Exploiting Process Lifetime Distributions for Dynamic Load Balancing

Mor Harchol-Balter* Allen B. Downey†

University of California at Berkeley
{harchol , downey}@cs.berkeley.edu

Abstract

We measure the distribution of lifetimes for UNIX processes and propose a functional form that fits this distribution well. We use this functional form to derive a policy for preemptive migration, and then use a trace-driven simulator to compare our proposed policy with other preemptive migration policies, and with a non-preemptive load balancing strategy. We find that, contrary to previous reports, the performance benefits of preemptive migration are significantly greater than those of non-preemptive migration, even when the memory-transfer cost is high. Using a model of migration costs representative of current systems, we find that preemptive migration reduces the mean delay (queueing and migration) by 35 – 50%, compared to non-preemptive migration.

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 ([ELZ88]) discourages 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. But simulation studies (which can use more realistic workload descriptions) and implemented systems have shown greater benefits for preemptive migration ([KL88] and [BSW93]). 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, *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.[1]

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 (e.g. [Sve90]) that specifies (by process name) which processes may be migrated.

In contrast, most preemptive migration policies do not use *a priori* information, since this it 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.

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). Since processes may be delayed while on the run queue or while migrating, the slowdown imposed on a process is

[*] Supported by National Physical Science Consortium (NPSC) Fellowship. Also supported by NSF grant number CCR-9201092.

[†] Partially supported by NSF (DARA) grant DMW-8919074.

[1] The other half of a load balancing strategy is the location policy — the selection a new host for the migrated process. Previous work ([Zho87] and [Kun91]), has suggested that choosing the target host with the shortest CPU run queue is both simple and effective. Our work confirms the relative unimportance of location policy.

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

where *wall-time*(p) is the total time p 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 $> 99\%$) 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 the results of [LO86], but this prior work has been incorporated in few subsequent analytic and simulator load balancing studies. This omission is unfortunate, since the results of these load balancing studies are quite sensitive to the lifetime model.

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 (the eligibility criterion doesn't rely on free parameters that must be hand-optimized). This tool is generally useful for analysis of system behavior.

Specifically, Section 3 shows the derivation 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 (from Section 3) with a non-preemptive policy based on name-lists. The simulator (see Section 5.1) 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 (Section 5.2) 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 (Section 5.3) we choose a specific model of preemptive and non-preemptive migration costs (described in 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.

Finally, in Section 5.4 we use the simulator to compare our preemptive migration strategy with other preemptive schemes in the literature.

We finish with a self-criticism of our model in Section 6 and conclusions in Section 7.

1.4 Related Work

1.4.1 Systems

Most existing systems provide some form of user-controlled remote execution, but relatively few provide automated load balancing. Of the ones that do, the majority are based on implicit remote execution of newborn processes; few use preemptive migration.

(The following taxonomy is based in large part on [Nut94].)

The following systems have implemented explicit remote execution and/or explicit preemptive migration; that is, both forms of migration are only performed at the user's request: Accent [Zay87], Locus [Thi91], Utopia [ZWZD93], DEMOS/MP [PM83], V [TLC85], NEST [AE87], and MIST [CCK+95].

Some other systems provide *implicit* remote execution, but perform preemptive migration only at the request of a user or for reasons other than load balancing (such as preserving autonomy): Amoeba [TvRaHvSS90], Charlotte [AF89], Sprite [DO91], Condor [LLM88], and Mach [Mil93]. Although these systems are capable of migrating active processes (with varying degrees of transparency), none have implemented a policy that specifies which processes should be preempted for purposes of load balancing.

Only a few systems have implemented automated load balancing policies with preemptive migration: MOSIX[BSW93] and RHODOS [GGI+91]. The MOSIX load balancing scheme is similar to the strategies recommended in this paper; our results support their claim that their scheme is effective and robust.

In general, non-preemptive load balancing strategies depend on *a priori* information about processes; e.g., explicit knowledge about the runtimes of processes or user-provided lists of migratable processes ([AE87], [LL90], [DO91], [ZWZD93]).

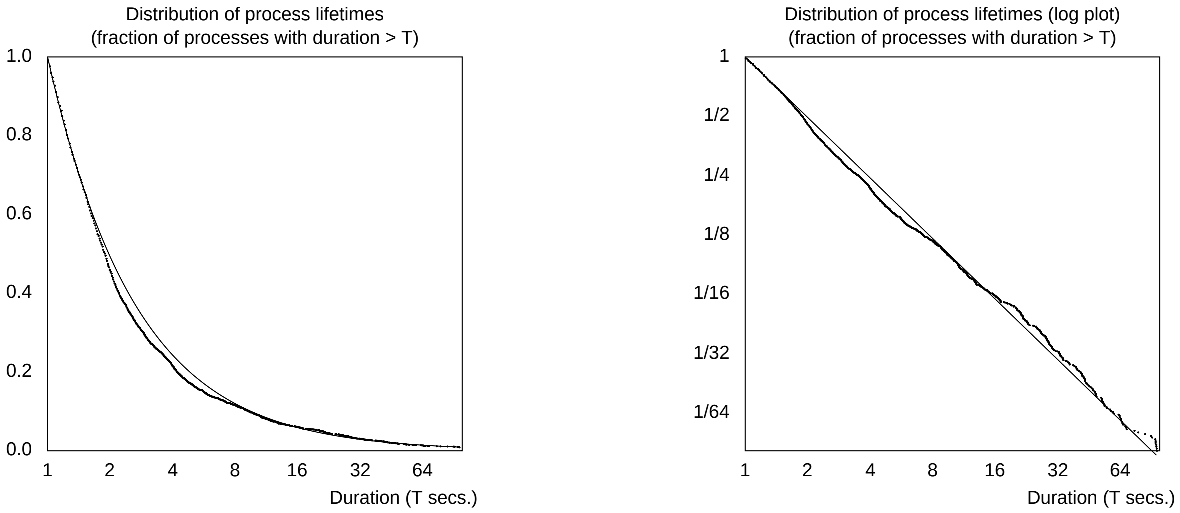

Figure 1: Distribution of process lifetimes for processes with lifetimes greater than 1 second, observed on machine "po" mid-semester. The dotted (thicker) line shows the measured distribution; the solid (thinner) line shows the least squares curve fit. To the right: the same distribution shown on a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

1.4.2 Studies

Although few systems use preemptive migration for load balancing, there have been many simulation studies and analytic models showing the performance benefits of various load balancing strategies. Some of these studies have focused on load balancing by remote execution ([LM82], [WM85], [CK87], [Zho87], [PTS88], [Kun91], [HJ90], [ELZ86]); others have compared the performance of systems with and without preemptive migration ([ELZ88], [KL88]).

Our work differs from [ELZ88] in both system model and workload description. [ELZ88] 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.

Also, [ELZ88] use a degenerate hyperexponential distribution of lifetimes that includes many jobs with zero lifetime, and far fewer short jobs (0 – 1 seconds) than we observed. For a more detailed explanation of this distribution and its effect on the study, see [DHB95].

[KL88] 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. Also, our migration policy differs from [KL88] in that our proposed migration policy uses preemptive migration exclusively, rather than in addition to, remote execution.

Like us, [BF81] discuss the distribution of process lifetimes and its effect on preemptive migration policy, but their hypothetical distributions are not based on system measurements. 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 in detail 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. Thus, in our estimates of slowdown, we will 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 [Ros65]: 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 [LO86] 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 concluded 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.[2] Specifically, they found that for $T > 3$ seconds, the probability of a process' lifetime exceeding T seconds is rT^k, where $-1.25 < k < -1.05$ (r normalizes the distribution).

In contrast to [LO86], Rommel ([Rom91]) claimed that his measurements show that "long processes have exponential service times."

Because of the importance of the process lifetime distribution to load balancing policies, we performed an independent study of this distribution, which we describe in Section 2.1.

[2] In contrast, the exponential distribution is memoryless; the expected remaining lifetime of a process is independent of age.

Name of Host	Total Number Procs. Studied	Num. Procs. with Age> 1	Estim. Lifetime Distrib. Curve	Std. Error	R^2 val
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 curve 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.

In our study the functional form proposed by [LO86] fits all our observed distributions well, for processes with lifetimes greater than 1 second. For the longest jobs ($>$ 1000 seconds), there are so few processes in our sample that the fit deteriorates, but overall the goodness of fit of the model is excellent.[3] 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 -1.0. Thus, as a *rule of thumb*,

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

2. 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.

Despite the [LO86] study, many researchers have continued to assume an exponential process lifetime distribution in their analysis of migration strategies (e.g., [MTS90], [BK90] [EB93], [LR93]). The reasons for assuming an exponential lifetime distribution include: (1) analytic tractability, and (2) the belief that the exponential distribution is close enough to real distributions that the results of the analyses are not affected.

In this paper, we make the following claims about lifetime distributions:

- The performance of various migration strategies (and other system features) depends strongly on the details of the workload description. For example, two distributions that match with respect to both mean and variance might still produce significantly different results.

- The properties of an exponential distribution are very different from those of the distributions we observed. For example, the distributions we observed all have a tail of long-lived jobs (i.e., the distributions have high variance). An exponential distribution with the same mean would have lower variance; it lacks the tail of long-lived jobs.

- Although the alternate functional form that we (and [LO86]) 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 3.2.

In previous work, some simulations and analyses have used a hyperexponential distribution of lifetimes (a hyperexponential distribution consists of two or more exponential branches). The motivation for this model is that by using more than one exponential distribution, it is possible to match an observed distribution more closely. In cases where the hyperexponential distribution has enough branches to fit the observed distribution well, as in [KL88], this model has been successful.

The remainder of this section focuses on our distribution measurements. We observed that long processes (with lifetimes greater than 1 second) have a predictable and consistent distribution. Section 2.1 describes this distribution. Section 2.2 makes some additional observations about shorter processes.

2.1 Lifetime distribution when lifetime $> 1s$.

To determine the probability distribution function 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 our process lifetime measurements on a heavily-used instructional machine in mid-semester. The plot shows only processes whose lifetimes exceed one second. The dotted (heavy) line indicates the measured distribution; the solid (thinner) line indicates the least squares curve fit. The straight line in log-log space indicates that the process lifetime distribution fits the curve T^k, where k is the slope of the line.

For all the machines we studied, the process lifetime data (for processes with age greater than one second) fit a curve of the form T^k, where k ranged from about -1.3 to $-.8$ for different machines. Table 1 shows the estimated lifetime distribution curve for each machine we studied. The parameters were estimated by an iteratively weighted least squares fit (with no intercept, in accordance with the functional model). The standard error associated with each estimated parameter gives a confidence interval for that parameter (all of these parameters are statistically significant at a high degree of certainty). Finally, the R^2 value indicates the goodness of

[3] Throughout this paper, we distinguish between observed lifetime distributions (taken from our measurements) and the proposed functional form (which fits the observed distribution over all but the longest processes). Although the functional form has infinite mean and variance, the observed distributions (necessarily) have finite mean and variance.

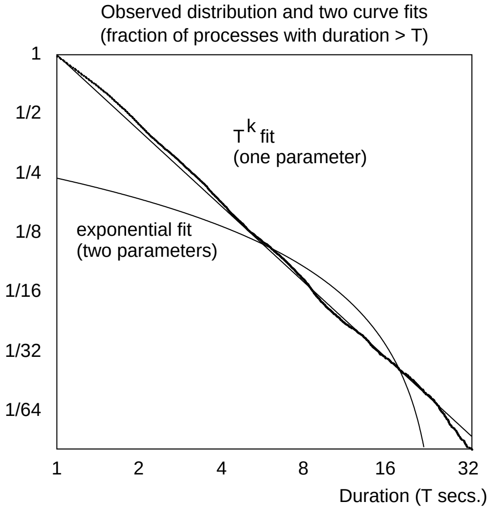

Figure 2: In log-log space, this plot shows the distribution of lifetimes for the $\sim$ 13000 processes with lifetimes > 1 second from our trace-driven simulation (see Section 5), and two attempts to fit a curve to this data. One of the fits is based on the model proposed in this paper, T^k. The other 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.

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 very high.

Although the range of parameters we observed is fairly broad, in the absence of measurements from a specific system, assuming a distribution of $1/T$ is substantially more accurate than assuming that process lifetimes are exponentially distributed, as shown by Figure 2.

Table 2 shows the lifetime distribution function, the corresponding density function, and the conditional distribution function. We will refer to the conditional lifetime distribution often during our analysis of migration strategies. The second column of Table 2 shows these functions when $k = -1$, which we will assume for our analysis in Section 3.

2.2 Process lifetime distribution in general

For completeness we discuss the lifetime distribution for processes with lifetimes less than one second. Since our measurements were made using the *lastcomm* command the shortest process we were able to measure was .01 seconds. For processes between .01 and 1 second, we did not find a consistent functional form, however for all machines we studied these processes had an even lower hazard rate than those of age > 1 second. That is, while the probability that a process of age $T > 1$ second lives another T seconds is approximately $1/2$, the probability that a process of age $T < 1$ second lives another T seconds is something greater than $1/2$.

Process Lifetime Distribution for Processes of Age ≥ 1 second	When $k = -1$
$\mathbf{Pr}\{\text{Proc. lifetime} > T \text{ sec} \mid \text{age} > 1 \text{ sec}\} = T^k$	$= 1/T$
$\mathbf{Pr}\{\text{Lifetime} = T \text{ sec} \mid \text{age} = 1 \text{ sec}\} = -kT^{k-1}$	$= 1/T^2$
$\mathbf{Pr}\{\text{Lifetime} > a \text{ sec} \mid \text{age} = b > 1 \text{ sec}\} = \left(\frac{a}{b}\right)^k$	$= \frac{b}{a}$

Table 2: The cumulative distribution function, probability density function, and conditional distribution function of processes lifetimes. The second column shows the functional form of each for the typical value $k = -1.0$.

3 Migration Policy

A migration policy is based on two decisions: when to migrate processes and which processes to migrate. The focus of this paper is the second question (we will touch on the first question in Section 5.1):

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

Our heuristic is to *choose the process that has highest probability of running longer than its migration time.*

The motivation for this heuristic is twofold. From the host's perspective, a large fraction of the migration time is spent at the host (packaging the process). The host would only choose to migrate processes that are likely to be more expensive to run than to migrate. From the process' 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.

Most existing migration policies only migrate newborn processes (no preemption), because these processes have no allocated memory and thus their migration cost is less (see Section 4).[4] The problem with this policy is that, according to the process lifetime distribution (Section 2), these newborn processes are unlikely to live long enough to justify the cost of remote execution.

Thus a "newborn" migration policy is only justified if the system has prior knowledge about the processes and can selectively migrate only those processes likely to be CPU hogs. However, the ability of the system to predict process lifetimes by name is limited, as shown in Section 5.3.1.

Can we do better? The lifetime distribution points us towards migrating *older* processes, since they have the highest probability of living long enough to justify the cost of migration, but there are two potential problems with this strategy: (1) since the vast majority of processes are short, there might not be enough long-lived processes to have a significant load balancing effect, and (2) the additional cost of migrating old processes (the memory transfer cost) might overwhelm the benefit of migrating longer-lived processes.

The following sections address these concerns. Section 3.2 also proposes a new preemptive migration strategy based on the lifetime distribution.

[4] 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.

3.1 Moving Enough Work

If only old processes are eligible for migration, and the majority of processes are short-lived, there might not be enough old processes to produce a significant load balancing effect.

In fact, although there are few old processes, they account for a large part of the total CPU load. According to our process lifetime measurements (Section 2), typically fewer than 3.5% 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 (see Figure 2). [LO86] make a similar observation.

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 large job on a busy host imposes slowdowns on many small processes.

3.2 Our Migration Policy

The obvious disadvantage of preemptive migration is the need to transfer the memory associated with the migrant process; thus, 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.

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.[5]

Whenever more than one process is running on a host and one process migrates away, the expected slowdown of the others decreases, regardless of the duration of the processes or the cost of migration. But the slowdown of the migrant process might increase, if the time spent migrating is greater than the time saved by running on a less-loaded host. Thus we will perform migration only if it improves the expected slowdown of the migrant process.

If there is no process on the host that satisfies this 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 now show how to 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:

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

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

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

This analysis extends easily 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 running processes at the target 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's thesis, [Zho87] and with [Kun91].

The MOSIX migration policy [BSW93] is based on a restriction that is similar to the criterion we are proposing: the age of the process must exceed the migration cost. Thus, the slowdown imposed on the migrant process (due to migration) must be less than 2.0. This bound is based on the worst case, in which the migrant process completes immediately upon arrival at the target.

The MOSIX requirement is likely 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). Secondly, it is based on the worst-case slowdown rather than (as shown above) the expected slowdown. We will explicitly compare the MOSIX policy with ours in Section 5.4.

4 Model of migration costs

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

[5] 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.

We model the cost of migrating an active process as the sum of a *fixed migration cost* for migrating the process' system state plus a *memory transfer cost* that is proportional to the amount of the process' memory that must be transferred. We model *remote execution cost* as a fixed cost; it is the same for all processes.

Throughout this paper, we refer to the following parameters:

- r: the cost of remote execution, in seconds
- f: the fixed cost of preemptive migration, in seconds
- b: the memory transfer bandwidth, in MB's per second
- m: the memory size of migrant processes, in MB

and thus:

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

We refer to the quotient m/b as the memory transfer cost.

4.1 Memory transfer costs

The amount of a process' memory that must be transferred during preemptive migration depends on properties of the distributed system. [DO91] have an excellent discussion of this issue, and we borrow from them here.

At the most, it might be necessary to transfer a process' 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 the V [TLC85] system), 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 ([Zay87]).

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. In this section, we will present reported values for parameters on a variety of real systems. Later we will use a trace-driven simulator to evaluate the effect of these parameters on system performance.

The cost of remote execution, r, on a typical UNIX workstation connected to an Ethernet is 1 – 4 seconds. Systems that use remote execution for load sharing have made an effort to reduce this cost. On Sprite [DO91] $r \approx .33$ seconds. Similarly for GLUNIX [VGA94], an operating system designed for networks of workstations connected by an ATM network, $r = .25 - .5$ seconds [Vah95]. The Utopia System takes $\sim$ 1.0 seconds to establish a connection between source and target hosts, but once this is done, subsequent remote executions can take as little as .1 seconds [ZWZD93].

Sprite was implemented on a network of SPARCstation 1 workstations connected by a 10Mb/second Ethernet. On Sprite preemptive migrations took $f = .33$ seconds plus $1/b = 2.0$ seconds per megabyte of memory transferred.

By implementing migration at the kernel level (on a cluster of Pentium-90 and i486/DX66 workstations), MOSIX reduces the fixed cost, f, to only 6 ms; the inverse memory transfer bandwidth, $1/b$, is .44 seconds per megabyte [Bra95].

The MIST system ([CCK$^+$95]) is implemented on a network of HP9000/720 workstations running HP-UX 9.03 and connected by a 10Mb/second Ethernet. On this system, preemptive migration takes $f = .24$ seconds plus $1/b = .99$ seconds per megabyte of process memory.

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 3.2) and a non-preemptive strategy that migrates newborn processes according to the process name (similar to strategies proposed by [WZKL93] and [Sve90]). Although we use a simple name-based strategy, we give it the benefit of several unfair 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 parameters 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. Lastly, in Section 5.4, we evaluate the analytic criterion for migration age (proposed in Section 3.2) used in our preemptive migration strategy, compared to criteria used in the literature.

5.1 The simulator

We have implemented a trace-driven simulation of a network of six identical workstations.[6] 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 simulate 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 process arrival 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 $\sim$ 15000 processes submitted to the six simulated hosts; Run 7 has $\sim$ 30000 processes. The average

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

duration of processes (for all runs) is $\sim .4$ seconds. Thus the total utilization of the system, ρ, is between .27 and .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 the process 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[7], 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. are never blocked on I/O). During a given time interval, we divide CPU time equally among the processes on the host.

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 a pessimistic assumption for advocates of preemptive migration.

5.1.1 Strategies

We compare a non-preemptive migration strategy with our proposed preemptive migration strategy (from Section 3.2). 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 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 target 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 3.2, 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. We have modeled strategies that allow migration at other times, and they do improve the performance of the preemptive strategy, but we have omitted them here to facilitate comparison between the two migration strategies.

As described in Section 3.2, 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.

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 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 $.5/1.0 = .5$: a 50% reduction in delay.

Mean slowdown alone, however, 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:

1. 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 will be dominated by this majority.

2. 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.

[7] In our informal study of processes in our department, we did not detect any correlations between memory size and process CPU usage. [KL88] make the same observation in their department.

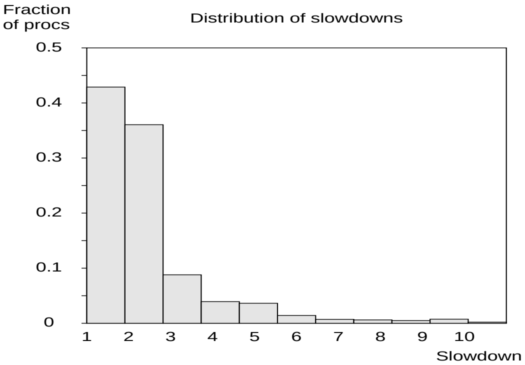

Figure 3: Distribution of process slowdowns for run 0 (with no migration). Most processes suffer small slowdowns, but the processes in the tail of the distribution are more noticeable and annoying to users.

Therefore, we will also consider the following two metrics:

variance of slowdown : This metric is often cited as a measure of the unpredictability of response time [SPG94], 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.

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.

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.

The shape of the memory transfer cost 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. 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 [HP90] of the eight ratios.

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). As mentioned above, the cost of non-preemptive migration (r) is fixed at .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 .33 seconds and the MMTC is 2 seconds, the normalized mean slowdown with preemptive migration is $\sim 40\%$ 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 in Figure 4b 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 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, .3 seconds
- f: the fixed cost of preemptive migration, .3 seconds
- b: the memory transfer bandwidth, .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 base case of no migration).

Non-preemptive migration reduces the normalized mean slowdown (Figure 5a) by less than 20% for most runs (and $\sim 40\%$ 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%.

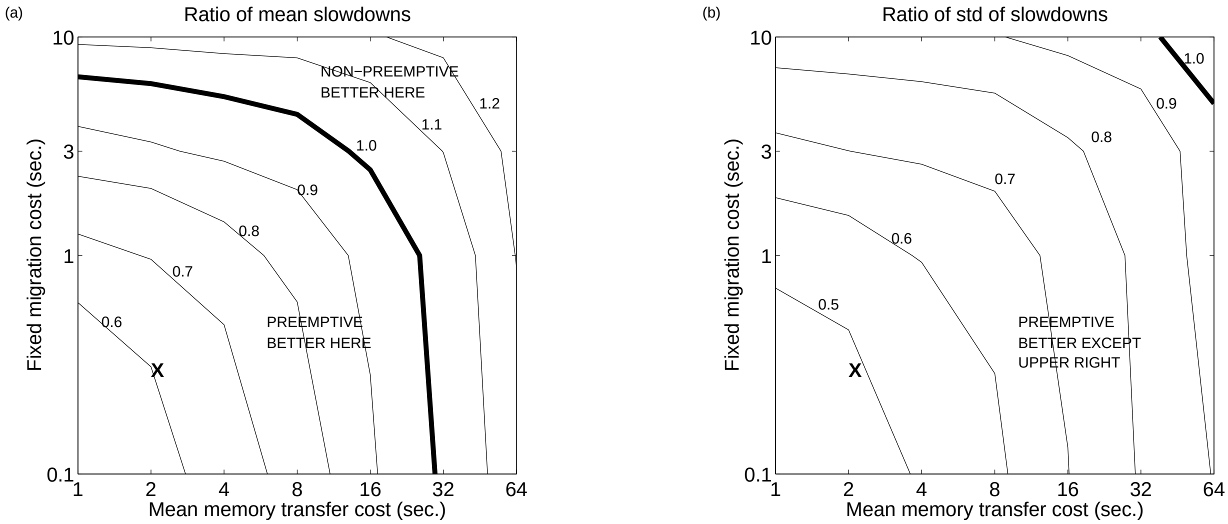

Figure 4: (a) The performance of preemptive migration relative to non-preemptive migration deteriorates as the cost of preemptive migration increases. The two axes are the two components of the preemptive migration cost. The cost of non-preemptive migration is held fixed. The "X" marks the particular 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.

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.

5.3.1 Shortcomings of 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 that are possible for either migration strategy:

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 eliminates this type of error by guaranteeing that the performance of a migrant improves in expectation.

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 brief program. Preemptive migration can correct these errors by migrating long jobs later in their lives.

Even occasional mistakes of the second 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.

In our simulations the second type of error was more significant: most severely-slowed jobs suffered because they were forced to run on a heavily-loaded host, not because they suffered migration delays. Specifically, under non-preemptive migration almost all (99.2%) processes that suffered slowdowns greater than 3.0 were short processes (< 1 second) that never migrated; their slowdowns were caused by running on a busy host.

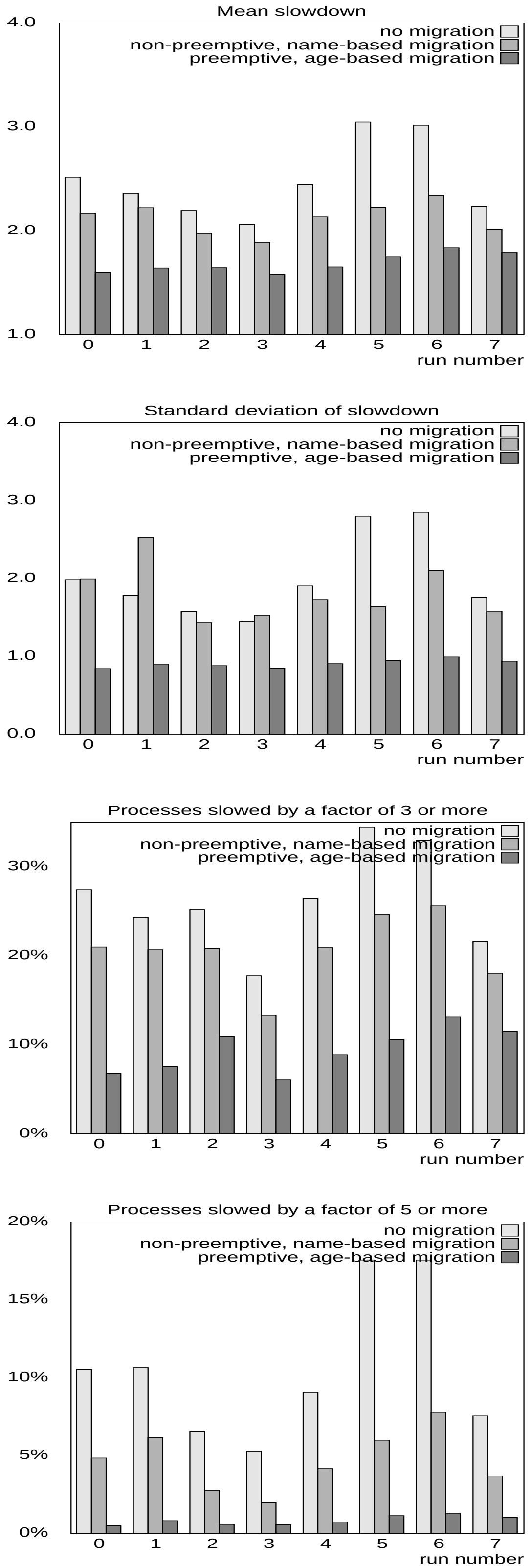

Figure 5: (a) Mean slowdown. (b) Standard deviation of slowdown. (c) Percentage of processes slowed by a factor of 3 or more. (d) Percentage of processes slowed by a factor of 5 or more. All shown at cost point X from Figure 4.

Preemptive migration is able to help some of these processes by performing more — and more effective — migrations, but still 96% of severe slowdowns are due to high loads, not migration delays. This suggests that there are additional performance benefits to be gained from improving load balance, even at the cost of additional migration.

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. In our simulations, the average lifetime of migrant processes under non-preemptive migration was between 1.5 and 2.1 seconds (the mean lifetime for all processes is 0.4 seconds). Our preemptive migration policy was better able to identify long jobs; the average age of migrant processes was between 4.1 and 5.7 seconds.

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 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, $\sim$ 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 rechecking loads at the end of a memory transfer and, if the load at the target is too high, aborting the migration and restarting the process on the source host.

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 Evaluation of analytic migration criterion

As derived in Section 3.2, the minimum age for a migrant process according to the *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 the potential migrant).

In order to evaluate the performance of this criterion, we will compare it with the *fixed parameter criterion*:

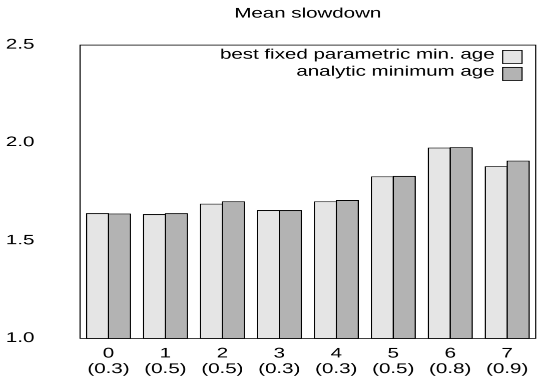

Figure 6: The mean slowdown for eight runs, using the two criteria for minimum migration age. The value of the best fixed parameter α is shown in parentheses for each run.

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

where α is a free parameter. For comparison, we will use the *best fixed parameter*, which is, for each run, the best parameter for that run, chosen empirically by excuting the run with a range of parameter values (of course, this gives the fixed parameter criterion a considerable advantage).

As discussed in Section 3.2, MOSIX uses the parameter $\alpha = 1.0$, based on a worst-case analysis of the slowdown imposed on the migrant. Although this age threshold offers a strict limit on the slowdown seen by a migrant process, it imposes greater slowdowns on the processes that would have benefited if a younger process were allowed to migrate away. A previous simulation study [KL88] chose a lower value for this parameter ($\alpha = 0.1$), but did not explain how it was chosen.

Figure 6 compares the performance of the analytic minimum age criterion with the best fixed parameter (α). 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 (and sometimes better than) 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.

6 Weaknesses of the model

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

environment : Our migration strategy takes advantage of the used-better-than-new property of process lifetimes. In an enviroment with a different distribution, this strategy will not be effective. We are currently examining the distribution of lifetimes on a Cray C90 at the San Diego Supercomputer Center.

I/O : Our model considers all jobs CPU-bound; thus, their response time necessarily improves if they run on a less-loaded host. For I/O bound jobs, however, CPU contention has little effect on response time. These jobs would benefit less from migration.

dependencies : Our model of migration cost considers only the cost of transferring a process, and not the additional costs imposed by future interaction and other I/O. For some jobs, these additional costs might be significant. To see how large a role this plays, 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.

memory size : One weakness of our model is that we choose memory sizes from a measured distribution and therfore our model ignores any correlation between memory size and other process characteristics. This choice however allows us to control the mean memory size as a parameter and examine its effect on system performance. In this paper we've made the pessimistic simplification that a migrant's entire memory must be transferred, although this is not always the case.

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 Conclusions

- 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.

- 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 make a more accurate prediction about the duration of a process (based on the its age) and, more importantly, if the prediction is wrong, it can recover by migrating the process later.

- 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.

- 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 preemptive load balancing.

8 Acknowledgements

We'd like to thank Tom Anderson and the members of the NOW group for comments and suggestions on our experimental setup, as well as on the preparation of this paper. 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

[AE87] Rakesh Agrawal and Ahmed Ezzet. Location independent remote execution in NEST. *IEEE Transactions on Software Engineering*, 13(8):905–912, August 1987.

[AF89] Y. Artsy and R. Finkel. Designing a process migration facility: The Charlotte experience. *IEEE Computer*, pages 47–56, September 1989.

[BF81] Raymond M. Bryant and Raphael A. Finkel. A stable distributed scheduling algorithm. In *2nd International Conference on Distributed Computing Systems*, pages 314–323, 1981.

[BK90] Flavio Bonomi and Anurag Kumar. Adaptive optimal load balancing in a nonhomogeneous multiserver system with a central job scheduler. *IEEE Transactions on Computers*, 39(10):1232–1250, October 1990.

[Bra95] Avner Braverman, 1995. Personal Communication.

[BSW93] Amnon Barak, Guday Shai, and Richard G. Wheeler. *The MOSIX Distributed Operating System:Load Balancing for UNIX*. Springer Verlag, Berlin, 1993.

[CCK$^+$95] Jeremy Casas, Dan L. Clark, Ravi Konuru, Steve W. Otto, Robert M. Prouty, and Jonathan Walpole. Mpvm: A migration transparent version of pvm. *Computing Systems*, 8(2):171–216, Spring 1995.

[CK87] Thomas L. Casavant and Jon G. Kuhl. Analysis of three dynamic distributed load-balancing strategies with varying global information requirements. In *7th International Conference on Distributed Computing Systems*, pages 185–192, September 1987.

[DHB95] Allen B. Downey and Mor Harchol-Balter. A note on "The limited performance benefits of migrating active processes for load sharing". Technical Report UCB//CSD-95-888, University of California, Berkeley, November 1995.

[DO91] Fred Douglis and John Ousterhout. Transparent process migration: Design alternatives and the sprite implementation. *Software – Practice and Experience*, 21(8):757–785, August 1991.

[EB93] D. J. Evans and W. U. N. Butt. Dynamic load balancing using task-transfer probablilites. *Parallel Computing*, 19:897–916, August 1993.

[ELZ86] Derek L. Eager, Edward D. Lazowska, and John Zahorjan. Adaptive load sharing in homogeneous distributed systems. *IEEE Transactions on Software Engineering*, 12(5):662–675, May 1986.

[ELZ88] Derek L. Eager, Edward D. Lazowska, and John Zahorjan. The limited performance benefits of migrating active processes for load sharing. In *SIGMETRICS*, pages 662–675, May 1988.

[GGI$^+$91] G.W. Gerrity, A. Goscinski, J. Indulska, W. Toomey, and W. Zhu. RHODOS–a testbed for studying design issues in distributed operating systems. In *Towards Network Globalization (SICON 91): 2nd International Conference on Networks*, pages 268–274, September 1991.

[HJ90] Anna Hać and Xiaowei Jin. Dynamic load balancing in a distributed system using a sender-initiated algorithm. *Journal of Systems Software*, 11:79–94, 1990.

[HP90] John L. Hennessy and David A. Patterson. *Computer Architecture A Quantitative Approach*. Morgan Kaufmann Publishers, San Mateo, CA, 1990.

[KL88] Phillip Krueger and Miron Livny. A comparison of preemptive and non-preemptive load distributing. In *8th International Conference on Distributed Computing Systems*, pages 123–130, June 1988.

[Kun91] Thomas Kunz. The influence of different workload descriptions on a heuristic load balancing scheme. *IEEE Transactions on Software Engineering*, 17(7):725–730, July 1991.

[LL90] M. Litzkow and M. Livny. Experience with the Condor distributed batch system. In *IEEE Workshop on Experimental Distributed Systems*, pages 97–101, 1990.

[LLM88] M.J. Litzkow, M. Livny, and M.W. Mutka. Condor - a hunter of idle workstations. In *8th International Conference on Distributed Computing Systems*, June 1988.

[LM82] Miron Livny and Myron Melman. Load balancing in homogeneous broadcast distributed systems. In *ACM Computer Network Performance Symposium*, pages 47–55, April 1982.

[LO86] W. E. Leland and T. J. Ott. Load-balancing heuristics and process behavior. In *Proceedings of Performance and ACM Sigmetrics*, volume 14, pages 54–69, 1986.

[LR93] Hwa-Chun Lin and C.S. Raghavendra. A state-aggregation method for analyzing dynamic load-balancing policies. In *IEEE 13th International Conference on Distributed Computing Systems*, pages 482–489, May 1993.

[Mil93] Dejan S. Milojicic. *Load Distribution: Implementation for the Mach Microkernel*. PhD Dissertation, University of Kaiserslautern, 1993.

[MTS90] Ravi Mirchandaney, Don Towsley, and John A. Stankovic. Adaptive load sharing in heterogeneous distributed systems. *Journal of Parallel and Distributed Computing*, 9:331–346, 1990.

[Nut94] Mark Nuttall. Survey of systems providing process or object migration. Technical Report DoC94/10, Imperial College Research Report, 1994.

[PM83] M.L. Powell and B.P. Miller. Process migrations in DEMOS/MP. In *ACM-SIGOPS 6th ACM Symposium on Operating Systems Principles*, pages 110–119, November 1983.

[PTS88] Spiridon Pulidas, Don Towsley, and John A. Stankovic. Imbedding gradient estimators in load balancing algorithms. In *8th International Conference on Distributed Computing Systems*, pages 482–490, June 1988.

[Rom91] C. Gary Rommel. The probability of load balancing success in a homogeneous network. *IEEE Transactions on Software Engineering*, 17:922–933, 1991.

[Ros65] Robert F. Rosin. Determining a computing center environment. *Communications of the ACM*, 8(7), 1965.

[SPG94] A. Silberschatz, J.L. Peterson, and P.B. Galvin. *Operating System Concepts, 4th Edition*. Addison-Wesley, Reading, MA, 1994.

[Sve90] Anders Svensson. History, an intelligent load sharing filter. In *IEEE 10th International Conference on Distributed Computing Systems*, pages 546–553, 1990.

[Thi91] G. Thiel. Locus operating system, a transparent system. *Computer Communications*, 14(6):336–346, 1991.

[TLC85] Marvin M. Theimer, Keith A. Lantz, and David R Cheriton. Preemptable remote execution facilities for the V-System. In *ACM-SIGOPS 10th ACM Symposium on Operating Systems Principles*, pages 2–12, December 1985.

[TvRaHvSS90] A.S. Tanenbaum, R. van Renesse adn H. van Staveren, and G.J. Sharp. Experiences with the Amoeba distributed operating system. *Communications of the ACM*, pages 336–346, December 1990.

[Vah95] Amin Vahdat, 1995. Personal Communication.

[VGA94] Amin M. Vahdat, Douglas P. Ghormley, and Thomas E. Anderson. Efficient, portable, and robust extension of operating system functionality. Technical Report UCB//CSD-94-842, University of California, Berkeley, 1994.

[WM85] Yung-Terng Wang and Robert J.T. Morris. Load sharing in distributed systems. *IEEE Transactions on Computers*, c-94(3):204–217, March 1985.

[WZKL93] J. Wang, S. Zhou, K.Ahmed, and W. Long. LS-BATCH: A distributed load sharing batch system. Technical Report CSRI-286, Computer Systems Research Institute, University of Toronto, April 1993.

[Zay87] E. R. Zayas. Attacking the process migration bottleneck. In *ACM-SIGOPS 11th ACM Symposium on Operating Systems Principles*, pages 13–24, 1987.

[Zho87] Songnian Zhou. *Performance studies for dynamic load balancing in distributed systems.* PhD Dissertation, University of California, Berkeley, 1987.

[ZWZD93] S. Zhou, J. Wang, X. Zheng, and P. Delisle. Utopia: a load-sharing facitlity for large heterogeneous distributed computing systems. *Software – Practice and Expeience*, 23(2):1305–1336, December 1993.