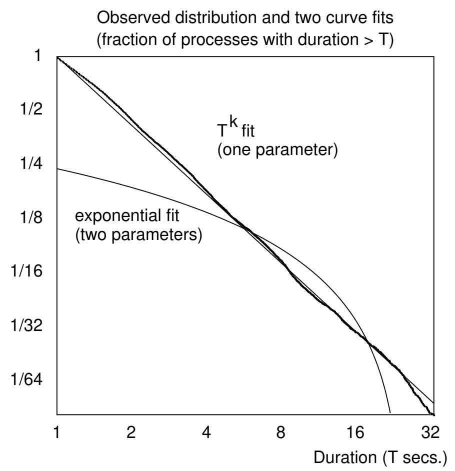

Fig. 2. In log-log space, this plot shows the distribution of lifetimes for the 13000 processes from our traces with lifetimes greater than second, and two attempts to fit a curve to this data. One of the fits is based on the model proposed in this paper, T^k. The other fit is an exponential curve, $c \cdot e^{-\lambda T}$. Although the exponential curve is given the benefit of an extra free parameter, it fails to model the observed data. The proposed model fits well. Both fits were performed by iteratively-weighted least squares.

Distribution of lifetimes for processes > 1 sec	When $k = -1$
$\mathbf{Pr}\{L > T \text{ sec}\} = T^k$	$1/T$
$\mathbf{Pr}\{T < L < T + dT \text{ sec}\} = -kT^{k-1}dT$	$1/T^2 \cdot dT$
$\mathbf{Pr}\{L > b \text{ sec} \mid \text{age} = a\} = \left(\frac{b}{a}\right)^k$	a/b

Table 2. The cumulative distribution function, probability density function, and conditional distribution function for the process lifetime L. The second column shows the functional form of each for the typical value $k = 1$.

Ott propose cannot be used in queueing models as easily as an exponential distribution, it nevertheless lends itself to some forms of analysis, as we show in Section 3.1.

Regarding the second point, we argue that the particular shape of the lifetime distribution affects the performance of migration policies, and therefore that it is important to model this distribution accurately. Specifically, the choice of a migration policy depends on how the expected remaining lifetime of a job varies with age. In our observations we found a distribution with the UBNE property — the expected remaining lifetime of a job increases linearly with age. As a result, we chose a migration policy that migrates only old jobs.

But different distributions yield in different relationships between the age of a process and its remaining lifetime. For example, a uniform distribution has the NBUE property — the expected remaining lifetime *decreases* linearly with age. Thus if the distribution of lifetimes were uniform, the migration policy should choose to migrate only young processes. In this case, we expect non-preemptive migration to perform better than preemptive migration.

As another example, the exponential distribution is memoryless — the remaining lifetime of a job is independent of its age. In this case, since all processes have the same expected lifetimes, the migration policy might choose to migrate the process with the lowest migration cost, regardless of age.

As a final example, processes whose lifetimes are chosen from a uniform log distribution (a uniform distribution in log-space) have a remaining lifetime that increases up to a point and then begins to decrease. In this case, the best migration policy might be to migrate jobs that are old enough, but not too old.

Thus different distributions, even with the same mean and variance, can lead to different migration policies. In order to evaluate a proposed policy, it is critical to choose a distribution model with the appropriate relationship between expected remaining lifetime and age.

Some studies have used hyperexponential distributions to model the distribution of lifetimes. These distributions may or may not have the right behavior, depending on how accurately they fit observed distributions. [Krueger and Livny 1988] use a three-stage hyperexponential with parameters estimated to fit observed values. This distribution has the appropriate UBNE property. But the two-stage hyperexponential distribution [Eager et al. 1988] use is memoryless; the remaining lifetime of a job is independent of its age (for jobs with nonzero lifetimes). According to this distribution, migration policy is irrelevant; all processes are equally good candidates for migration. This result is clearly in conflict with our observations.

Assuming the wrong lifetime distribution may also underestimate the benefits of preemptive migration. The heavy tail of our measured lifetime distribution implies that a tiny fraction of the jobs require more CPU than all the other jobs combined. As we'll discuss in Section 5.4, part of the power of preemptive migration is its ability to identify those few hogs. In a lifetime distribution without such a heavy tail, preemptive migration might not be as effective.

3. MIGRATION POLICY

A migration policy is based on two decisions: when to migrate processes and which processes to migrate. The first question concerns how often or at what times the system checks for eligible migrants. We address this issue briefly in Section 5.1.1. The focus of this paper is the second question, also known as the *selection policy*:

> Given that the load at a host is too high, how do we choose *which* process to migrate?

Our heuristic is to migrate processes that are expected to have long remaining lifetimes. The motivation for this heuristic is twofold. From the process's perspective, migration time has a large impact on response time. A process would choose to migrate only if the migration overhead could be amortized over a longer lifetime. From the perspective of the source host, it takes a significant amount of work to package a process for migration. The host would only choose to migrate processes that are likely to be more expensive to run than to migrate.

Many existing migration policies only migrate newborn processes (no preemption), because these processes have no allocated memory and thus their migration cost is low. The idea of migrating newborn processes might also stem from the fallacy that process lifetimes have an exponential distribution, implying that all processes have equal expected remaining lifetimes regardless of their age, so one should migrate the cheapest processes. The problem with only migrating newborn processes is that, according to the process lifetime distribution, newborn processes are unlikely to live long enough to justify the cost of remote execution. In fact, our measurements show that over 70% of processes have lifetimes smaller than the smallest non-preemptive migration cost (see Table 3).

Thus a newborn migration policy is only justified if the system has prior knowledge about processes and can selectively migrate processes likely to be CPU hogs. We have found, though, that the ability of the system to predict process lifetimes by name is limited (Section 5.4).

Can we do better? The distribution of lifetimes implies that we expect and old process to run longer than a young process; thus, it is preferable to migrate old processes.

There are two potential problems with this approach. First, since the vast majority of processes are short, there might not be enough old processes to have a significant load balancing effect. In fact, although there are few long-lived processes, they account for a large part of the total CPU load. According to our measurements, typically fewer than 4% of processes live longer than 2 seconds, yet these processes make up more than 60% of the total CPU load. This is due to the long tail of the process lifetime distribution. Furthermore, we will see that the ability to migrate even a few large jobs can have a large effect on system performance, since a single long process on a busy host imposes slowdowns on many short processes.

A second problem with migrating old processes is that the migration cost for an active process is much greater than the cost of remote execution. If preemptive migration is done carelessly, this additional cost might overwhelm the benefit of migrating processes with longer expected lives.

3.1 Our Migration Policy

For this reason, we propose a strategy that guarantees that every migration improves the expected performance of the migrant process and the other processes at the source host. This strategy migrates a process *only if it improves the expected slowdown of the process*, where slowdown is defined as in Section 1.2. Of course, processes on the target host are slowed by an arriving migrant, but on a moderately-loaded system there are almost always idle hosts; thus the number of processes at the target host is usually zero. In any case, the number of processes at the target is always less than the number at the source.

If there is no process on the host that satisfies the above migration criterion, no migration is done. If migration costs are high, few processes will be eligible for migration; in the extreme there will be no migration at all. But in no case is the performance of the system worse (in expectation) than the performance without migration.

Using the distribution of process lifetimes, we calculate the expected slowdown imposed on a migrant process, and use this result to derive a minimum age for migration based on the cost of migration. Denoting the age of the migrant process by a; the cost of migration by c; the (eventual total) lifetime of the migrant by L, the number of processes at the source host by n; and the number of processes at the target host (including the migrant) by m, we have:

$$
\begin{aligned}
&\mathbf{E}\,\{\text{slowdown of migrant}\}\\
&= \int_{t=a}^{\infty} \mathbf{Pr}\left\{\begin{array}{c}\text{Lifetime of}\\ \text{migrant is t}\end{array}\right\} \cdot \left\{\begin{array}{c}\text{Slowdown given}\\ \text{lifetime is t}\end{array}\right\} dt\\
&= \int_{t=a}^{\infty} \mathbf{Pr}\,\{t \le L < t+dt | L \ge a\} \cdot \frac{na+c+m(t-a)}{t}\\
&= \int_{t=a}^{\infty} \frac{a}{t^2} \cdot \frac{na+c+m(t-a)}{t} dt\\
&= \frac{1}{2}\left(\frac{c}{a}+m+n\right)
\end{aligned}
$$

If there are n processes at a heavily loaded host, then a process should be eligible for migration only if its expected slowdown after migration is less than n (which is the slowdown it expects in the absence of migration).

Thus, we require $\frac{1}{2}(\frac{c}{a}+m+n) < n$, which implies

$$
\boxed{\text{Minimum migration age} = \frac{\text{Migration cost}}{n-m}}
$$

We can extend this analysis to the case of heterogeneous processor speeds by applying a scale factor to n or m.

This analysis assumes that current load predicts future load; that is, that the load at the source and target hosts will be constant during the migration. In an attempt to evaluate this assumption, and possibly improve it, we considered a number of alternative load predictors, including (1) taking a load average (over an interval of time), (2) summing the ages of the processes running on the host, and a (3) calculating a prediction of survivors and future arrivals based on the distribution model proposed here. We found that current (instantaneous) load is the best single predictor, and that using several predictive variables in combination did not greatly improve the accuracy of prediction. These results are in accord with Zhou [Zhou 1987] and Kunz [Kunz 1991].

3.2 Prior Preemptive Policies

Only a few preemptive strategies have been implemented in real systems or proposed in prior studies. The three that we have found are, like ours, based on the principle that a process should be migrated if it is old enough.

In many cases, the definition of *old enough* depends on a "voodoo" constant[1]: a free parameter whose value is chosen without explanation, and that would need to be re-tuned for a different system or another workload.

Under Leland and Ott's policy, a process p is eligible for migration if

$$\text{age}(p) \; > \; \text{ages of } k \text{ younger jobs at host}$$

where k is a free parameter called MINCRIT [Leland and Ott 1986]. Krueger and Livny's policy, like ours, takes the job's migration cost into account. A process p is eligible for migration if

$$\text{age}(p) \; > 0.1 * \text{migration cost}(p)$$

but they do not explain how they chose the value 0.1 [Krueger and Livny 1988]. The MOSIX policy is similar [Barak et al. 1993]; a process is eligible for migration if

$$\text{age}(p) \; > 1.0 * \text{migration cost}(p).$$

The choice of the constant (1.0) in the MOSIX policy ensures that the slowdown of a migrant process is never more than 2, since in the worst case the migrant completes immediately upon arrival at the target.

Despite this justification, the choice of the maximum slowdown (2) is arbitrary. We expect the MOSIX policy to be too restrictive, for two reasons. First, it ignores the slowdown that would be imposed at the source host in the absence of migration (presumably there is more than one process there, or the system would not be attempting to migrate processes away). Second, it is based on the worst-case slowdown rather than the expected slowdown. In Section 5.5, we show that the best choice for this parameter, for our workload, is usually near 0.4, but it depends on load.

[1] This term was coined by Professor John Ousterhout at U.C. Berkeley.

4. MODEL OF MIGRATION COSTS

Migration cost has such a large effect on the performance of preemptive load balancing; this section presents the model of migration costs we use in our simulation studies.

We model the cost of migrating an active process as the sum of a *fixed migration cost* for migrating the process's system state and a *memory transfer cost* that is proportional to the amount of the process's memory that must be transferred.

We model *remote execution cost* as a fixed cost; it is the same for all processes. The cost of remote execution includes sending the command and arguments to the remote host, logging in or otherwise authenticating the process, and creating a new shell and environment on the remote host.

Throughout this paper, we use the following notation:

—r: the cost of remote execution, in seconds

—f: the fixed cost of preemptive migration, in seconds

—b: the memory transfer bandwidth, in MB per second

—m: the memory size of migrant processes, in MB

and thus:

$$\text{cost of remote execution} = r$$
$$\text{cost of preemptive migration} = f + m/b$$

where the quotient m/b is the memory transfer cost.

4.1 Memory transfer costs

The amount of a process's memory that must be transferred during preemptive migration depends on properties of the distributed system. Douglis and Ousterhout [Douglis and Ousterhout 1991] have an excellent discussion of this issue, and we borrow from them here.

At the most, it might be necessary to transfer a process's entire memory. On a system like Sprite, which integrates virtual memory with a distributed file system, it is only necessary to write dirty pages to the file system before migration. When the process is restarted at the target host, it will retrieve these pages. In this case the cost of migration is proportional to the size of the resident set rather than the size of memory.

In systems that use precopying, such as V [Theimer et al. 1985], pages are transferred while the program continues to run at the source host. When the job stops execution at the source, it will have to transfer again any pages that have become dirty during the precopy. Although the number of pages transferred might be increased, the delay imposed on the migrant process is greatly decreased. Additional techniques can reduce the cost of transferring memory even more [Zayas 1987].

4.2 Migration costs in real systems

The specific parameters of migration cost depend not only on the nature of the system (as discussed above) but also on the speed of the network. Tables 3 and 4 show reported costs from a variety of real systems. Later we will use a trace-driven simulator to evaluate the effect of these parameters on system performance. We

System	Hardware	Cost of rexec, r
Sprite [Douglis and Ousterhout 1991]	SPARCstation1 10Mb/sec Ethernet	0.33 sec
GLUNIX [Vahdat et al. 1994] [Vahdat 1995]	HP workstations ATM network	0.25 to 0.5 sec
MIST [Prouty 1996]	HP9000/720 10Mb/sec Ethernet	0.33 sec
Utopia [Zhou et al. 1993]	DEC 3100 and SPARC IPC Ethernet	0.1 sec

Table 3. Cost of non-preemptive migration in various systems. Some of these numbers were obtained from personal communication with the authors.

System	Hardware	Fixed Cost, f	Inverse Bandwidth, $1/b$
Sprite [Douglis and Ousterhout 1991]	SPARCstation1 10Mb/sec Ethernet	0.33 sec	2.00 sec/MB
MOSIX [Barak et al. 1993] [Braverman 1995]	Intel Pentium 90MHz and 486 66MHz	0.006 sec	0.44 sec/MB
MIST [Prouty 1996]	HP9000/720s 10Mb/sec Ethernet	0.24 sec	0.99 sec/MB

Table 4. Values for preemptive migration costs from various systems. Many of these numbers were obtained from personal communication with the authors. The memory transfer cost is the product of the inverse bandwidth, $1/b$, and the amount of memory that must be transferred, m.

will make the pessimistic simplification that a migrant's entire memory must be transferred, although, as pointed out above, this is not necessarily the case.

5. TRACE-DRIVEN SIMULATION

In this section we present the results of a trace-driven simulation of process migration. We compare two migration strategies: our proposed age-based preemptive migration strategy (Section 3.1) and a non-preemptive strategy that migrates newborn processes according to the process name (similar to strategies proposed by [Wang et al. 1993] and [Svensson 1990]). With the intention of finding a conservative estimate of the benefit of preemptive migration, we give the name-based strategy the benefit of several unrealistic advantages; for example, the name-lists are derived from the same trace data used by the simulator.

Section 5.1 describes the simulator and the two strategies in more detail. We use the simulator to run three experiments. First, in Section 5.2, we evaluate the sensitivity of each strategy to the migration costs r, f, b, and m discussed in Section 4. Next, in Section 5.3, we choose values for these parameters that are representative of current systems and compare the performance of the two strategies in detail. In Section 5.4 we discuss why the preemptive policy outperforms the non-preemptive policy. Lastly, in Section 5.5, we evaluate the analytic criterion for migration age proposed in Section 3.1, compared to criteria used in previous studies.

5.1 The simulator

We have implemented a trace-driven simulation of a network of six identical workstations.[2] We selected six daytime intervals from the traces on machine po (see Section 2.1), each from 9:00 a.m. to 5:00 p.m. From the six traces we extracted the start times and CPU durations of the processes. We then simulated a network where each of six hosts executes (concurrently with the others) the process arrivals from one of the daytime traces.

Although the workloads on the six hosts are homogeneous in terms of the job mix and distribution of lifetimes, there is considerable variation in the level of activity during the eight-hour trace. For most of the traces, every arriving process finds at least one idle host in the system, but in the two busiest traces, a small fraction of processes (0.1%) arrive to find all hosts busy. In order to evaluate the effect of changes in system load, we divided the eight-hour trace into eight one-hour intervals. We refer to these as *runs* 0 through 7, where the runs are sorted from lowest to highest load. Run 0 has a total of 15000 processes submitted to the six simulated hosts; Run 7 has 30000 processes. The average duration of processes (for all runs) is 0.4 seconds. Thus the total utilization of the system, ρ, is between 0.27 and 0.54.

The birth process of jobs at our hosts is burstier than a Poisson process. For a given run and a given host, the serial correlation in interarrival times is typically between .08 and .24, which is significantly higher than one would expect from a Poisson process (uncorrelated interarrival times yield a serial correlation of 0.0; perfect correlation is 1.0).

Although the start times and durations of the processes come from trace data, the memory size of each process, which determines its migration cost, is chosen randomly from a measured distribution (see Section 5.2). This simplification obliterates any correlations between memory size and other process characteristics, but it allows us to control the mean memory size as a parameter and examine its effect on system performance.

In our system model, we assume that processes are always ready to run; i.e. they are never blocked on I/O. During a given time interval, we divide CPU time equally among the processes on the host (processor sharing).

In real systems, part of the migration time is spent on the source host packaging the transferred pages, part in transit in the network, and part on the target host unpacking the data. The size of these parts and whether they can be overlapped depend on details of the system. In our simulation we charge the entire cost of migration to the source host. This simplification is conservative in the sense that it makes preemptive migration less effective.

5.1.1 *Strategies.* We compare the preemptive migration strategy proposed in Section 3.1 with a non-preemptive migration strategy, where the non-preemptive strategy is given unfair advantages. For purposes of comparison, we have tried to make the policies as simple and as similar as possible. For both types of migration, we consider performing a migration only when a new process is born, even though a

[2] The trace-driven simulator and the trace data are available at http://http.cs.berkeley.edu/~harchol/loadbalancing.html.

preemptive strategy might benefit by initiating migrations at other times. Also, for both strategies, a host is considered heavily-loaded any time it contains more than one process; in other words, any time it would be sensible to consider migration. Finally, we use the same location policy in both cases: the host with the lowest instantaneous load is chosen as the target host (ties are broken by random selection).

Thus the only difference between the two migration policies is which processes are considered eligible for migration:

Name-based non-preemptive migration. A process is eligible for migration only if its name is on a list of processes that tend to be long-lived. If an eligible process is born at a heavily-loaded host, the process is executed remotely on the selected host. Processes cannot be migrated once they have begun execution.

The performance of this strategy depends on the list of eligible process names. We derived this list by sorting the processes from the traces according to name and duration and selecting the 15 common names with the longest *mean durations*. We chose a threshold on mean duration that is empirically optimal (for this set of runs). Adding more names to the list detracts from the performance of the system, as it allows more short-lived processes to be migrated. Removing names from the list detracts from performance as it becomes impossible to migrate enough processes to balance the load effectively. Since we used the trace data itself to construct the list, our results may overestimate the performance benefits of this strategy.

Age-based preemptive migration. A process is eligible for migration only if it has aged for some fraction of its migration cost. Based on the derivation in Section 3.1, this fraction is $\frac{1}{n-m}$, where n (respectively m) is the number of processes at the source (target) host. When a new process is born at a heavily-loaded host, all processes that satisfy the migration criterion are migrated away.

This strategy understates the performance benefits of preemptive migration, because it does not allow the system to initiate migrations except when a new process arrives.

As described in Section 3.1, we also modeled other location policies based on more complicated predictors of future loads, but none of these predictors yielded significantly better performance than the instantaneous load we use here.

We also considered the effect of allowing preemptive migration at times other than when a new process arrives. Ideally, one would like to initiate a migration whenever a process becomes eligible (since the eligibility criterion guarantees that the performance of the migrant will improve in expectation). One of the strategies we considered performs periodic checks of each process on a heavily-loaded host to see if any satisfy the criterion. The performance of this strategy is significantly better than that of the simpler policy (migrating only at process arrival times).

5.1.2 *Metrics.* We evaluate the effectiveness of each strategy according to the following performance metrics:

Mean slowdown. Slowdown is the ratio of wall-clock execution time to CPU time (thus, it is always greater than one). The average slowdown of all jobs is a common metric of system performance. When we compute the ratio of mean slowdowns (as from different strategies) we will use normalized slowdown, which is the ratio of

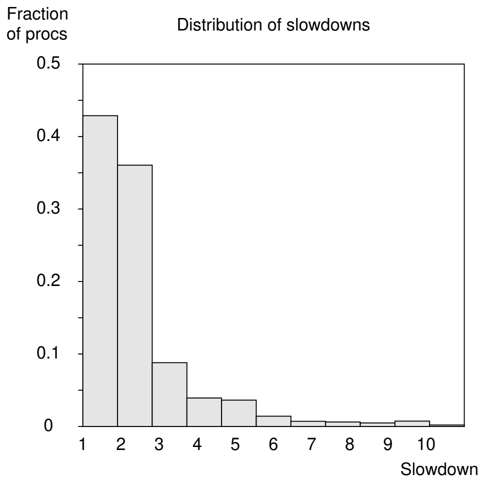

Fig. 3. Distribution of process slowdowns for run 0 (with no migration). Most processes suffer small slowdowns, but the processes in the tail of the distribution are more noticeable and annoying to users.

inactive time (the *excess* slowdown caused by queueing and migration delays) to CPU time. For example, if the (unnormalized) mean slowdown drops from 2.0 to 1.5, the ratio of normalized mean slowdowns is $0.5/1.0 = 0.5$: a 50% reduction in delay.

Mean slowdown alone is not a sufficient measure of the difference in performance of the two strategies; it understates the advantages of the preemptive strategy for these two reasons:

—Skewed distribution of slowdowns: Even in the absence of migration, the majority of processes suffer small slowdowns (typically 80% are less than 3.0. See Figure 3). The value of the mean slowdown is dominated by this majority.

—User perception: From the user's point of view, the important processes are the ones in the tail of the distribution, because although they are the minority, they cause the most noticeable and annoying delays. Eliminating these delays might have a small effect on the mean slowdown, but a large effect on a user's perception of performance.

Therefore, we will also consider the following three metrics:

Variance of slowdown. This metric is often cited as a measure of the unpredictability of response time [Silberschatz et al. 1994], which is a nuisance for users trying to schedule tasks. In light of the distribution of slowdowns, however, it may be more meaningful to interpret this metric as a measure of the length of the tail of the distribution; i.e. the number of jobs that experience long delays. (See Figure 5b).

Number of severely slowed processes. In order to quantify the number of noticeable delays explicitly, we consider the number (or percentage) of processes that are severely impacted by queueing and migration penalties. (See Figures 5c and 5d).

Mean slowdown of long jobs. Delays in longer jobs (those with lifetimes greater than .5 seconds) are more perceivable to users than delays in short jobs. (See Figure 6).

5.2 Sensitivity to migration costs

In this section we compare the performance of the non-preemptive and preemptive strategies over a range of values of r, f, b and m (the migration cost parameters defined in Section 4).

For the following experiments, we chose the remote execution cost $r = .3$ seconds. We considered a range for the fixed migration cost of $.1 < f < 10$ seconds.

The memory transfer cost is the quotient of m (the memory size of the migrant process) and b (the bandwidth of the network). We chose the memory transfer cost from a distribution with the same shape as the distribution of process lifetimes, setting the mean memory transfer cost (MMTC) to a range of values from 1 to 64 seconds. The shape of this distribution is based on an informal study of memory-use patterns on the same machines from which we collected trace data. The important feature of this distribution is that there are many jobs with small memory demands and a few jobs with very large memory demands. Empirically, the exact form of this distribution does not affect the performance of either migration strategy strongly, but of course the mean (MMTC) does have a strong effect.

Figures 4a and 4b are contour plots of the ratio of the performance of the two migration strategies using normalized slowdown. Specifically, for each of the eight one-hour runs we calculate the mean (respectively standard deviation) of the slowdown imposed on all processes that complete during the hour. For each run, we then take the ratio of the means (standard deviations) of the two strategies. Lastly we take the geometric mean of the eight ratios (for discussion of the geometric mean, see [Hennessy and Patterson 1990]).

The two axes in Figure 4 represent the two components of the cost of preemptive migration, namely the fixed cost, f, and the MMTC, m/b. The cost of non-preemptive migration, r, is fixed at 0.3 seconds. As expected, increasing either the fixed cost of migration or the MMTC hurts the performance of preemptive migration. The contour line marked 1.0 indicates the crossover where the performance of preemptive and non-preemptive migration is equal (the ratio is 1.0). For smaller values of the cost parameters, preemptive migration performs better; for example, if the fixed migration cost is 0.3 seconds and the MMTC is 2 seconds, the normalized mean slowdown with preemptive migration is almost 40% lower than with non-preemptive migration. When the fixed cost of migration or the MMTC are very high, almost all processes are ineligible for preemptive migration; thus, the preemptive strategy does almost no migrations. The non-preemptive strategy is unaffected by these costs so the non-preemptive strategy can be more effective.

Figure 4b shows the effect of migration costs on the standard deviation of slowdowns. The crossover point — where non-preemptive migration surpasses preemptive migration — is considerably higher here than in Figure 4a. Thus there is a region where preemptive migration yields a higher mean slowdown than non-preemptive migration, but a lower standard deviation. The reason for this is that non-preemptive migration occasionally chooses a process for remote execution that turns out to be short-lived. These processes suffer large delays (relative to their

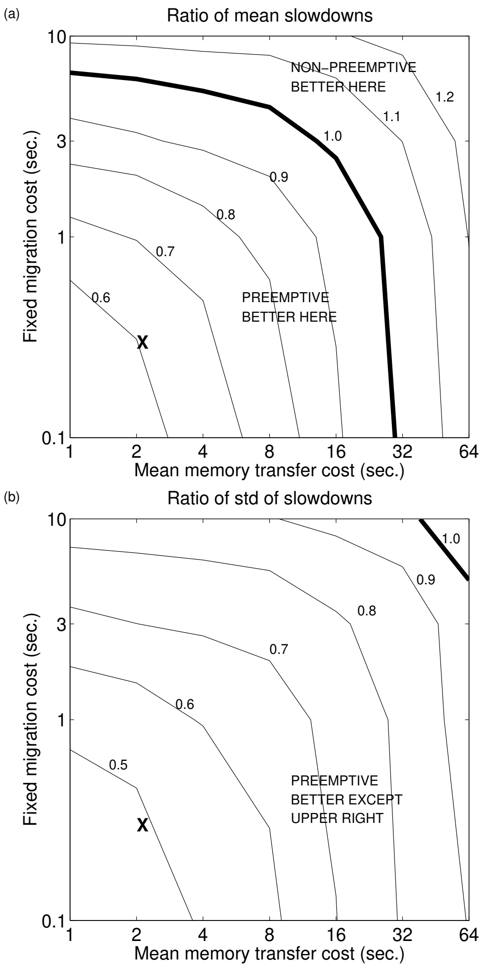

Fig. 4. (a) The performance of preemptive migration relative to non-preemptive migration deteriorates as the cost of preemptive migration increases. The two axes are the two components of the preemptive migration cost. The cost of non-preemptive migration is held fixed. The X marks the particular set of parameters we will consider in the next section. (b) The standard deviation of slowdown may give a better indication of a user's perception of system performance than mean slowdown. By this metric, the benefit of preemptive migration is even more significant.

run times) and add to the tail of the distribution of slowdowns. In the next section, we show cases in which the standard deviation of slowdowns is actually worse with non-preemptive migration than with no migration at all (three of the eight runs).

5.3 Comparison of preemptive and non-preemptive strategies

In this section we choose migration cost parameters representative of current systems (see Section 4.2) and use them to examine more closely the performance of the two migration strategies. The values we chose are:

—r: the cost of remote execution, 0.3 seconds

—f: the fixed cost of preemptive migration, 0.3 seconds

—b: the memory transfer bandwidth, 0.5 MB per second

—m: the mean memory size of migrant processes, 1 MB

In Figures 4a and 4b, the point corresponding to these parameter values is marked with an `X`. Figure 5 shows the performance of the two migration strategies at this point (compared to the case of no migration).

Non-preemptive migration reduces the normalized mean slowdown (Figure 5a) by less than 20% for most runs (and 40% for the two runs with the highest loads). Preemptive migration reduces the normalized mean slowdown by 50% for most runs (and more than 60% for two of the runs). The performance improvement of preemptive migration over non-preemptive migration is typically between 35% and 50%.

As discussed above, we feel that the mean slowdown (normalized or not) understates the performance benefits of preemptive migration. We have proposed other metrics to try to quantify these benefits. Figure 5b shows the standard deviation of slowdowns, which reflects the number of severely impacted processes. Figures 5c and 5d explicitly measure the number of severely impacted processes, according to two different thresholds of acceptable slowdown. By these metrics, the benefits of migration in general appear greater, and the discrepancy between preemptive and non-preemptive migration appears much greater. For example in Figure 5d, in the absence of migration, 7 – 18% of processes are slowed by a factor of 5 or more. Non-preemptive migration is able to eliminate 42–62% of these, which is a significant benefit, but preemptive migration consistently eliminates nearly all (86–97%) severe delays.

An important observation from Figure 5b is that for several runs, non-preemptive migration actually makes the performance of the system worse than if there were no migration at all. For the preemptive migration strategy, this outcome is nearly impossible, since migrations are only performed if they improve the slowdowns of all processes involved (in expectation). In the worst case, then, the preemptive strategy will do no worse than the case of no migration (in expectation).

Another benefit of preemptive migration is graceful degradation of system performance as load increases (as shown in Figure 5). In the presence of preemptive migration, both the mean and standard deviation of slowdown are nearly constant, regardless of the overall load on the system.

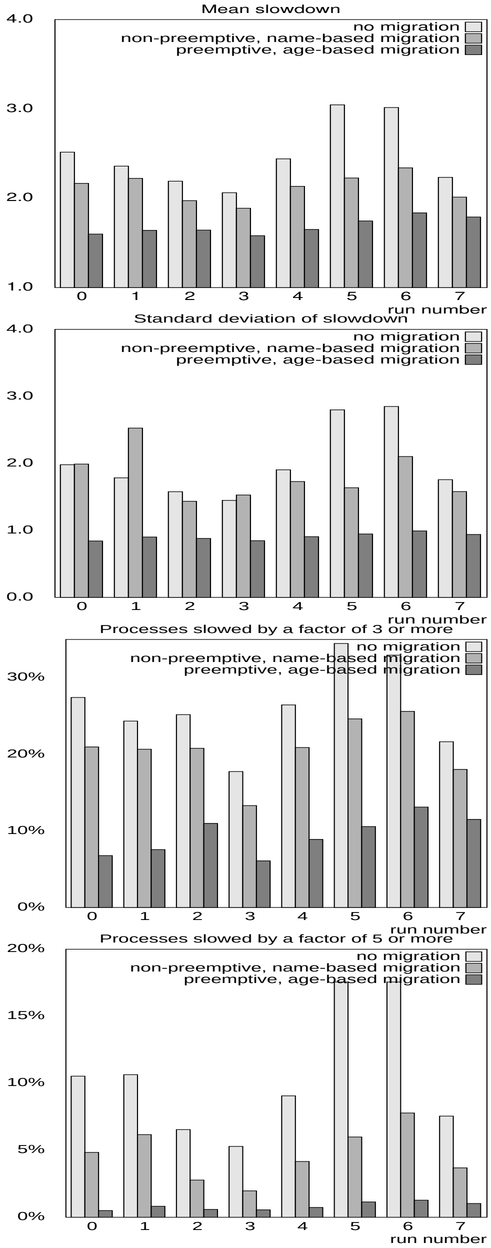

Fig. 5. (a) Mean slowdown. (b) Standard deviation of slowdown. (c) Percentage of processes slowed by a factor of 3 or more. (d) Percentage of processes slowed by a factor of 5 or more.

5.4 Why preemptive migration outperforms non-preemptive migration

The alternate metrics discussed above shed some light on the reasons for the performance difference between preemptive and non-preemptive migration. We consider two kinds of mistakes a migration system might make:

Failing to migrate long-lived jobs. This type of error imposes moderate slowdowns on a potential migrant, and, more importantly, inflicts delays on short jobs that are forced to share a processor with a CPU hog. Under non-preemptive migration, this error occurs whenever a long-lived process is not on the name-list, possibly because it is an unknown program or an unusually long execution of a typically short-lived program. Preemptive migration can correct these errors by migrating long jobs later in their lives.

Migrating short-lived jobs. This type of error imposes large slowdowns on the migrated process, wastes network resources, and fails to effect significant load balancing. Under non-preemptive migration, this error occurs when a process whose name is on the eligible list turns out to be short-lived. Our preemptive migration strategy all but eliminates this type of error by guaranteeing that the performance of a migrant improves in expectation.

Even occasional mistakes of the first kind can have a large impact on performance, because one long job on a busy machine will impede many small jobs. This effect is aggravated by the serial correlation between arrival times (see Section 5.1), which suggests that a busy host is likely to receive many future arrivals.

Thus, an important feature of a migration policy is its ability to identify long-lived jobs for migration. To evaluate this ability, we consider the average lifetime of the processes chosen for migration under each policy. Under non-preemptive migration, the average lifetime of migrant processes was 2.0 seconds (the mean lifetime for all processes is 0.4 seconds), and the median lifetime of migrants was 0.9 seconds. The non-preemptive policy migrated about 1% of all jobs, which accounted for 5.7% of the total CPU.

The preemptive migration policy was better able to identify long jobs; the average lifetime of migrant processes under preemptive migration was 4.9 seconds; the median lifetime of migrants was 2.0 seconds. The preemptive policy migrated 4% of all jobs, but since these migrants were long-lived, they accounted for 55% of the total CPU.

Thus the primary reason for the success of preemptive migration is its ability to identify long jobs accurately and to migrate those jobs away from busy hosts.

The second type of error did not have as great an impact on the mean slowdown for all processes, but it did impose large slowdowns on some small processes. These outliers are reflected in the standard deviation of slowdowns — because the non-preemptive policy sometimes migrates very short jobs, it can make the standard deviation of slowdowns worse than with no migration (see Figure 5b). The age-based preemptive migration criterion eliminates most errors of this type by guaranteeing that the performance of the migrant will improve in expectation.

There is, however, one type of migration error that is more problematic for preemptive migration than for non-preemptive migration: stale load information. A target host may have a low load when a migration is initiated, but its load may have increased by the time the migrant arrives. This is more likely for a preemptive migration because the migration time is longer. In our simulations, we found that these errors do occur, although infrequently enough that they do not have a severe impact on performance.

Specifically, we counted the number of migrant processes that arrived at a target host and found that the load was higher than it had been at the source host when migration began. For most runs, this occurred less than 0.5% of the time (for two runs with high loads it was 0.7%). Somewhat more often, 3% of the time, a migrant process arrived at a target host and found that the load at the target was greater than the *current* load at the source. These results suggest that the performance of a preemptive migration strategy might be improved by a reservation system as in MOSIX.

One other potential problem with preemptive migration is the volume of network traffic that results from large memory transfers. In our simulations, we did not model network congestion, on the assumption that the traffic generated by migration would not be excessive. This assumption seems to be reasonable: under our preemptive migration strategy fewer than 4% of processes are migrated once and fewer than .25% of processes are migrated more than once. Furthermore, there is seldom more than one migration in progress at a time.

In summary, the advantage of preemptive migration — its ability to identify long jobs and move them away from busy hosts — overcomes its disadvantages (longer migration times and stale load information).

5.4.1 *Effect of migration on short and long jobs.* We have claimed that identifying long jobs and migrating them away from busy hosts helps not only the long jobs (which run on more lightly-loaded hosts) but also the short jobs that run on the source host. To test this claim, we divided the processes into three lifetime groups and measured the performance benefit for each group due to migration. The number of jobs in short group is roughly ten times the number in the medium group, which in turn is roughly ten times the number in the long group. Figure 6 shows that migration reduces the mean slowdown of all three groups: for non-preemptive migration the improvement is the same for all groups; under preemptive migration the long jobs enjoy a slightly greater benefit.

This breakdown by lifetime group is useful for evaluating various metrics of system performance. The metric we are using here, slowdown, gives equal weight to all jobs; as a result, the mean slowdown metric is dominated by the most populous group, short jobs. Another common metric, residence time, effectively weights jobs according to their lifetimes. Thus the mean residence time metric reflects, primarily, the performance benefit for long jobs. Under the mean residence time metric, then, preemptive migration appears even more effective.

5.5 Evaluation of analytic migration criterion

As derived in Section 3.1, the minimum age for a migrant process according to our *analytic criterion* is

$$\begin{array}{c}\text{Minimum}\\ \text{migration age}\end{array} = \frac{\text{Migration cost}}{n - m}$$

where n is the load at the source host and m is the load at the target host (including

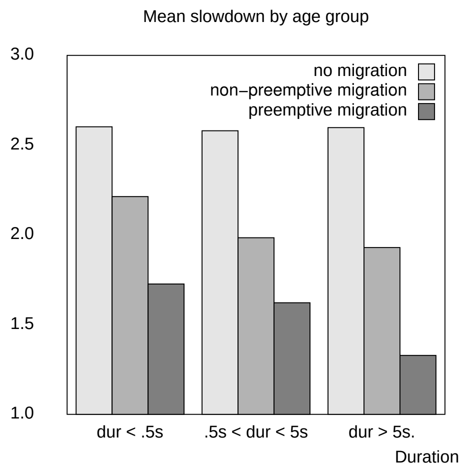

Fig. 6. Mean slowdown broken down by lifetime group.

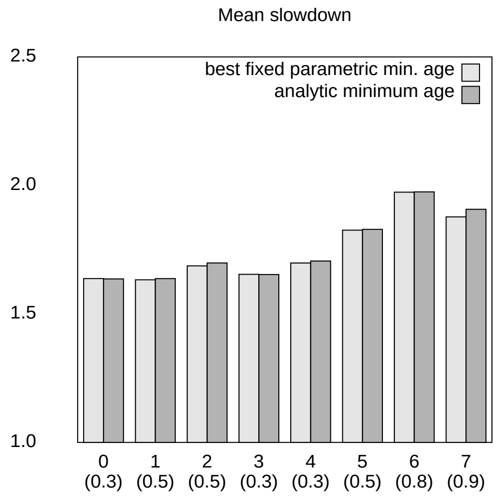

Fig. 7. The mean slowdown for eight runs, using the two criteria for minimum migration age. The value of the best fixed parameter α is shown in parentheses for each run.

the potential migrant).

We compare the analytic criterion with the *fixed parameter criterion*:

$$\begin{array}{c}\text{Minimum}\\ \text{migration age}\end{array} = \alpha * \text{Migration cost}$$

where α is a free parameter. This parameter is meant to model preemptive migration strategies in the literature, as discussed in Section 3.2. For comparison, we will use the *best fixed parameter*, which is, for each run, the value that yields the smallest mean slowdown. Of course, this gives the fixed parameter criterion a considerable advantage.

Figure 7 compares the performance of the analytic minimum age criterion with the best fixed parameter. The best fixed parameter varies considerably from run to run, and appears to be roughly correlated with the average load during the run (the runs are sorted in increasing order of total load).

The performance of the analytic criterion is always within a few percent of the performance of the best fixed value criterion. The advantage of the analytic criterion is that it is parameterless, and therefore more robust across a variety of workloads. We feel that the elimination of one free parameter is a useful result in an area with so many (usually hand-tuned) parameters.

This result also suggests that the parameter used by Krueger and Livny ($\alpha = 0.1$) is too low, and the parameter used in MOSIX ($\alpha = 1.0$) is too high, at least for this workload (see Section 3.2).

6. WEAKNESSES OF THE MODEL

Our simulation ignores a number of factors that would affect the performance of migration in real systems:

CPU-bound jobs only. Our model considers all jobs CPU-bound; thus, their response time necessarily improves if they run on a host with a lighter load. For I/O bound jobs, however, CPU contention has little effect on response time. These jobs would benefit less from migration. To see how large a role this plays in our results, we noted the names of the processes that appear most frequently in our traces (with CPU time greater than 1 second, since these are the processes most likely to be migrated). The most common names were cc1plus and cc1, both of which are CPU bound. Next most frequent were: trn, cpp, ld, jove (a version of emacs), and ps. So although some jobs in our traces are in reality interactive, our simple model is reasonable for many of the most common jobs. In Section 7 we discuss further implications of a workload including interactive, I/O-bound, and non-migratable jobs.

Environment. Our migration strategy takes advantage of the used-better-than-new property of process lifetimes. In an environment with a different distribution, this strategy will not be effective.

Local scheduling. We assume that local scheduling on the hosts is similar to round-robin. Other policies, like feedback scheduling, can reduce the impact of long jobs on the performance of short jobs, and thereby reduce the need for load balancing. We explore this issue in more detail in Section 7 and find that preemptive migration is still beneficial under feedback scheduling.

Memory size. One weakness of our model is that we chose memory sizes from a measured distribution and therefore our model ignores any correlation between memory size and other process characteristics. To justify this simplification, we conducted an informal study of processes in our department, and found no correlation between memory size and process CPU usage. Krueger and Livny report a similar observation [Krueger and Livny 1988]. Thus, this may be a reasonable simplification.

Network contention. Our model does not consider the effect of increased network traffic as a result of process migration. We observe, however, that for the load levels we simulated, migrations are occasional (one every few seconds), and that there is seldom more than one migration in progress at a time.

7. FUTURE WORK

In our workload model we have assumed that all processes are CPU-bound. Of primary interest in future work is including interactive, I/O-bound, and non-migratable jobs into our workload.

In a workload that includes interactive jobs, I/O-bound jobs, and daemons, there will be some jobs that should not or cannot be migrated. An I/O-bound job, for example, will not necessarily run faster on a more lightly-loaded host, and might run slower if it is migrated away from the disk or other I/O device it uses. A migrated interactive job might benefit by running on a more lightly-loaded host if it uses significant CPU time, but will suffer performance penalties for all future interactions. Finally, some jobs (e.g. many daemons) cannot be migrated away from their hosts.

The policy we proposed for preemptive migration can be extended to deal appropriately with interactive and I/O bound jobs by including in the definition of migration cost the additional costs that will be imposed on these jobs after migration, including network delays, access to non-local data, etc. The estimates of these costs might be based on the recent behavior of the job; e.g. the number and frequency of I/O requests and interactions. Jobs that are explicitly forbidden to migrate could be assigned an infinite migration cost.

The presence of a set of jobs that are either expensive or impossible to migrate might reduce the ability of the migration policy to move work around the network and balance loads effectively. However, we observe that the majority of long-lived jobs are, in fact, CPU-bound, and it is these long-lived jobs that consume the majority of CPU time. Thus, even if the migration policy were only able to migrate a subset of the jobs in the system, it could still have a significant load-balancing effect.

Another way in which the proposed migration policy should be altered in a more general environment is that n (the number of jobs at the source) and m (the number of jobs at the target host) should distinguish between CPU-bound jobs and other types of jobs, since only CPU-bound jobs affect CPU contention, and therefore are significant in CPU load balancing.

Another important consideration in load balancing is the effect of local scheduling at the hosts. Most prior studies of load balancing have assumed, as we do, that the local scheduling is round-robin (or processor-sharing). A few assume first-come-first-serve (FCFS) scheduling, but fewer still have studied the effect of feedback scheduling, where processes that have used the least CPU time are given priority over older processes. We simulated feedback scheduling and found that it greatly reduced mean slowdown (from approximately 2.5 to between 1.2 and 1.7, depending on load) even without migration. Thus, the potential benefit of either type of migration is greatly reduced.

We evaluated the non-preemptive migration policy from Section 5.1.1 under feedback scheduling, and found that it often makes things worse, increasing the mean slowdown in 5 of the 8 runs, and only decreasing it by 11% in the best case (highest load).

To evaluate our preemptive policy, we had to change the migration criterion to reflect the effect of local scheduling. Under processor sharing, we assume that the slowdown imposed on a process is equal to the number of processes on the host. Under feedback scheduling, the slowdown is closer to the number of *younger* processes, since older processes have lower priority. Thus, we modified the migration criterion in Section 3.1 so that n and m are the number of processes at the source and target hosts that are younger than the migrant process. Using this criterion, preemptive migration reduces mean normalized slowdown by 12–32%, and reduces the number of severely slowed processes (slowdown greater than 5) by 30–60%.

An issue that remains unresolved is whether feedback scheduling is as effective in real systems as it was in our simulations. For example, decay-usage scheduling as used in UNIX has some characteristics of both round-robin and feedback policies [Epema 1995]. Young jobs do have some precedence, but old jobs that perform interaction or other I/O are given higher priority, which allows them to interfere with short jobs. In our experiments on a SPARC workstation running SunOS, we found that a long-running job that performs periodic I/O can obtain more than 50% of the CPU time, even if it is sharing a host with much younger processes. The more recent lottery scheduling behaves more like processor-sharing [Waldspurger and Weihl 1994]. To understand the effect of local scheduling on load balancing requires a process model that includes interaction and I/O.

8. CONCLUSIONS

—To evaluate migration strategies, it is important to model the distribution of process lifetimes accurately. Assuming an exponential distribution can underestimate the benefits of preemptive migration, because it ignores the fact that old jobs are expected to be long-lived. Even a lifetime distribution that matches the measured distribution in both mean and variance may be misleading in designing and evaluating load balancing policies.

—Preemptive migration outperforms non-preemptive migration even when memory-transfer costs are high, for the following reason: non-preemptive name-based strategies choose processes for migration that are expected to have long lives. If this prediction is wrong, and a process runs longer than expected, it cannot be migrated away, and many subsequent small processes will be delayed. A preemptive strategy is able to predict lifetimes more accurately (based on age) and, more importantly, if the prediction is wrong, the system can recover by migrating the process later.

—Migrating a long job away from a busy host helps not only the long job, but also the many short jobs that are expected to arrive at the host in the future. A busy host is expected to receive many arrivals because of the serial correlation (burstiness) of the arrival process.

—Using the functional form of the distribution of process lifetimes, we have derived a criterion for the minimum time a process must age before being migrated. This criterion is parameterless and robust across a range of loads.

—Exclusive use of mean slowdown as a metric of system performance understates the benefits of load balancing as perceived by users, and especially understates the benefits of preemptive load balancing. One performance metric which is more related to user perception is the number of severely slowed processes. While non-preemptive migration eliminates half of these noticeable delays, preemptive migration reduces them by a factor of ten.

—Although preemptive migration is difficult to implement, several systems have chosen to implement it for reasons other than load balancing. Our results suggest these systems would benefit from incorporating a preemptive load balancing policy.

ACKNOWLEDGMENTS

We would like to thank Tom Anderson and the members of the NOW group for comments and suggestions on our experimental setup. We are also greatly indebted to the anonymous reviewers from SIGMETRICS and SOSP, and to John Zahorjan, whose comments greatly improved the quality of this paper.

REFERENCES

Abrahams, D. M. and Rizzardi, F. 1988. *The Berkeley Interactive Statistical System*. W. W. Norton and Co.

Agrawal, R. and Ezzet, A. 1987. Location independent remote execution in NEST. *IEEE Transactions on Software Engineering 13*, 8 (August), 905–912.

Ahmad, I., Ghafoor, A., and Mehrotra, K. 1991. Performance prediction of distributed load balancing on multicomputer systems. In *Supercomputing* (1991), pp. 830–839.

Artsy, Y. and Finkel, R. 1989. Designing a process migration facility: The Charlotte experience. *IEEE Computer*, 47–56.

Barak, A., Shai, G., and Wheeler, R. G. 1993. *The MOSIX Distributed Operating System: Load Balancing for UNIX*. Springer Verlag, Berlin.

Bonomi, F. and Kumar, A. 1990. Adaptive optimal load balancing in a nonhomogeneous multiserver system with a central job scheduler. *IEEE Transactions on Computers 39*, 10 (October), 1232–1250.

Braverman, A. 1995. Personal Communication.

Bryant, R. M. and Finkel, R. A. 1981. A stable distributed scheduling algorithm. In *2nd International Conference on Distributed Computing Systems* (1981), pp. 314–323.

Cabrera, F. 1986. The influence of workload on load balancing strategies. In *Proceedings of the Usenix Summer Conference* (June 1986), pp. 446–458.

Casas, J., Clark, D. L., Konuru, R., Otto, S. W., Prouty, R. M., and Walpole, J. 1995. Mpvm: A migration transparent version of pvm. *Computing Systems 8*, 2 (Spring), 171–216.

Casavant, T. L. and Kuhl, J. G. 1987. Analysis of three dynamic distributed load-balancing strategies with varying global information requirements. In *7th International Conference on Distributed Computing Systems* (September 1987), pp. 185–192.

Chowdhury, S. 1990. The greedy load sharing algorithm. *Journal of Parallel and Distributed Computing 9*, 93–99.

De Paoli, D. and Goscinski, A. 1995. The rhodos migration facility. *Journal of Systems and Software*. Submitted. See also http://www.cm.deakin.edu.au/rhodos/.

Douglis, F. and Ousterhout, J. 1991. Transparent process migration: Design alternatives and the sprite implementation. *Software – Practice and Experience 21*, 8 (August), 757–785.

Downey, A. B. and Harchol-Balter, M. 1995. A note on "The limited performance benefits of migrating active processes for load sharing". Technical Report UCB/CSD-95-888 (November), University of California, Berkeley.

Eager, D. L., Lazowska, E. D., and Zahorjan, J. 1986. Adaptive load sharing in homogeneous distributed systems. *IEEE Transactions on Software Engineering 12*, 5 (May), 662–675.

Eager, D. L., Lazowska, E. D., and Zahorjan, J. 1988. The limited performance benefits of migrating active processes for load sharing. In *ACM Sigmetrics Conference on Measuring and Modeling of Computer Systems* (May 1988), pp. 662–675.

Epema, D. 1995. An analysis of decay-usage scheduling in multiprocessors. In *ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems* (1995), pp. 74–85.

Evans, D. J. and Butt, W. U. N. 1993. Dynamic load balancing using task-transfer probabilites. *Parallel Computing 19*, 897–916.

Hać, A. and Jin, X. 1990. Dynamic load balancing in a distributed system using a sender-initiated algorithm. *Journal of Systems Software 11*, 79–94.

Hennessy, J. L. and Patterson, D. A. 1990. *Computer Architecture A Quantitative Approach.* Morgan Kaufmann Publishers, San Mateo, CA.

Krueger, P. and Livny, M. 1988. A comparison of preemptive and non-preemptive load distributing. In *8th International Conference on Distributed Computing Systems* (June 1988), pp. 123–130.

Kunz, T. 1991. The influence of different workload descriptions on a heuristic load balancing scheme. *IEEE Transactions on Software Engineering 17*, 7 (July), 725–730.

Larsen, R. J. and Marx, M. L. 1986. *An introduction to mathematical statistics and its applications* (2nd ed.). Prentice Hall, Englewood Cliffs, N.J.

Leland, W. E. and Ott, T. J. 1986. Load-balancing heuristics and process behavior. In *Proceedings of Performance '86 and ACM Sigmetrics*, Volume 14 (1986), pp. 54–69.

Lin, H.-C. and Raghavendra, C. 1993. A state-aggregation method for analyzing dynamic load-balancing policies. In *IEEE 13th International Conference on Distributed Computing Systems* (May 1993), pp. 482–489.

Litzkow, M. and Livny, M. 1990. Experience with the Condor distributed batch system. In *IEEE Workshop on Experimental Distributed Systems* (1990), pp. 97–101.

Litzkow, M., Livny, M., and Mutka, M. 1988. Condor - a hunter of idle workstations. In *8th International Conference on Distributed Computing Systems* (June 1988).

Livny, M. and Melman, M. 1982. Load balancing in homogeneous broadcast distributed systems. In *ACM Computer Network Performance Symposium* (April 1982), pp. 47–55.

Milojicic, D. S. 1993. *Load Distribution: Implementation for the Mach Microkernel.* PhD Dissertation, University of Kaiserslautern.

Mirchandaney, R., Towsley, D., and Stankovic, J. A. 1990. Adaptive load sharing in heterogeneous distributed systems. *Journal of Parallel and Distributed Computing 9*, 331–346.

Powell, M. and Miller, B. 1983. Process migrations in DEMOS/MP. In *6th ACM Symposium on Operating Systems Principles* (November 1983), pp. 110–119.

Prouty, R. 1996. Personal Communication.

Pulidas, S., Towsley, D., and Stankovic, J. A. 1988. Imbedding gradient estimators in load balancing algorithms. In *8th International Conference on Distributed Computing Systems* (June 1988), pp. 482–490.

ROMMEL, C. G. 1991. The probability of load balancing success in a homogeneous network. *IEEE Transactions on Software Engineering 17*, 922–933.

ROSIN, R. F. 1965. Determining a computing center environment. *Communications of the ACM 8*, 7.

SILBERSCHATZ, A., PETERSON, J., AND GALVIN, P. 1994. *Operating System Concepts, 4th Edition*. Addison-Wesley, Reading, MA.

SVENSSON, A. 1990. History, an intelligent load sharing filter. In *IEEE 10th International Conference on Distributed Computing Systems* (1990), pp. 546–553.

TANENBAUM, A., VAN RENESSE, R., VAN STAVEREN, H., AND SHARP, G. 1990. Experiences with the Amoeba distributed operating system. *Communications of the ACM*, 336–346.

THEIMER, M. M., LANTZ, K. A., AND CHERITON, D. R. 1985. Preemptable remote execution facilities for the V-System. In *10th ACM Symposium on Operating Systems Principles* (December 1985), pp. 2–12.

THIEL, G. 1991. Locus operating system, a transparent system. *Computer Communications 14*, 6, 336–346.

VAHDAT, A. 1995. Personal Communication.

VAHDAT, A. M., GHORMLEY, D. P., AND ANDERSON, T. E. 1994. Efficient, portable, and robust extension of operating system functionality. Technical Report UCB//CSD-94-842, University of California, Berkeley.

WALDSPURGER, C. A. AND WEIHL, W. E. 1994. Lottery scheduling: Flexible proportional-share resource management. In *Proceedings of the First Symposium on Operating System Design and Implementation* (November 1994), pp. 1–11.

WANG, J., ZHOU, S., K.AHMED, AND LONG, W. 1993. LSBATCH: A distributed load sharing batch system. Technical Report CSRI-286 (April), Computer Systems Research Institute, University of Toronto.

WANG, Y.-T. AND MORRIS, R. J. 1985. Load sharing in distributed systems. *IEEE Transactions on Computers c-94*, 3 (March), 204–217.

ZAYAS, E. R. 1987. Attacking the process migration bottleneck. In *11th ACM Symposium on Operating Systems Principles* (1987), pp. 13–24.

ZHANG, Y., HAKOZAKI, K., KAMEDA, H., AND SHIMIZU, K. 1995. A performance comparison of adaptive and static load balancing in heterogeneous distributed systems. In *Proceedings of the 28th Annual Simulation Symposium* (April 1995), pp. 332–340.

ZHOU, S. 1987. *Performance studies for dynamic load balancing in distributed systems.* Ph.D. Dissertation, University of California, Berkeley.

ZHOU, S. AND FERRARI, D. 1987. A measurement study of load balancing performance. In *IEEE 7th International Conference on Distributed Computing Systems* (October 1987), pp. 490–497.

ZHOU, S., WANG, J., ZHENG, X., AND DELISLE, P. 1993. Utopia: a load-sharing facility for large heterogeneous distributed computing systems. *Software – Practice and Expeience 23*, 2 (December), 1305–1336.