INTERNATIONAL COMPUTER SCIENCE INSTITUTE

1947 Center St. • Suite 600 • Berkeley, California 94704-1198 • (510) 643-9153 • FAX (510) 643-7684

Exploiting Process Lifetime Distributions for Dynamic Load Balancing

Mor Harchol-Balter* Allen B. Downey †

TR-95-021

May 1995

Abstract

We propose a preemptive migration scheme that assumes no prior knowledge about the behavior of processes, and show that it significantly outperforms more traditional non-preemptive migration schemes.

Our scheme migrates a process only if the process' expected remaining lifetime justifies the cost of migration. To quantify this heuristic, we perform empirical studies on the distribution of process lifetimes and the distribution of memory use (which dominates migration cost) for a variety of workloads. We use these results to derive a robust criterion for selecting processes for migration.

Using a trace-driven simulation based on actual job arrival times and lifetimes, we show that under our preemptive policy the mean slowdown of all processes is 40% less than under an optimistic non-preemptive migration scheme that uses name lists. Furthermore, the preemptive policy reduces the number of severely delayed processes by a factor of ten, compared with the non-preemptive scheme.

*Computer Science Division, UC Berkeley, CA 94720. Supported by National Physical Science Consortium (NPSC) Fellowship. Also supported by NSF grant number CCR-9201092. Email: harchol@cs.berkeley.edu

†Computer Science Division, UC Berkeley, CA 94720. Supported by NSF (DARA) grant DMW-8919074. Email: downey@cs.berkeley.edu

1 Introduction

In most existing systems, load balancing is non-preemptive and based on *a priori* knowledge of process behavior. Previous analytic studies have shown that the additional benefit offered by preemptive migration is small compared with the benefits of simple, non-preemptive schemes [ELZ88]. 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 tries to resolve these conflicting results.

In addition, we explore in depth one of the questions raised by preemptive schemes – *which processes should be eligible for migration?* To answer this question, we derive the functional form of the distribution of process lifetimes and study the distribution of memory use. Using these, together with a fairness principle which states that one process should not suffer for the benefit of others, we derive (analytically) locally optimal eligibility criteria for migration.

We present a trace-driven simulation based on actual job arrival times, lifetimes and memory use. The simulator shows:

- The preemptive migration policy reduces the mean slowdown by $35-45\%$, and cuts the number of significantly slowed processes by a factor of ten, compared to a reasonably optimistic non-preemptive scheme.
- The performance of our analytic eligibility criterion for migration is similar, over a range of loads, to the performance of an optimal parameterized eligibility criterion (See Section 4.1.1)

1.1 Load balancing

Given a network of time-sharing host machines, *load balancing* is the idea of migrating processes across the network from hosts with high workloads to hosts with low workloads. The reasons for load balancing include minimizing the average completion time of processes and maximizing 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 1.6).

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: *implicit*[2] *remote execution* (also called *non-preemptive* migration), in which some new processes are 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. Migration policies may also be classified as having *a priori knowledge* about the function of the processes, or how long they will take, or having *no prior knowledge* about the function of the processes. Prior knowledge is often implemented as a name-list (e.g. [Sve90]) that specifies (by name) which processes are eligible for migration. These name-lists may have been created by a user or by the system. Policies that don't rely on any prior knowledge only have system data to work with, 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 prior information** about processes.

1.2 Our Model/Definitions

Migration primarily benefits cpu-bound processes. Throughout this paper, we model a process as using two resources: cpu and memory. Our model does not include I/O. Thus, we use "age" and "lifetime" to mean cpu age (the cpu time a process has used thus far) and cpu lifetime (the total cpu time it uses during its life). Since we assume that hosts time-share among processes, 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 either getting cpu or waiting on cpu during its life.

1.3 Distribution of process lifetimes

Previous efforts to measure the distribution of process lifetimes have produced conflicting results. [SPG94] claims that process lifetimes have an exponential distribution, which implies that the expected remaining lifetime of a process is independent of its current age. [SPG94] states

> "...The duration of cpu bursts have been measured. Although they vary greatly

[1] The other half of a load balancing strategy is the location policy, namely how to choose a new host for the migrated process to run on. 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.

[2] Implicit refers to the fact that the system is migrating the processes rather than the user (explicit).

from process to process and computer to computer, they tend to have a frequency curve generally characterized as exponential or hyperexponential[3]".

Rommel, [Rom91], also claims his measurements show that "long processes have exponential service times." On the other hand, [LO86] measured 9.5 million Unix processes between 1984 and 1985 and 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. Specifically, they found that for $T > 3$ seconds, the probability of a process' lifetime exceeding T seconds is rT^{-c}, where $1.05 < c < 1.25$ and r is the normalization factor.

Many of the location policies and migration policies proposed in the literature (see for example [ELZ88], [MTS90] [BK90], [EB93], [LR93], [LM82], [WM85]) are based on the assumption that process lifetimes are exponential or hyperexponential.

We show that the function form of the process lifetime distribution proposed by [LO86] is correct for processes older than 1 second. This functional form is consistent across a variety of machines and workloads, and although the parameter of this function varies (from .8 to 1.3), 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 $\frac{1}{T}$.
2. The probability that a process with age T seconds uses an additional T seconds of cpu time is about $\frac{1}{2}$.

Note how slowly this function drops off compared to the exponential distribution, for which the probability that a process uses an additional *one* second of cpu time is $\frac{1}{2}$, and the probability that a process uses an additional T seconds of cpu time (regardless of its age) is e^{-T}. This and other empirical results we obtain on process lifetimes impact our choice of migration policy.[4]

1.4 Migration policies: Which processes are *worth* migrating?

As described in Section 1.6, most existing migration policies are non-preemptive, meaning that only newborn processes are migrated.

Migrating newborn processes is less expensive than migrating running processes, because the newborn has no allocated memory, and easier to implement, because the newborn has no state in the system (such as open files or interprocess communication channels).

Unfortuntely, the UBNE distribution of process lifetimes (described in Section 1.3) implies that newborn processes have the shortest expected lifetimes and are least likely to run long enough to justify the cost of migration. Migrating short-lived processes not only imposes large slowdowns on the migrated processes; it also consumes CPU time and network bandwidth without effectively transferring work away from overloaded hosts.

Thus a "newborn" migration policy is justified only if the system has knowledge about the processes and can selectively migrate only those processes likely to be cpu hogs.

In the absence of this kind of prior knowledge, however, the UBNE distribution of lifetimes suggests migrating older processes, since they have longer remaining life expectancies. There are two potential problems with such a strategy:

1. There is a danger that the additional cost of migrating older processes (i.e. transferring memory and system state) could be prohibitive.
2. There is a danger that if *only* old processes (older than, say, one second) can migrate, it might not be possible to migrate enough processes to have a significant load balancing effect.

Our measurements indicate that both of these concerns are unfounded. In regard to the first, we will show that the average memory size is small and uncorrelated with age; thus, older processes have a higher probability of outliving their migration times than new processes. In regard to the second concern, we will show that more than 50% of total cpu time is spent on processes with ages greater than four seconds; thus, a migration policy that migrates only these processes can have a significant load-balancing effect.

These results are not surprising considering that the observed lifetime distribution has a much longer tail than the exponential distribution. This tail implies that there are many long-lived processes, and that the expected remaining lifetimes for these processes is high. Therefore, whatever the migration cost is for these processes, it may still pay to migrate them.

1.5 Trace-driven simulation

We use a trace-driven simulation to evaluate the performance advantages of our preemptive migration policy over non-preemptive migration. For the non-preemptive

[3] The hyperexponential distribution is defined as the combination of two or more exponential distributions. [Wol89]

[4] The rule of thumb implies that the expected remaining lifetime of all processes is infinite (a consequence of integrating the tail of the $\frac{1}{T}$ distribution from zero to infinity). For practical purposes, of course, there is a finite bound on the lifetimes of processes. In order to circumvent this complication, we consider median (rather than average) remaining lifetimes.

policy, we assume optimistically that we have name-lists telling us which processes are eligible for migration. In our preemptive policy, we assume no prior knowledge of the processes. We find that

1. The performance benefits of preemptive migration are significantly greater than the benefits of non-preemptive migration.
2. The workload description has a great effect on the simulation results.
3. Models based on exponential service times underestimate the performance benefits of preemptive migration.
4. The standard metric of system performance, mean slowdown, understates the performance benefits of migration.

1.6 Related Work

Most existing systems provide some form of user-controlled remote execution, but relatively few provide automated load balancing by migrating processes. Of the ones that do, the majority are based on implicit remote execution of new processes; few provide preemptive migration. (The taxonomy below is based in large part on [Nut94].)

The systems that have implemented user-controlled remote execution and migration include: Accent [Zay87], Locus [Thi91], Utopia [ZZWD93], DEMOS/MP [PM83], V [TLC85], and NEST [AE87]. Several of these provide some form of automated location policy.

Some other systems provide automated remote execution, but perform preemptive migration only at the explicit 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.

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.

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 ([BF81], [ELZ88], [KL88]).

Our work differs from both [ELZ88] and [KL88] in that we use trace-driven simulations rather than synthetic workloads. This approach eliminates as a source of error the unrealistic workload descriptions that are necessary for analysis. Furthermore, our migration policy differs from [KL88] in that our proposed migration policy uses preemptive migration exclusively, rather than in addition to remote execution (which they call "placement").

Many load-balancing systems depend on *a priori* information about processes; for example, explicit knowledge about the runtimes of processes or user-provided lists of migratable processes ([Sve90], [ZF87], [Zho89], [ZZWD93], [DO91], [LL90], [AE87]).

1.7 Organization of this paper

In Section 2, we measure the process lifetime distribution, compute the conditional lifetime distribution, and make several empirical observations about process lifetimes. These results will be used in developing our load migration policy.

Section 3 proposes a preemptive load-balancing policy based on the principle of choosing processes for migration that are most likely to run long enough to justify the cost of migration. We analytically derive the optimal minimum age at which a process should be eligible for migration.

Section 4 presents the results of a trace-driven simulation showing the performance benefits of our preemptive migration policy compared with a typical non-preemptive migration strategy, and discusses its relevance to previously implemented strategies.

2 Probability distribution function for Unix process lifetimes

To determine the probability distribution functions 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.

We observed that long processes (with lifetimes greater than 1 second) have a predictable and consistent distribution. Section 2.1 describes this distribu-

tion. Section 2.2 makes some additional observations about shorter processes. Lastly in Section 2.3 we make some general empirical observations about process cpu usage that we will need in developing our process migration strategy.

2.1 Process lifetime distribution when lifetime > 1 second

Figure 1a is an impulse plot showing our process lifetime measurements on a heavily used instructional machine, po, during mid-semester. The plot depicts 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 is accurately described by the curve T^c, where c is the slope of of the line.

For all the machines we studied, the process lifetime data (for processes of age exceeding one second) is described by a curve of the form T^c, where c ranges from about $-.8$ to -1.3 for the different machines. Table 1 shows the estimated lifetime distribution curve for our measurements on 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 essentially perfect.

Although the range of parameters we observed is fairly broad, in the absence of measurements from a specific system, assuming a distribution of $\frac{1}{T}$ is substantially more accurate than assuming that process lifetimes are exponentially distributed.

Table 2 shows the lifetime distribution function, the corresponding density function, and the corresponding conditional lifetime 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 $c = -1$, which we will assume in our analysis in Section 3.

2.2 Measuring the process lifetime distribution in general

In this section, we will see that processes of age < 1 second complete at an even slower rate than those of

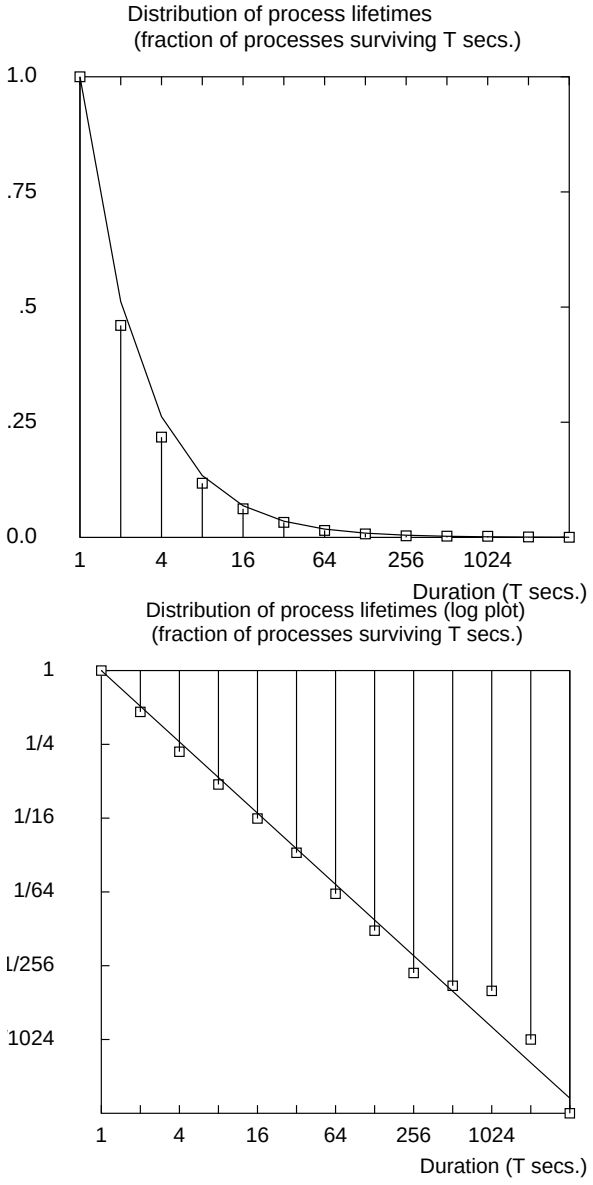


Figure 1: *a)Distribution of process lifetimes for processes measured on machine po, mid semester. The impulses show the measured data; the curve shows the fitted values. b) The same distribution as in (a), shown on a log-log plot. The straight line in log-log space indicates that the process lifetime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

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
porsc	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 respectively. Cory is a heavily-used machine, used for coursework and research. Porsc is a less frequently-used machine, used primarily for numerical analysis research. 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.*

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

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 of c.*

Seconds (T)	Num processes that live $> T$ secs	$s(T)$	Constant Region
0	77440	52%	
2^{-6}	40117	57%	
2^{-5}	22991	77%	
2^{-4}	17808	76%	
2^{-3}	13581	73%	
2^{-2}	9980	66%	
2^{-1}	6632	62%	
2^{0}	4107	46%	$\surd$
2^{1}	1890	47%	$\surd$
2^{2}	894	54%	$\surd$
2^{3}	483	53%	$\surd$
2^{4}	255	53%	$\surd$
2^{5}	134	46%	$\surd$
2^{6}	62	50%	$\surd$
2^{7}	31	45%	$\surd$
2^{8}	14	79%	
2^{9}	11	91%	
2^{10}	10	40%	
2^{11}	4	25%	
2^{12}	1		

Table 3: *Process lifetimes measured on machine po, mid-semester. These measurements correspond to Figure 1.*

age ≥ 1 second.

That is,

$$\mathbf{Pr}\{\text{Process lifetime} > a \text{ sec} \mid \text{age} = b < 1 \text{ sec}\} > \left(\frac{a}{b}\right)^c$$

To see this, let

$$s(T) = \mathbf{Pr}\{\text{ Process lifetime} > 2T \mid \text{lifetime} > T\}$$

For each machine we studied, we found that $s(T)$ was constant when $T \geq 1$. We refer to this constant as the machine's *survival ratio* and denote it by s^*. In the range $T < 1$, $s(T)$ varies, but always $s(T) \geq s^*$. Table 3 illustrates this point; the third column, $s(T)$, is nearly constant for processes whose cpu lifetimes exceed one second. (We ignore the last few entries in the table, since they involve too few processes to be statistically useful.)

In the previous section, we showed empirically that in this constant region ($T \geq 1$) the distribution of process lifetimes has the functional form T^{-c}. The following analysis shows how to derive this functional form from the survival ratio s^*.

$$\mathbf{Pr}\{\text{ Process lifetime} > 2T \mid \text{Process lifetime} > T\} = s^*$$

$$\Longrightarrow \mathbf{Pr}\{\text{Process lifetime} > 2T\} = s^* \cdot \mathbf{Pr}\{\text{lifetime} > T\}$$

Iterating the above recurrence, we obtain the formulas in Table 2:

$$\mathbf{Pr}\{\text{ lifetime} > T \mid \text{lifetime} > 1\} \quad = \quad (s^*)^{\lg T}$$

$$= T^{\lg(s^*)}$$
$$= \frac{1}{T}, \text{ when } s^* = .5$$

2.3 More Empirical Observations

Lastly, we make the following empirical observations that we will need when discussing migration policies.

- For all machines we studied,
  $> 50\%$ of all cpu time was used by processes of age at least 4 seconds.
  $> 60\%$ of all cpu time was used by processes of age at least 2 seconds.
  This is due to the long tail of $\frac{1}{T}$.
- For all machines we studied,
  Pr[Process lifetime $> .33$ sec] $< .15$.

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 4.1.1):

> 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. First, from the host's perspective, a large fraction of the migration time is spent at the host (packaging the process). It only makes sense for the host to invest this time to get rid of the process if it is less than the expected remaining CPU time. Secondly, from the process' perspective, the migration time has a large impact on response time. If the expected total lifetime of the process is longer, the overhead imposed by migration can likely 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 3.1).[5] The problem with this policy is that the process lifetime distribution (Section 2) tells us that these newborn processes actually have the shortest expected remaining lifetimes.

A "newborn" migration policy is thus 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 is quite limited, as shown in Section 4.

Can we do better? The lifetime distribution shown in Section 2 points us towards migrating *older* processes. However, there are two potential problems with this strategy. First, the additional cost of migrating old processes (the memory transfer cost) might overwhelm the benefit of migrating longer-lived processes. Secondly, since the vast majority of processes are short, there might not be enough long-lived processes to have a significant load-balancing effect.

Section 3.1 addresses the first issue by quantifying the difference in migration costs between preemptive and non-preemptive migration. We show that the memory transfer cost for most processes is small and that there is no correlation between a process' age and its virtual memory size. Thus, old processes have the highest chance of living long enough to justify their migration cost.

Section 3.2 shows that although there are few long-lived processes, they represent a significant part of the total CPU load; thus, migrating old processes is sufficient to balance the system load.

In Section 3.3 we use the conditional distribution of process lifetimes (from Section 2) to derive a lower bound on the age of a migrant process.

3.1 Migration cost as a function of process age.

The cost of migrating a process can be modelled as the sum of a *fixed migration cost* for migrating all the state, except for the virtual memory, plus a *memory migration cost*. These costs vary depending on the system. For Sprite ([DO91]), the average fixed cost has been measured at .33 seconds and the memory migration cost is 2 seconds per MByte of memory transferred. We will use these numbers as a working example throughout the rest of this paper.

$$\begin{aligned} \text{Migration cost} &= \text{Fixed migration cost} + \text{Memory migration cost} \\ &= .33 \text{ s} + 2 \text{ s/MB memory transferred} \end{aligned}$$

The amount of a process' memory that must be transferred during 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 virtual memory. With a distributed file system (as in Sprite) it is only necessary to write dirty pages to the file system before migration. When the

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

process is restarted at the target host, it will retrieve these pages. In systems that use precopying (such as the V system), some of the virtual memory might need to be copied over more than once, but total delay imposed by the migration is usually reduced.

For the trace-driven simulation (Section 4) we use the following model of migration cost

$$\text{Migration cost} = .33 \text{ s} + 2 \text{ s/MB resident VM}$$

By Section 2.3, the probability that a newborn process lives for another period equal to its migration cost is less than 15%, so migrating newborn processes doesn't make sense.

To evaluate the additional cost of migrating active processes, we examine the distribution of memory sizes for UNIX processes, and the relationship between a process' age and its memory size. We find that for the majority of processes ($> 85\%$), the size of the resident set is less than .5 MB, and thus (for our cost model) the cost of migrating an active process is roughly four times the cost of migrating a newborn. Furthermore, we find that for processes over age one second, there is no correlation between a process' age and its virtual memory size.

Figures 2 and 3 show the memory use of processes on the two machines in our sample with the largest and smallest average memory size, "po" and "hera." Po and is a heavily-used instructional machine; hera is used for academic administration. Each figure shows the $85th$, $95th$ and $99th$ percentiles for resident set size. Although the $95th$ and $99th$ percentile lines fluctuate with age, the $85th$ percentile line is roughly constant and below 0.5 MB, regardless of process age.

We also examine the distribution of total memory sizes and found, similarly, that the $85th$ percentile line is below 1MB regardless of the age of the process.

We made our measurements using the UNIX "ps" command to take snapshots of all processes at one hour intervals (thus, some long-lived processes were sampled more than once) over the course of 5 days. We ignored processes whose resident set sizes were zero, because these processes are not active and hence not available for migration. Some processes are not shown in the figures for purposes of scaling, but the percentile lines are based on data from all processes.

3.2 Moving Enough Work

The second concern mentioned above is that if only old processes are eligible for migration, there might not be enough of them to produce a significant load-balancing effect.

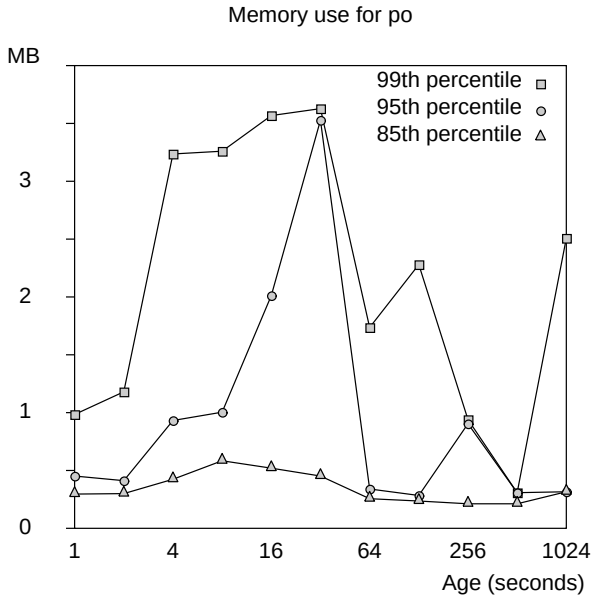


Figure 2: *From machine po: Scatterplot of resident set size of 7731 processes as a function of the age of the process. The processes have been grouped by age (rounded up to the next power of two). The lines show percentiles of memory use within each group. For example, in the above figure, for processes between 8 and 16 seconds of age, 85% are using less than half a megabyte and 95% are using less than two megabytes. (The 366 discarded processes with resident set size zero are not shown.)*

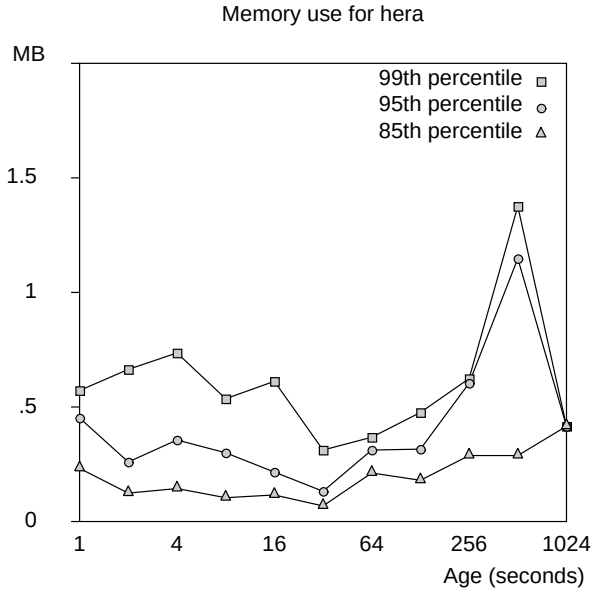


Figure 3: *From machine hera (this machine is used by the Computer Science staff for administrative work) : Scatterplot of resident set size of 13977 processes as a function of the age of the process. (The 2306 discarded processes with resident set size zero are not shown.)*

Although there are few old processes, they account for a large part of the total CPU load. In the distribution shown in Table 2, fewer than 3% of processes live longer than 2 seconds, yet these processes make up more than 60% of the total CPU load (see Section 2.3).

3.3 Choosing the Age Parameter

So far, we have shown that it is desirable to migrate older processes, because they have a greater load-balancing effect, and undesirable to migrate newborn processes, because they are unlikely to live long enough to amortize the cost of migration.

We have suggested a load-balancing strategy based on migrating the oldest available process, but we have not addresses the question of specifically how old a process must be before it is eligible for migration. We will answer this question by applying the following fairness principle:

> *One process should not have to suffer for the benefit of other processes.*

Specifically, if the migration strategy indicates that one of the processes should be moved away from a heavily-loaded host, then fairness dictates that the slowdown imposed on the migrant process should be no higher than the slowdown it would suffer by staying at the source host. If no such process is available (because all processes are too young), we shouldn't migrate any process.

Using the distribution of process lifetimes, we will now derive a lower bound on the age of migrant processes that satisfies this requirement. In section 4.1.1, we will show results from a trace-driven simulation that suggest that this bound is also optimal for general system performance. In other words, a migration policy that is locally fair is also globally optimal.

First we derive the expected slowdown imposed on a migrant process as a function of its age. Denote the age of the migrant process by a, the cost of migrating the process by c, the lifetime of the migrant by L, the number of processes at the source host by n, and the number of processes at the target host (including the migrant) by m, we have

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

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 would experience in the absence of migration).

Thus, by the fairness principle, $\frac{c}{2a} + \frac{m}{2} + \frac{n}{2} < n$, which implies

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

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 must be less than 2.0. This bound is based on the worst case, in which the migrant process dies immediately upon arrival at the target.

The MOSIX requirement is likely to be too harsh, 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 MOSIX's policy with ours in Section 4.1.1.

4 Trace-driven Simulation

In this section we present the results of a trace-driven simulation of process migration. We compare two migration strategies, and show that the proposed age-based preemptive migration strategy (Section 3) performs significantly better than an optimistic version of a non-preemptive strategy that migrates newborn processes according to the process name (similar to strategies proposed by [ZZWD93] and [Sve90]).

We also suggest that the most common metric of system performance, average slowdown over all processes, understates users' perception of the benefit of process migration. We suggest alternative metrics intended to quantify the number of noticeable slowdowns the user suffers. By these metrics, the benefits of preemptive migration appear far more significant. We feel that this result should inspire reappraisal of earlier work ([ELZ88]) that demonstrated limited benefits for preemptive process migration.

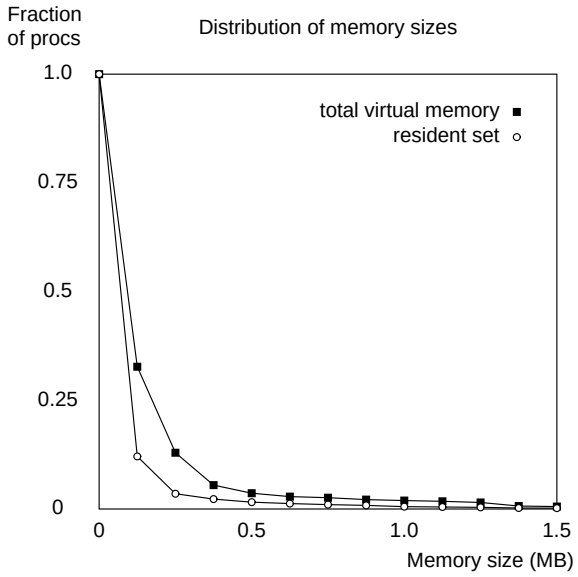


Figure 4: *The distribution of virtual memory use and resident set size for all processes measured (includes processes shown in Figures 2 and 3).*

4.1 The Simulator

We have implemented a trace-driven simulation of a network of six workstations[6]. Processes are submitted to the cluster with start times and durations taken from real machine traces (the same traces presented above in the derivation of the distribution of process lifetimes). We selected 6 periods of high activity 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 mixture of jobs and the approximate level of activity, there is considerable variation during the eight-hour trace. At any given time, one of the six hosts is usually idle; however, during busy interval, all six are active. In order to evaluate the effect of these variations, we divided the eight-hour trace into eight one-hour intervals.

Although the start times and durations of the processes come from trace data, the virtual memory size of each process is chosen randomly from the distribution in Figure 4. In accordance with the results shown in Figures 2 and 3, this memory size is uncorrelated with the age of the process.

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.

age-based preemptive migration A process is considered eligible for migration only if it has aged for some fraction of its migration cost. Every time a new process is born at a heavily-loaded host, the oldest eligible process on that host is migrated away.

The performance of name-based nonpreemptive migration 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 names with the longest mean durations. 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 significantly overestimate the performance benefits of this strategy.

4.1.1 Choosing migration parameters

Both migration strategies depend on the decision of when processes are migrated; i.e. 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 using 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.

For the preemptive migration policy, we also need to choose a minimum age criterion – how old a process has to be to be eligible for migration. According to the fairness principle discussed in Section 3.3, then, the minimum age for a migrant process is

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

where n is the load at the source host and m is the load at the target host (including the potential migrant).

In Section 3.3, we claimed that this criterion not only satisfies the fairness principle; it is also globally optimal. In order to evaluate this claim, we compared the performance of this criterion with the performance of the alternative criterion

$$\text{Minimum migration age} = f * \text{Migration cost}$$

[6] The trace-driven simulator and the trace data are available by anonymous FTP at (address deleted from draft to avoid identification of authors)

where f is a system parameter. As we discussed in Section 3.3, MOSIX choose the parameter $f = 1.0$, 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 (0.1), but did not explain how it was chosen.

Figures 10a and 10b compare the performance of the analytic minimum age criterion with the optimal fixed parameter (f). The optimal 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 minimum age is often better than (and always within a few percent of) the performance of the optimal fixed value. The advantage of the analytic minimum age criterion is that is 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 that is plagued with so many parameters.

4.1.2 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. We will also consider normalized slowdown, which is the ratio of inactive time (time in queue and time during migration) to CPU time. In this context, the normalized slowdown is always one less than the slowdown.

variance of slowdown This metric is often cited as a measure of the unpredictability of response time [SPG94]. In light of the observation that the distribution of slowdowns is skewed (see Figure 9), it might provide greater insight to interpret this as a measure of the size of the tail of the distribution; that is, the number of processes that suffer unusual and noticeable 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 assumed that processes are always ready to run (i.e. are never blocked on I/O). During a given time slice, we divide CPU time equally among the processes on the host.

Our model of migration cost is based on Sprite [DO91]: the total cost of migration is .33 seconds plus 2.0 seconds per megabyte of resident virtual memory (for new procs, of course, the resident set size is zero). As a further simplification, we charge the entire cost of migration to the source host. This is, of course, a pessimistic assumption for advocates of preemptive migration.

4.2 Simulator Results

4.2.1 Performance of migration strategies

Figures 5-8 show the performance of the two migration strategies relative to the base case of no migration. Non-preemptive migration reduces the mean slowdown (see Figure 7) by 20%[7] for most traces (and almost 50% for the two traces with the highest loads). Preemptive migration reduces the mean slowdown by 50% for most traces (and 70% for the two high-load traces). Thus, the performance improvement of preemptive migration over non-preemptive migration is typically between 35% and 45%.

This improvement is somewhat greater than that predicted by previous analytic models [ELZ88]. The primary reason for this discrepancy is that the workload model required by queueing-theoretic analysis doesn't describe real workloads. In reality, the variance of process lifetimes is higher than that of the exponential and hyperexponential distributions used, and process inter-arrival times are more correlated (bursty) than the usual memoryless Poisson arrivals.

Our results are in accord with previous simulator results [KL88] which used a more accurate distribution of process lifetimes than [ELZ88], but which used randomly-generated workloads with Poisson arrivals rather than trace data.

4.2.2 Alternative metrics

Although mean slowdown is the most common metric of system performance, we feel that the comparison of slowdowns (as above) understates the benefits of migration as perceived by users.

[7] For purposes of comparing the strategies, we will use normalized slowdowns (see definition above). Thus a mean slowdown of 1.5 is said to be 50% better than a mean slowdown of 2.0, since it is 50% closer to 1.0, which is the minimum possible value. The other metrics, including the standard deviation, are not affected by this normalization.

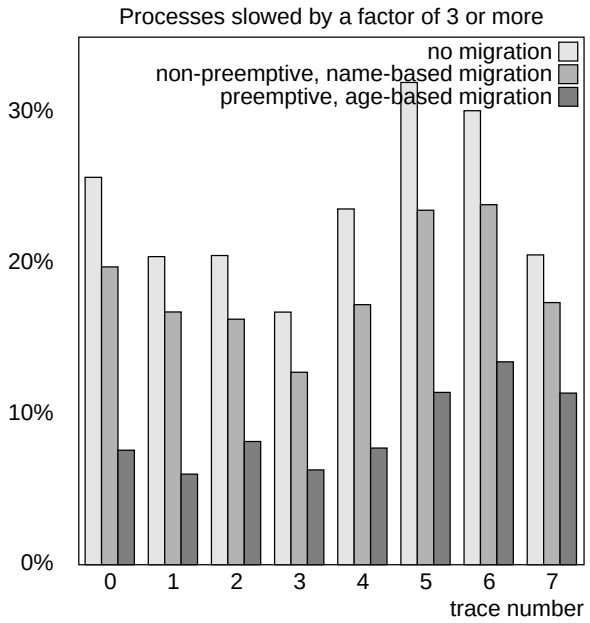


Figure 5: *Percentage of processes slowed by a factor of 3 or more.*

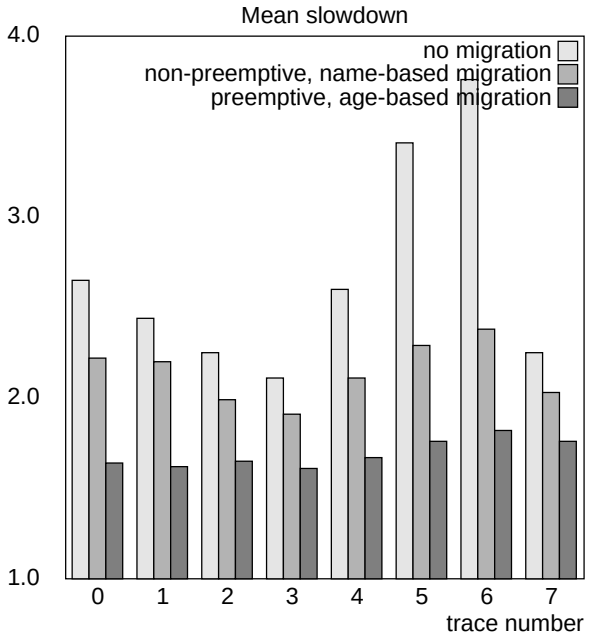


Figure 7: *Mean slowdown of all processes (minimum possible slowdown is 1.0).*

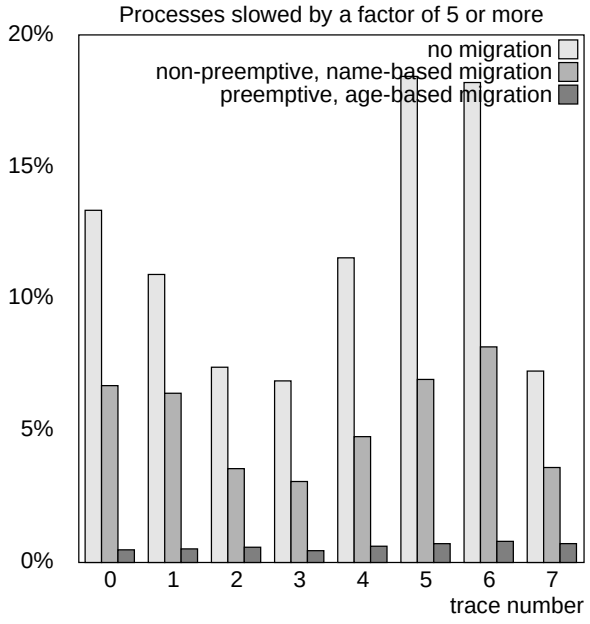


Figure 6: *Percentage of processes slowed by a factor of 5 or more.*

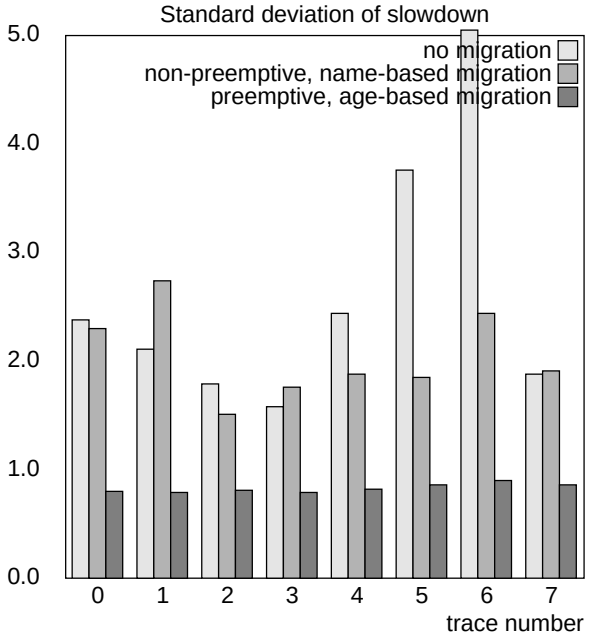


Figure 8: *Standard deviation of slowdown.*

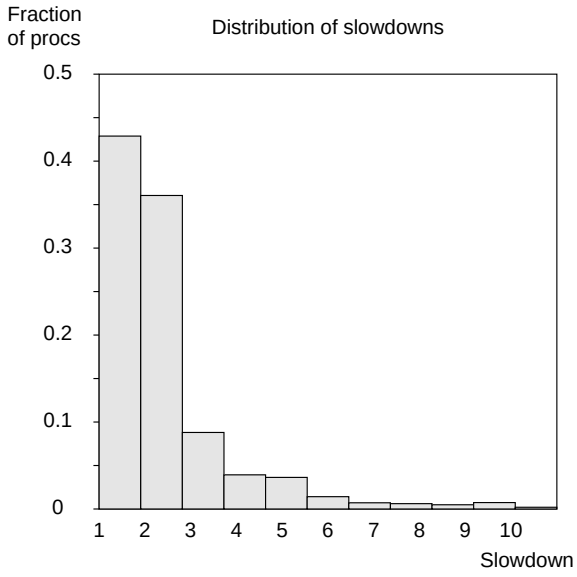
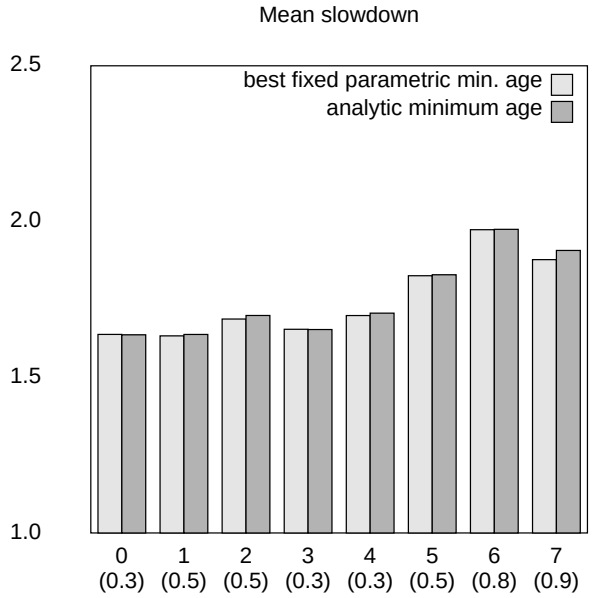


Figure 9: *Distribution of process slowdowns for trace 0 (with no migration). Most processes suffer small slowdowns, but the processes in the tail of the distribution are more noticeable and annoying to users.*

As shown is Figure 9, even in the absence of migration, the majority of processes suffer small slowdowns (typically 80% are less than 3.0 – see Figure 5). The value of the mean slowdown will be dominated by this majority. *From the user's point of view, however, 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.*

We propose three metrics to attempt to quantify these delays. Figures 5 and 6 explicitly measure the number of severely impacted processes, according to two different thresholds of acceptable slowdown. Figure 8 shows the standard deviation of slowdowns, which reflects not only the number of severely impacted processes, but also the annoyance of unpredictable response times.

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 6, 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 $41-62\%$ of these, which is a significant benefit, but preemptive migration consistently eliminates nearly all $(90-96\%)$ severe delays!



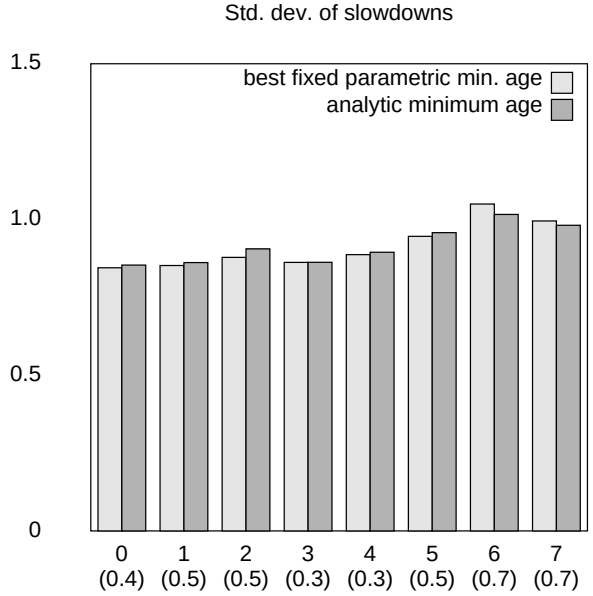


Figure 10: *Comparison of two criteria for choosing minimum migration age. The Best Fixed Parameter Criterion involves using one fixed ratio of age to migration cost to decide the eligiblity of a process for migration, where this fixed constant is the best possible fixed parameter for that trace, as determined by the simulator performance. Observe that the best fixed parameter changes with each trace. The Analytic Criterion is our parameterless criterion, described in Section 3.3.*

4.2.3 Preemptive vs. non-preemptive migration

The alternate metrics discussed above shed some light on the reasons for the discrepancy between the performance of 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.

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.

Both of these errors are reflected in the standard deviation of slowdowns (Figure 6): for most of the traces, name-based migration offers little improvement over no migration. In three traces it actually degrades the performance of the system.

One other benefit of preemptive migration is graceful degradation of system performance as load increases (as shown in figures 5-8). 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.

4.2.4 Potential complications

Previous preemptive load-balancing strategies (e.g. [BSW93]) have included mechanisms to avoid perverse behaviors such as repeated migration of a single long-lived process or bulk migrations back and forth between machines. For the most part, we found that simple strategies were stable and well-behaved, and no special checks were necessary to avoid these behaviors.

For example, we found that the number of migrations necessary to maintain good load balance is small; thus, there is little danger of rampant, system-wide oscillation. For name-based migration, fewer than 2% of processes are migrated; for age-based migration, fewer than 5% of processes are migrated (except when the system load is quite high). Furthermore, it is rare that more than one migration is in progress at a given time.

Another potential problem for dynamic load-balancing strategies is the danger that placement decisions might grow stale during a migration; that is, that by the time a process arrives at the target host of a migration, the load there might have increased. This problem is especially troublesome for preemptive migration strategies, where migration times are longer (due to to need to transport the migrant process' memory).

We found, though, that a simple placement policy (choosing the host with the lowest current load) worked well for both preemptive and non-preemptive migration. For name-based migration, 95% of migrant processes arrived at a host with no load, and the remaining 5% arrived at a host with one other process. For age-based migration, $70-80\%$ of processes arrived at a host with no load, but up to 10% arrived at hosts that were themselves heavily-loaded.

This result confirms that the increased migration times required by preemptive migration make it more difficult to choose the best target host. It also suggests that as the system load increases, the impact of location policy on system performance becomes more significant. Nevertheless, for the system loads we modelled, the performance of simple locations policies was adequate.

5 Future Work

We are currently undertaking projects to explore more completely the following topics:

- the distribution of memory sizes and resident set sizes in environments other than academic computers
- the effect of different cost models (cost of remote execution vs. cost of preemptive migration) on the relative performance of preemptive and non-preemptive migration

We feel that an important direction for future work in this area will be the addition of I/O to the model of system behavior and a study of its effect on the benefit of preemptive migration.

6 Conclusion

The main points of this paper are:

- The process lifetime distribution indicates that old processes are likely to have long remaining lifetimes. Thus it is often preferable for a load-balancing scheme to migrate old processes (preemptive migration) rather than newborn processes (remote execution), even if the cost of preemptive migration is much higher.
- Migration has a more significant load-balancing effect by moving a small number of long-lived

processes rather than many short-lived processes. Preemptive migration is more effective than non-preemptive name-based migration because it migrates exactly the processes that consume the most CPU time. In our simulations, fewer than 5% of processes are migrated once and fewer than .25% of all processes are migrated more than once.

- Using the observed distribution of process lifetimes, we have derived an analytic criterion for the minimum time a process must age before being migrated. This criterion is parameterless and robust across a range of workloads.

- We observe that exclusive use of mean slowdown as a metric of system performance understates the benefits of load-balancing schemes as perceived by users.

We now recapitulate our derivation of a preemptive migration scheme that assumes no prior knowledge of the behavior of processes.

Without *a priori* knowledge, the only information available for migration decisions is system state, e.g. the process' cpu age and memory use. To figure out how to use this information, we studied the distribution of process lifetimes and the distribution of memory use.

The general shape of the lifetime distribution suggested that migration schemes should migrate older processes, because they have the longest expected remaining lifetimes, and therefore are best able to amortize their migration cost.

The problem with migrating older processes, though, is that their migration cost is higher, because of the need to transfer the memory associated with the process. Our study of the distribution of memory sizes showed that the memory use (and hence the migration cost) of most processes is small and uncorrelated with age. This gave support to the strategy of migrating older processes.

We then answered the following two unresolved questions:

1. How old should a process be before it is eligible for migration? Is there a lower bound on a migrant's age, or should we just migrate the oldest process available?

2. If we're only migrating old processes, are there enough old processes around to have a significant load balancing effect?

To answer the first question, we proposed a fairness principle that limits the slowdown that can be imposed on a migrant process, and then used the functional form of the distribution of process lifetimes to derive a lower bound on the age of a migrant process (as a function of its migration cost). We showed that, for the load threshold 2.0, one should migrate only processes whose age exceeds half their migration cost. Being more strict or more lenient is suboptimal.

We addressed the second question with the observation that the oldest 2% of all processes account for over half the total cpu usage and therefore create a significant load balancing effect.

Using a trace-driven simulation, we compared the performance of the proposed preemptive migration policy with an optimistic name-based non-preemptive policy. The arrival times and duration of processes in the simulation are taken from real workloads. Under the preemptive policy, the mean slowdown of all processes is 40% less than under the non-preemptive policy.

In addition, we proposed alternative metrics intended to quantify users' perception of system performance. For example, while users might not notice a small change in the average response time, they would appreciate a reduction in the number of long, noticeable delays. By these alternative metrics, the discrepancy between preemptive and non-preemptive policies is even greater. In our simulated system, preemptive migration cuts the number of severely delayed processes by a factor of ten, compared to non-preemptive migration.

7 Acknowledgements

We'd like to thank Tom Anderson, John Ousterhout, and Keith Vetter for helpful comments on earlier drafts.

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 Rapael 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.

[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.

[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.

[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, May 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.

[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.

[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.

[Wol89] R. W. Wolff. *Stochastic Modeling and the Theory of Queues*. Prentice Hall, Englewood Cliffs, NJ, 1989.

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

[ZF87] Songnian Zhou and Domenico Ferrari. A measurement study of load balancing performance. In *IEEE 7th International Conference on Distributed Computing Systems*, pages 490–497, October 1987.

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

[ZZWD93] S. Zhou, X. Zheng, J. Wang, and P. Delisle. Utopia: a load-sharing facitlity for large heterogeneous distributed computing systems. *Software – Practice and Expeience*, 23(2):1305–1336, December 1993.