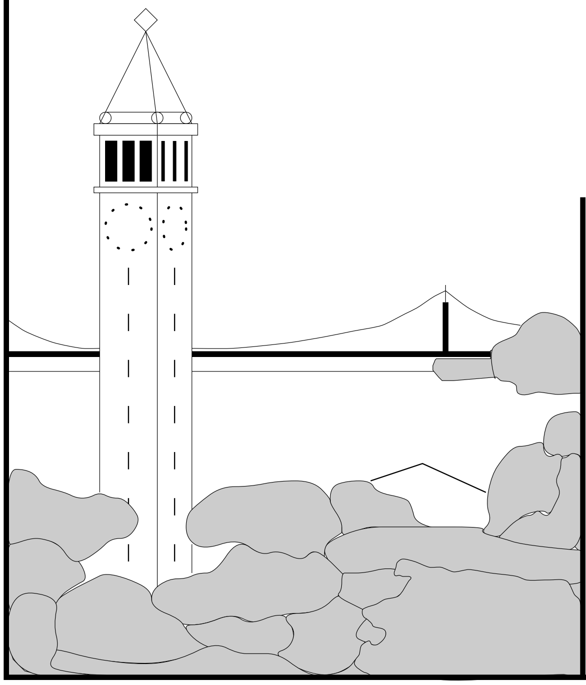
Process Lifetimes are Not Exponential, more like $\frac{1}{T}$: Implications on Dynamic Load Balancing

Mor Harchol-Balter



Report No. UCB/CSD-94-826

August 1994

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

Process Lifetimes are Not Exponential, more like $\frac{1}{T}$: Implications on Dynamic Load Balancing

Mor Harchol-Balter*
Computer Science Division
University of California at Berkeley
Berkeley, CA 94720

August 1994

Abstract

The two decisions defining any dynamic load balancing strategy are:

1. How do we choose a new host (target host) for the migrated process to run on?
2. How do we select which processes to migrate?

Previous load balancing literature has either assumed the process CPU lifetime distribution function is exponential, or has been vague about its properties, and this has greatly influenced the way in which the above two questions have been answered. In this paper we first determine the lifetime distribution function and then *explicitly use it* in our load balancing strategy.

Our measurements of Unix processes show that the lifetime distribution function varies widely among different workloads, however a rule of thumb (which is far more accurate than assuming an exponential distribution) is

$$\Pr[\text{lifetime} > a \mid \text{lifetime} > b\,] \begin{cases} = b/a & \text{if } b \geq 1 \text{ second} \\ > b/a & \text{if } 2^{-6} < b < 1 \text{ second} \end{cases}$$

In particular $\Pr[\text{lifetime} > T \text{ seconds} \mid \text{lifetime} \geq 1 \text{ second}] = \frac{1}{T}$.

In answering question 1 above, we find that measurements of the loads on all potential target hosts become very stale during the time it takes to migrate the process to its new host. Our solution to this problem is: rather than measure the current load on all potential target hosts, we use our lifetime distribution function to compute the *expected future load* (i.e. the load T seconds from now, where T is the time until the migrated process arrives at the new host and is ready to run).

In answering question 2 above, we first analyze previous approaches, for example migrating newborn processes (this solution stems from the misconception that process lifetimes are exponentially distributed). Our lifetime distribution function shows that newborn processes have less than 20% chance of even living for a period equal to the time they spent migrating. We show that we can get away with only migrating processes whose lifetimes exceed four seconds and still have a significant load balancing effect.

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

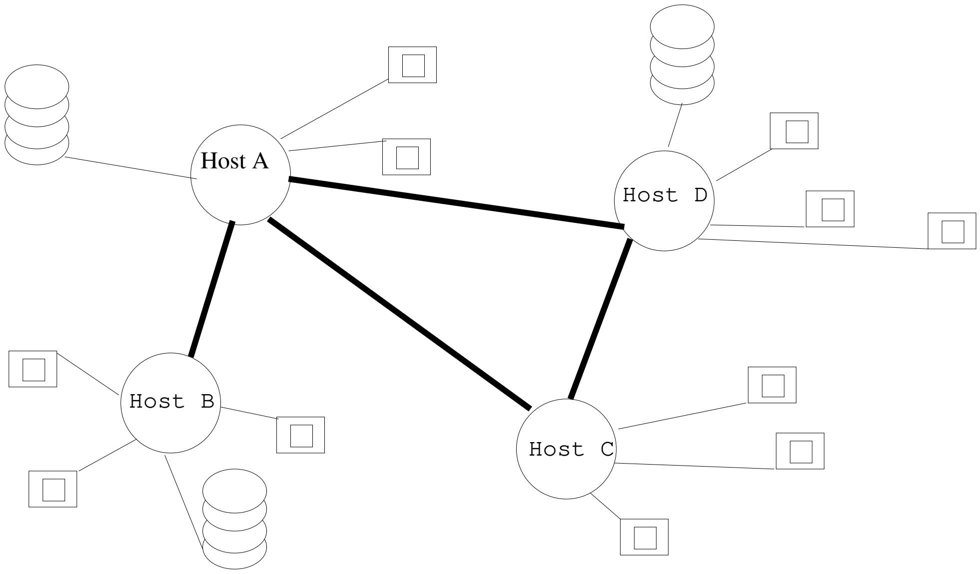


Figure 1: *Network of host machines. Each host consists of a time-sharing processor, some main memory, a few used terminals, and possibly a local disk. At any point in time, several processes may be running on each host machine.*

1 Introduction

1.1 Load balancing

Given a network of time-sharing host machines, as shown in Figure 1, *load balancing* is the idea of migrating processes across the network from a host with a very high workload to a host with a very low workload. The reasons for load balancing include minimizing the average completion time of processes in the network and maximizing the use of the processors in the network. There are two important decisions defining any load balancing strategy:

1. How do we choose a new host for the migrated process to run on?

2. How do we select which processes to migrate?

The first question above is commonly referred to as the *location policy*. We'll refer to the second question as the *migration policy*.

We claim that knowledge of the process cpu lifetime distribution function (namely the probability that a process' cpu lifetime exceeds T seconds) is crucial in answering these two questions.

1.2 Process CPU lifetime distribution

Almost all load balancing literature assumes that process CPU lifetimes have an exponential distribution. See, for example, [BK 90], [EB 93], [CK 68], [ELZ 86], [LR 93], [LM 82], [WM 85]. This belief may stem from [SPG 91]'s statement, "...The duration of CPU bursts have been measured. Although they vary greatly from process to process and computer to computer, they tend to have a frequency curve generally characterized as exponential or hyperexponential[1].

Many of the location policies and migration policies proposed in the literature are based on the assumption that process lifetimes are exponential, and consequently these policies are suboptimal.

We show that the process lifetime distribution function for Unix processes is far from exponential. In fact, for processes which have already used up one second of CPU time, the probability that their lifetime exceeds T seconds is much closer to $\frac{1}{T}$ than e^{-T}. This and other empirical results we obtain on process lifetimes greatly impacts our location and migration policies.

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

Existing migration policies often migrate newly born processes. For example, the migration policies of [HJ 90] and [ELZ 86] only migrate a process if, when born, the load of its host exceeds some threshold. The policy of migrating newborn processes stems from the fallacy that process lifetimes have an exponential distribution, therefore all processes have equal expected remaining lifetimes regardless of their age, so why not migrate newborn processes? Our process lifetime distribution function, however indicates that the longer a process has lived, the longer its expected remaining lifetime. In particular, we show that newborn processes are not worth migrating in the sense that the probability that the migrated process lives for a period equal to its migration time is less than .18.

[BSW 93] and [BF 81] suggest migrating a process only if it has already lived for a period of time equal to its migration time. Our measured process lifetime distribution indicates that under this policy a migrated process has about a $\frac{1}{2}$ chance of living for a period of time equal to its migration time. However is this the best we can do? Are we migrating enough processes to have a significant load balancing effect? Can we afford to be more selective? These are questions that can only be answered by having a complete understanding of the process lifetime distribution. We use this knowledge to develop our own migration policy and to analyze those suggested by others.

1.4 Location Policies: Selecting a new host given a *stale* load index

Previously studied location policies typically involve choosing as new host the host with lowest *load index*, where the load index is some measure of the load at a host as defined by the location policy. By far the most common definition for the load index of a host is the CPU queue length at the host, or the CPU queue length averaged over one second (see for example [Z 87], [HJ 90], [ELZ 86], [BSW 93]). The second most common definition for the load index of a host is the sum of the CPU time used so far by the processes at the host (see for example [HJ 90]).

[1] The hyperexponential distribution is defined as the mixture of two exponentials. The hyperexponential distribution has more variance than the exponential distribution. [W 89]

No previous location policy has addressed the issue of the load index growing stale. We show that the time between which the source host queries its neighbors for their load index and the chosen target host actually receives the migrated process (denoted by the *stale time*) is large with respect to process lifetimes. Suppose for example the load index being used is the CPU queue length. Existing location policies don't take into account the fact that processes die during the stale time, so that by the time the target host actually receives the migrated process it may no longer have anywhere near the lowest load index. Under the assumption of exponentially distributed process lifetimes, this is not a problem because all processes at all hosts are equally likely to die, so the load indices of all the hosts drop at the same rate, so the host with smallest CPU queue length before the stale time is the same as the host with smallest CPU queue length after the stale time. In reality, however, the longer a process has lived so far, the longer it is expected to continue living. Therefore, the relative ranking of load indices can change significantly during the stale time!

To remedy this problem we propose a location policy whereby rather than transmitting its load index, a host transmits what it expects its load index to be T seconds from now, where T is the stale time. We show how our knowledge of the process lifetime distributions can be used to compute the expected future load index for the case where the load index is the CPU queue length, and for the case where the load index is the sum of the CPU time used so far by the processes at the host. We also use the process lifetime distribution to compute the fraction of outside arrivals to the host during the stale time which outlive the stale time, and therefore affect the load index.

1.5 Previous study ignored

After completing our study and writing this paper, we stumbled upon a paper by Leland and Ott which reported doing a study on CPU process lifetimes back in 1984, [LO 86]. Their measurements weren't too different from ours. Specifically, they found that the probability a process' CPU time exceeds x equals $rx^{-c}, 1.05 < c < 1.25$, provided x exceeds 3 seconds. (Our process lifetime measurements are summarized in Section 2.5. They differ from [LO 86] in both c and the 3 second criteria. Also, our study included several other process lifetime measurements which we found relevant to developing load balancing strategies.) Surprisingly, as we've pointed out, this 1984 study seems to have gone unnoticed by future literature on load balancing. Perhaps this is because the load balancing strategy [LO 86] suggested was based on trying to create a spiral arrangement of processes on the hosts, and it therefore did not make use of the exact probability distribution function.

1.6 Organization of this paper

This paper is structured into two halves: The first half (Sections 2 and 3) includes all the facts and formulas we will need for developing our load balancing policy. In the second half (Sections 4 and 5) we formulate our load balancing policy.

In Section 2, we measure the process lifetime distribution, compute the conditional lifetime distribution, and make several empirical observations about process lifetimes. Every fact/formula in Section 2 will be used in developing out load balancing policy, so for easy reference we have summarized Section 2 in Section 2.5 .

In Section 3 we discuss the factors contributing to the cost of migrating a process and argue that the time for migrating a process is high with respect to process lifetimes.

In Section 4 we describe our location policy. We refer to Section 3 in analyzing how stale the load index can get, and then we use the formulas of Section 2 to show how to compute the expected future load index.

In Section 5 we describe our migration policy. We discuss how choosing a process to migrate is a function of both the process' expected remaining lifetime (as described in Section 2) and its migration cost (as described in Section 3) as well as a few other factors. Taking all these factors into consideration, we develop our own migration policy and compare it analytically with migration policies in the literature.

2 Probability distribution function for Unix process lifetimes

In this section we describe how we determined the probability distribution function for Unix process lifetimes, as well as some other empirical observations about Unix process lifetimes. In Table 4 of this section we contrast the Unix process lifetime distribution with the exponential distribution. Because this section contains so many different formulas and observations, we have summarized the results of this section in Section 2.5, and we'll only refer to this list of summarized results in future sections.

2.1 Process lifetime distribution when lifetime > 1 second

We began by measuring process lifetimes on just one machine, "po", a heavily used instructional machine. Measurements were made by turning on Unix accounting on po and using the "lastcomm" command, which outputs the CPU time used by each completed process. Tables 1, 2, and 3 show measurements made on po mid-semester, late-semester, and end-semester, respectively. The first two columns of Tables 1, 2, and 3 indicate the number of processes whose CPU lifetimes exceeded 2^i seconds, for i ranging from -6 to 12. The ith entry in the third column indicates the fraction of processes whose CPU lifetimes exceeded 2^i seconds, which lived on to have a CPU lifetime exceeding 2^{i+1} seconds.

All three tables have one thing in common: **The third column is fairly constant for processes whose CPU lifetimes exceed one second.** (We naturally ignore the last few entries in each table, since they involve too few proceses to hold any statistical significance.) We call this constant the *drop*, and denote it by d. That is

$$d = constant = \frac{\text{no. processes whose lifetime} > 2T}{\text{no. processes whose lifetime} > T}, \text{ for ANY } T > 1 \text{ second}$$

For example, for Table 3, $d \approx 44\%$.

The fact that d is (fairly) constant immediately implies a probability distribution on process lifetimes (as a function of d), as shown in the claim below:

Seconds (T)	No. Processes Live $> T$ Secs	% Processes Which Live Another T Secs	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 1: *Process lifetimes measured on machine po, mid-semester. These measurements correspond to Figure 2.*

Seconds (T)	No. Processes Live $> T$ Secs	% Processes Which Live Another T Secs	Constant Region
0	154368	67%	
2^{-6}	102929	60%	
2^{-5}	61968	79%	
2^{-4}	49568	78%	
2^{-3}	38599	73%	
2^{-2}	28007	72%	
2^{-1}	20222	57%	
2^{0}	11468	44%	$\surd$
2^{1}	5024	38%	$\surd$
2^{2}	1911	39%	$\surd$
2^{3}	740	42%	$\surd$
2^{4}	309	36%	$\surd$
2^{5}	110	37%	$\surd$
2^{6}	41	32%	$\surd$
2^{7}	13	54%	
2^{8}	7	71%	
2^{9}	5	60%	
2^{10}	3	33%	
2^{11}	1		

Table 2: *Process lifetimes measured on machine po, late-semester. These measurements correspond to Figure 3.*

Seconds (T)	No. Processes Live $> T$ Secs	% Processes Which Live Another T Secs	Constant Region
0	111997	66%	
2^{-6}	73721	61%	
2^{-5}	44929	79%	
2^{-4}	35365	76%	
2^{-3}	26949	71%	
2^{-2}	19111	71%	
2^{-1}	13509	56%	
2^{0}	7524	49%	$\surd$
2^{1}	3745	44%	$\surd$
2^{2}	1640	44%	$\surd$
2^{3}	715	42%	$\surd$
2^{4}	297	38%	$\surd$
2^{5}	114	42%	$\surd$
2^{6}	48	35%	$\surd$
2^{7}	17	47%	$\surd$
2^{8}	8	38%	$\surd$
2^{9}	3	67%	
2^{10}	2	100%	
2^{11}	2	50%	
2^{12}	1		

Table 3: *Process lifetimes measured on machine po, end-semester. These measurements correspond to Figure 4.*

Claim 1 *On a machine with drop d,*

$$Pr[\textit{Process lifetime} > T \mid \textit{Process lifetime} > 1] = T^{\lg d}$$

Proof:

$$\begin{aligned} & \Pr[\text{ Process lifetime } > 2T \mid \text{Process lifetime } > T] = d \\ \Longrightarrow\ & \Pr[\text{Process lifetime } > 2T] = d \cdot \Pr[\text{Process lifetime } > T] \end{aligned}$$

Iterating the above recurrence[2] , we have

$$\Pr[\text{Process lifetime } > T] = d^{\lg T} = T^{\lg d}$$

[2] A more formal proof is : Guess that the form of the probability distribution function is

$$\Pr[\text{Process lifetime} > T] = T^k, \text{ for some k}$$

Substituting this guess into the above equation, we get:

$$\begin{aligned} & (2T)^k = d \cdot T^k \\ \Longrightarrow\ & 2^k = d \\ \Longrightarrow\ & k = \lg d \end{aligned}$$

$L \sim$ Process Lifetime Distribution		$L \sim$ Exponentially
drop d	$d = .5$	
$\Pr[L > 2T \mid L > T] = d$	$\Pr[L > 2T \mid L > T] = .5$	$\Pr[L > T+1 \mid L > T] = e^{-1}$
$\Pr[L > T \mid L > 1\ ] = T^{\lg d}$	$\Pr[L > T \mid L > 1\ ] = \frac{1}{T}$	$\Pr[L > T \mid L > 1] = e^{-(T-1)}$
$\Pr[L > T] > \frac{1}{2} d^6 T^{\lg d}, \quad \forall T > 2^{-6}$	$\Pr[L > T] > 2^{-7} \frac{1}{T}, \quad \forall T > 2^{-6}$	$\Pr[L > T] > e^{-T}$

Table 4: *Contrasting Process Lifetime Distribution with Exponential Distribution*

■

So, for example, if $d = 50\%$, we know that the probability of a process having a CPU lifetime of $> T$ seconds, given that it has already used up 1 second of CPU time is $= \frac{1}{T}$.

To emphasize why the exponential distribution is such a poor model for process lifetimes, in Table 4 we contrast the above distribution with the exponential distribution.

Figures 2,3,and 4 are impulse plots of the process lifetime measurements from Tables 1, 2, and 3, respectively. Each plot depicts only processes whose lifetimes exceeded 1 second. The impulse (line) at 2^i seconds indicates the *fraction* of processes we counted whose lifetimes exceeded 2^i seconds. The purpose of these plots is to show how closely our data is approximated by the probability distribution function, $T^{\lg d}$. Observe that in all three plots, the data fits between two curves, both of the form $T^{\lg d}$, having very close values of d.

We next decided to measure process lifetimes on other machines to see how the distribution function varied. Figures 5,6,7,8 show our measurements for four more machines. For these machines too, we found that the drop, d, became fairly constant, but only for processes whose lifetimes exceeded one second, therefore our plots only include processes whose lifetimes exceed one second.

2.2 Measuring the drop, d

Observe that the drop, d, differs for the different workloads. In our study, d ranged from .41 to .7, and therefore $\Pr[\text{Process lifetime} > T \mid \text{lifetime} > 1\ ]$ ranged from $T^{-1.3}$ to $T^{-.52}$. The drop d may be determined for a particular machine directly from the definition, namely,

$$d = \text{constant} = \frac{\text{no. processes whose lifetime} > 2T}{\text{no. processes whose lifetime} > T}, \text{ for ANY } T > 1$$

Obviously, the more processes we look at the more accurate our measure of d.

It's important to note that even if we don't bother estimating d at all, a good rule of thumb is d $= .5$, so $\Pr[\text{lifetime} > T \mid \text{lifetime} > 1\ ] = \frac{1}{T}$. As we've seen from Table 4, this is substantially more accurate than assuming process lifetimes are exponential.

2.3 Measuring process lifetime distribution in general

Now, suppose we want to know the probability distribution on process lifetimes when we're not restricting ourselves to processes whose lifetimes exceed one second.

Let

$$r(T) = \frac{\text{no. processes whose lifetime} > 2T}{\text{no. processes whose lifetime} > T}$$

$r(T)$ is only a constant for $T \geq 1$ second. However, in all the studies we've done, $r(T) > d$, $\forall T > 2^{-6}$ seconds. (Note, we could not make measurements under 2^{-6} seconds using the lastcomm command.) Using this fact, together with the same reasoning as in the proof of Claim 1, we have:

$$\Pr[\text{ lifetime} > a | \text{ lifetime} > b\ ] \begin{cases} = d^{\lg\left(\frac{a}{b}\right)} = \left(\frac{a}{b}\right)^{\lg d} & \text{if } b \geq 1 \\ > d^{\lg\left(\frac{a}{b}\right)} = \left(\frac{a}{b}\right)^{\lg d} & \text{if } 2^{-6} \leq b < 1 \end{cases}$$

(Observe that if we substitute $(b+k)$ in for a in the above formula, we see that the longer a process has already lived, the greater the probability that it will live an additional k seconds.)

The above formula can be used to obtain a lower bound on the general probability distribution as follows:

$$\begin{aligned} \forall T > 2^{-6},\ \Pr[\text{lifetime} > T] &= \Pr[\text{ lifetime} > T | \text{ lifetime} > 2^{-6} \text{ seconds}] \\ &\quad \cdot \Pr[\text{ lifetime} > 2^{-6} \text{ seconds}] \\ &> d^{\lg(T/2^{-6})} \cdot \Pr[\text{ lifetime} > 2^{-6} \text{ seconds}] \end{aligned}$$

Now, in all our studies, $\Pr[\text{ Process lifetime} > 2^{-6} \text{ seconds}] \approx \frac{2}{3}$, and, more specifically, $50\% < \Pr[\text{ Process lifetime} > 2^{-6} \text{ seconds}] < 78\%$ So,

$$\begin{aligned} \forall T > 2^{-6},\ \Pr[\text{Process lifetime} > T] &> d^{\lg(T/2^{6})} \cdot 50\% \\ &= \frac{1}{2} \cdot d^{6} \cdot T^{\lg d} \end{aligned}$$

Once again, a rule of thumb is $d = .5$, so

$$\forall T > 2^{-6},\ \Pr[\text{ Process lifetime} > T \text{ seconds}] > 2^{-7} \cdot \frac{1}{T}$$

To contrast this with the exponential distribution, see Table 4.

2.4 More empirical observations of process lifetimes

We now make a couple more observations from our measurements which we will need when discussing load balancing policies. Firstly, for all machines we studied, $< 1\%$ of the processes accounted for over 50% of the CPU usage. These are the processes whose CPU lifetime exceeds 4 seconds. Secondly, for all machines we studied, $.13 < \Pr[\text{lifetime} > .25 \text{ sec}] < .18$.

2.5 Summary

To summarize, we have seen in this section that:

1. For each machine we studied, we found

$$r(T) = \frac{\text{no. processes whose lifetime} > 2T}{\text{no. processes whose lifetime} > T}$$

   is constant for all $T \geq 1$. We call this constant the machine's *drop* and denote it by d throught this paper. For $2^{-6} \leq T < 1$, we found $r(T) > d$.

2. We found that although d varied from machine to machine (ranging from .41 to .7), the simple rule of thumb, $d = .5$ was still far more accurate than assuming process lifetimes are exponentially distributed.

3. From formula 1 above, we derived the following expressions regarding process lifetimes:

   (a) $\Pr[\text{Process lifetime} > T | \text{ lifetime} > 1] = T^{\lg d} \quad (= \frac{1}{T} \text{ if } d = .5)$

   (b) $\Pr[\text{ lifetime} > a | \text{ lifetime} > b\ ] \begin{cases} = \left(\frac{a}{b}\right)^{\lg d} & \text{if } b \geq 1 \\ > \left(\frac{a}{b}\right)^{\lg d} & \text{if } 2^{-6} \leq b < 1 \end{cases}$

   (c) $\Pr[L > T] > \frac{1}{2} d^6 T^{\lg d}, \quad \forall T > 2^{-6} \quad (= 2^{-7}\frac{1}{T}, \text{ if } d = .5)$

4. By substituting $(b + k)$ in for a in formula 3b above, we see that the longer a process has already lived, the greater the probability that it will live an additional k seconds.

5. For all machines we studied, $< 1\%$ of the processes accounted for over 50% of the CPU usage. These are the processes whose CPU lifetime exceeds 4 seconds.

6. For all machines we studied, $.13 < \Pr[\text{Process lifetime} > .25 \text{ sec}] < .18$.

Throught the rest of this paper, whenever we refer to this section, we will only refer to the above summary list.

3 Cost of migrating a process

In this section we describe the factors contributing to the cost of migrating a process. The relevance of this section will become apparent during our discussion of effective load balancing policies in Sections 4 and 5.

Assume the scenario shown in Figure 1. We have a network of host machines. Each host consists of a time-sharing processor, some main memory, a few user terminals, and possibly a local disk. At any point in time, several processes may be running on each host machine. In this section we will discuss the costs of process migration, as surveyed in [DO 91].

Migrating a process is only complicated because a process has large amounts of state associated with it. The trend in operating systems seems to be towards increasing state size. The largest part of a process' state is usually its virtual memory.

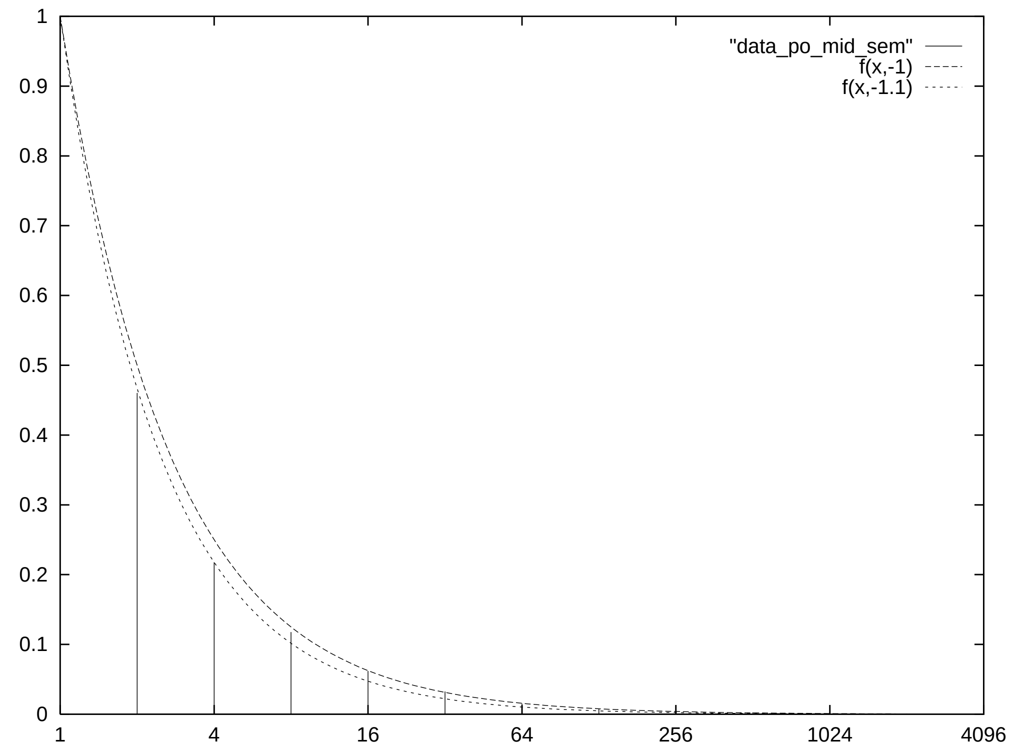


Figure 2: *Process CPU lifetimes, measured on machine "po", measured mid-semester. Po is a very heavily used DECserver5000/240 machine, used primarily for undergraduate coursework. The sample space is the processes whose CPU lifetimes exceeded* 1 *second. Sample space size = 4107. The impulses (lines) indicate our data. The x-axis is measured in seconds. The impulse at* 2^i *seconds indicates the fraction of processes whose CPU lifetimes exceeded* 2^i *seconds. The data is upper-bounded by* $T^{\lg .5} = T^{-1}$ *and lower-bounded by* $T^{\lg .47} = T^{-1.1}$.

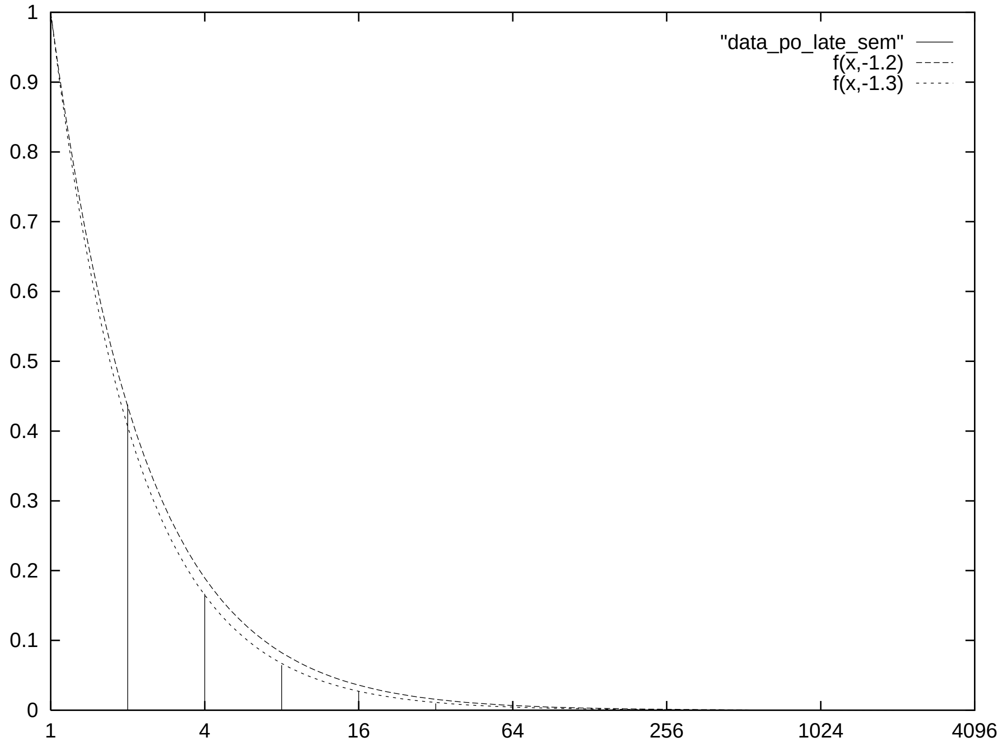


Figure 3: *Process CPU lifetimes, measured on machine "po", measured late-semester. Po is a very heavily used DECserver5000/240 machine, used primarily for undergraduate course-work. The sample space is the processes whose CPU lifetimes exceeded* 1 *second. Sample space size = 11468. The impulses (lines) indicate our data. The x-axis is measured in seconds. The impulse at* 2^i *seconds indicates the fraction of processes whose CPU lifetimes exceeded* 2^i *seconds. The data is upper-bounded by* $T^{\lg .44} = T^{-1.2}$ *and lower-bounded by* $T^{\lg .41} = T^{-1.3}$.

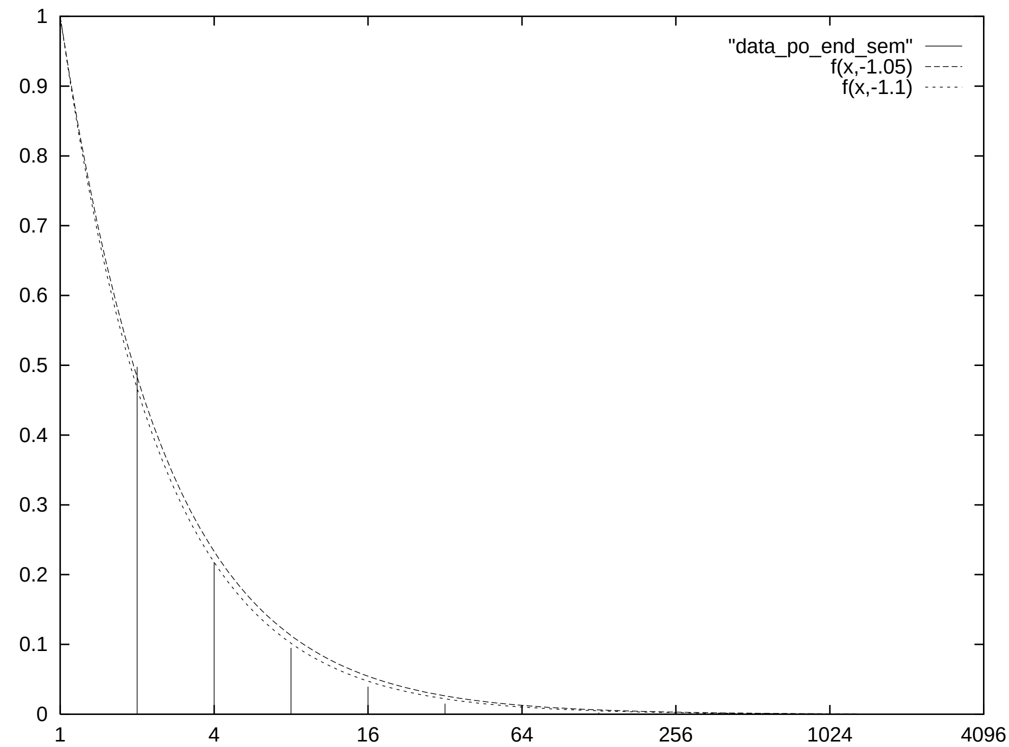


Figure 4: *Process CPU lifetimes, measured on machine "po", measured end-semester. Po is a very heavily used DECserver5000/240 machine, used primarily for undergraduate coursework. The sample space is the processes whose CPU lifetimes exceeded* 1 *second. Sample space size = 7524. The impulses (lines) indicate our data. The x-axis is measured in seconds. The impulse at* 2^i *seconds indicates the fraction of processes whose CPU lifetimes exceeded* 2^i *seconds. The data is upper-bounded by* $T^{\lg .48} = T^{-1.05}$ *and lower-bounded by* $T^{\lg .47} = T^{-1.1}$.

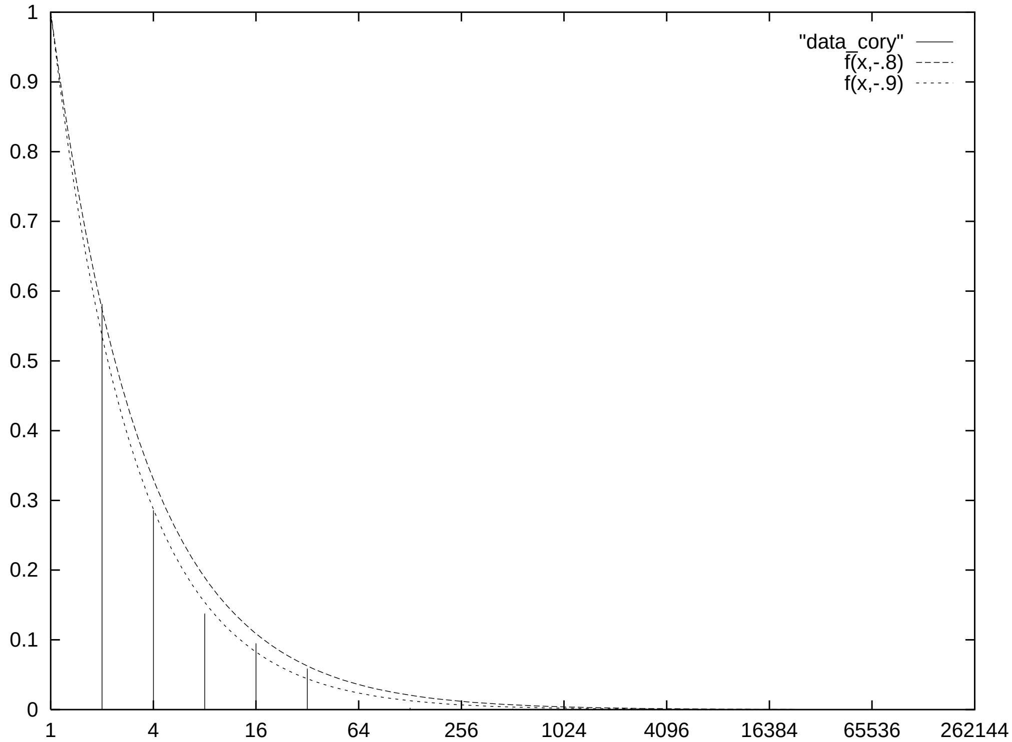


Figure 5: *Process CPU lifetimes, measured on machine "cory." Cory is a very heavily used machine, used primarily for undergraduate coursework. The sample space is the processes whose CPU lifetimes exceeded* 1 *second. Sample space size = 14253. The impulses (lines) indicate our data. The x-axis is measured in seconds. The impulse at* 2^i *seconds indicates the fraction of processes whose CPU lifetimes exceeded* 2^i *seconds. The data is upper-bounded by* $T^{\lg .57} = T^{-.8}$ *and lower-bounded by* $T^{\lg .54} = T^{-.9}$.

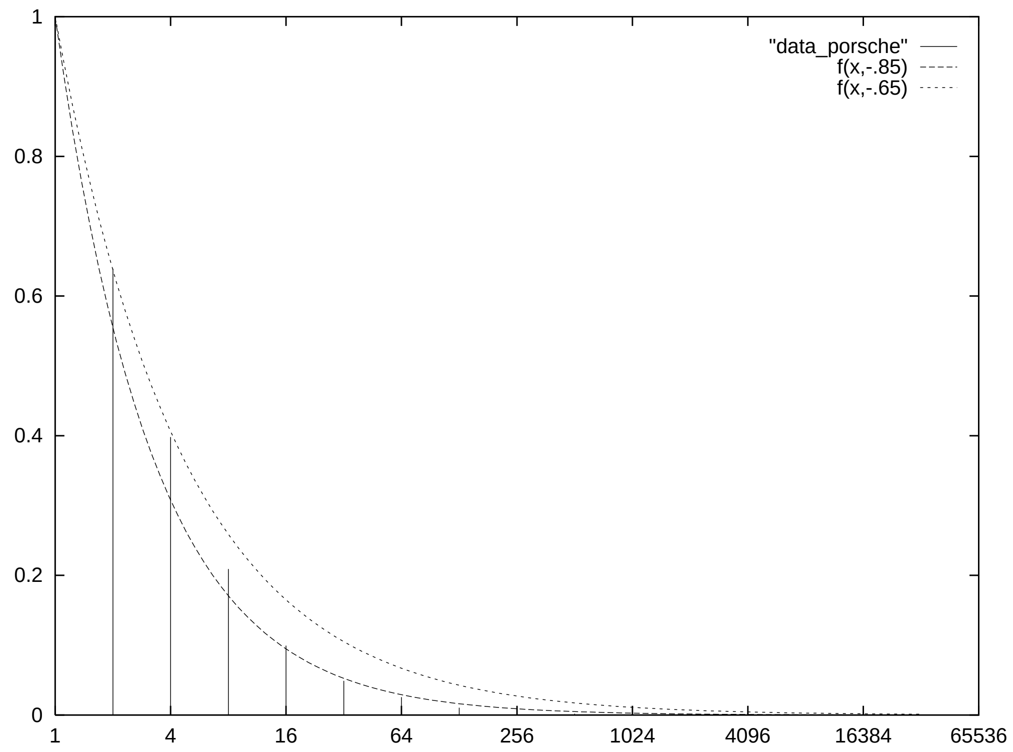


Figure 6: *Process CPU lifetimes, measured on machine "porsche." Porsche is a less-frequently used machine, used primarily for numerical analysis research. The sample space is the processes whose CPU lifetimes exceeded* 1 *second. Sample space size = 10402. The impulses (lines) indicate our data. The x-axis is measured in seconds. The impulse at* 2^i *seconds indicates the fraction of processes whose CPU lifetimes exceeded* 2^i *seconds. The data is upper-bounded by* $T^{\lg .55} = T^{-.85}$ *and lower-bounded by* $T^{\lg .64} = T^{-.65}$.

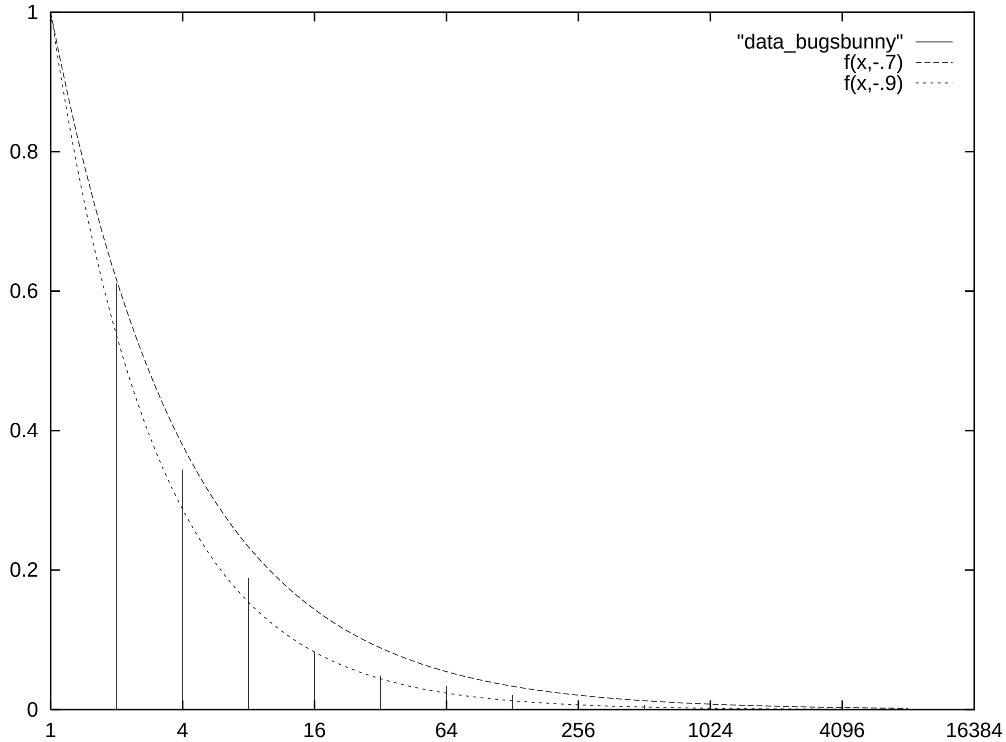


Figure 7: *Process CPU lifetimes, measured on machine "bugsbunny." Bugsbunny is a heavily used machine, used primarily for multimedia research. The sample space is the processes whose CPU lifetimes exceeded* 1 *second. Sample space size = 4940. The impulses (lines) indicate our data. The x-axis is measured in seconds. The impulse at* 2^i *seconds indicates the fraction of processes whose CPU lifetimes exceeded* 2^i *seconds. The data is upper-bounded by* $T^{\lg .62} = T^{-.7}$ *and lower-bounded by* $T^{\lg .54} = T^{-.9}$.

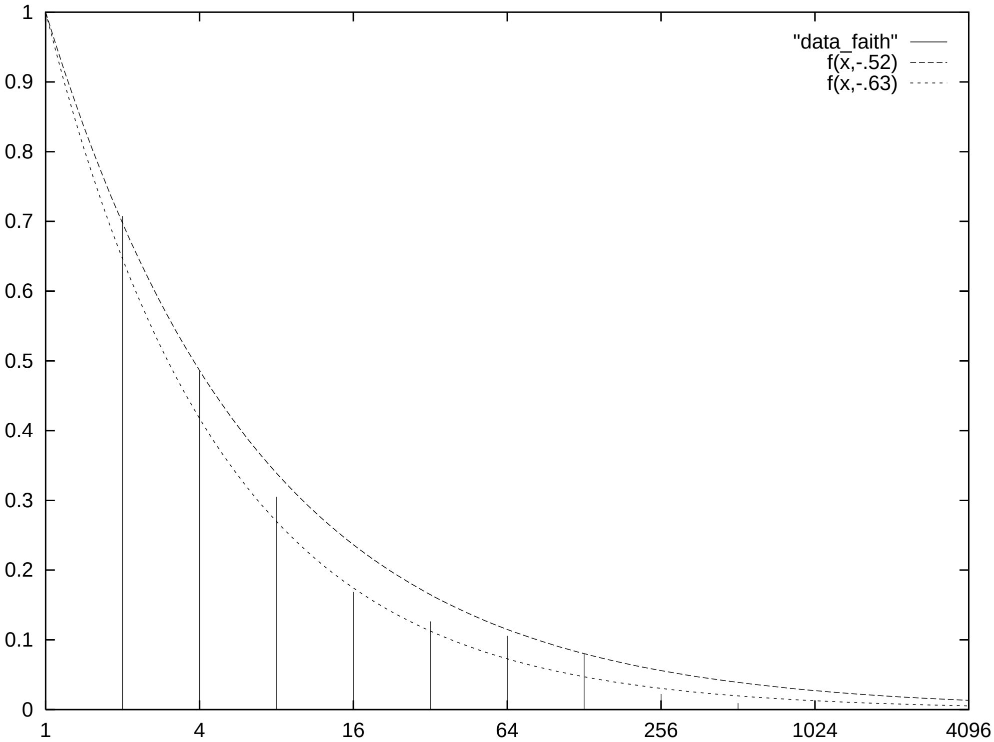


Figure 8: *Process CPU lifetimes, measured on machine "faith." Faith is an infrequently used machine, used for both video applications and system administration. The sample space is the processes whose CPU lifetimes exceeded* 1 *second. Sample space size = 3328. The impulses (lines) indicate our data. The x-axis is measured in seconds. The impulse at* 2^i *seconds indicates the fraction of processes whose CPU lifetimes exceeded* 2^i *seconds. The data is upper-bounded by* $T^{\lg .70} = T^{-.52}$ *and lower-bounded by* $T^{\lg .65} = T^{-.63}$.

The cost of migrating a process can be expressed as the sum of a *fixed migration cost* for migrating all the state, except for the virtual memory, plus a *memory migration cost.* Obviously these costs vary depending on the network. For Sprite, the average fixed cost has been measured at .330 seconds and the memory migration cost is 2 seconds per MByte of VM transfered. We will use these numbers as an example throughout the rest of this paper.

$$\begin{aligned}\text{Migration cost} &= \text{Fixed migration cost} + \text{Memory migration cost} \\ &= .33 \text{ seconds} + 2\text{seconds/MByte of VM transfered}\end{aligned}$$

The fixed cost is incurred at the time of migration. However, there are some options as to *when* the memory transfer cost is incurred, depending on the virtual memory transfer mechanism. Below we list the four common memory transfer mechanisms: (We will not discuss them now, but rather they will come up when we discuss load balancing in the next two sections.)

1. Transfer all the process' memory at migration time. (examples: Charlotte and LOCUS)
2. Precopying: Allow the process to continue executing while its address space is being transfered. (example: V System)
3. Lazy copying: Only copy over the pages as they are referenced. (example: Accent)
4. The file system is distributed. To migrate a process, flush the dirty pages back to the file system. Then migrate the process without its address space. At the target machine, page in the pages as the process needs them. (example: Sprite)

4 Our Location Policy

The purpose of a location policy is to determine a target host for the migrated process. Previously studied location policies typically involve choosing as new host the host with lowest *load index*, where the load index is some measure of the load at a host as defined by the location policy. The most common definitions for the load index of a host are

1. The number of processes at the host waiting for the CPU.
2. The sum of the CPU time used up so far by the processes at the host.

No previous work has addressed the issue of the load index growing stale. Suppose, for example, process p resides on host A, and after A has queried all its neighboring hosts for their load index, A concludes host B has the lowest load index. By the time p is ready to run at B, B may no longer have anywhere near the lowest load index. We denote the time between which A's neighbors send off their load indices and B receives p as the *stale time.*

In Section 4.1, we show that the stale time can be very large, causing the load index to become very out of date as processes complete and new processes are born. To remedy this problem, we propose a location policy where rather than transmitting the load index, *a host transmits what it expects its load index to be T seconds from now*, where T is the stale time associated with the migrated process. In Section 4.2, we show how to compute the expected future load index at a host, where the load index is the CPU queue length. In Section 4.3, we show how to compute the expected future load index at a host, where the load index is the sum of the CPU time used up so far by all processes at the host. Note that our goal is not to decide which load index of all those being used is "best", but rather we show, for whichever load index one wishes to use, how to compute the expected load index T seconds from now.

4.1 Stale time can be very large

In Section 3 we discussed the cost associated with migrating a process p from its source host A to its target host B. The *stale time* differs from the *process migration time* in two ways: Firstly, the stale time also includes the time it takes to transfer the load index information (we call this the *load index transfer time*). Secondly, the stale time only includes the part of the migration time required to *start* process p running on host B (we call this p's *startup time*.

$$\text{stale time} = \text{load index transfer time} \ + \ p\text{'s startup time}$$

To differentiate between p's startup time and p's migration time, consider the four possible virtual memory transfer mechanisms suggested in Section 3: If all p's memory is transfered with p at migration time (option 1), then p's startup time equals p's migration time, namely,

$$\begin{aligned} p\text{'s startup time} &= \text{fixed migration cost} + \text{time to migrate p's VM} \\ &= .33 \text{ sec} + 2 \text{ sec/MByte of } p\text{'s VM} \end{aligned}$$

If precopying is used (option 2), then p's startup time may be even greater since often parts of p's VM will end up having to be copied twice. Option 3, lazy copying, we ignore because it leaves behind too many residual dependencies which will become a huge complexity problem when a process is migrated repeatedly. Option 4, the Sprite migration mechanism, minimizes p's startup time, since it is fastest in getting p running on its new host. In this case,

$$\begin{aligned} p\text{'s startup time} &= \text{fixed migration cost} + \text{time to flush } p\text{'s dirty VM} \\ &= .33 \text{ sec} + 2\text{sec /MByte of } p\text{'s } \textit{dirty} \text{ VM} \end{aligned}$$

Thus we've seen that the stale time is at the very least .33 seconds, and might be much longer. Considering that less than 18% of all processes even live .25 seconds (see formula 6, section 2.5), it's clear that the stale time can have a large effect on the load index. Suppose for example the load index is the CPU queue length. Imagine two hosts B_1 and B_2 with equal CPU queue lengths, except that B_1 has all short jobs and B_2 has all long jobs. From what we've seen in Section 2, after stale time T, B_2's load index will be much higher than B_1's, since B_1's jobs have a much lower probability of living another T seconds than B_2's jobs.

4.2 Computing future load index, when load index = cpu queue length

Suppose our load index is the CPU queue length. Then the expected CPU queue length at a host T seconds from now is:

$$
\begin{aligned}
&E[\text{No. processes at host T secs from now}] = \\
&\quad E[\text{No. processes alive now which live} > T \text{ more secs}] + \qquad (1)\\
&\quad E[\text{No. processes born at host during next } T \text{ secs \& still alive } T \text{ secs from now}] \quad (2)
\end{aligned}
$$

To compute (1) above, let $p_1, p_2, \ldots, p_n$, be the process at the host and suppose that process p_i has already used up t_i seconds of CPU time.

$$
\begin{aligned}
&E[\text{Number of processes alive now which live} > T \text{ more secs}]\\
&= \sum_{i=1}^{n} \Pr[\ p_i \text{ has lifetime} > t_i + T | p_i \text{ has lifetime} > t_i]\\
&> \sum_{i=1}^{n} \left(\frac{t_i+T}{t_i}\right)^{\lg d} \text{(by formula 3b, Section 2.5)}
\end{aligned}
$$

Observe that if we take into account only processes for which $t_i > 1$, then the last inequality is a strict equality. Note that it is not unreasonable to only account for processes for which $t_i > 1$ for two reasons: Firstly, for large T, the term $\left(\frac{t_i+T}{t_i}\right)^{\lg d}$ is only significant when $t_i > 1$, anyway. Secondly, it is far less time-consuming computationally to only account for processes for which $t_i > 1$.

To compute (2) above, we require a measure of λ the average rate of arrival of new jobs to the host from outside the network. Assuming that jobs arrive at the host from outside according to a Poisson process with rate λ, we have

$$
\begin{aligned}
&E[\text{Number outside arrivals during [0,T] which are still alive at time } T]\\
&= \int_{t=0}^{t=T} E[\text{No. outside arrivals during (t,t+dt) which live at least another } T-t]\\
&= \int_{t=0}^{t=T} E[\text{No. outside arrivals during (t,t+dt)}] \cdot \Pr[\text{ newborn lives} > T-t]\\
&= \text{(by formula 3c, section 2.5)} \int_{t=0}^{t=T} (\lambda dt) \cdot (.5d^6(T-t)^{\lg d})\\
&> .5\lambda d^6 \frac{T^{\lg d+1}}{\lg d+1}
\end{aligned}
$$

4.3 Computing future load index, when load index = sum of CPU time

Now suppose our load index is the sum of the CPU time used up by all processes on the host.

E[Total CPU time at host T secs from now] =

$$E[\text{Total CPU time } T \text{ secs from now due to processes alive now}] + \tag{3}$$
$$E[\text{Total CPU time } T \text{ secs from now due to processes born w/i next } T \text{ secs }] \tag{4}$$

To compute (3) above, we use the same notation as in Section 4.2.

$$\begin{aligned}
& E[\text{Total CPU time } T \text{ secs from now due to processes alive now}] \\
&= \sum_{i=1}^{n} \Pr[\ p_i \text{ has lifetime} > t_i + T | p_i \text{ has lifetime} > t_i] \cdot (t_i + T) \\
&> \sum_{i=1}^{n} \left(\frac{t_i + T}{t_i}\right)^{\lg d} \cdot (t_i + T)
\end{aligned}$$

Again, as in Section 4.2, observe that the last inequality is not far from an equality. Also, observe that if $d = .5$, this simplifies to $\sum_{i=1}^{n} t_i$, independent of T. That is , the total CPU time due to processes alive now stays constant, regardless of the distribution of the t_i's.

To compute (4) above, we require a measure of λ the average rate of arrival of new jobs to the host from outside the network. Assuming that jobs arrive at the host from outside according to a Poisson process with rate λ, we have

$$\begin{aligned}
& E[\text{Total CPU time accumulated by arrivals during [0,T] which live to time } T] \\
&= \int_{t=0}^{t=T} E[\text{No. outside arrivals during (t,t+dt) which live another } T - t] \cdot (T - t) \\
&= \int_{t=0}^{t=T} E[\text{No. outside arrivals during (t,t+dt)}] \cdot \Pr[\text{newborn lives} > T - t] \cdot (T - t) \\
&= \text{(by formula 3c, section 2.5)} \int_{t=0}^{t=T} (\lambda dt) \cdot (.5d^6 (T - t)^{\lg d}) \cdot (T - t) \\
&> .5\lambda d^6 \frac{T^{\lg d + 2}}{\lg d + 2}
\end{aligned}$$

5 Our Migration Policy

Our goal in this section is to determine a simple, effective migration policy.

What information do we know about a process that might influence its suitability for migration?

1. CPU time used so far by the process.

2. Process' migration time (The time to migrate a process is defined in Section 3).

3. How interactive the process has been so far

We will ignore (3) for now, and discuss only (1) and (2). We'll come back to (3) at the end of this section.

Ideally, we would like to migrate processes for which (2) is low, so not much time is lost on the migration, and (1) is high, because those processes have the highest expected remaining lifetime (see formula 4, Section 2.5) and therefore stand to obtain the maximum benefit by being move to a new host with a very low workload. The problem is that high (1) usually implies high (2). That is, the longer a process has lived, usually the greater the memory it has accumulated.

If we migrate only processes with high (1) and low (2), we may not end up doing enough migration to have any real load-balancing effect.

One might consider a migration policy which migrates processes with low (1) and low (2). This is a very commonly used policy (see the Introduction), in which only newborn processes are migrated. This policy stems from the belief that process lifetimes are exponentially distributed, namely that the expected remaining CPU lifetime of a process is independent of how much cpu time it has used up so far, therefore why not migrate newborn processes? The flaw is that the process lifetime distribution we saw in Section 2 tells us that these newborn processes actually have the shortest expected remaining lifetimes. In fact, since migration fixed cost is so high (330 ms), the probability that a newborn process even ends up living 330 ms is $< .18$ (see formula 6, Section 2.5), that is, a newborn process has $< .18$ chance of even living as long as it spent migrating.

Fortunately, we've found that by choosing (1) just right, we can satisfy our goal of making the migration time an insignificant part of a process' overall lifetime, but still ensure that we're doing enough migration to have an effect on load-balancing.

Recall in formula 5, Section 2.5 we state that 50% of all CPU time is used up by $< 1\%$ of all processes. These are the processes whose CPU lifetimes exceed 4 seconds. Therefore, by limiting our migration to processes whose CPU lifetimes exceed 4 seconds, we still have a significant load-balancing effect. Furthermore, since the lifetimes of these processes are *so* large, they have a much higher probability of living another T seconds, where T is the migration time. Let's consider a "worst-case" example of the cost of migrating one of these processes (see Section 3). Suppose the virtual memory of the process is large, say 1 MByte, and suppose our memory transfer mechanism requires all the process' VM to be migrated with it. Then, (by Section 3) the cost of migrating this process is 2.33 seconds. Now, let's compute the probability that the process once migrated continues to live for at least as long a period as it spent migrating. Using formula 3b from Section 2.5, we have:

$$\Pr[\text{Process lifetime } > (4 + 2.33) | \text{ lifetime } > 4] = \left(\frac{6.33}{4}\right)^{\lg d} = d^{.66}$$

Clearly if the process we're migrating has lived > 4 seconds or has accumulated less virtual memory or if our memory transfer mechanism only requires migrating dirty pages, then this probability is higher.[3]

[3] It's interesting to contrast our migration policy with that suggested by [BSW 93] (see the Introduction), which migrates only processes whose current lifetime exceeds their estimated migration time. From what we've learned about the process lifetime distribution, [BSW 93]'s policy seems pretty good, since it guarantees probability d that the migrated process continues to live for a period of time equal to its migration time (see formula 1, Section 2.5). The difference between our migration policy and [BSW 93] is that we are being

Lastly, let's discuss (3). We'll assume that if a process originated at host A, then if there is a user interacting with the process, that user is sitting at a terminal at host A (see Figure 1). Under this assumption, we might not want to migrate a process which is expected to interact soon (especially if the interaction involves a lot of data), because that interaction must travel through the network, making it slow. We therefore need to know the probability distribution function on the time until a process' next interaction. We speculate that the time between successive interactions in a process has the same type of distribution as we saw for process CPU lifetimes. That is, *if the time since a process' last interaction* $> T$ *seconds, then with some significantly high constant probability, the time until the process' next interaction will be* $> 2T$ *seconds.* If our hypothesis turns out to be true, then we will be able to use the same distribution function for the time until the next interaction as we used for CPU lifetime (with perhaps a different value for the drop, d). It will then make sense to migrate the process with has run the longest time since its last interaction. (We have not yet tested our hypothesis.)

6 Acknowledgements

I would like to thank Tom Anderson, Allen Downey, John Ousterhout, and Keith Vetter for carefully reading earlier drafts of this paper, and for their useful comments.

References

[BSW 93] Amnon Barak and Guday Shai and Richard G. Wheeler. The MOSIX Distributed Operating System Load Balancing for UNIX. Springer Verlag, Berlin, 1993. pp. 133-168.

[BK 90] Flavio Bonomi and Anurag Kumar. Adaptive Optimal Load Balancing in a Non-homogeneous Multiserver System with a Centra Job Scheduler. *IEEE Transactions on Computers*, vol. 39, no.10, Oct. 1990, pp.1232 - 1250.

[BF 81] Raymond M. Bryant and Raphael A. Finkel. A Stable Distributed Scheduling Algorithm. *2nd International Conference on Distributed Computing Systems.* 1981. pp. 314 - 323.

[CK 68] Edward G. Coffman and Leonard Kleinrock. Feedback Queueing Models for Time-Shared Systems. *Journal of the ACM.* Vol. 15, No. 4, Oct. 1968. pp. 549-576.

[EB 93] D.J. Evans and W.U.N. Butt. Dynamic load balancing using task-transfer probabilities. *Parallel Computing*, vol. 19, August 1993, pp.897-916.

much more selective by limiting our migration to processes which have live > 4 seconds. This ensures us a *higher* probability that the migrated process continues to live for a period of time equal to its migration time (for example, in our "worst-case" example above $d^{.66} > d$.)

[DO 91] Fred Douglis and John Ousterhout. Transparent Process Migration: Design Alternatives and the Sprite Implementation. Software - Practice and Experience, Aug. 1991, vol.21, (no.8):757-85.

[ELZ 86] Derek L. Eager, Edward D. Lazowska, and John Zahorjan. Adaptive Load Sharing in Homogeneous Distributed Systems. *IEEE Transactions on Software Engineering.* vol. se-12, no. 5, May 1986. pp. 662-675.

[LR 93] Hwa-Chun Lin and C.S. Raghavendra. A state-aggregation method for analyzing dynamic load-balancing policies. *IEEE 13th International Conference on Distributed Computing Systems.* Pittsburgh, PA, May 1993, pp. 482-489.

[HJ 90] Anna Hać and Xiaowei Jin. Dynamic Load Balancing in a Distributed System using a Sender-Initiated Algorithm. *Journal of Systems Software.* vol 11, 1990. pp. 79-94.

[LM 82] Miron Livny and Myron Melman. Load balancing in homogeneous broadcast distributed systems. *ACM Computer Network Performance Symposium*, College Park, MD, April 82, pp. 47-55.

[LO 86] W.E. Leland and T.J Ott. Load-balancing heuristics and process behavior. Performance Evaluation Review, May 1986, vol.14, no.1. pp.54-69.

[SPG 91] A. Silberschatz, J.L. Peterson, P.B. Galvin. Operating System Concepts, 3rd edition. Reading, Mass., Addison-Wesley c1991.

[WM 85] Yung-Terng Wang and Robert J.T. Morris. Load Sharing in Distributed Systems. *IEEE Transactions on Computers.* Vol. c-94, no. 3, March 1985. pp. 204-217.

[W 89] R.W. Wolff. Stochastic Modeling and the Theory of Queues. Prentice Hall, Englewood Cliffs, NJ, 1989. p. 266.

[Z 87] Songnian Zhou. Performance studies for dynamic load balancing in distributed systems. (PhD Dissertation. U.C. Berkeley) 1987.