An Analysis of Decay-Usage Scheduling in Multiprocessors

D.H.J. Epema
Department of Mathematics and Computer Science
Delft University of Technology
P.O. Box 356, 2600 AJ Delft, The Netherlands
e-mail: epema@cs.tudelft.nl

Abstract

Priority-ageing or decay-usage scheduling is a time-sharing scheduling policy capable of dealing with a workload of both interactive and batch jobs by decreasing the priority of a job when it acquires CPU time, and by increasing its priority when it does not use the (a) CPU. In this paper we deal with a decay-usage scheduling policy in multiprocessor systems modeled after widely used systems. The priority of a job consists of a base priority and a time-dependent part based on processor usage. Because the priorities in our model are time dependent, a queueing-theoretic analysis, for instance for the mean response time, seems impossible. Still, it turns out that as a consequence of the scheduling policy, the shares of available CPU time obtained by jobs converge, and a deterministic analysis for these shares is feasible: for a fixed set of jobs with very large (infinite) processing demands, we derive the relation between their base priorities and their steady-state shares. In addition, we analyze the relation between the values of the parameters of the scheduler and the level of control it can exercise over the steady-state shares. We validate the model by simulations and by measurements of actual systems.

1 Introduction

Time-sharing systems that support both interactive and batch jobs are steadily becoming of less importance in this age of workstations. Instead, for compute-intensive jobs, the use of multiprocessors and clusters of uniprocessors is increasing. Currently, most of the latter systems run some variation of the UNIX™ operating system, which employs a classical time-sharing policy. In this paper we analyze a scheduling policy in multiprocessor systems modeled after scheduling in UNIX and Mach, – which has a scheduler similar to the one in UNIX –, under a workload consisting of long, compute-intensive jobs.

SIGMETRICS '95, Ottawa, Ontario, Canada
© 1995 ACM 0-89791-695-6/95/0005 ..$3.50

In this scheduling policy, the priority of a job is equal to the sum of its fixed base priority and a time-dependent component. The latter part, the *accumulated and decayed CPU usage*, increases in a linearly proportional fashion with CPU time obtained, and is periodically (at the end of every *decay cycle*) divided by the *decay factor* (a *low* priority corresponds to a *high* level of precedence.) Jobs with equal base priorities belong to the same *class*. Jobs are selected for service in a processor-sharing fashion according to their priorities. Because the priorities are time dependent, a queueing-theoretic analysis of the policy, for instance solving for the mean response time, possibly assuming a Poisson arrival process and an exponential service-time distribution for each class, seems infeasible. However, provided that the set of jobs in the system is fixed and that all jobs are present from the same time onwards, a deterministic analysis *is* feasible.

Our main results are proving the convergence of this scheduling policy in the sense that the fractions of CPU time obtained by jobs (the service fractions or *shares*) have limits (the *steady-state shares*), and deriving the relation between the base priorities and the steady-state shares.

In order to validate our model and to assess the impact of the continuous-time hypothesis without any influence of implementation details of actual systems, we simulate the UNIX scheduler. In general, the results of the simulations agree well with the model when the quantum size is small enough. In addition, we performed measurements on uni- and multiprocessors under UNIX versions based on 4.3 Berkeley Software Distributions (4.3BSD) UNIX, and under Mach. The results for the 4.3BSD uniprocessor agree extremely well with the model, for the 4.3BSD multiprocessor they do not agree, but detailed traces of this system show that its scheduling policy does not comply completely with the model. For the Mach systems, the measurements match the results of the model as we will describe it in Section 2 not very well; however, after adapting the model in an obvious way to include the somewhat different way in which Mach effectuates decay, the measurements match the (adapted) model very well indeed.

In classical time-sharing systems, the two main performance objectives are short response times of interactive work and high throughput of large jobs. For a compute-intensive workload, it is a natural objec-

tive to deliver pre-specified shares of the total compute power to groups of jobs (e.g., all jobs belonging to a single user). Schedulers for uniprocessors with this objective have been treated in the literature under the name of *fair-share schedulers* ([3, 7, 8]), though *share schedulers* would be more appropriate. Our results show how to achieve share scheduling for long, compute-intensive jobs in UNIX systems without scheduler modifications, while in each of [3, 7, 8], an additional term in the priorities of jobs is needed. In [11], a probabilistic share scheduler is presented.

Our analysis extends the analysis by J.L. Hellerstein [5] of scheduling in UNIX System-V uniprocessors and the short treatment by Black [2] of Mach multiprocessors to a unified treatment of scheduling in uni- and multiprocessors running UNIX System V, 4.3BSD UNIX, or Mach. The extension to 4.3BSD and Mach involves more complicated formulas for the priorities; the main difficulty in the extension to multiprocessors is to prove the convergence of the scheduling policy and to determine how many processors serve each of the classes in the steady state. Hellerstein [6] gives a more detailed exposition of the results in [5], and also deals with control issues relating to the range of possible ratios of steady-state shares and the granularity of service-rate control. In [5, 6], measurements are presented for uniprocessors under UNIX System V; also they match the predictions of the model remarkably well.

For lack of space, we cannot go into the convergence rate of our decay-usage scheduling policy. Suffice it to say here that when a system with a fixed set of jobs (as in our model) is considered, or when an arrival or departure occurs, for reasonable numbers of jobs (say less than 10 per processor), we can prove that it takes at most a few tens of seconds, and usually much less, before the jobs start receiving their steady-state shares.

2 The Model

In this section, we describe scheduling in UNIX and in Mach, our model of this type of scheduling, and the differences between them.

2.1 Scheduling in UNIX Systems

Scheduling in UNIX follows a general pattern, but details vary among versions, such as UNIX System V [1], 4.3BSD UNIX [10], and the related operating system Mach [2], which is the basis of the operating system of the Open Software Foundation (OSF)[1]. The description below applies to both uni- and multiprocessors (it does not apply to System V Release 4). UNIX employs a round-robin scheduling algorithm with multilevel feedback and priority ageing. Time is divided into clock ticks, and CPU time is allocated in time quanta of some fixed number of clock ticks; in most current UNIX and Mach systems, a clock tick is 10 ms. When a time quantum expires, the scheduler selects the process at the head of the highest non-empty run queue (lowest value of priority) to run. The set of priorities (typically $0, 1, \ldots, 127$) is partitioned into the kernel-mode priorities (usually 0 through 49) and the user-mode priorities (50 through 127). A job (in UNIX called a process) is represented by a job descriptor, with the following three scheduling-related fields (also for Mach, we use UNIX terminology):

1. *p_nice*: The nice-value, which has range 0 through 19.
2. *p_cpu*: The *accumulated and decayed CPU usage*, which is incremented by 1 for every clock tick received, and is periodically decremented (*priority ageing*, see below).
3. *p_usrpri*: The priority of a job, which depends on *p_nice* and *p_cpu*. A high value corresponds to a low precedence. In the sequel, in the context of UNIX, 'priority' refers to *p_usrpri*.

The fields *p_cpu* and *p_usrpri* of the running process(es) are regularly updated. In many implementations, there are 32 instead of 128 run queues; then, the two least-significant bits of *p_usrpri* are ignored when deciding to which run queue to append a process. The priority of a process is equal to

$$p\_usrpri := PUSER + R \cdot p\_cpu + \gamma \cdot p\_nice, \qquad (1)$$

with $PUSER$ a constant (usually 50) separating kernel-mode priorities from user-mode priorities, and R and γ system parameters. At fixed intervals (a *decay cycle*, usually a second), the following recomputation is performed for every process:

$$p\_cpu := p\_cpu/D + \delta \cdot p\_nice,$$

with $D > 1$ the *decay factor*, and with $\delta \geq 0$, and *p_usrpri* is set according to (1). In System V, $R = 0.5$, $D = 2$, $\gamma = 1$, and $\delta = 0$. In 4.3BSD, $R = 0.25$, $D = (2 \cdot load + 1)/(2 \cdot load)$, $\gamma = 2$, and $\delta = 1$, with *load* equal to the number of jobs in the run queues divided by the number of processors as sampled by the system. In Mach, there is a direct match between priorities, which range from 0 through 31, and the 32 run queues. Here, $PUSER = 12$, and $R = lR'/T$, with l, the *load factor*, defined by $l = \max(1, N/P)$, where N is the number of processes and P the number of processors, with R' the increment of the priority due to one second of received CPU time when $l = 1$ ($R' \approx 3.8$), and with T the number of clock ticks per second. The effect of the load factor in R is to keep the priorities in the same range regardless of the number of processes and processors (see Subsection 4.2, Remark 3). Furthermore, $D = 1.6$, $\gamma = 1$, and $\delta = 0$. The way decay is effectuated in Mach is somewhat different from that in UNIX, but until we discuss Mach measurements in Section 6, we assume Mach operates in the same way as UNIX. In 4.3BSD UNIX ([10], p. 87) and Mach, a time quantum is 100 ms (10 clock ticks), in System V, it is 10 ms (1 tick). Because we will only consider systems in their steady states and with more processes than processors, the definitions of *load* for 4.3BSD and of the load factor l for Mach coincide, and we will in the sequel denote both by l. Currently, $T = 100$ in almost all implementations of these operating systems the author knows of, and throughout this paper we will assume this value for T.

[1] Open Software Foundation, Inc , 11 Cambridge Center, Cambridge, Ma , USA

2.2 The Decay-Usage Scheduling Model

Our model of decay-usage scheduling is defined as follows.

1. There are P processors of equal speed.

2. There are K classes of jobs. We consider the model from time $t = 0$ onwards. There is a fixed set of jobs, all present at $t = 0$; no arrivals or departures occur. There are M_k class-k jobs, $k = 1, \ldots, K$. We write $\hat{M}_k = \sum_{l=1}^{k} M_l$, and assume $\hat{M}_K > P$. Let k_0 be the index such that $\hat{M}_{k_0} \leq P$ and $\hat{M}_{k_0+1} > P$. If $M_1 > P$, then let $k_0 = 0$.

3. A class-k job has *base priority* b_k, with b_k real and non-negative, and $b_k < b_l$ when $k < l$, $k, l = 1, \ldots, K$. The *priority* of a class-k job at time t is

$$q_k(t) = \gamma b_k + R v_k(t), \tag{2}$$

with γ and R positive, real constants, and $v_k(t)$ explained below.

4. Time is divided into intervals of length T, called *decay cycles*, from $t = 0$ onwards. Let $t_n = nT, n = 1, 2, \ldots$. The n-th decay cycle $[t_{n-1}, t_n]$ is denoted by T_n. The scheduling policy is a variation of a policy known as *priority processor sharing* [9] and also as *discriminatory processor sharing* [4], that is, jobs simultaneously progress at possibly different rates (called their *processor shares*), which may change over time. The functions $v_k(\cdot)$ are defined as follows: $v_k(0) = 0$, and if during an interval $[t_1, t_2]$ contained in one decay cycle a job of class k receives a constant fraction $f \leq 1$ of a processor, then

$$v_k(t) = v_k(t_1) + f \cdot (t - t_1), \text{for } t_1 \leq t \leq t_2. \tag{3}$$

Furthermore, at the end of a decay cycle, the following recomputation is performed:

$$v_k(nT) := v_k(nT)/D + \delta b_k, k = 1, \ldots, K, \tag{4}$$

where D and δ are real constants, $D > 1$, $\delta \geq 0$.

5. At any point in time t, the set of jobs that receive service and their processor shares are determined as follows. Order the classes such that $q_{k_i}(t) \leq q_{k_{i+1}}(t), i = 1, \ldots, K-1$, with $\{k_i | i = 1, \ldots, K\} = \{1, 2, \ldots, K\}$. Let r be the lowest index such that $q_{k_r}(t) < q_{k_{r+1}}(t)$, that $\sum_{i=1}^{r} M_{k_i} \leq P$, and that $\sum_{i=1}^{r+1} M_{k_i} > P$, and let s be the index such that $q_{k_{r+1}}(t) = q_{k_s}(t)$ and that $q_{k_s}(t) < q_{k_{s+1}}(t)$. If such an r does not exist, let $r = 0$; if such an s does not exist, let $s = K$. Now at time t, each of the jobs of classes $k_1, \ldots, k_r$ has a dedicated processor, the jobs of classes $k_{r+1}, \ldots, k_s$ evenly share the remaining $P - \sum_{i=1}^{r} M_{k_i}$ processors, and classes $k_{s+1}, \ldots, k_K$ do not receive service.

There are a few things to note about this scheduling policy. Because all processors have equal speeds, any number M of jobs can be given equal shares on any number P' of processors, by means of processor sharing, possibly also *across* processors, when $M \geq P'$, which shows the feasibility of 5. above. Shares only change during a decay cycle when two priorities $q_k(t)$ and $q_l(t)$ become equal. If at time t, jobs (of possibly different classes) have equal priorities, they will receive equal shares at any time during the remainder of the decay cycle, and their priorities remain equal. As a consequence, there are at most $K - 1$ points in a decay cycle where shares change (these are then recomputed according to 5.) Intervals between two consecutive points where this happens are called *epochs*. The l-th epoch of decay cycle n is denoted by $T_n(l)$. For $n \geq 2$ we have

$$q_k(0) + R\delta b_k \sum_{j=0}^{n-2} D^{-j} \leq q_k(t_n^-) \leq$$

$$\leq q_k(0) + RT \sum_{j=0}^{n-1} D^{-j} + R\delta b_k \sum_{j=0}^{n-2} D^{-j}. \tag{5}$$

The lower (upper) bound is only reached when class-k jobs starve (have dedicated processors) during decay cycles $1, \ldots, n$ (to see this, we use (2), (3) with $f = 0$ or 1, and (4)).

The dimension of the parameter T is time, expressed as elapsed time, or as the number of clock ticks delivered by a processor during a decay cycle. This gives us the opportunity to compare the behavior of our model for processors of different speeds by choosing different values for T (different numbers of clock ticks in a decay cycle, each clock tick representing the same amount of useful work). The parameters R and D may depend on other parameters (such as P and the M_k), but should be constant in any instance of the model. In the sequel we will use the terms base priority and nice-value interchangeably.

There are three points where our model differs from real UNIX scheduling (except for the shift in priority by $PUSER$, which is inconsequential), viz.: (1) In our model we use continuous time and a continuous range for the parameters in the scheduler, while actual systems use discrete time and integers; (2) In many UNIX systems, the two least-significant bits of *p_usrpri* are ignored when determining to which run queue to append a process; and (3) The UNIX scheduler uses *priority clamping*: *p_cpu* and *p_usrpri* have maximum values. For instance, in 4.3BSD, *p_cpu* cannot exceed 255 and *p_usrpri* cannot exceed 127 ([10], p. 87). The issue of clamping in the scheduler, in particular of *p_usrpri*, is addressed in [6] (see also Subsection 4.2, Remark 4).

3 The Operation of the Model

In this section, we describe the operation of our decay-usage scheduling model. At the beginning of the first decay cycle T_1, all jobs in classes $1, 2, \ldots, k_0$ get dedicated processors, and if $P > \hat{M}_{k_0}$, the jobs of class $k_0 + 1$ share the remaining processors. The priorities of classes $1, \ldots, k_0$ all increase at the same rate (R), so if $P = \hat{M}_{k_0}$, this operation continues until either $q_{k_0}(t)$ is equal to $q_{k_0+1}(t)$ or until T_1 finishes, whichever occurs first. If $P > \hat{M}_{k_0}$, this operation continues until one of four things happens:

1. The priority of class k_0 becomes equal to the priority of class $k_0 + 1$. Then, the jobs in classes

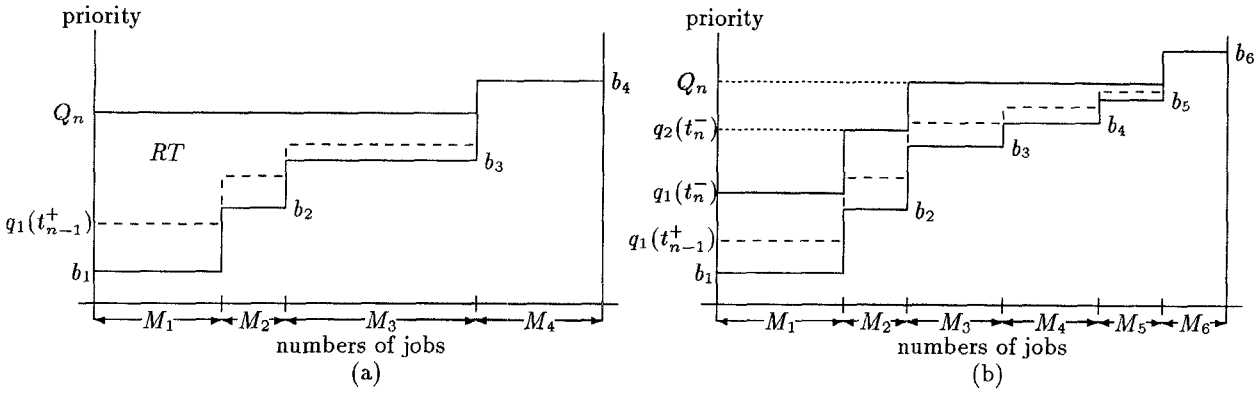


Figure 1: Examples of decay-usage scheduling on (a) a uniprocessor and (b) a multiprocessor.

$1, 2, \ldots, k_0 - 1$ continue having dedicated processors, and the jobs in classes k_0 and $k_0 + 1$ start sharing $P - \hat{M}_{k_0-1}$ processors.

2. The priority of class $k_0 + 1$ becomes equal to the priority of class $k_0 + 2$. In this case, the jobs in classes $1, 2, \ldots, k_0$ continue having dedicated processors, and the jobs in classes $k_0 + 1$ and $k_0 + 2$ start sharing $P - \hat{M}_{k_0}$ processors.

3. The priorities of classes k_0 and $k_0 + 1$ become equal to the priority of class $k_0 + 2$ at the same time. Then, the jobs in classes $1, 2, \ldots, k_0 - 1$ continue having dedicated processors, and the jobs in classes k_0, k_0+1 and k_0+2 start sharing $P-\hat{M}_{k_0-1}$ processors.

4. Before any of 1.-3. happens, T_1 finishes.

Continuing in this way, it is clear that T_1 consists of at most K epochs $T_1(1), T_1(2), \ldots$, with service delivered as follows. During $T_1(l)$, jobs in classes $1, 2, \ldots, i_1(l)$ have dedicated processors, jobs in classes $i_1(l) + 1, \ldots, j_1(l)$ share $P - \hat{M}_{i_1(l)}$ processors, and the remaining classes do not receive service, for some $i_1(l), j_1(l)$. In addition, recalling 1.-4. above, we have $i_1(l+1) = i_1(l)$ or $i_1(l+1) = i_1(l)-1$, and $j_1(l+1) = j_1(l)$ or $j_1(l+1) = j_1(l) + 1$.

We conclude that there are values i_1 and j_1 such that jobs of classes $1, \ldots, i_1$ have dedicated processors during the entire first decay cycle T_1, jobs of classes $i_1 + 1, \ldots, j_1$ receive processor time during T_1 but do not have dedicated processors during at least part of T_1, and jobs of classes $j_1 + 1, \ldots, K$ do not receive service during T_1. Also,

$$q_k(t_1^-) < q_{k+1}(t_1^-), k < i_1, k > j_1, \quad (6)$$

$$q_k(t_1^-) \leq q_{k+1}(t_1^-), k = i_1, j_1, \quad (7)$$

$$q_k(t_1^-) = q_{j_1}(t_1^-), k = i_1 + 1, \ldots, j_1 - 1. \quad (8)$$

It easily follows that $q_k(t_1^+) < q_{k+1}(t_1^+), k = 1, 2, \ldots, K-1$. As a consequence, the operation of the scheduling algorithm during T_2 is analogous, though the starting values of the priorities, the lengths of corresponding epochs, and even the number of epochs may be different. Putting $\beta = \gamma(D-1)/RD + \delta$, in general, we have for $k = 1, \ldots, K$:

$$q_k(t_n^+) = q_k(t_n^-)/D + \beta R b_k, n = 1, 2, \ldots, \quad (9)$$

and so by induction on n, $q_k(t_n^+) < q_{k+1}(t_n^+), k = 1, \ldots, K-1, n \geq 1$, and therefore, the operation of the model in T_n is analogous to that in T_1. We now define i_n as the index such that jobs of classes $1, \ldots, i_n$ have dedicated processors during T_n, and those of class $i_n + 1$ do not (if there are no such classes, we set $i_n = 0$), j_n as the highest index such that jobs of class j_n receive a positive amount of processor time during T_n, and $Q_n = q_{j_n}(t_n^-)$ as the highest priority attained at the end of T_n by a class that receives service. Obviously, $i_n \leq j_n$ and $i_n \leq k_0$. If $\hat{M}_{k_0} = P$, then $j_n \geq k_0$, otherwise $j_n > k_0$.

The operation of the model is illustrated in Figure 1. The dotted lines indicate the priorities $q_k(t_{n-1}^+)$ (we take $\gamma = 1$). In Figure 1 for $P = 1$, class 1 catches up with class 2 at the end of $T_n(1)$, classes 1 and 2 catch up with class 3 at the end of $T_n(2)$, but T_n ends before class 4 gets any service ($i_n = 0, j_n = 3$). In Figure 2 for a multiprocessor, $\hat{M}_2 < P < \hat{M}_5$, and $i_n = 2, j_n = 5$. Note that the area between the graphs of the priorities at the beginning and end of a decay cycle is RPT.

4 Analysis of the Model

In this section we analyze our decay-usage model. First, we derive a set of equations and inequalities, indexed by the decay-cycle number, showing how to compute the shares obtained during T_n and the priorities of all classes at the end of T_n, given the priorities at the start of T_n, for any $n \geq 1$. Because the latter only depend on the priorities at the end of T_{n-1}, this allows us to compute the shares in any decay cycle iteratively.

Next we show that the decay-usage policy converges in the sense that the priorities of all classes at the end of T_n have a limit for $n \to \infty$. It follows that the shares of the jobs of all classes in a decay cycle and the priorities at the start of a decay cycle also have a limit. This result enables us to suppress the decay-cycle index in the set of equations, and solve for the limits of the shares.

4.1 Formulation of the Solution

We introduce the following notation for $k = 1, \ldots, K$:

$$c_k(n) = (q_k(t_n^-) - q_k(t_{n-1}^+))/R, n = 1, 2, \ldots, \quad (10)$$

$$v_k(n) = v_k(t_n^-), n = 1, 2, \ldots. \quad (11)$$

By (2) and (3), $c_k(n)$ is the amount of CPU time obtained by a class-k job during T_n. From (4), (10) and

(11), we have

$$v_k(n) = \frac{v_k(n-1)}{D} + \delta b_k + c_k(n), \quad n = 1, 2, \ldots. \quad (12)$$

From the description of the scheduling policy it is clear that $\{c_1(n), \ldots, c_K(n), i_n, j_n\}$ is the solution of

$$\begin{cases} c_k(n) = T, k = 1, \ldots, i_n, & (13) \\ q_k(t_{n-1}^+) + Rc_k(n) = q_{i_n+1}(t_{n-1}^+) + Rc_{i_n+1}(n), & \\ \qquad k = i_n + 2, \ldots, j_n, & (14) \\ c_k(n) = 0, k = j_n + 1, \ldots, K, & (15) \\ \sum_{k=1}^{K} M_k c_k(n) = PT, & (16) \\ c_{i_n+1}(n) < T, & (17) \\ q_{i_n}(t_{n-1}^+) + RT \le q_{i_n+1}(t_{n-1}^+) + & \\ \qquad + Rc_{i_n+1}(n), & (18) \\ c_{j_n}(n) > 0, & (19) \\ q_{j_n}(t_{n-1}^+) + Rc_{j_n}(n) \le q_{j_n+1}(t_{n-1}^+). & (20) \end{cases}$$

4.2 Convergence

In this subsection we sketch a proof that the decay-usage scheduling policy converges in the sense described in the introduction of Section 4. The main step is to prove that the set of classes with dedicated processors is non-increasing and that the set of classes that receive CPU time is non-decreasing in successive decay cycles, from decay cycle T_2 onwards (Proposition 1, the proof is omitted for lack of space).

Proposition 1. *$i_{n+1} \le i_n$ and $j_{n+1} \ge j_n$, for $n = 2, 3, \ldots$.*

Remark 1. The condition $n \ge 2$ in Proposition 1 is necessary; it is easy to construct a counter example for $n = 1$. One can prove that for System V, 4.3BSD, and Mach, it does hold for $n = 1$.

Corollary. *There exist $N > 0, i_0 \ge 0, j_0 \le K$, such that $i_n = i_0$ and $j_n = j_0$ for all $n \ge N$.*

In fact, from T_N onwards, the scheduling algorithm operates as if the P-way multiprocessor were partitioned in $\hat{M}_{i_0}$ uniprocessors, one for each job of classes $1, \ldots, i_0$, and a $(P - \hat{M}_{i_0})$-way multiprocessor serving classes $i_0 + 1, \ldots, j_0$.

Theorem. *The decay-usage scheduling policy converges in the sense that the limits $c_k = \lim_{n\to\infty} c_k(n)$, $v_k = \lim_{n\to\infty} v_k(n)$, and $q_k = \lim_{n\to\infty} q_k(t_n^-)$ exist. For $k = 1, \ldots, i_0$ we have*

$$q_k = \left(\gamma + \frac{R\delta D}{D-1}\right) b_k + \frac{RTD}{D-1},$$
$$c_k = T, \qquad v_k = \frac{(\delta b_k + T)D}{D-1},$$

for $k = i_0 + 1, \ldots, j_0$ we have (defining $\bar{b} = \sum_{k=i_0+1}^{j_0} M_k b_k / (\hat{M}_{j_0} - \hat{M}_{i_0})$):

$$q_k = \left(\gamma + \frac{R\delta D}{D-1}\right) \bar{b} + \frac{R(P - \hat{M}_{i_0})DT}{(\hat{M}_{j_0} - \hat{M}_{i_0})(D-1)}, \quad (21)$$

$$c_k = \frac{D-1}{RD} q_k - \beta b_k, \qquad v_k = (q_k - \gamma b_k)/R, \quad (22)$$

and for $k = j_0 + 1, \ldots, K$ we have

$$q_k = \left(\gamma + \frac{R\delta D}{D-1}\right) b_k, \quad c_k = 0, \quad v_k = \frac{\delta b_k D}{D-1}.$$

PROOF. Because of Proposition 1 and by the Corollary, from T_N onwards, jobs of classes $1, \ldots, i_0$ always have dedicated processors, the jobs of classes $i_0 + 1, \ldots, j_0$ are jointly served by $P - \hat{M}_{i_0}$ processors and have equal priorities at the end of T_n, for $n \ge N$, and classes $j_0 + 1, \ldots, K$ starve. For the first and last of these three groups of classes, q_k can be computed from (5); for the second, q_k can be computed from (2)-(4). For $k \le i_0$ and $k > j_0$, the value of c_k is obvious and v_k can be found from

$$v_k = \frac{(\delta b_k + c_k)D}{D-1}, \quad (23)$$

which is obtained by taking the limit for $n \to \infty$ in (12). For $k = i_0 + 1, \ldots, j_0$, c_k and v_k can be derived from $q_k = \gamma b_k + Rv_k$ and (23).

Remark 2. For $n \ge N$, $q_k(t_n^-) = Q_n$ for $k = i_0 + 1, \ldots, j_0$, so $q_k(t_n^+) - q_l(t_n^+) = \beta R(b_k - b_l)$ for $i_0 + 1 \le k < l \le j_0$, and so $c_k(n) = c_k$ for $n \ge N+1$. We say that the model is in the *steady state* when the allocation to classes does not change anymore, i.e., once $i_n = i_0$ and $j_n = j_0$. While the limiting priorities of the Theorem are never attained, in the steady state, the shares are equal to their limits.

Remark 3. Assume that $i_0 = 0, j_0 = K$, and let $Q = q_k, k = 1, \ldots, K$. Then by (21), in System V, $Q = \bar{b} + 100/l$, in 4.3BSD, $Q = (9/4 + l/2)\bar{b} + 25/l + 50$, and in Mach, $Q = \bar{b}/2 + R'D/(D-1) \approx \bar{b}/2 + 10$. Note that in Mach, the priority Q is invariant with respect to the load, to P, and to T.

Remark 4. With the help of the Theorem, one can show that the bounds of *p_cpu* and *p_usrpri* are hardly ever reached in System V and Mach. However, in 4.3BSD, the bound of *p_cpu* is easily reached. One can show that then the jobs of classes with low base priorities get larger shares than predicted by the model.

4.3 Steady-State Shares

Assuming the numbers of jobs $M_k, k = 1, \ldots, K$, to be fixed, we now show how to compute the *steady-state shares* $s_k = c_k/PT$, given the base priorities b_k, and conversely, how to compute the base priorities b_k (or rather the differences $b_k - b_1$), given the required shares s_k (or the c_k).

In order to find the c_k, we have to solve for $c_1, \ldots, c_K$, i_0, j_0 the set of equations and inequalities obtained by taking the limit for $n \to \infty$ in (13)-(20). Using (9) (taking the limit for $n \to \infty$) and (23) in (14), (18), and (20), we find

input: $P, T, R, D, \gamma, \delta, K, M_k, b_k, k = 1, 2, \ldots, K$
output: $c_k, k = 1, 2, \ldots, K, i_0, j_0$

s1: $i := k_0 + 1$; **if** $(\hat{M}_{k_0} = P)$ **then** $j := k_0$ **else** $j := k_0 + 1$;

s2: **for** $k := 1$ **to** k_0 **do** $c_k := T$ **od**;

s3: **for** $k := j + 1$ **to** K **do** $c_k := 0$ **od**;

s4: **do**

s5: $i := i - 1; j := j - 1$

s6: **do**

s7: $j := j + 1$

s8: Solve for $c_k, k = i+1, \ldots, j$, the following set of equations:

$$\begin{cases} c_k = c_{i+1} - \beta(b_k - b_{i+1}), k = i+2, \ldots, j & (32) \\ \sum_{k=i+1}^{j} M_k c_k = (P - \hat{M}_i)T & (33) \end{cases}$$

until ($(j = K)$ **or** $(c_j \leq \beta(b_{j+1} - b_j))$)

until ($(i = 0)$ **or** $(c_{i+1} \geq T + \beta(b_i - b_{i+1}))$)

s9: $i_0 = i; j_0 = j$

Figure 2: Algorithm for the computation of the steady-state shares.

$$\begin{cases} c_k = T, k = 1, \ldots, i_0, & (24) \\ c_k = c_{i_0+1} - \beta(b_k - b_{i_0+1}), \\ \quad k = i_0 + 2, \ldots, j_0, & (25) \\ c_k = 0, k = j_0 + 1, \ldots, K, & (26) \\ \sum_{k=1}^{K} M_k c_k = PT, & (27) \\ c_{i_0+1} < T, & (28) \\ c_{i_0+1} \geq T + \beta(b_{i_0} - b_{i_0+1}), & (29) \\ c_{j_0} > 0, & (30) \\ c_{j_0} \leq \beta(b_{j_0+1} - b_{j_0}). & (31) \end{cases}$$

It seems that there is no closed-form expression for the c_k, but they can be computed by the algorithm in Figure 2. We start by assuming that the jobs in classes $1, \ldots, k_0$ have dedicated processors throughout a decay cycle in the steady state (step s2), and that if $\hat{M}_{k_0} = P$ ($\hat{M}_{k_0} < P$), classes $k_0 + 1, \ldots, K$ ($k_0 + 2, \ldots, K$) starve (steps s1 and s3). Whenever step s8 is executed, i and j indicate the highest-numbered class that is assumed to have dedicated processors, and the highest-numbered class that is assumed to receive service, respectively. In step s8, we solve the linear system consisting of Equations (24)-(27), which can be rewritten as in (32) and (33).

Proposition 2. (a) *The amounts $c_k, k = 1, \ldots, K$, of CPU time and the class indices i_0 and j_0 are correctly computed by the algorithm in Figure 2.*
(b) *For $k = i_0 + 1, \ldots, j_0$, we have*

$$\frac{s_k}{s_l} = \frac{(P - \hat{M}_{i_0})T + \beta \sum_{i=i_0+1}^{j_0} M_i(b_i - b_k)}{(P - \hat{M}_{i_0})T + \beta \sum_{i=i_0+1}^{j_0} M_i(b_i - b_l)}. \quad (34)$$

PROOF. (a) Clearly, the solution produced by the algorithm satisfies Equations (24)-(27), and (29) and (31). One can show that $((c_{i+1} < T)$ **and** $(c_j > 0))$ is an invariant of the algorithm.
(b) Solving the linear system in step s8 of the algorithm by substituting (32) in (33) yields (34).

Conversely, one may want to set the c_k, the s_k, or the total shares $M_k s_k$ of classes, and compute a suitable set of base priorities. Obviously, we can then assume $T > c_1 > \cdots > c_K > 0$ (or $1/P > s_1 > \cdots > s_K > 0$), so $i_0 = 0, j_0 = K$, and $\sum_{k=1}^{K} M_k c_k = PT$ (or $\sum_{k=1}^{K} M_k s_k = 1$). Then, inverting (25), we find

$$b_k - b_1 = (c_1 - c_k)/\beta, k = 2, \ldots, K, \quad (35)$$

which for $P = 1$ coincides with Eq. 20 of [6].

In [6], the behavior of the decay-usage policy (for a uniprocessor) is also analyzed in the *underloaded case*, characterized by $\sum M_k c_k < PT$, and the *overloaded case*, defined by $\sum M_k c_k > PT$, with $c_k, k = 1, \ldots, K$, the *required* amounts of CPU time in a decay cycle. It is shown that in either case, $s'_k - s_k = s'_1 - s_1, k = 1, \ldots, K$, where the s_k, s'_k denote the *required* and the *obtained* steady-state shares, respectively, provided that no starvation occurs in the overloaded case, i.e., that $s'_K > 0$. Measurements in [6] show that the policy indeed behaves in this way in practice. This property of equal differences between required and obtained shares clearly carries over to multiprocessors in those cases when no class has dedicated processors and no class starves: the underloaded and overloaded cases correspond to lengthening or shortening the decay cycle, which in general amounts to lengthening or shortening the last epoch, in which all jobs of all classes evenly share all processors. So the decay-usage scheduling policy is *fair* in the sense that an excess or deficit of capacity is spread equally over all jobs. It would perhaps be a more desirable, and fairer, policy if it enjoyed the property that $s'_k/s_k = s'_1/s_1, k = 1, \ldots, K$.

5 Control Considerations

In this section we deal with the level of control that can be exercised by the decay-usage scheduling policy over the share ratios.

5.1 Uniprocessors versus Multiprocessors

Consider the share ratios given by (34). An important thing to note is that it is immaterial whether there are P processors of equal speed (each delivering T clock ticks per decay cycle), or one processor which is P times as fast (delivering PT clock ticks per decay cycle): in either case, an amount PT of processor capacity is delivered in one decay cycle, and the steady-state shares are equal.

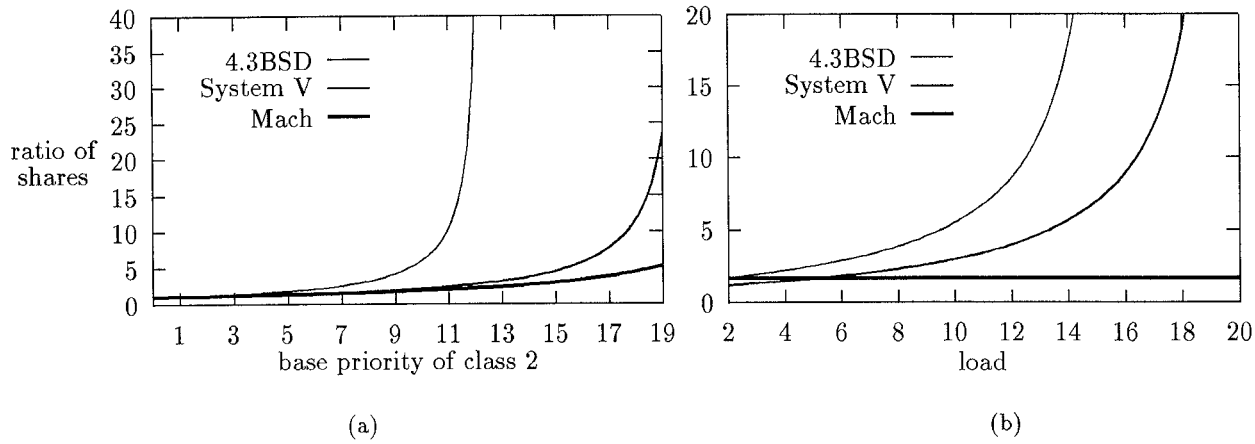


Figure 3: Ratio of shares (s_1/s_2) versus (a) the base priority of class 2 (b_2) and (b) the load (l) for different parameterizations of the decay-usage scheduler ($K = 2, b_1 = 0$; in (a), $M_1 = 5P, M_2 = P$; in (b), $M_1 = M_2 = MP, l = 2M, b_2 = 10$).

5.2 Scheduler Parameterizations and the Range of Share Ratios

In this subsection we will trace the impact of the values of the parameters in the scheduler on the share ratios given by (34). Let us first assume that R and D are constants. Then, as far as the steady-state shares are concerned, $P, T, R, D, \gamma, \delta$ are not independent: only the value of β/PT is relevant. The larger the value of β/PT, the larger the range of possible share ratios. In fact, $\beta = \gamma(D-1)/RD + \delta$ can take any positive value, and there is no loss of control by taking $D = 2, \gamma = 1, \delta = 0$ (these are the values in System V, which has $R = 0.5$, and so $\beta = 1$). So, assuming P and T to be fixed, the scheduler could be parameterized by one instead of four parameters. Also, in order to have the same range for the ratios of steady-state shares, the range of base priorities has to be proportional to both the number of processors P and the processor speed T. In multiprocessors, one may have the opportunity to partition logically the system into a set of multiprocessors with a smaller number of processors each, for instance, in order to reduce contention for the central run queue or so as to assign parts of the machine to different applications. Here we see that in a partitioned multiprocessor, one can exercise more control over the steady-state shares when the ranges of base priorities are the same.

In 4.3BSD and Mach, β depends on the other parameters. In 4.3BSD, $\beta = (2l+9)/(2l+1)$, so β varies from 11/3 when $l = 1$ to 1 when the load is very high. In Mach, $R = \hat{M}_K R'/PT, D = 1.6, \gamma = 0.5$, and $\delta = 0$, so $\beta = 3PT/16\hat{M}_K R' \approx 5/l$, and by (34),

$$\frac{s_k}{s_l} = \frac{16\hat{M}_K R' + 3\sum_{i=1}^{K} M_i(b_i - b_k)}{16\hat{M}_K R' + 3\sum_{i=1}^{K} M_i(b_i - b_l)}, k, l = 1, 2, .., \quad (36)$$

invariant with respect to P and T, so the range of base priorities does not have to be adjusted for multiprocessors or for different speeds. Note that in Mach, the ratios s_k/s_l only depend on the *ratios* M_k/M_l (and the base priorities). For the case of two classes, the levels of control for the three systems are depicted in Figure 3 for varying base priority of class 2 and for varying load. In either case, 4.3BSD has the highest level of control, and Mach the lowest.

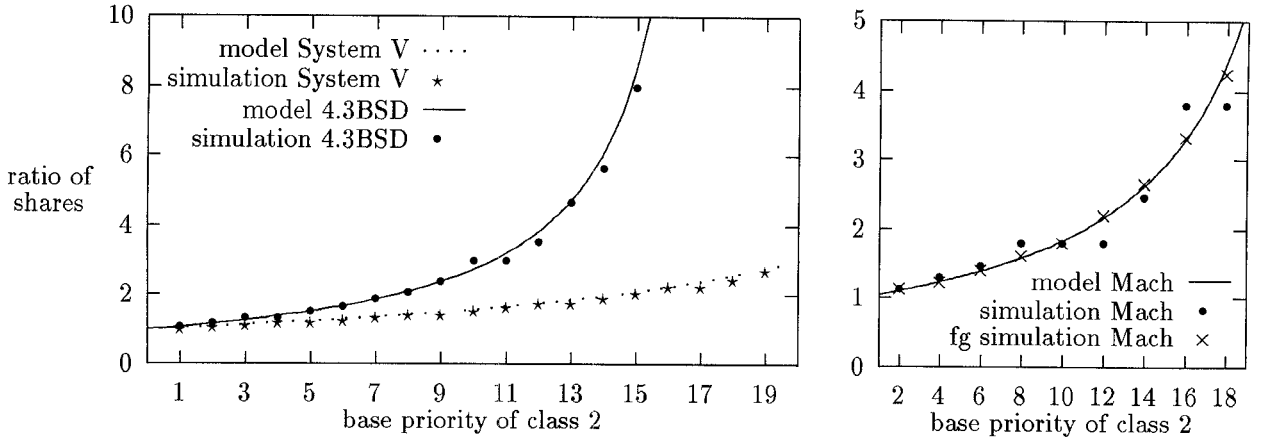
6 Validation: Simulation and Measurements

In order validate our model, we have done measurements on UNIX and Mach systems. In addition, in order to exclude any effects of implementation details of such systems so that we might isolate the influence of discrete time versus continuous time, we have built a simple simulator which mimics the UNIX scheduler. We will compare the ratios of shares s_k/s_l as predicted by our model, as obtained from the simulations, and as measured in actual systems. For the model, the ratios of shares are computed by means of a program implementing the algorithm in Figure 2. Measurements were taken on 1- and 4-processor Sun Sparcstations running SunOS 4.1.3, on a 4-processor Sequent running DYNIX version V3.0.17.9, – all based on 4.3BSD –, and on a Sequent running Mach 3.0 with 1,4,8 or 16 CPUs enabled. We found that the 4-processor Sun uses $l = \hat{M}_K$ (i.e., *not* divided by $P = 4$); it can easily be shown with (22) that then the bound 255 of *p_cpu* for class 1 is reached for *any* set of jobs. As a consequence, measurements on the 4-processor Sun are worthless for our purposes.

Also in DYNIX did we find a deviation from the scheduling policy, and the way Mach effectuates decay deviates somewhat from the description in Section 2. We will discuss the details and the consequences of these differences between the model and these systems below. Because $\gamma = 0.5$ in Mach, we only consider even values of base priorities in that system.

For the measurements, the jobs ran for (at least) 20 minutes in each experiment, and the ratios s_k/s_l are computed as the quotient of the average amounts of CPU time obtained during the last 10 minutes by jobs of classes k and l, respectively. The reason for having the jobs first run 10 minutes before actually measuring is to let the system build up the load l.

In the simulations, the simulated time in each experiment was also 20 minutes, and the ratios of shares were computed in the same way as for the measurements. In *coarse-grained* (cg) simulations, we used the



(a) (b)

Figure 4: Ratio of shares (s_1/s_2) versus the base priority of class 2 (b_2) ($K = 2, M_2 = P, b_1 = 0$; in (a), $P = 1, 4, M_1 = 3P$; in (b), $P = 1, 4, 8, 16, M_1 = 5P$).

values of γ, δ given in Section 2, put $T = 100$, and let the number of clock ticks per quantum be 1 (System V) or 10 (4.3BSD and Mach). It turned out that the results of such simulations can deviate greatly from the model output. Therefore, in some cases we also ran *fine-grained* (fg) simulations, in which $T = 1000$, γ, δ, and for Mach R', were replaced by 10γ, 10δ, and $10R'$, respectively, and the number of clock ticks per quantum was always 10; by (34), this does not change the ratios of shares, but continuous time is approximated more closely. Simulations were coarse-grained, unless otherwise stated.

In our model, the ratios of steady-state shares s_k/s_l are realized during *every single decay cycle*. Clearly, if a steady state is reached in a simulation, in the sense that at the start of successive decay cycles all priorities have the same values and the jobs have the same order in the run queues, there is only a very restricted set of possible share ratios, and *we can expect large deviations from the model.*

In both the simulations and the measurements, there was little variability in the share ratios, so we simply always report the results of one simulation run or one measurement. Also the amounts of CPU time obtained by single jobs within a class in one experiment did not vary much.

6.1 The Model versus Simulation

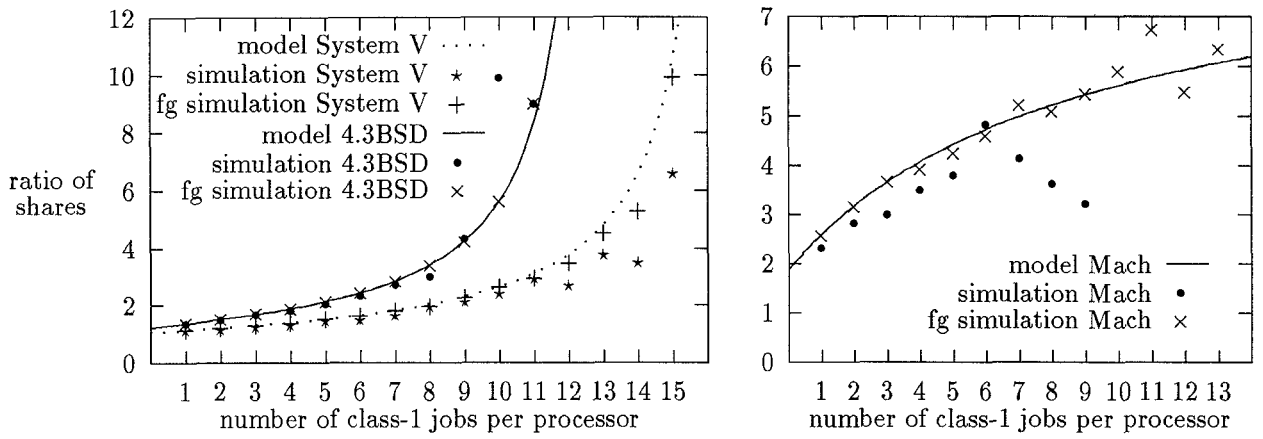
We compare the model output with simulation results for three representative sets of experiments.

I) *Two classes, fixed numbers of jobs, increasing base priority of class 2 (Figure 4).* It turns out that for all three systems, the simulation output is identical for the shown values of P (as is of course the model output). The model output and the simulation results match quite well. Furthermore, 4.3BSD discriminates much more between jobs with the same difference in base priorities than System V does, as was to be expected (cf. Section 5.2). For Mach, the (cg) simulations deviated somewhat from the model, so we also ran fg simulations, which match the model better.

II) *Two classes, fixed base priorities, increasing number of jobs with the lowest base priority (Figure 5).* The deviation between the model and the (cg) simulations for some values of M_1/P is considerable, and is caused by the discrete values in the scheduler, as we found from simulation traces. For instance, for System V, when $P = 1, M_1 = 14, M_2 = 1$, the model gives $s_1/s_2 = 6.63$ and the simulation gives $s_1/s_2 = 3.50$, and it turns out that in the simulation a steady state is reached in the sense described above, and that the class-1 jobs obtain 7 clock ticks in a decay cycle, and the class-2 job gets 2 ticks (one of which is the last one in the decay cycle). For 4.3BSD, it is even more difficult to achieve the ratios predicted by the model because there are only 10 quanta per second, so it is not strange that for $M_1/P = 10$, the (cg) simulation is far from the model. Note that in the fg simulations, which approximate the model much more closely, in 4.3BSD the number of quanta per second is 100 instead of 10, but for System V it remains 100; in the latter system, the better approximation of the model is solely due to a finer distinction in priorities.

For Mach, the fg simulations are close to the model, but the cg simulations are not (when no point is depicted for cg simulations, starvation of class 2 occurs). Traces of the cg simulations showed that for $M_1/P = 6, 7, 8, 9$, the class-2 jobs get almost exactly the same amount of CPU time: for $r = M_1/M_2 = 7, 8, 9$, the share ratios are virtually equal to $6/r$ times the ratio for $M_1/M_2 = 6$. What happens is that for each of these values of M_1/P, the following sequence of events during a decay cycle occurs on the average the same number of times: the class-1 jobs make their way to the queue with the class-2 jobs by receiving a quantum, and then the class-2 jobs get a quantum before the decay cycle finishes. For $M_1/P \geq 10$, in every decay cycle, each of the class-1 jobs gets a quantum before it reaches the queue with the class-2 jobs, and so the latter starve.

III) *Three classes.*
Table 1 shows the ratios of shares and the limiting priority Q at the end of a decay cycle (in the simulations taken as the average priority of all jobs at the end of the last decay cycle in the simulation, and including



(a) (b)

Figure 5: Ratio of shares versus (s_1/s_2) versus the number of class-1 jobs *per processor* (M_1/P) $(K = 2, M_2 = P, b_1 = 0$; in (a), $P = 1, 4, b_2 = 6$; in (b), $P = 1, 4, 8, 16, b_2 = 18)$.

$PUSER = 50$) for a set of experiments with three classes on a uniprocessor with 4.3BSD parameters. The simulations with four classes on a four-way multiprocessor that we carried out, produced results with an equally small error margin, except when the share ratios became very large (higher than 20).

6.2 Model Adaptation for Mach

In the next subsection we will find that especially when there are many jobs (per processor), the Mach measurements may deviate considerably from the model output. We now argue that this is probably due to a combination of the way decay is effectuated in Mach, which is somewhat different from that described in Section 2, and the large quantum size. Mach maintains a global 1-second counter C, and for each process p, a local decay counter C_p. Whenever at a clock interrupt at a processor, the running process p finds $C_p \neq C$, it performs decay with decay factor D^{C-C_p} (let's call this local decay) and sets C_p equal to C. At the end of every 2-second interval, decay is performed for every process in the same way (call this global decay, and call such a 2-second interval an extended decay cycle). The effect is that during the first half of an extended decay cycle, local decay does not occur, and that when a job is selected for execution for the first time during the second half of an extended decay cycle, it is chosen based on its priority *to which no (local or global) decay has been applied yet* during the current extended decay cycle. In addition, if a job gets an amount c of CPU time during either half of an extended decay cycle and nothing during the other half (which happens frequently when there are many jobs), then, denoting the decayed CPU usage of the job at the end of an extended decay cycle by v, we have $v = v/D^2 + c$, so the job experiences a decay factor of D^2 and a decay-cycle length of $2T$.

We conclude that Mach is much more closely modeled with a decay factor $D = 2.56$, equal to the square of the original one, and with a decay cycle of $T = 200$ clock ticks. However, as the increment in priority due to one second of CPU time is built into the system, it remains the same, and we still have $R = lR'/100$. We refer to the model of Mach with these values for D, T, and R as the *adapted model*, the share ratios for which can still be found with (34). We included the way in which Mach effectuates decay in our simulator, and found that such *coarse-grained* simulations proved to be rather close to the *adapted model*, while such *fine-grained* simulations were close to the *original* model.

Two other possible causes for deviations between the model and the measurements are that in our experiments (1) the reported *load* is usually higher than that caused by our jobs, and (2) that the percentage of CPU time obtained by our jobs is about 75-85%, 92-95%, 95%, and 97% for $P = 1, 4, 8, 16$, respectively. In one set of experiments below, we also depict the impact of these two phenomena (labeled *adapted model (measured load and percentage CPU)*), which turns out to be small.

6.3 Measurements

We compare measurements of 4.3BSD-based UNIX systems and Mach systems with model output for two representative sets of experiments. For System V, measurements of uniprocessors were reported in [5, 6]. It can be shown with (21) and (22) that in all experiments reported below, unless explicitly stated otherwise, the model is valid, i.e., that the bounds of *p_cpu* and *p_usrpri* in the scheduler are not reached.

I) *Two classes, fixed base priorities, increasing number of jobs with the lowest base priority (Figures 6 and 7).* On the 4.3BSD-based systems, for $M_1 = P$, the model is invalid, and indeed, measurement and model differ, at least for $P = 1$. For the Sequent under DYNIX, the measurements show larger share ratios than the model. Traces of this system revealed that jobs may get longer time slices than 100 ms, even up to 600 ms, while their priorities do not justify this. Because high-priority processes (i.e., with low base priorities) are eligible for running earlier during a decay cycle, this favors the lower-numbered classes. We have observed the

M_1	M_2	M_3	s_1/s_2		s_1/s_3		Q	
			model	simulation	model	simulation	model	simulation
1	1	1	1.58	1.58	3.75	3.71	142.1	140.7
1	1	2	1.58	1.53	3.78	3.83	154.1	152.5
1	2	1	1.68	1.67	5.25	5.00	144.5	142.8
1	2	2	1.67	1.68	5.11	5.00	156.3	154.6
2	1	1	1.82	1.85	10.07	9.25	134.9	134.0
2	1	2	1.78	1.78	7.98	8.01	147.7	145.8
2	2	1	1.92	1.92	24.11	22.59	139.2	137.2
2	2	2	1.87	1.87	14.66	14.04	151.4	149.7

Table 1: Ratios of shares and the limiting priority Q (4.3BSD, $P = 1, K = 3, b_1 = 0, b_2 = 9, b_3 = 18$).

same phenomenon - higher share ratios - in some measurements of the same Sequent/DYNIX system with one CPU enabled, and we conclude that the deviation is not a multiprocessor issue.

On Mach, the match between the adapted model and the measurements is only reasonable. For this set of experiments, the impact of adapting the model is considerable (compare Figures 5.(b) and 7).

II) *Three classes (Figures 8 and 9).*
On the Sun, in Figure 8.(b), the model is invalid for $M_1 \leq 2$.

7 Conclusions

We have analyzed a decay-usage scheduling policy for multiprocessors, modeled after different variations of UNIX and Mach. Our main results are the convergence of the policy and the relation between the base priorities and the steady-state shares. Our simulations validate our analysis, but also show that discrete time may be a source of considerable deviation from the model. The measurements of the 4.3BSD uniprocessor and of Mach match the model remarkably well, those of the 4.3BSD multiprocessor do less so, which is probably caused by an implementation detail we do not know of.

We have shown that share scheduling can be achieved in UNIX, but unfortunately, in the decay-usage scheduling policy we have analyzed, the shares depend on the numbers of jobs in the classes in an intricate way. We have seen that in order to have the same range of share ratios in a system that does not employ the Mach load-factor technique, the range of base priorities should be proportional to both the number and the speed of the processors, or in other words, the leverage of decay-usage scheduling with the same range of base priorities is much larger in small or slow multiprocessors than in large or fast ones. Also, of the systems considered, 4.3BSD has the highest level of control over the share ratios, and Mach the lowest. Finally, a scheduler with a constant decay factor and a constant increase of priority due to a clock tick of obtained CPU time, only needs one parameter instead of four, at least as far as the steady-state behavior is concerned.

8 Acknowledgments

The support of the High Performance Computing Department, managed by W.G. Pope, of the IBM T.J. Watson Research Center in Yorktown Heights, NY, USA, where part of the research reported on in this paper was performed, and of IBM The Netherlands, is gratefully acknowledged. In addition, the author owes much to stimulating discussions with J.L. Hellerstein of the IBM Research Division. Furthermore, the author thanks the OSF Research Institute in Grenoble, France, for the opportunity to perform measurements on their multiprocessor Mach system, and Andrei Danes and Philippe Bernadat of OSF for their help.

References

[1] M.J. Bach, *The Design of the UNIX Operating System*, Prentice-Hall, 1986.

[2] D.L. Black, *Scheduling and Resource Management Techniques for Multiprocessors*, Report CMU-CS-90-152, Carnegie Mellon University, 1990.

[3] R.B. Essick, "An Event-Based Fair Share Scheduler," *USENIX*, Winter, 147-161, 1990.

[4] G. Fayolle, I. Mitrani, and R. Iasnogorodski, "Sharing a Processor among Many Job Classes," *J. of the ACM*, Vol. 27, 519-532, 1980.

[5] J.L. Hellerstein, "Control Considerations for CPU Scheduling in UNIX Systems," *USENIX*, Winter, 359-374, 1992.

[6] J.L. Hellerstein, "Achieving Service Rate Objectives With Decay-Usage Scheduling," *IEEE Trans. on Softw. Eng.*, Vol. 19, 813-825, 1993.

[7] G.J. Henry, "The Fair Share Scheduler," *AT&T Bell Lab. Techn. Journal*, Vol. 63, 1845-1857, 1984.

[8] J. Kay and P. Lauder, "A Fair Share Scheduler," *Comm. of the ACM*, Vol. 31, 44-55, 1988.

[9] L. Kleinrock, "Time-Shared Systems: A Theoretical Treatment," *J. of the ACM*, Vol. 14, 242-261, 1967.

[10] S.J. Leffler, M.K. McKusick, M.J. Karels, and J.S. Quarterman, *The Design and Implementation of the 4.3BSD UNIX Operating System*, Addison-Wesley, 1989.

[11] C.A. Waldspurger and W.E. Weihl, "Lottery Scheduling: Flexible Proportional-Share Resource Management," *Proc. of the First USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 1-11, 1994.

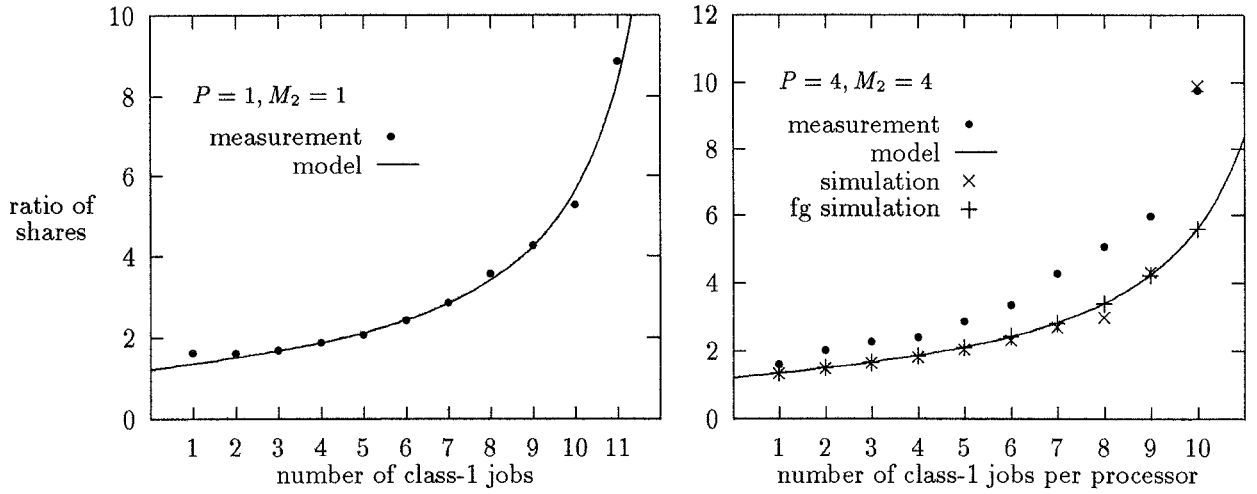


Figure 6: Ratio of shares (s_1/s_2) versus the number of class-1 jobs per processor (M_1/P) (4.3BSD, $K = 2, b_1 = 0, b_2 = 6$).

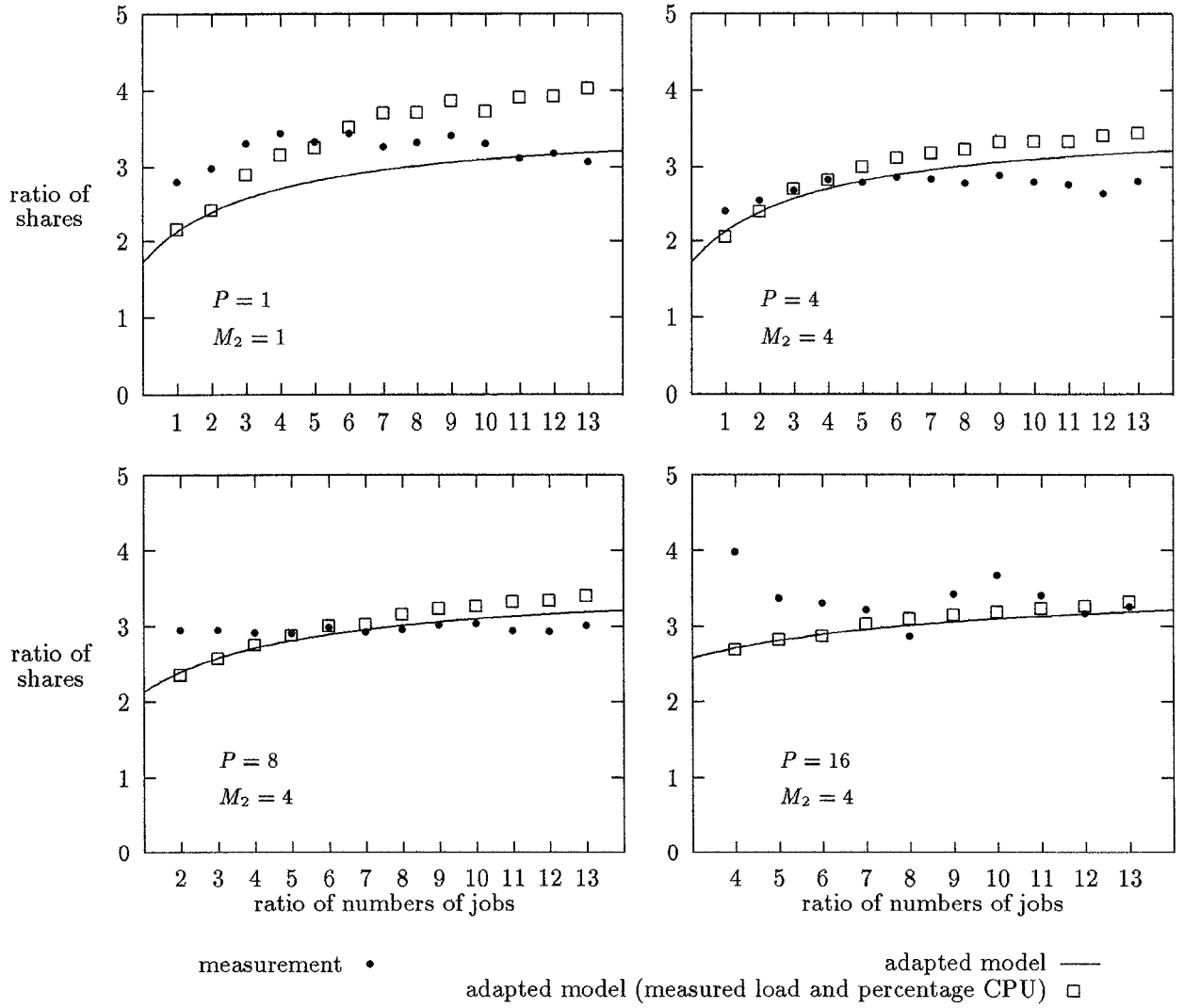


Figure 7: Ratio of shares (s_1/s_2) versus the ratio of the numbers of jobs of classes 1 and 2 (M_1/M_2) (Mach, $K = 2, b_1 = 0, b_2 = 18$).

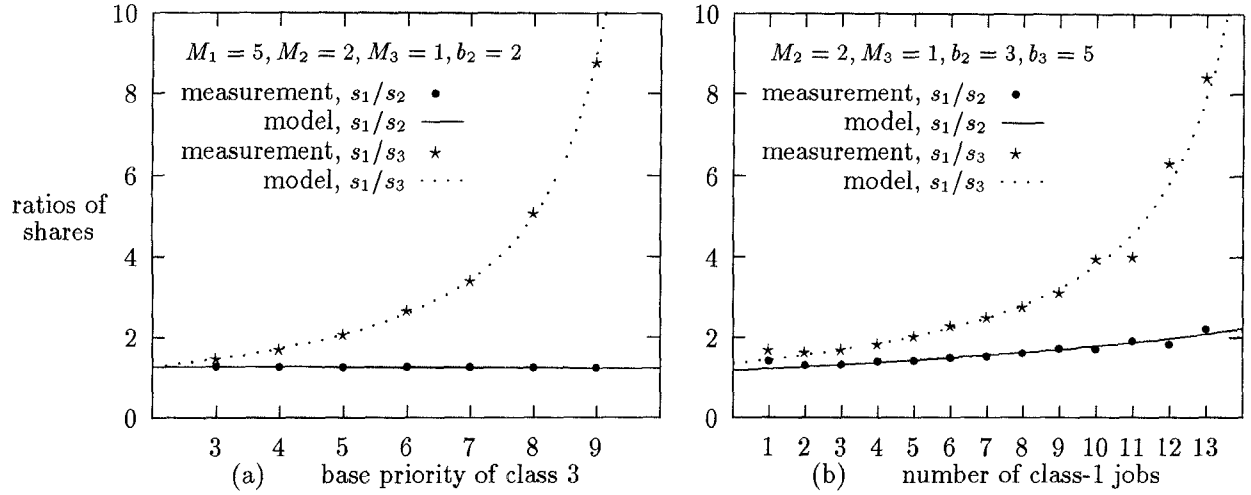


Figure 8: Ratios of shares ($s_1/s_k, k = 2, 3$) versus (a) the base priority of class 3 (b_3) and (b) the number of class-1 jobs (M_1) (4.3BSD, $P = 1, K = 3, b_1 = 0$).

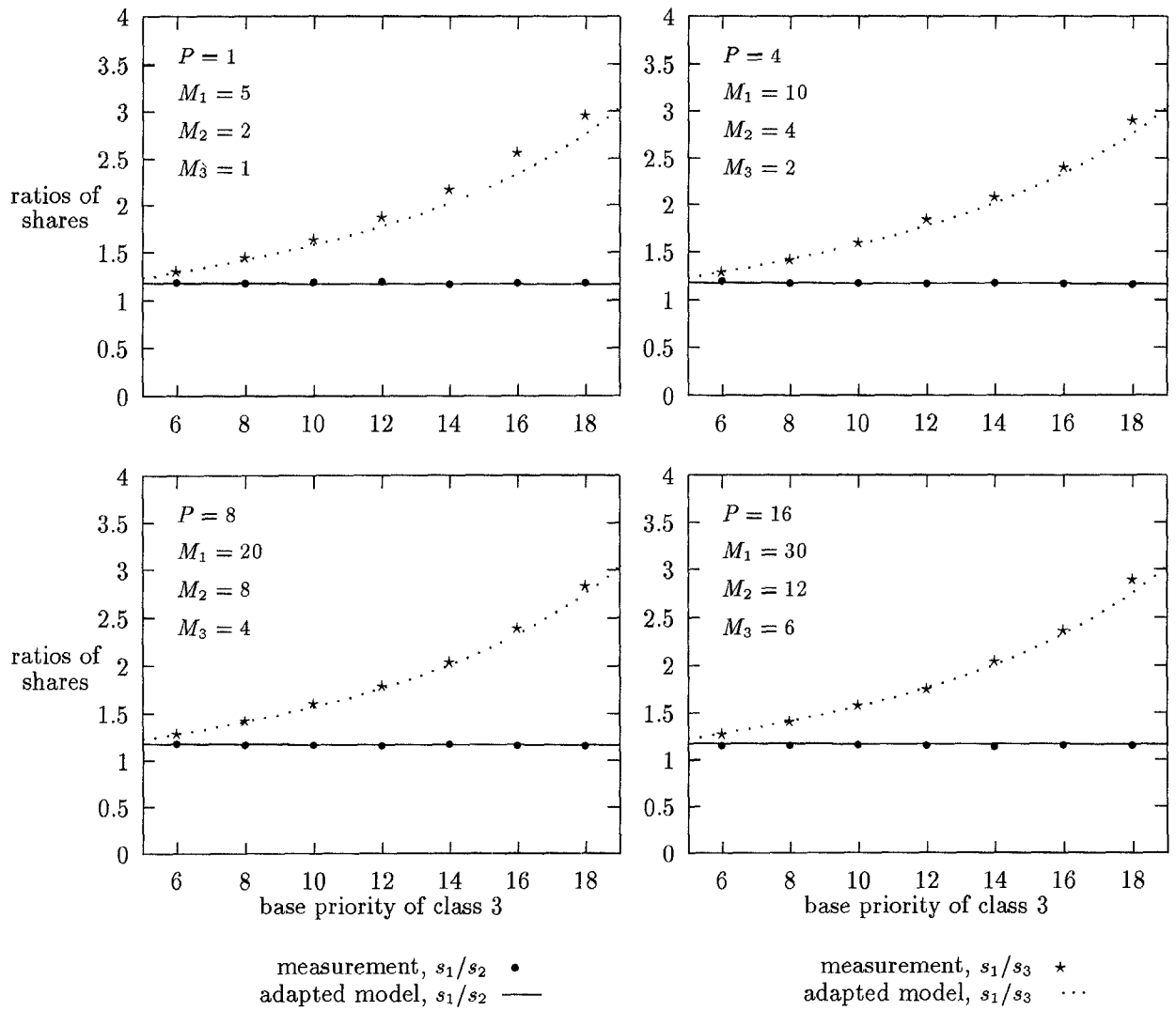


Figure 9: Ratios of shares ($s_1/s_k, k = 2, 3$) versus the base priority of class 3 (b_3) (Mach, $K = 3, b_1 = 0, b_2 = 4$).