Edgar H. Sibley
Panel Editor

Central-processing-unit schedulers have traditionally allocated resources fairly among processes. By contrast, a fair Share scheduler allocates resources so that users get their fair machine share over a long period.

A FAIR SHARE SCHEDULER

J. KAY and P. LAUDER

One of the critical requirements of a scheduler is that it *be fair*. Traditionally, this has been interpreted in the context of *processes* and has meant that schedulers were designed to share resources fairly between processes [3, 9, 10]. More recently, it has become clear that schedulers need to be fair to *users* rather than just processes. This is reflected in work such as [5], [11], [12], [15], and [19].

The context for the development of Share was that of the principal teaching machine in a computer science department. We had a large user community, dominated by 1000 undergraduates in several classes, as well as a staff of about 100. Our load was almost exclusively interactive and frequently had extreme peaks when major assignments were due. On a typical day, there were 60–85 active terminals, mainly engaged in editing, compiling, and (occasionally) running small-to-medium Pascal programs. All this activity was supported by a DEC VAX 11/780 running AUSAM, a local version of UNIX® [17] that is oriented toward a student environment.

UNIX provides a fairly typical scheduler [1]. It was inadequate in our environment, however, for a number of reasons:

(1) It gave more of the machine to users with more processes, which meant that users could easily increase their share of the machine simply by creating more processes.
(2) It did not take into account the long-term history of a user's activity. Thus, if a student used the machine heavily for approximately two hours, the same machine share was allocated as to a student who had not used the machine for some time.
(3) When one class had an assignment due and all the students in that class wanted to work very hard, everyone, including other students and staff, suffered with poor response.
(4) If someone needed good response for activities like practical examinations or project demonstrations, it was difficult to ensure they would get that response without denying all other users access to the machine.

The first three of these manifest the inequitable way the process scheduler affects users.

On many systems these problems are partially addressed by the *charging* mechanism [14, 16]. Typically, charging systems involve allocation of a budget to each user, and as users consume resources, they are charged for them. We might call this the *fixed-budget* model, in that each user has a fixed-size budget allotment. Then, as the resources are used, the budget is reduced, and when it is empty, the user cannot use the machine at all. A process can get a better machine share if the user who owns it is prepared to pay more for the resources used by that process. The fixed-budget model can be refined so that each user has several budgets, each associated with different resources.

We control allocation of some resources with a fixed-budget charging mechanism. In particular, we use this approach in these cases:

(1) For resources such as disk space, each user has a limit.[1]

UNIX is a registered trademark of AT&T Bell Laboratories.

[1] If a user exceeds this limit, the user is warned at the time and then at each log in for three log ins. After that, the user is not allowed to log in.

(2) Resources such as printer pages are allocated to each user as a budget that has a tight upper bound and is updated each day.[2]
(3) Daily connect limits are available to prevent individuals from exceeding their limit on the machine within a single day.
(4) Weekly connect limits are sometimes used to prevent students from spending too much time on computing (compared to other subjects), and to encourage students to work steadily on assignments from the first week they are set, right through to the last week. This, however, commonly has the effect of denying students any machine access near the assignment deadline, even though the machine is lightly loaded and students would like more machine access to finish and improve their programs.

There are also other utilities to help allocate resources, including a terminal-booking program that allows students to reserve a terminal at particular times each week.

All of these measures helped control consumption of resources, but did not deal with the problems of central-processing-unit (CPU) allocation we described earlier. It was for these reasons that we developed the Share scheduler. Although Share was motivated by our particular problems in a student environment, it equally serves the needs of any user community that shares a machine that is not operating for financial profit. Indeed, Share has been implemented in a research environment with many users from different organizations that have chosen to share the capital and running costs of a machine.[3]

To date, Share has been used exclusively to allocate CPU time, though it takes into account the consumption of all resources. We realize that Share may be applicable to the scheduling of resources other than CPU, but for simplicity, this article concentrates on CPU scheduling.

OBJECTIVES OF SHARE

The only way many systems link charging and scheduling is where users can specify the processes they agree to be charged more for, in return for being given preference in scheduling. Indeed, UNIX offers this kind of mechanism in *nice*, an attribute of a process that a user can adjust to alter its scheduling priority. On an in-house machine, this approach is adequate. A more natural approach, however, is to regard each user as having an entitlement to a fair share of the machine, relative to other users. Then, the task of the scheduling and charging systems must ensure that

- individuals cannot get more than their fair share of the machine in the long term, and
- the machine can be well utilized.

In addition, we extended the notion of fair shares to cover groups of individuals so that Share can allow the sharing of a machine between independent organizations.

To achieve fair sharing and be practicable for the individual and independent groups that share the machine, the objectives of Share were that it

(1) seem fair;
(2) be understandable;
(3) be predictable;
(4) accommodate special needs: Where a user needs excellent response for a short time, it should permit this with minimum disruption to other users;
(5) deal well with peak loads;
(6) encourage load spreading;
(7) give interactive users reasonable response;
(8) give some resources to each process so that no process is postponed indefinitely; and
(9) be inexpensive to run.

USER'S VIEW OF SHARE

Essentially Share is based on the principle that everyone should be scheduled for some resources

- according to their *entitlement*,
- as defined by their *shares*, and
- as defined by their resource *usage history*.

This is illustrated in Figure 1, which shows that a user can expect poorer response if the user has had his or her fair machine share. This, in turn, gives other users a chance to get their fair share.

A user's *shares* indicate his or her entitlement to use the machine. The more shares a user has, the greater the entitlement. This should operate in a linear fashion so that, if user A has twice as many shares as user B, then in the long run, user A should be able to do twice as much work as user B.

Every user has a *usage*, which is a *decayed* measure of

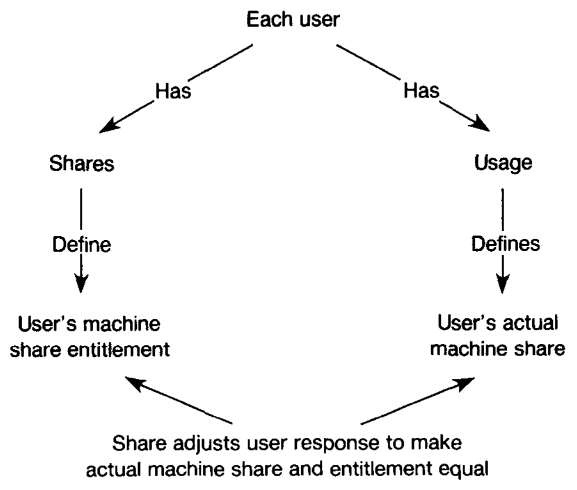

Each user
Has
Has
Shares
Usage
Define
Defines
User's machine share entitlement
User's actual machine share
Share adjusts user response to make actual machine share and entitlement equal

FIGURE 1. A User's View of Share

[2] For example, a user might have a printer page bound of 10 pages and a daily increment of 2 pages. This means the user starts with a budget of 10 pages; if, say, 3 pages are printed in one day, the budget for the rest of the day is 7 pages, and provided no more printing is done that day, the budget at the beginning of the next day will be 9 pages.

[3] A Cray X-MP at AT&T Bell Laboratories.

the work the user has done. The *decay rate* is defined by an administrative decision that determines how long a user's history of resource consumption affects his or her response. For example, in its first implementation in a student environment, the decay rate was set so that usage had a half-life of three days to encourage students to spread their machine use over the week.

Although it is normal for schedulers to use decayed CPU usages, Share's use of decayed resource usage in *charging* is a departure from traditional approaches. Where a machine is solely for in-house use, the only need for a raw (undecayed) resource consumption tally is to monitor machine performance and throughput, and to observe patterns of user behavior.

The decayed usage is also *normalized* by the user's shares. This might look as though it makes the machine less expensive to users with more shares. In essence, *Share attempts to keep the actual machine share defined by normalized usage the same as the machine entitlement defined by shares.* From the user's point of view, Share gives worse response to users who have had more than their fair share so that they cannot work as much as users who have not had their fair share. *So users see that, as their normalized usage increases, their response becomes worse.* (This assumes allowance is made for machine load.) We provide a simple command that displays a user's profile that includes their usage and the machine share they can expect.

This approach contrasts strikingly with conventional charging and scheduling systems that schedule processes equally, provided the user who owns them has a nonempty budget. In the fixed-budget model, the users who consume their fair share, by emptying their budgets, do not get any available resources. In the extreme case, there may not be any users because everyone who wants to use the machine has empty budgets. For an in-house machine, this does not make sense, and can generate substantial administrative overheads as users seek extra allocations.

The number of shares allocated to a user is, essentially, an administrative decision. In a situation where independent organizations share a machine, however, the shares that should be allocated to individual users depend both on the entitlement that their organization has and the individual's entitlement within the organization. For simplicity, we describe Share first in terms of a simple situation where there are no independent organizations involved: All users' shares are simply defined to indicate their right to work compared to other users. We deal with the more complex situation where the combined usage of groups of users must be considered, in the description of hierarchical Share.

Another factor in scheduling is the individual users' rights to alter the relative scheduling priority of their processes. We have preserved the UNIX *nice*, a number in the range 0–19 that a user can associate with a process. When users assign a nonzero *nice* value to a process, they indicate that poorer response is acceptable. The larger the *nice* value, the poorer the response. The way that this affects charging is another administrative decision: The name, *nice*, suggests that users who do not need fast response for a process will use nice out of generosity. In our environment we felt that it was worthwhile to give users an incentive to use nice, so we reduced the costs charged for processes with larger nice values.

Finally, the *charges* that Share uses are defined by the relative costs of different resources. For example, we associate a charge with memory occupancy, another with systems calls, another with CPU use, and so on. Note that this is another difference between Share and conventional schedulers that define a process's scheduling priority only on the process's consumption of CPU time. In Share, CPU scheduling priority is affected by total resource consumption.

In addition, we set charges at different levels at different times of the day. This is another administrative decision. For example, during the time the university is in session, we charge a peak rate during normal work hours, somewhat less for the hours before and after, and much less at off-peak hours.

Note that Share represents a radical departure from the traditional approaches to charging as described in [8]: "Prices should not be changed too frequently, since stability is one of the accepted requirements of a charging system."

We agree that users need to understand the charging system and see it as stable, but we argue that this does not require constant behavior. It can also be achieved by behavior that changes steadily, as in Share, where response steadily degrades as a user's resource consumption increases relative to other users.

We must emphasize that the *administrative decisions* are very important to Share's fairness. For example, we have just noted that we charge less at off-peak times and this seems to help spread the machine load. Another important factor in setting this policy, however, is that users consider it fair that they be charged less for the inconvenience of working after normal hours. Some of the administrative decisions are not easy though. The fixed-budget model has the advantage that one can easily supplement empty budgets and the initial budget size may not be as critical. By contrast, in Share, the shares allocated define the right to do work so that, when we allocate each first-year student half the shares given to a second-year student, we are defining the relative amount of work we expect each to extract from the machine.

OVERVIEW OF THE IMPLEMENTATION

As one might expect, conceptually there are two main components, one at the user level and the other at the process level.

User Level

User-level scheduling

- **Update usage for each user by adding charges incurred by all their processes since the last update and decaying by the appropriate constant.**
- **Update accumulated records of resource consumption for planning, monitoring, and policy decisions.**

At this point, Share computes the charges due to a user for the resources the user has consumed during the last cycle of the user-level scheduler. The charges are for all resources consumed and are lower at off-peak periods. This part of the scheduler does not need to run frequently because usage generally changes slowly.

Each user can get an estimate of their share of the machine by comparing their usage with that of all active users. Since this is a convenient and intuitive indication of the response a user can expect, we provide a percentage estimate of the user's machine share as part of the standard user profile information.

Process Level

The remainder of Share operates at the process level. Each process has a priority, and the *smaller* its value, the better the scheduling priority. We also introduce the term *active process* to describe any process that is ready to run, and at any point, the active process that actually has control of the CPU is called the *current process*. There are three types of activity at the process level:

(1) that associated with the activation of a new process;
(2) the regular and frequent adjustment of the priority of the current process; and
(3) the regular, but less frequent, decaying of the priorities of all processes.

The first occurs in a number of situations, including times when a process relinquishes control of the CPU, times when the active process is interrupted for some reason, and at regular times when the scheduler usurps the currently active process to hand control to the lowest priority process that is ready to run.

Process activation

- Update costs incurred by the current process.
- Select the process with lowest priority, and set it running.

Next is the adjustment to the priority of the current process, which defines the resolution of the scheduler. This ensures that the CPU use of the current process increases (worsens) its priority.

Priority adjustment

- Increase the priority of current process in proportion to the user's usage, shares, and number of active processes.

Finally, there is the regular decaying of all process priorities, which must be done frequently compared to the user-level scheduler, but with longer time intervals than the scheduler's resolution.

Decay of process priorities

- Decay all process priorities, with slower decay for processes with nonzero *nice* values.

DETAILED IMPLEMENTATION

The implementation of Share is shown in Figure 2. The remainder of this section explains each component, including the setting of the various parameters (that can be altered as the system runs).

```
Every t1 seconds: User-level scheduling
For each user,
  decay usage and update with costs incurred in last t1 seconds:
      $\mathtt{usage}_{user} = \mathtt{usage}_{user} \times \mathtt{K1} + \mathtt{charges}_{user};$
  reset cost tally:
      $\mathtt{charges}_{user} = 0.$
Every t2 seconds: Decay of process priorities
For each process,
      $\mathtt{priority}_{process} = \mathtt{priority}_{process} \times \mathtt{K2} \times (\mathtt{nice}_{process} + \mathtt{K2'}).$
Every t3 seconds: Priority adjustment
      $\mathtt{priority}_{current\_process} = \mathtt{priority}_{current\_process} + \frac{\mathtt{usage}_{current\_user} \times \mathtt{active\_processes}_{current\_user}}{\mathtt{shares}^2_{current\_user}}$
At each scheduling event: Current process selection
      $\mathtt{charges}_{current\_user} = \mathtt{charges}_{current\_user} + \mathtt{cost}_{event}$
Run process with lowest priority.
```

FIGURE 2. Share Implementation

User-Level Scheduling
The user-level scheduler is invoked every t1 seconds. The value of t1 defines the granularity of changes in a user's usage as he or she uses the machine. Since usage is generally very large compared to the resources consumed in a second, t1 can be of the order of a few seconds without compromising the fairness of the scheduler. The advantage of making t1 reasonably large is that somewhat costly computations can be made at this level without prejudicing the time efficiency of Share. Our VAX implementation makes t1 four seconds, which is 240 times the scheduler's resolution. On the Cray, we have found that four seconds (400 times Share's CPU charging resolution) is also acceptable.

The first component of the user-level scheduler decays each user's usage. This ensures that usages remain bounded and the value of the constant K1 in combination with t1 defines the rate of decay. We generally consider the effect of K1 in terms of the half-life of usage. In a student environment, we have used a half-life of three days. In other contexts, it has been much shorter, but generally on the order of several hours. At a conceptual level, this step is performed for all users. In fact, the effect of the calculation is computed as each user logs in, and the actual calculation need only be performed for active users.

The next part of user-level scheduling involves updating the usage of active users by the charges they have incurred in the last t1 seconds and resetting the charges tally.

Process-Level Scheduling
From this point on, we discuss the low-level scheduler that deals with processes. It operates in terms of the *priority* of each process. As is common practice in process schedulers, the priority defines the order in which processes are entitled to be allocated CPU resources. Accordingly, it

- schedules CPU resources to the process with the smallest priority, which corresponds to the process being at the head of the queue;
- increases the priority of a process each time it is allocated CPU time, which can be viewed as putting the process further down the queue; and
- decays all process priorities steadily so that one might view all processes as slowly drifting toward the front of the queue.

Share combines these activities with user-level scheduling in the following ways:

Decay of Process Priorities. The decay of process priorities ages the processes so that those that have not had the CPU achieve better and better (smaller) priority values. The value of t2 and K2 combined defines the rate at which processes age. We need to make t2 small, compared to t1, because priority values change rapidly. In our VAX implementation, t2 is set at 1 second, which is 60 times the resolution of the scheduler. (On the Cray, t2 is also 1 second.)

The rate at which processes age is affected by their *nice* value. We note that Share preserves the approach of the UNIX scheduler to *nice*: It assumes users normally want the best response possible (which corresponds to a *nice* value of 0), but there are also times when a user is happy to accept lesser response, which they indicate in terms of a *nice* value that is a small integer. (Its range is from 0, the default, to 19, which gives the worst response.) We define the value of K2 as

$$\mathrm{K2} = \frac{\mathrm{K2}''}{\mathrm{K2}' + \mathrm{max\_nice}},$$

where max_nice is the largest *nice* value (19). This ensures that the priority of processes with *nice* set to max_nice is decayed by K2″ every t2 seconds, and that the priority of processes with *nice* set to 0 is decayed somewhat faster. The values of K2′ and K2″ must be large enough to ensure that priorities are well spread and remembered long enough to prevent large numbers of processes from having 0 priority.

Priority Adjustment. At the finest resolution of the scheduler, t3, the current process has its priority increased by the usage and active-process count of the user who owns the process. (The scheduler resolution, t3, is a sixtieth of a second on the VAX version, and one hundredth of a second on the Cray.) Typically, schedulers increase the priority by a constant. Intuitively, one might view the difference between Share and typical schedulers as follows:

- A typical scheduler adjusts the priority of the current process by pushing it down the queue of processes by a *constant* amount.
- Share pushes the current process down the queue by an amount proportional to the usage and number of active processes of the process's owner, and inversely proportional to the square of that user's shares. Processes belonging to higher usage (more active) users are pushed further down the queue than processes belonging to lower usage (less active) users. This means that a process belonging to a user with high usage takes longer to drift back up to the front of the queue. (The priority needs longer to decay to the point that it is the lowest.)

We also want users to be able to work at a rate proportional to their shares. This means that the charges they incur must be allowed to increase in proportion to the square of the shares (which gives a derivative, or *rate* of work done, proportional to the shares).

The formula also takes account of the number of active processes (processes on the priority queue) for the user who owns the current process. This is necessary since a priority increment that involved just usage and shares would push a single process down the queue far enough to ensure that the user gets no more than his or her fair share. If the user has more than one active process, we need to penalize each of them to ensure that the user's share is spread between them, and we do this by multiplying the priority increment by the active-process count. This is the crux of the Share mechanism for making long-term usage, over all resources that attract charges, affect the user's response and rate of work.

Although the model we have described may be adequate for implementations where process priorities have a large range of values, on the machines where we have implemented Share, process priorities are small integers and cannot be used directly. We need to normalize the *share* priorities into a range that is appropriate for real process priorities. In addition, where the range in priority values is quite small, we need to ensure that the normalization procedure does not allow a single very large Share priority value to reduce all other normalized priorities to 0. To avoid this, we define a bound on the Share priority. This is calculated in the process-level scheduler as shown in Figure 3. K4 is determined by the largest priority available to the low-level scheduler. Note that the Share priority bound does, somewhat unfairly, favor very heavy users. They still suffer the effect of their slowly decaying large usage, however, and are still treated more severely than everyone else. On the other hand, it helps prevent marooning.

Process Activation. At each scheduling event, Share updates the current user's charges by the costs associated with the event and selects the lowest priority process to run. This aspect of Share is typical of CPU schedulers.

```
Find greatest Share priority for normalization:
        $max\_priority = 0.$
For each process,
  if
    $max\_priority < priority_{process} \leq priority\_bound,$
  then
    $max\_priority = priority_{process}.$

For each process, scale priority to appropriate range:
  if
    $priority_{process} \leq max\_priority,$
  then
    $normalized\_priority_{process} = (K4 - 1) \times \frac{priority_{process}}{max\_priority},$
  else
    $normalized\_priority_{process} = K4.$
```

FIGURE 3. Priority Normalization

Multiple Processors

Multiprocessor machines do not affect the implementation provided the kernel still uses a single priority queue for processes. The only difference is that processes are selected to run from the front of the queue more often, and incur charges more frequently, than if only one processor were present.

Efficiency

The implementation shown in Figure 2 should only be seen as a model of the actual code. For efficiency, some of the calculations shown at the level of the process scheduler are actually precalculated elsewhere.

Edge Effects

In general, it is important to avoid edge effects on scheduler behavior. In particular, if a user enters the system with zero usage they could effectively consume 100 percent of the resources, at least for one cycle of the user-level scheduler. Since this is a comparatively long time (a few seconds), this would be unacceptable. We now examine why this undue favoritism might occur and how Share deals with the problem.

First, we define the relative proportion of the machine due to a user by virtue of the allocation of shares:

$$machine\_proportion\_due_{user} = \frac{shares_{user}}{\sum_{u=1}^{active\_users} shares_u}.$$

This defines the proportion of the machine due to a user in the short term. Now we can also predict the short-term future proportion of the machine that a user should get by virtue of their usage:

$$near\_future\_machine\_proportion_{user} = \frac{shares_{user}^2/usage_{user}}{\sum_{u=1}^{active\_users} shares_u^2/usage_u}.$$

If everyone is getting their fair share, these two formulas will give the same value for each active user. Indeed, Share works to push these two formulas to the same value for each user. In the case where a user has zero usage (or near zero usage), we need to interfere to prevent that user from being unduly favored (while other users are ignored). We do this by altering the usage value in the user-level scheduler as shown in Figure 4. We have set K5 to 2.

System Processes. Processes that run in support of the operating system must be given all the resources they need. In effect, system processes are given a 100 percent share of the resources, although it is assumed they will not use it most of the time. Share is intended to arbitrate fairly between users, *after* the system has taken all the resources it needs.

Marooning. It is possible for a user to achieve a very large usage during a relatively idle period. If new users then become active, the original user's share becomes so small that they are unable to work effectively. This

```
Every t1 seconds:  User-level scheduler
for each user,
  if
    $near\_future\_machine\_proportion_{user} > K5 \times machine\_proportion\_due_{user}$,
  then
    $usage_{user} = usage_{user} \times \frac{near\_future\_machine\_proportion_{user}}{K5 \times machine\_proportion\_due_{user}}$.
```

FIGURE 4. Avoiding Edge Effects

user's processes are effectively *marooned* with insufficient CPU allocation even to exit. Marooning is avoided by the combination of bounds on the normalized form of priority, the process priority decay rate, and the granularity of the process-level scheduler.

HIERARCHICAL SHARE

Although the simple version of Share we have described served well for several years, it was inadequate for a machine that is shared between organizations or independent groups of users. For instance, consider a situation where organizations need to share a machine not only between users, but also at the organizational level. Share as described above is fine for this situation provided we can make the following assumptions:

(1) The total allocation of shares for each organization is strictly maintained in the proportions that the machine split is made. For example, if a machine is to be split equally between two organizations, the total shares for each organization must be the same.
(2) The users in each organization are equally active.
(3) K1 is acceptable at the organizational level and constant for all users.
(4) Costs for resources are consistent for all users, and the other parameters of Share, including K2, t1, t2, and t3, are accepted for all users.

Let us now consider how the simple Share is adjusted to account for each of these factors.

Shares in a Hierarchical Share Scheduler

It would be impractical to require that the total shares for each organization be maintained at a fixed value. This would mean the arrival of a new user would require adjustments to the shares of all users in that organization. This would be a serious problem that might rule out organizational sharing with the simple form of Share.

So that each organization appears to be operating their own machine, we allow that users be allocated shares just as in the simple Share. We cannot, however, directly compare such shares across organizations. We need to convert them to a comparable measure. The approach we take is to calculate each user's *machine share*, the proportion of the machine that their allocation of shares make them eligible to receive. We start at the root of the Share hierarchy tree and convert the shares allocated to each child node into their machine share, using the following formula:

$$m\_share_{node} = m\_share_{parent} \times \frac{shares_{node}}{\sum_{n=1}^{siblings} shares_n + share_{node}}.$$

This calculation is repeated recursively down the hierarchy tree until the m_share of each node has been calculated, and m_share is then used instead of shares is the user-level scheduler.

Varying Levels of Activity

It is not reasonable to assume that users are equally active at all times. This means that, as users log in and out, they alter the m_share value of all users in their scheduling group (and if they are the first user in their group to log in, or the last to log out, they alter the m_share of all users who descend from their grandparent node in the hierarchy tree). In terms of the operation of Share, this means that some m_share values will usually be recalculated at each log in or log out. This poses a small but acceptable overhead.

Share acts fairly under full load, but a light load can distort it. Consider, for example, the situation depicted in Table I. This shows a case where there are two organizations A and B with an equal share of the resources, where organization A has one active user, A1, and organization B has two users, B1 with a large share and inactive, and B2 with a small share running a CPU-bound process. The *effective* share of the two active users, A1 and B2, differ by a factor of 10, and yet the scheduler should divide the resources equally between the two groups, A and B.

TABLE I. User Activity that Distorts Group Sharing

	m_share	Description of user activity
Organization A		
User A1	0.50	Active
Organization B		
User B1	0.45	Logged in but inactive
User B2	0.05	CPU bound

First, we define the relative proportion of the machine due to a group by virtue of its allocation of shares:

$$\texttt{machine\_proportion\_due}_{group} = \frac{\texttt{shares}_{group}}{\sum_{g=1}^{active\_groups} \texttt{shares}_g}.$$

Now we can also calculate the actual share of resources consumed by a group for the most recent scheduling period:

$$\texttt{actual\_machine\_proportion}_{group} = \frac{\texttt{charges}_{group}}{\sum_{g=1}^{active\_groups} \texttt{charges}_g}.$$

If each group is getting its fair share, these two formulas give the same value for each active group. In the case described above, we need to interfere if group B (and hence user B2) is to get its fair share. This is done in the user-level scheduler by reducing the costs of resources consumed by a group that is getting less than a certain amount of its share (see Figure 5). This decreases the usage for active users in the group and allows them to increase their share and the group's share. This calculation applies only to the dynamic usage value: The long-term usage for the user incurs full costs. K6 is set to allow a group's allocated share to fall below its effective share by some small amount. We chose 10 percent.

Differential Decay Rates for Usage

We have shown that the simple Share used the same rate of decay for the usages of all users. It follows that users within an organization should have the same usage decay rate. We do not need to do this between organizations. This can be illustrated in terms of the simple Share system operating in the university context where it is deemed appropriate to a set a three-day half-life for usage in the case of a machine used by undergraduates, but for the research-support machine, an acceptable half-life value is 12 hours. When different organizations share a machine, the right to define different decay rates may be important.

In practice, we have not dealt with this problem. There is a simple administrative solution if the organizations can agree to a constant decay rate within each organization and negotiate the organization machine share allocations to take this into account. An alternate, more complex approach, is a dynamic correction for differential decay rates by keeping two forms of usage: one for each user as we currently do, and another for each organization with a common decay rate applied to all organizational usage values. Then we could make a further adjustment to each group's m_share value (and hence each user's) to account for any imbalances in the group-level usage value.

Other Parameters

We have not allowed for variability per group or per user in any of these.

EVALUATION OF SHARE

Some parts of the design we have described were evaluated [2] before implementation in 1985. This evaluation with synthetic loads was mainly intended to guide the development of a computational model for the scheduler before it was put into active service on a heavily used machine. This preliminary work smoothed the introduction of the scheduler.

Once Share had been put into service, we used two forms of evaluation. First, we used several monitoring tools to watch it in operation. These have also been useful for administration and users. They indicate

- *resource usage between groups*: shows the effective share and actual resource consumption by group;
- *resource usage between users*: shows the actual resource consumption for every user;
- *effective share distribution*: plots a graph of users versus normalized usages; a non-Poisson distribution probably indicates problems, such as a class of users (not necessarily in the same group) that are consuming a disproportionately large amount of the resources;
- *resource event frequency*: provides feedback on active resource consumptions;
- *long-term charges*: provides details on the share of the resources between groups and users over a long time period.

In addition, we have run synthetic tests with pure CPU bound processes, to check that Share preserves the proper relationships between users with different shares, usage, and number of processes.

In view of the difficulties of the creation of valid

```
For each group (descend hierarchy),
    if
        $\texttt{actual\_machine\_proportion}_{group} < K6 \times \texttt{machine\_proportion\_due}_{group}$,
    then, for each user in the group (descend hierarchy),
        $\texttt{charges}_{user} = \texttt{charges}_{user} \times \frac{\texttt{actual\_machine\_proportion}_{group}}{K6 \times \texttt{machine\_proportion\_due}_{group}}$.
```

FIGURE 5. Group Adjustment

simulation models and synthetic loads [4], we consider the users' reactions to Share in real operation the most important evaluation of the system.

DESIGN GOALS

Design Goal: That It Be Fair

We aimed to achieve this goal in terms of a secondary goal: that users be allocated shares that defined their relative machine share and that users getting more than their machine share should be penalized with poorer response. On simple tests with synthetic jobs, we observed that Share met this design goal [2]. More important, however, users deemed the scheduler to be treating them fairly.

Even with the simple, nonhierarchical Share, we have observed a number of situations where Share has dealt with potentially disastrous situations to the satisfaction of most users. For example, in our student environment, we allocate shares to students on the basis of the relative machine share they should need. If a class is given an assignment that demands significantly more machine resources, only the students in that class will find the machine slow. With a conventional scheduler, everyone suffered in this situation. Share has proved useful for this problem in that the source of the problem is patently obvious, as is the identity of the person responsible for creating it.

A similar example, with the hierarchical Share system, involved a user who initiated a long running CPU-bound process. Share ensured that users in other groups were unaffected by the problem.

Design Goal: That It Be Understandable

Figure 1 indicates the user's view of Share. Our users appear to be able to appreciate this view, and interpret relatively poor response as an indication they have exceeded their machine share. They also become alert to the relative costs of various processes they create since it is directly reflected in their relative response from the machine.

Design Goal: That It Be Predictable

Each user's personal profile lists their effective machine share, that they quickly learn to interpret. Users speak of a certain machine share as being adequate to do one task, but not another.

Design Goal: The Scheduler Should Accommodate Special Needs

Share accommodates situations where brief periods of excellent response are guaranteed for individuals or groups of users. One simply allocates a relatively large number of shares to the relevant user's (or group's) account for the duration of the special needs. This is a simple procedure that the system administrator can set up to run at the required times.

Clearly, this sort of activity disrupts other users as they have to share a smaller part of the machine than usual. In fact, we observe that the favored users may only make major demands of the machine for very brief periods. Typically, other users suffer only small periods of reduced response. Although this facility is only necessary on rare but critical occasions,[4] it is an attractive benefit of Share.

Design Goal: That It Should Deal Well with Peak Loads

In our design environment, one of the classic causes of a peak load is the deadline for an assignment. Because we stagger the deadlines for different classes, one class of students may try to work harder as the deadline approaches. In pre-Share days, everyone suffered, and the machine would grind to a halt. With Share, the individuals in the class that is working to the deadline are penalized as their usage grows. Meanwhile, other students get good response and are often unaware of the other class's deadline. In effect, under heavy load, heavy users suffer most.

Design Goal: That It Should Encourage Load Spreading

The most direct observation of Share's load-spreading effect is that users do give up when they get poor response, especially when it is bad relative to other users. We would like to report that our students now start their assignments early and work on them steadily: this unfortunately is not the case. Given that one class deadline cannot disrupt another, however, allows students to plan their work and enables them to predict that they will get reasonable response, if they work steadily.

Design Goal: That It Should Give Interactive Users Reasonable Response

We can ensure this goal by combining Share with a check at log-in time that only allows users to log in if they can get reasonable response. In practice, we have not utilized this facility unless there is a very large number of users (over 70 on the VAX). Those who have high usage do get poor response, and if the machine is heavily loaded, the poor response may well be intolerable for tasks such as using a screen editor. We view this as an inevitable consequence of Share being fair to users whose fair share is really very small.

In general, Share does ensure good throughput for the small processes that typify interactive use.

Design Goal: No Process Should Be Postponed Indefinitely

Since Share allocates some resources to every process, this goal is also achieved.

Design Goal: That It Should Be Inexpensive to Run

Since most of the costly calculations are performed infrequently (in the user-level scheduler), Share creates only a small overhead relative to the conventional scheduler.

THE ESSENTIAL SHARE

Our description mirrors Share as we have implemented it. The aspect that is essential to Share is that it shares

[4] These include demonstrations of software to funding agencies and, in the teaching context, practical examinations.

resources fairly between users, rather than just processes. Other aspects can be altered within the Share framework.

In particular, several parameters are defined by administrative decisions and need to be set according to the particular requirements of each machine. For example, we set the constant K1 to make usage decay quite slowly: Its half-life has ranged from a few hours to three days. It could equally be of the same order as the process priority decay rate. Since the function of usage is to ensure that process priorities reflect the total activity of the user who owns them, it can do that equally well with a short half-life if that is what is required. To date, we have used Share in environments where a long-usage half-life has been regarded as fair.

Other such parameters that can be altered include the various constants, the frequency with which the user level and process schedulers run, and the way charges are calculated. On the last of these, charges should be selected to reflect the administrator's view of the costs of each resource. This may well change in light of monitoring information or with changes in the hardware configuration.

Similarly, the time variance of charges could be altered. In our experience, it seems best to have fixed costs at particular times of day so that users can plan their work in terms of these. In other situations, it may be appropriate to take some other approach. Cost could be dynamically altered on the basis of load so that the machine would become more costly to use at peak times, whenever they occurred, or one could have fixed costs at all times. Such changes should be taken with care. For example, the suggestions that costs change dynamically may, at first glance, seem attractive and sensible. Nevertheless, it violates the principle of predictability, a sacrifice that should not be taken lightly.

CONCLUSION

Users perceive the scheduler as fair in practice, and tend to blame poor response more on their past usage, rather than on system overloading. The strengths of Share are that it

- is fair to *users* and to *groups*, in that users cannot cheat the system, and groups of users are protected from each other;
- gives a good prediction of the response that a user might expect;
- gives meaningful feedback to users on the cost of various services; and
- helps spread load.

Share has proved useful in practice, both in teaching and research contexts. Other contexts are possible, such as sharing access to a file server to prevent any one client from monopolizing the service.

Appendix

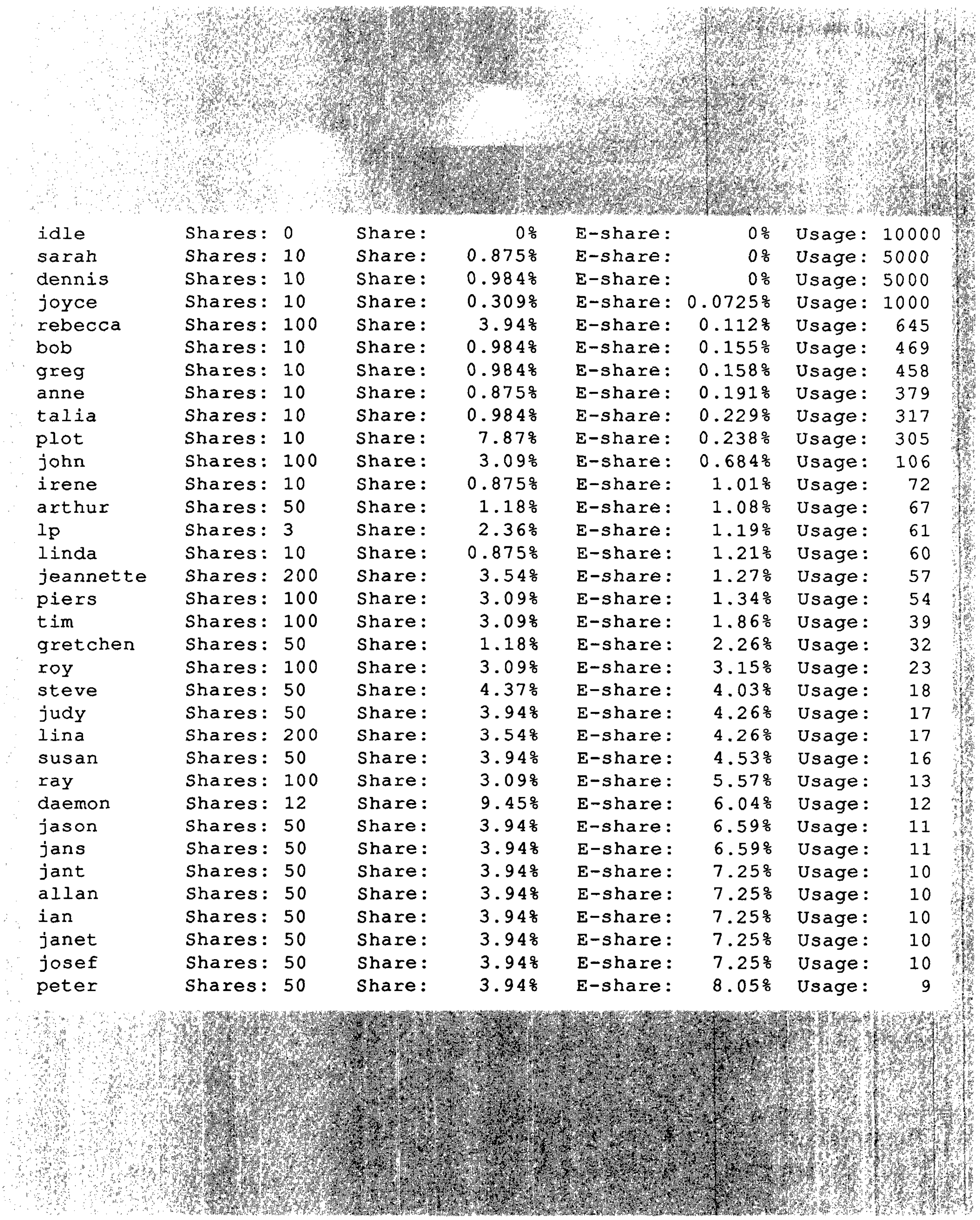
Below is sample output from some of the monitoring utilities. They illustrate some of the information available to users and system maintainers. The actual displays have been edited to give fictitious names to users and groups.

Display from a Hierarchical Share System on a DEC-VAX

Figures 6 and 7 have hash signs to represent resource consumption, and I to show allocated share.

```
Group      No. %Rate                         Wed Oct 14 16:48:08 1987
System      1    1.0 #                                              |
other       1    0.0 #I                                             |
idle        1    0.0 #                                              |
support     1    0.0 #  I                                           |
tutor       5   15.2 ###I####                                       |
staff       6    0.0 #          I                                   |
maint       2    0.0 I                                              |
office      2    0.0 #  I                                           |
pgrad       4   24.3 #######I####                                   |
hons        4    0.0 #I                                             |
prog        6   57.8 #######I#####################                  |
daemon      1    1.6 ##  I                                          |
lp          1    0.0 I                                              |
Total =>   35
```

FIGURE 6. A Display of the Rate of Use by Groups on a VAX



```
idle       Shares: 0    Share:      0%  E-share:      0%  Usage: 10000
sarah      Shares: 10   Share:  0.875%  E-share:      0%  Usage: 5000
dennis     Shares: 10   Share:  0.984%  E-share:      0%  Usage: 5000
joyce      Shares: 10   Share:  0.309%  E-share: 0.0725%  Usage: 1000
rebecca    Shares: 100  Share:   3.94%  E-share:  0.112%  Usage:  645
bob        Shares: 10   Share:  0.984%  E-share:  0.155%  Usage:  469
greg       Shares: 10   Share:  0.984%  E-share:  0.158%  Usage:  458
anne       Shares: 10   Share:  0.875%  E-share:  0.191%  Usage:  379
talia      Shares: 10   Share:  0.984%  E-share:  0.229%  Usage:  317
plot       Shares: 10   Share:   7.87%  E-share:  0.238%  Usage:  305
john       Shares: 100  Share:   3.09%  E-share:  0.684%  Usage:  106
irene      Shares: 10   Share:  0.875%  E-share:   1.01%  Usage:   72
arthur     Shares: 50   Share:   1.18%  E-share:   1.08%  Usage:   67
lp         Shares: 3    Share:   2.36%  E-share:   1.19%  Usage:   61
linda      Shares: 10   Share:  0.875%  E-share:   1.21%  Usage:   60
jeannette  Shares: 200  Share:   3.54%  E-share:   1.27%  Usage:   57
piers      Shares: 100  Share:   3.09%  E-share:   1.34%  Usage:   54
tim        Shares: 100  Share:   3.09%  E-share:   1.86%  Usage:   39
gretchen   Shares: 50   Share:   1.18%  E-share:   2.26%  Usage:   32
roy        Shares: 100  Share:   3.09%  E-share:   3.15%  Usage:   23
steve      Shares: 50   Share:   4.37%  E-share:   4.03%  Usage:   18
judy       Shares: 50   Share:   3.94%  E-share:   4.26%  Usage:   17
lina       Shares: 200  Share:   3.54%  E-share:   4.26%  Usage:   17
susan      Shares: 50   Share:   3.94%  E-share:   4.53%  Usage:   16
ray        Shares: 100  Share:   3.09%  E-share:   5.57%  Usage:   13
daemon     Shares: 12   Share:   9.45%  E-share:   6.04%  Usage:   12
jason      Shares: 50   Share:   3.94%  E-share:   6.59%  Usage:   11
jans       Shares: 50   Share:   3.94%  E-share:   6.59%  Usage:   11
jant       Shares: 50   Share:   3.94%  E-share:   7.25%  Usage:   10
allan      Shares: 50   Share:   3.94%  E-share:   7.25%  Usage:   10
ian        Shares: 50   Share:   3.94%  E-share:   7.25%  Usage:   10
janet      Shares: 50   Share:   3.94%  E-share:   7.25%  Usage:   10
josef      Shares: 50   Share:   3.94%  E-share:   7.25%  Usage:   10
peter      Shares: 50   Share:   3.94%  E-share:   8.05%  Usage:    9
```

FIGURE 7. A Display of the Scheduling Information for Users on a VAX

Acknowledgments. This work was the product of much discussion over a long period. In typical academic tradition, many people had a lot to say about the changes Share brought to their lives. Many of those comments were very useful. In addition, flaws in the initial design were identified by several people to whom we are grateful.

The first versions of Share [7, 13] grew from the ideas described by Larmouth [11, 12] and the basic work by Hume [6]. Chris Maltby played a critical role in the implementation and monitoring of the first version. Sandy Tollasepp [18] helped to analyze the performance of the first version, and John Brownie, the second. Carroll Morgan's suggestions were the basis for revising the whole approach of the first version of Share, and they made for the simplicity of the current version. Rob Pike and Allan Bromley independently identified an error in an earlier form of the Share normalization procedure. Glenn Trewitt suggested the current form of taking account of the user-supplied *nice* value.

REFERENCES

1. Bach, M.J. *The Design of the UNIX Operating System.* Prentice-Hall, Englewood Cliffs, N.J., 1986.
2. Brownie, J. Analysis and simulation of Share systems. Honors thesis, Computer Science Dept., Univ. of Sydney, Australia, 1984.
3. Coffman, E.G., Jr., and Kleinrock, L. Computer scheduling methods and their countermeasures. In *Proceedings of the Spring Joint Computer Conference,* vol. 32 (Atlantic City, N.J., Apr. 30–May 2). AFIPS Press, Reston, Va., 1968, pp. 11–21.
4. Heidelberger, P., and Lavenberg, S.S. Computer performance evaluation methodology. *IEEE Trans. Comput.* C-33, 12 (Dec. 1984), 1195–1220.
5. Henry, G.J. The fair share scheduler. *Bell Syst. Tech. J.* 63, 8, Part 2 (Oct. 1984), 1845–1857.
6. Hume, A. A Share scheduler for Unix. AUUG Newsl., Australian UNIX User's Group, Sydney, Australia, 1979.
7. Kay, J., Lauder P., Maltby, C., and Tollasepp S. The Share charging and scheduling system. Tech. Rep. 174, Basser Dept. of Computer Science, Univ. of Sydney, Australia, May 1982.
8. Kleijnen, A. J. V. Principles of computer charging in a university-type organization. *Commun. ACM 26* 11 (Nov. 1983), 926–932.
9. Kleinrock, L. A continuum of time-sharing scheduling algorithms. In *Proceedings of the AFIPS Spring Joint Computer Conference,* vol. 36 (Atlantic City, N.J., May 5–7). AFIPS Press, Reston, Va., 1970, pp. 453–458.
10. Lampson, B.W. A scheduling philosophy for multiprocessor systems. *Commun. ACM 11,* 5 (May 1968), 347–360.
11. Larmouth, J. Scheduling for a share of the machine. *Softw. Pract. Exper. 5,* 1 (Jan. 1975), 29–49.
12. Larmouth, J. Scheduling for immediate turnaround. *Softw. Pract. Exper. 8,* 5 (Sept.-Oct. 1978), 559–578.
13. Lauder, P. Share scheduling works! AUUG News. 1980.
14. McKell, L.J., Hansen, J.V., and Heitger, L.E. Charging for computing resources. *ACM Comput. Surv. 11,* 2 (June 1979), 105–120.
15. Newbury, J.P. Immediate turnround—An elusive goal. *Softw. Pract. Exper. 12,* 10 (Oct. 1982) 897–906.
16. Nielsen, N.R. The allocation of computing resources—Is pricing the answer? *Commun. ACM 13,* 8 (Aug. 1970), 467–474.
17. Ritchie, D.M., and Thompson, K. The UNIX timesharing system. *Bell Syst. Tech. J. 57,* 6 (July-Aug. 1978), 1905–1929.
18. Tollasepp, S. The SHARE resource allocation system. Honors thesis, Computer Science Dept., Univ. of Sydney, Australia, 1981.
19. Woodside, C.M. Controllability of computer performance tradeoffs obtained using controlled-share queue schedulers. *IEEE Trans. Softw. Eng. SE-12,* 10 (Oct. 1986), 1041–1048.

CR Categories and Subject Descriptors: C.4 [**Computer Systems Organization**]: Performance of Systems—*design studies; performance attributes; reliability, availability, and serviceability;* D.4.1 [**Operating Systems**]: Process Management—*scheduling*

General Terms: Performance

Additional Key Words and Phrases: User scheduler, fair share scheduler, hierarchical CPU scheduler, charging, resource allocation

Received 4/87; accepted 9/87

Authors' Present Address: J. Kay and P. Lauder, Basser Dept. of Computer Science, Madsen Building, F09, University of Sydney, N.S.W., Sydney 2006, Australia.