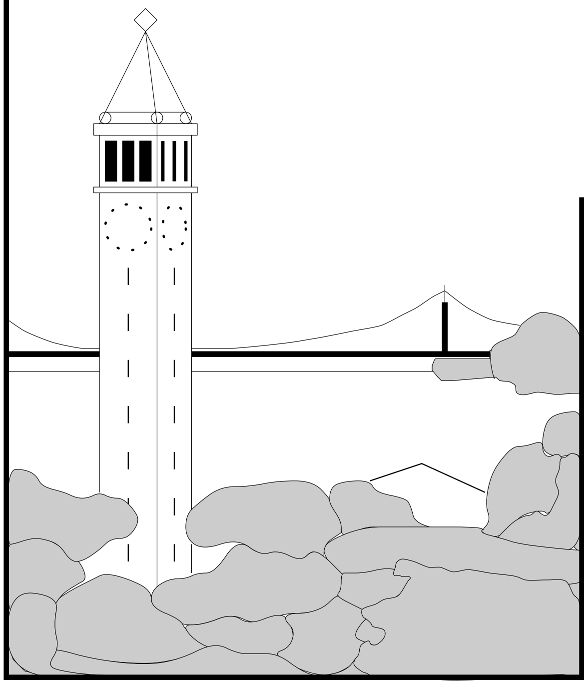
Exploiting Process Lifetime Distributions for Dynamic Load Balancing

Mor Harchol-Balter and Allen B. Downey



Report No. UCB/CSD-95-887

November 1995

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Exploiting Process Lifetime Distributions for Dynamic Load Balancing

Mor Harchol-Balter* Allen B. Downey †

November 1995

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. If the cost of remote execution is significant relative to the lifetimes of processes, then implicit non-preemptive policies require some *a priori* information about job lifetimes. This information is often implemented as an eligibility list

*Supported by National Physical Science Consortium (NPSC) Fellowship. Also supported by NSF grant number CCR-9201092. Email: harchol@cs.berkeley.edu

†Partially supported by NSF (DARA) grant DMW-8919074.Email: downey@cs.berkeley.edu

[1] The other half of a load balancing strategy is the location policy — the selection a new host for the migrated process. Previous work ([Zho89] 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.

(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

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

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 UNIX network of 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 T seconds of CPU time is $1/T$ (see Figure 1).

Our measurements are consistent with the lifetime result of [LO86], but this prior work has been incorporated in few subsequent analytic and simulator load balancing studies. This omission is un-

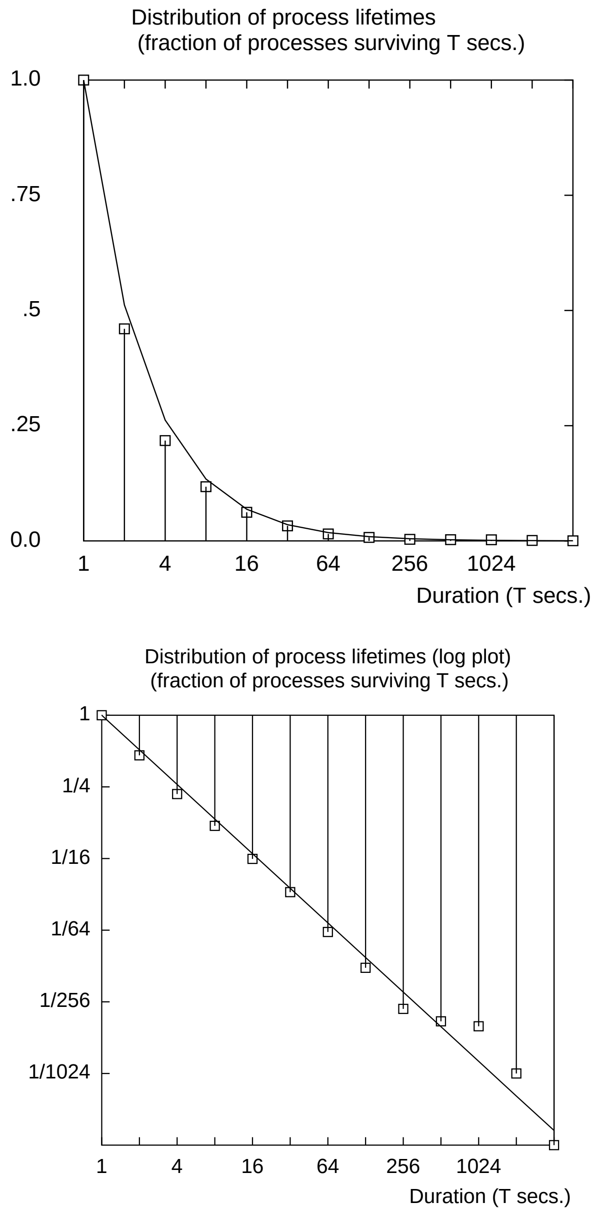


Figure 1: *a)Distribution of process lifetimes for processes measured on machine po, mid-semester. The impulses show the measured data; the curve shows the fitted values. b) The same distribution as (a), shown on a log-log scale. The straight line in log-log space indicates that the process lifetime distribution can be modelled by* T^k*, where* k *is the slope of of the line.*

fortunate, 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 which must be hand-optimized). This tool is generally useful for analysis of system behavior.

Specifically, Section 3 shows the derivation a migration eligiblity 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 suprisingly 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 far 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].)

Systems that have implemented *user-controlled* (explicit) remote execution and/or preemptive migration include: Accent [Zay87], Locus [Thi91], Utopia [ZWZD93], DEMOS/MP [PM83], V [TLC85], and NEST [AE87]. Several of these also provide some form of automated location policy.

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], and Condor [LLM88]. Although these systems are capable of migrating active processes (with varying degrees of transparency), none have implemented a policy that preempts processes for purposes of load-balancing.

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]).

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.

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], [Zho89], [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. In this context, nonpreemptive migration can maintain load balance simply by placing jobs at lightly-loaded hosts. 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 non-trivial.

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 distribution which approximates very 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] discusses 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 by far 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 for UNIX processes

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] presented the first results on a functional form for the process lifetime distribution. This functional form was based on measurements of the lifetimes of 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.

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 very largest jobs, the curve does not fit well, since there are very few of these jobs.[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 T seconds of cpu time is about $1/T$.

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

[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 a range of lifetimes). Although the functional form has infinite mean and variance, naturally the observed distributions have finite mean and variance.

2. The probability that a process with age T seconds uses 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 would not be impacted.

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 which match with respect to both mean and variance might still produce radically 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 Process lifetime distribution when lifetime > 1 second

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 is an impulse plot showing our process lifetime measurements on a heavily-used instructional machine in mid-semester. The plot shows only processes whose lifetimes exceed one second. The impulse (line) at 2^i seconds indicates the *fraction* of processes we counted whose lifetimes exceeded 2^i seconds. Figure 1b shows the same data on a log-log scale. The straight line in log-log space indicates that the process lifetime distribution fits the curve T^k, where k is the slope of 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 very high degree of certainty). Finally, 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 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 and the corresponding density function and 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.

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.*

2.2 Measuring the 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.

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 mi-

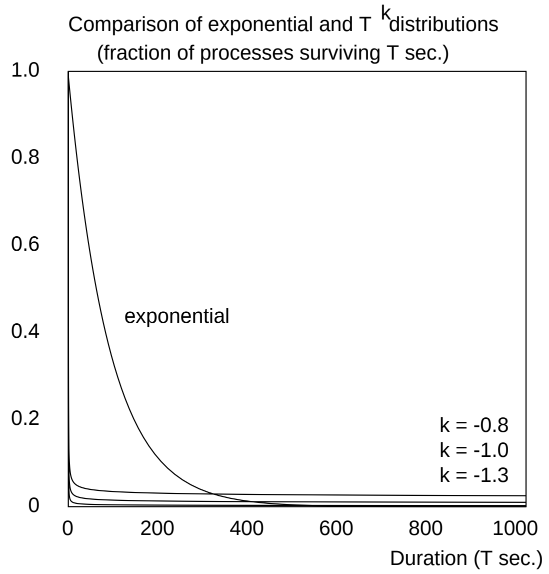


Figure 2: *The curve T^{-1} is a reasonable approximation of the curves T^k for a range of k from $-.8$ to -1.3. The exponential curve shown is $e^{-\lambda T}$, where the value of λ was chosen by a least-squares fit to 100 points on the curve $1/T$. Although the exponential model has one free parameter (compared to the zero-parameter $1/T$ model), it does not produce a reasonable approximation.*

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

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 $k = -1.0$.*

grate?

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 (non-preemptive), because these processes have no allocated memory and therefore 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.

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).

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 will impose slowdowns on many small processes.

3.2 Taking Migration Costs into Account: 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 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

[4] The idea of migrating newborn processes might also stem from the fallacy that process lifetimes have an exponential distribution, with all processes having equal expected remaining lifetimes regardless of their age.

[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.

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

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

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.

The MOSIX migration policy [BSW93] is based on a similar, but simpler restriction: 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 similation studies.

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* which 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 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 which 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 ~ 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].

On Sprite preemptive migrations took .33 seconds plus 2.0 seconds per megabyte of memory transferred. By implementing migration at the kernel level, MOSIX reduces the fixed cost, f, to only 6 ms; the inverse memory transfer bandwidth, $1/b$, is .44 seconds per megabyte [Bra95].

It appears that the fixed part of the cost of preemptive migration is determined by the implementation of migration; the memory transfer cost depends mostly on properties of the network.

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 which 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] Processes are submitted to hosts with start times and durations taken from real machine traces (from the same machines used to measure the distribution of process lifetimes). We selected 6 daytime intervals from these traces, each eight hours long, and executed them simultaneously on a simulated network.

Although the workloads on the six hosts are homogeneous in terms of the job mix and the approximate level of activity, there is considerable variation during the eight-hour trace. At any given time, at least one of the six hosts is usually idle; however, during busy intervals, all six are active. In order to evaluate the effect of these variations, 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 ~ 15000 processes; Run 7 has ~ 30000 processes. The average duration of processes (for all runs) is $\sim .4$ seconds. Thus the total utilization of the system, ρ, is between .27 and .54.

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 distribution (see Section 5.2).

The strategies we considered are:

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 arrives at a heavily-loaded host, the process is executed remotely on the host with the lowest load. Processes cannot be migrated once they have begun execution.

The performance of the name-based, non-preemptive strategy depends on the list of el-

[6] The trace-driven simulator and the trace data will be made available by anonymous FTP before publication.

igible 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 long *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 considered 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, any process that satisfies the migration criterion is migrated away. Since the system can only initate a migration when a new process arrives, it misses some migration opportunities. We have examined strategies that allow migration at times other than process arrivals, and they do improve the performance of the system, but we have omitted them here to facilitate comparison between the two migration strategies.

Both migration strategies depend on the definition of a heavily-loaded host. Like Zhou ([Zho89]) and others, we use a simple threshold on the number of jobs in the run queue. Alternative strategies that use more global load information improved system performance somewhat, but they are not the focus of this paper. We chose a load threshold of 2 processes; any time more than one process is running on a host, one of them may be migrated away.

5.1.1 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, and thus it is more meaningful to report a 50% reduction in delay than a 25% improvement in performance.

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 the user's perception of performance.

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.

For the sake of simplicity, 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.

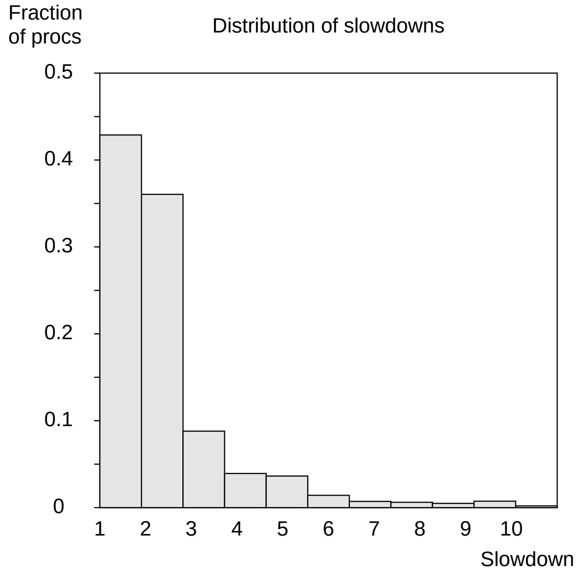


Figure 3: *Distribution of process slowdowns for run 0 (with no migration). Most processes suffer small slowdowns, but the processes in the tail of the distribution are more noticeable and annoying to users.*

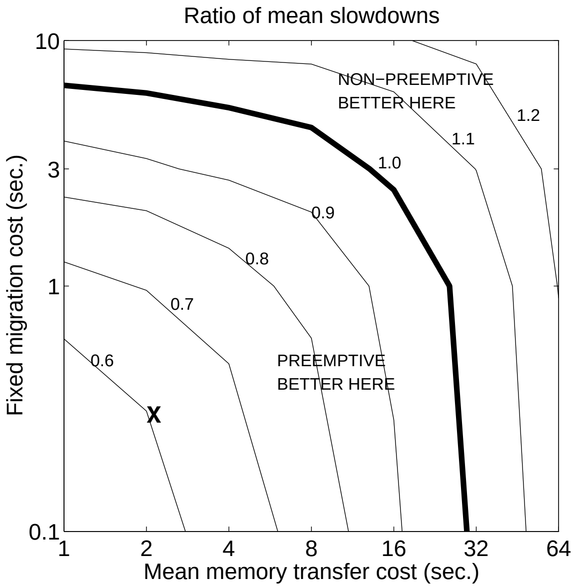


Figure 4: *The performance of preemptive migration relative to non-preemptive migration deteriorates as the cost of preemptive migration increases. The two axes are the two components of the preemptive migration cost. The cost of non-preemptive migration is held fixed. The "X" marks the particular set of parameters which we will consider in the next section.*

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.

5.2 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 4 and 5 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 Figures 4 and 5 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 mi-

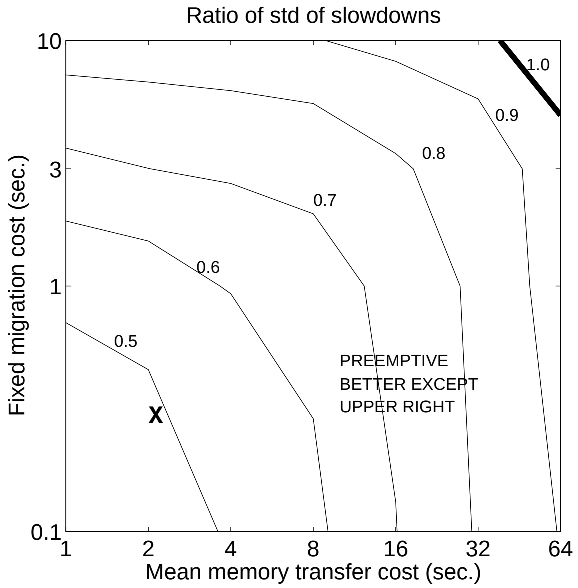


Figure 5: *The standard deviation of slowdown may give a better indication of the users' perception of system performance than mean slowdown. By this metric, the benefit of preemptive migration is even more significant.*

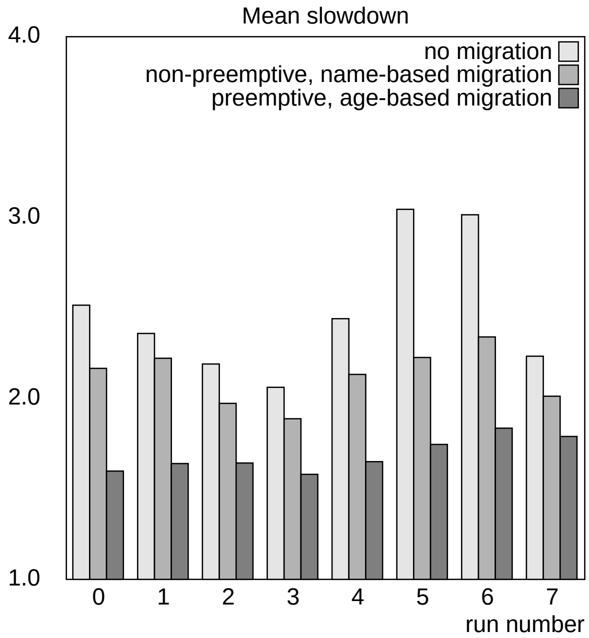


Figure 6: *Mean slowdown of all processes (minimum possible slowdown is 1.0).*

gration. 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 these 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 5 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 5 than in Figure 4. 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 slowdowns (relative to their runtimes) and add to the tail of the distribution of slowdowns. In the next section, we show cases in which the standard deviation of slowdowns in actually worse with preemptive migration than with no migration at all (three of the eight runs).

5.3 Comparison of preemptive and non-preemptive migration 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 4 and 5, the point corresponding to these parameter values is marked with an "X". Figures 6–9 show the performance of the two migration strategies at this point (compared to the base case of no migration).

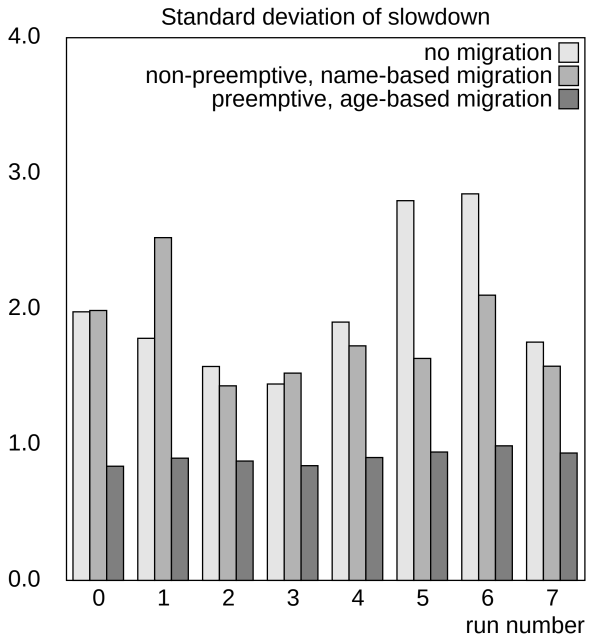


Figure 7: *Standard deviation of slowdown.*

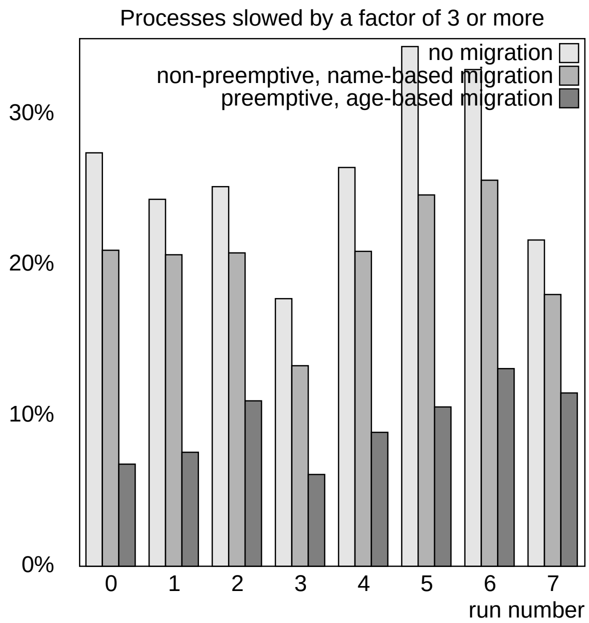


Figure 8: *Percentage of processes slowed by a factor of 3 or more.*

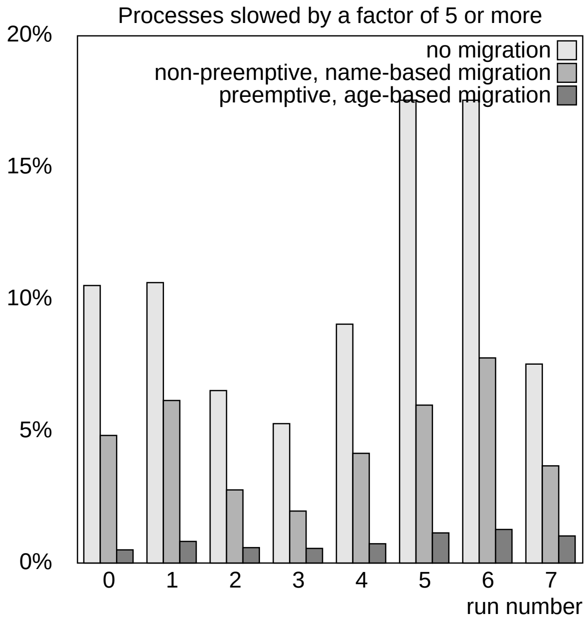


Figure 9: *Percentage of processes slowed by a factor of 5 or more.*

Non-preemptive migration reduces the normalized mean slowdown (Figure 6) by less than 20% for most traces (and $\sim 40\%$ for the two traces with the highest loads). Preemptive migration reduces the normalized mean slowdown by 50% for most traces (and more than 60% for two of the traces). 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 7 shows the standard deviation of slowdowns, which reflects the number of severely impacted processes. Figures 8 and 9 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 9, 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 7 is that for several traces, 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.

Another benefit of preemptive migration is graceful degradation of system performance as load increases (as shown in Figures 6–9). 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. Two kinds of mistakes are possible in a non-preemptive, name-based strategy that are eliminated by the age-based, preemptive strategy:

Migrating short-lived jobs whose names are on the list of eligible processes: This type of error imposes large slowdowns on the migrated process, wastes network resources, and fails to effect significant load-balancing. It might improve the performance of name-based systems to exclude names with high mean lifetimes but high variance.

Failing to migrate long-lived jobs whose names are not on the list: This type of error imposes moderate slowdowns on the potential migrant, and, more importantly, inflicts delays on short jobs that are forced to share a processor with a CPU hog.

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, and not because they suffered migration delays. 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. The primary benefit of preemptive migration is the ability to migrate long jobs away from busy hosts so that subsequent small jobs run unimpeded. Preemptive migration is effective because it is more successful at identifying these long jobs.

Because preemptive migration is not limited to a set of eligible migrants, the total number of migrations is higher under preemptive migration than under non-preemptive migration. Nevertheless, typically fewer than 4% of processes are migrated once and fewer than .25% of all processes are migrated more than once.

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*:

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

where α is a free parameter. For comparison, we will use the *best fixed parameter*, which is, for each trace, the best parameter for that trace, chosen empirically by running the trace 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.

Figures 10 and 11 compare the performance of the analytic minimum age criterion with the best fixed parameter (α). The best fixed parameter varies considerably from trace to trace, and appears to be roughly correlated with the average load during the trace (the traces 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.

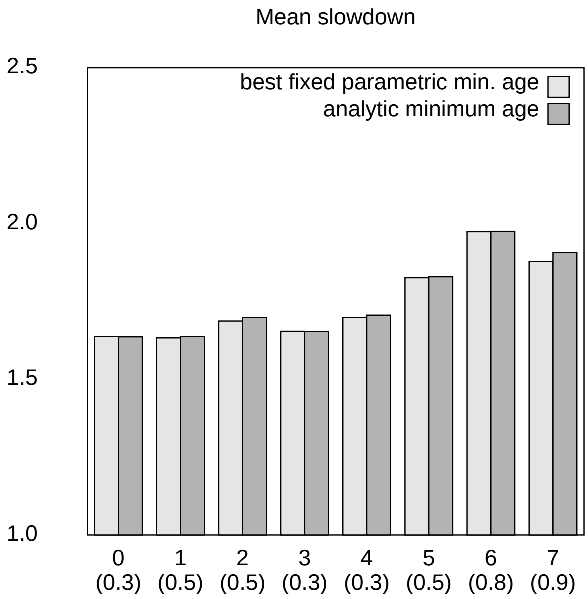


Figure 10: *The mean slowdown for eight runs, using the two criteria for minimum migration age. The value of the best fixed parameter α is shown in parentheses for each trace.*

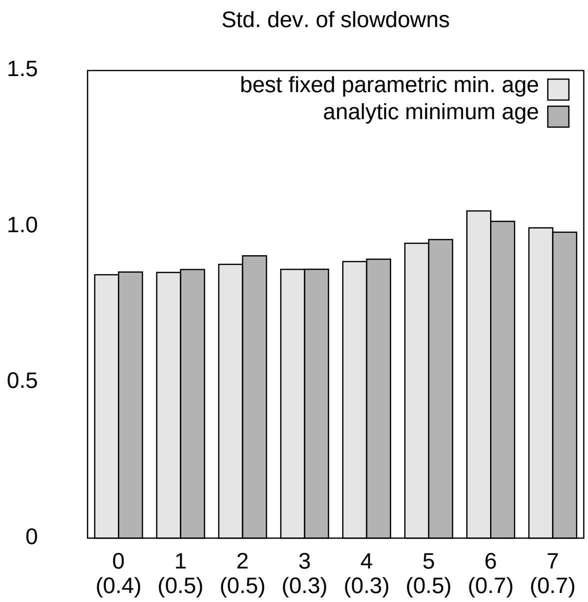


Figure 11: *The standard deviation of slowdowns using the two criteria for minimum migration age.*

6 Weaknesses of the model

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

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.

interaction : Our model of migration cost considers only the cost of transferring a process, and not the additional costs imposed on 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 which came up most frequently in our traces (with cpu time greater than 1 second, since these are the processes most likely to be migrated). The two most common names by far 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 there are some jobs in our traces that are in reality interactive, our model is reasonable for many of the most common jobs.

memory size : In this paper we've made the pessimistic simplification that a migrant's entire memory must be transferred. However, we have not been able to make a claim about the size of this memory for processes on current systems. It would be useful to examine memory transfer costs in more detail.

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

- 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.

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.

[CK87] Thomas L. Cassavant 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.

[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. LSBATCH: 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.

[Zho89] Songnian Zhou. *Performance studies for dynamic load balancing in distributed systems.* PhD Dissertation, University of California, Berkeley, 1989.

[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.