Technical Report

UCAM-CL-TR-13
ISSN 1476-2986

Number 13

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Resource allocation and job scheduling

Philip Hazel

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

Technical reports published by the University of Cambridge
Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/TechReports/

ISSN 1476-2986

SUMMARY

The mechanisms for sharing the resources of the Cambridge IBM 370/165 computer system among many individual users are described. File store is treated separately from other resources such as central processor and channel time. In both cases, flexible systems that provide incentives to thrifty behaviour are used. The method of allocating resources directly to individual users rather than in a hierarchical manner via faculties and departments is described, and its social acceptability is discussed.

CONTENTS

INTRODUCTION

"Have ye not known? have ye not heard? hath it not been told you from the beginning?"

(Isaiah)

The IBM 370/165 system was installed in Cambridge at the beginning of 1972. At that time the IBM operating system contained no mechanisms for sharing computing resources amongst a community of users, and the Computing Service had therefore to develop its own. Some versions of some of the systems that were developed have been described previously [1,2]; however, there have been many changes during the lifetime of the machine. This paper describes the two resource allocation systems, for processing resources and for disc store, as they were at the end of 1979. Appendix A contains a historical summary of the development of these systems.

The work of designing the algorithms for resource control and implementing them within the existing operating system has involved many members of the staff of the Computing Service. The present author has had little to do with this aspect of the system, and is acting here as raconteur only.

The existence of mechanisms for sharing resources means that administrative procedures for allocating resources amongst the users must exist. University Computer Centres in the U.K. are supported on public funds, largely by earmarked grants. A University, having made a case for, and subsequently purchased, a given computer system, must then share out the resources by whatever means it believes is best. The decoupling of Computer Centre funding from resource allocation introduces a degree of flexibility in that the use of real money as a means of control can be avoided. In many universities resource allocation is done by a hierarchical series of committees, which divide the resources between faculties, departments, sub-departments, and so on, down to the individual user. At Cambridge, in contrast, resources are allocated directly to individuals by Computing Service staff, after consultation with the user's department.

The next section contains brief descriptions of the 370/165 system, the facilities it provides, and the user population. The following two sections describe the mechanisms for sharing out processing resources and disc store respectively. These are followed by a discussion of the resource allocation procedure.

THE CAMBRIDGE 370/165 SYSTEM

"To give an accurate description of what has never occurred is not merely the proper occupation of the historian, but the inalienable privilege of any man of parts and culture."

(Oscar Wilde)

The hardware configuration of the Cambridge 370/165 is shown in figure 1. The system runs under OS/MVT release 21 with HASP. A number of modifications have been made to improve the performance and facilities of the system.

Time-sharing is provided using the basic framework of TSO, but with many modifications. The command interpreter has been replaced in order to provide the Phoenix command language, which is also available as an alternative to IBM's JCL in batch jobs. Thus three modes of access to the system exist: Phoenix sessions, Phoenix jobs and JCL jobs. Up to 120 simultaneously active Phoenix sessions are supported, in five swapping regions. The regions are necessarily small, to keep down the time spent swapping so that response times remain good. The main limitation on the number of simultaneous sessions is swap load; the current limit of 120 is made possible by local improvements to the swapping software, of which the most important is the use of spare main store for temporary swapping space.

The terminal user normally has around 34K bytes of store available in which to run interactive programs. The usual set of system utilities normally found in time-sharing systems is provided within this limit, including job submission, job queue interrogation, output collection, file store management and text editing. In some cases the programs are permanently resident in store, and the 34K swapped region is used purely as work space. The running of interactive user programs is severly inhibited by the smallness of the swapped regions, and in practice only a very few, specially written user programs are run interactively.

A local modification to TSO has been implemented to enable a time-sharing user to acquire a second, non-swapped region from the general pool of main store in order to run programs such as compilers that require more than 34K bytes of store. In order to minimize the real time that such store is in use, interaction with the terminal is normally forbidden to programs running in the second region. All input must be set up in disc files before the program is run. Output for the terminal is automatically spooled on disc by the Phoenix system, and copied to the terminal when the second region has been released. By this means, programs which require up to 250K bytes of store and 60 seconds of CPU time can be run directly from a terminal without impacting on the swap load. A very small number of privileged users are permitted to interact with programs running in a second region. Use of this facility can make main store unavailable for long periods of real time, which affects system throughput. Hence it is strictly controlled.

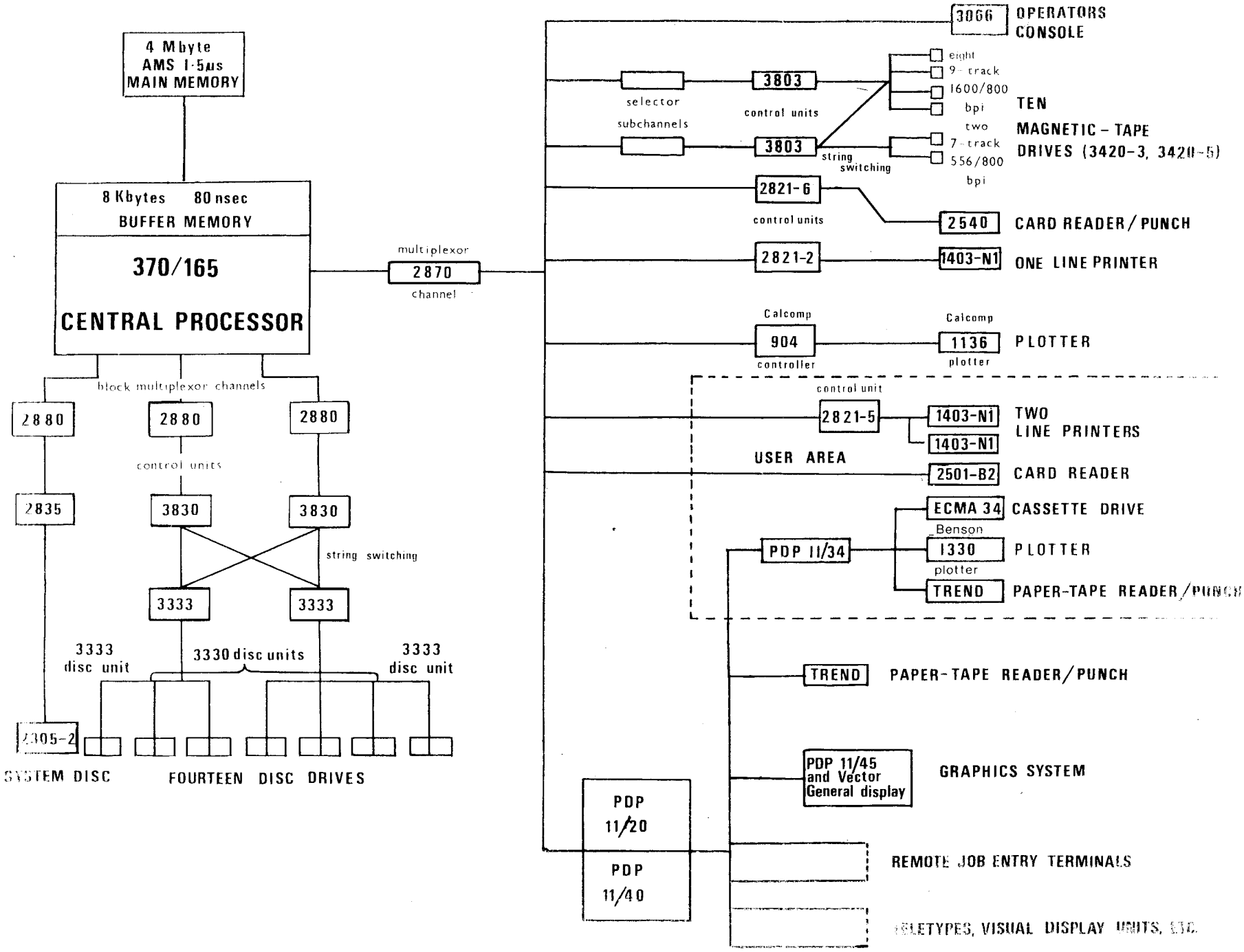


Figure 1: The Cambridge 370/165 system

The philosophy adopted for resource control is based on the premise that it is only necessary to ration that which is scarce. Some resources, such as disc store, are always scarce. Others, such as CPU time, are more scarce at certain times of day when demand for them is high. Some resources are not scarce at all, and need not therefore be controlled. It has not been found necessary to control the use of consumables such as cards and printer paper (other than by exhortation), nor the connect time for Phoenix time-sharing sessions. The maximum number of simultaneous sessions has for the most part been kept greater than the maximum demand, and so a logged-on user who is not consuming any resources causes no detriment to anybody else. However, if a session consumes no resources for a continuous period of ten minutes, it is logged off, a warning being given after five minutes. This prevents Phoenix session slots from being held by unattended terminals.

Resources are allocated to users in two categories: permanent disc space and processing resources. The latter include central processor (CPU) time, channel time, main store occupancy, and other items related to the running of programs. These are handled in combination, and the cost of running a program is measured in a derived unit called a Resource Unit. Appendix B contains a description of the formula used to calculate the number of resource units used by a job or session.

Disc space is treated separately for a number of reasons. It is not a wasting resource in the same way as CPU time is, and the cost of using it (in terms of detriment to the rest of the community) is not directly related to processing. In addition, there are entities to whom disc space but not processing resources are to be allocated, and vice versa.

The resources allocated for running programs are not divided between batch and time-sharing use in any way. The user is free to choose whether to do his work at a terminal or to submit jobs, subject to the resource control mechanism, and to the maximum size limits in either case. No job may use more than 20 minutes of CPU time or one megabyte of main store, while no time-sharing step may use more than one minute of CPU time or 250 kilobytes of store. There is no limit, however, on the duration of a time-sharing session, other than the automatic log off after ten idle minutes.

In practice, most users use a mixture of Phoenix sessions and jobs, the mix depending on their type of work. Over 90% of jobs are submitted from Phoenix sessions.

With very few exceptions the users of the 370/165 are doing academic work, and the major part of the workload consists of research computation. However, a large number of student teaching jobs are also run, both for Computer Science students and for students in other disciplines. Administrative work is not in general run on this machine.

THE SCHEDULING SYSTEM

"All men are created equal and independent"

(Thomas Jefferson)

"...but some are more equal than others."

(George Orwell)

The scheduling system is the mechanism by which the sharing of processing resources is controlled. We give first an informal, narrative description before going into the details. The system is described from the user's point of view and rigorous specifications of the scheduling algorithms are not included. These are to be separately documented.

From the point of view of the resource sharing scheduler, outstanding batch work is kept as a queue of jobs which is processed from the top down. In practice, multi-programming means that several jobs can be executing at once. Although in principle these should be the top jobs on the queue, resource conflicts may prevent this from being the case. For example, if there are eight tape decks available, there is no point running three jobs which each require three decks. The choice of which jobs to run from the top of the queue is handled by the machine dependent scheduler. This is not described in this document, and its existence is largely ignored in what follows.

Narrative Overview

"If it be done now, 'tis not to come; if it be not to come, it will be now; if it be not now, yet it will come: the readiness is all."

(Shakespeare)

The scheduling system works on the principle of controlling a user's rate of work, rather than the total amount. This is done by giving a longer turnround to the work of those users who have had more than their share of resources. The nature of the user population is such that there is virtually no work with fixed deadlines run on the 370/165. What little there is is dealt with in an ad hoc fashion. On the other hand, users do expect reasonable turnround for their work, and they also require to know when their work will be run.

Strictly speaking, processing resources are allocated to projects rather than users. One project may have many users, and one user may have access to more than one project. However, in the majority of cases there is a one-to-one correspondence between users and projects.

In early versions of the scheduling system, the turnround given to jobs for a particular project increased continuously as the amount of recent work on the project increased. While this system is attractive on paper, it neglects the human factors concerned with the way people work. A person is normally content to wait for a job that returns in 10 minutes, or maybe half an hour, but if the turnround is as long as an hour he tends to go

away and start some other activity which may take several hours. The machine might then just as well run the job in three hour's time as in one hour's time.

The current system aims to maximize the number of jobs which are given immediate turnround. A time-sharing session is treated as a job with immediate turnround; details of session scheduling are given in a later section. The general behaviour of the scheduler is to continue to give immediate turnround to a project's work, in the absence of any user request to the contrary, until the project becomes overworked. The turnround then increases discontinuously, normally to around three hours in the first instance.

More than 85% of all jobs submitted are scheduled for immediate turnround, and are termed fast jobs. About 30% of the CPU time delivered to jobs is consumed by fast jobs. By and large, therefore, fast jobs tend to be the smaller jobs, as one might expect. Large production work is typically scheduled to run at off-peak times, but can be run immediately if the project's allocation can sustain it. About 80% of fast jobs begin execution within one minute of submission. At first sight this figure might be thought to imply that the machine is underloaded. The following simple example illustrates the concept of maximizing immediate turnround, and shows that the implication does not hold.

Consider a uni-programming system which runs for 24 hours. Jobs arrive at the rate of one per minute for the first twelve hours. Thereafter the machine simply runs off the backlog (scheduling overheads are assumed negligible). Each job requires 2 minutes of CPU time, and hence the machine is fully loaded. Now consider two different scheduling strategies:

In the first strategy, jobs are scheduled on a first come, first served basis. The first job is run immediately, the second job receives a one minute turnround, the third a two minute turnround, and so on up to the last job, which has a 12 hour turnround. The average turnround is 6 hours, and out of 720 jobs, only the first 60 are run within an hour of submission.

In the second strategy, jobs are alternately scheduled for immediate running or 12 hour turnround. The average turnround is still 6 hours, but now 360 jobs, half of the total, are run immediately they are submitted. The remainder are delayed for 12 hours. This scheduling strategy is only sensible if it is known that the submission rate varies. It relies on a reasonable prediction of future submissions. As will be described below, the Cambridge scheduler also works on the basis of future load predictions. In passing, it should be noted that no scheduling strategy is much use if the submission rate is constant at all times. In these circumstances the queue will either grow without limit or diminish to nothing, except in the unlikely case when the submission rate equals the processing rate.

The fact that there are many users competing for processing resources is incorporated in the scheduling system by the use of a unit price for processing resources which varies according to the demand on the machine

as well as according to the amount of work the project has recently run. This price is not related to monetary values, either real or artificial, in any way. It is simply a factor for converting resource units into charging units. (The word priority, which is a misnomer, is used in other Cambridge documentation for historical reasons.) The unit price of a job always lies in the range 1-255 inclusive.

The price for processing a unit of work for a given project at a given time depends not only on the current level of demand for resources, but also on the amount of work recently run on the project. Thus an underloaded project may be able to run very cheaply even in prime shift, whereas an overused project will find any computing whatsoever relatively expensive. The details of how this is achieved are described in the next section.

Technical Description

"Merely corroborative detail intended to give artistic verisimilitude to an otherwise bald and unconvincing narrative."

(W.S. Gilbert)

All jobs submitted to the 370/165 begin with a group of statements called a job description. This contains information about the job such as the user identifier under which it is to be run, the magnetic tapes it uses, and so on. A complete description is given in appendix E; we shall consider here the three parameters which relate to job scheduling. These are as follows:

C1 - CPU limit for the job in seconds

The job may not use more CPU time than this limit, which may be specified in seconds or in minutes. A default of five seconds is assumed if no value is given.

P1 - unit price limit for the job

This value, which must lie in the range 1-255, specifies the maximum unit price the user is prepared to pay for the job. If it is not specified in the job description a default value is used which varies inversely with the size of the job according to the formula

P1 = 1275/C1

within the limits 1-255. By specifying a maximum unit price, the user is able to indicate to the system the relative urgency of particular jobs. The default formula assumes that large jobs (probably production runs) are less urgent than small ones (probably tests).

The third scheduling parameter is the turnround requested for the job. The user may specify this in a number of ways such as

```
TURNROUND 2 HOURS
TURNROUND BY 1700
```

A common request is OVERNIGHT, which is a synonym for BY 0600. The user may specify turnround AT a particular time, even though this may cost more than turnround BY the same time. Use of this facility is not however encouraged, and in any case, there is never any guarantee given. The default turnround, which may also be explicitly requested, is

TURNROUND NOW

The basic action of the scheduler is to compute a unit price for the job, and this depends on the turnround requested, the current or expected load, and the amount of recent work on the project. The following general rules apply:

- The larger the project's allocation, the lower the unit price
- The greater the recent usage of the project,) the
  The greater the load on the system,) higher the
  The bigger the job,) unit price

When a job is scheduled for 'immediate turnround' it is placed at the end of the so-called 'fast queue', which exists at the head of the main job queue. Most jobs on this queue begin execution within one minute of submission (see appendix C).

In the case of a turnround request relative to the time of submission or a request for turnround by a given time of day, the job is scheduled to run at a time between the time of submission and the requested turnround time which results in the lowest price. If this is achievable at a number of times, the earliest is chosen. Users can inspect the turnround estimates for jobs in the queue by means of a Phoenix command or from workstation consoles. When it schedules jobs for future running, the scheduler modifies the expected load estimates to take account of changes in the load from previous days.

To implement the scheduling algorithms the system keeps the following quantities for each project:

S - the number of shares

S is a number allocated by the management, and is a measure of the proportion of processing resources available to the project. The proportion is not relative to the total number of shares issued, but to those belonging to currently active projects. Typical values for S lie between 1 (for an undergraduate hobbyist project) and several hundred (for a heavily used research project). For most projects S lies between 10 and 100.

U - the current usage

U is a measure of the current rate of work on the project. Whenever processing resources are used, U is incremented by the product of resource units times unit price, which is termed the charge for the job. In addition, the system decays U with time, and so resources consumed in the past have a smaller and smaller effect as time goes on. The decay takes

place only when the 370/165 is running, and is such that for an unused project, U decreases by about 10% per day.

W - outstanding work

W records the estimated charge for outstanding work held in the job queue. Whenever a job is accepted by the system, W for the relevant project is incremented; whenever a job is run W is decremented, and the actual charge added to U. The two quantities appear as U+W in the scheduling formula, thus ensuring that outstanding work is taken into account when analysing scheduling requests. The estimation of the size of jobs is very crude. Users are required to specify CPU and main store limits for jobs, but not channel time. Observation of the actual workload shows that, on average, the number of resource units used by a job is very roughly equal to its CPU usage in seconds. Hence the value of the CPU limit (C1) is used as an estimate of a job's resource usage.

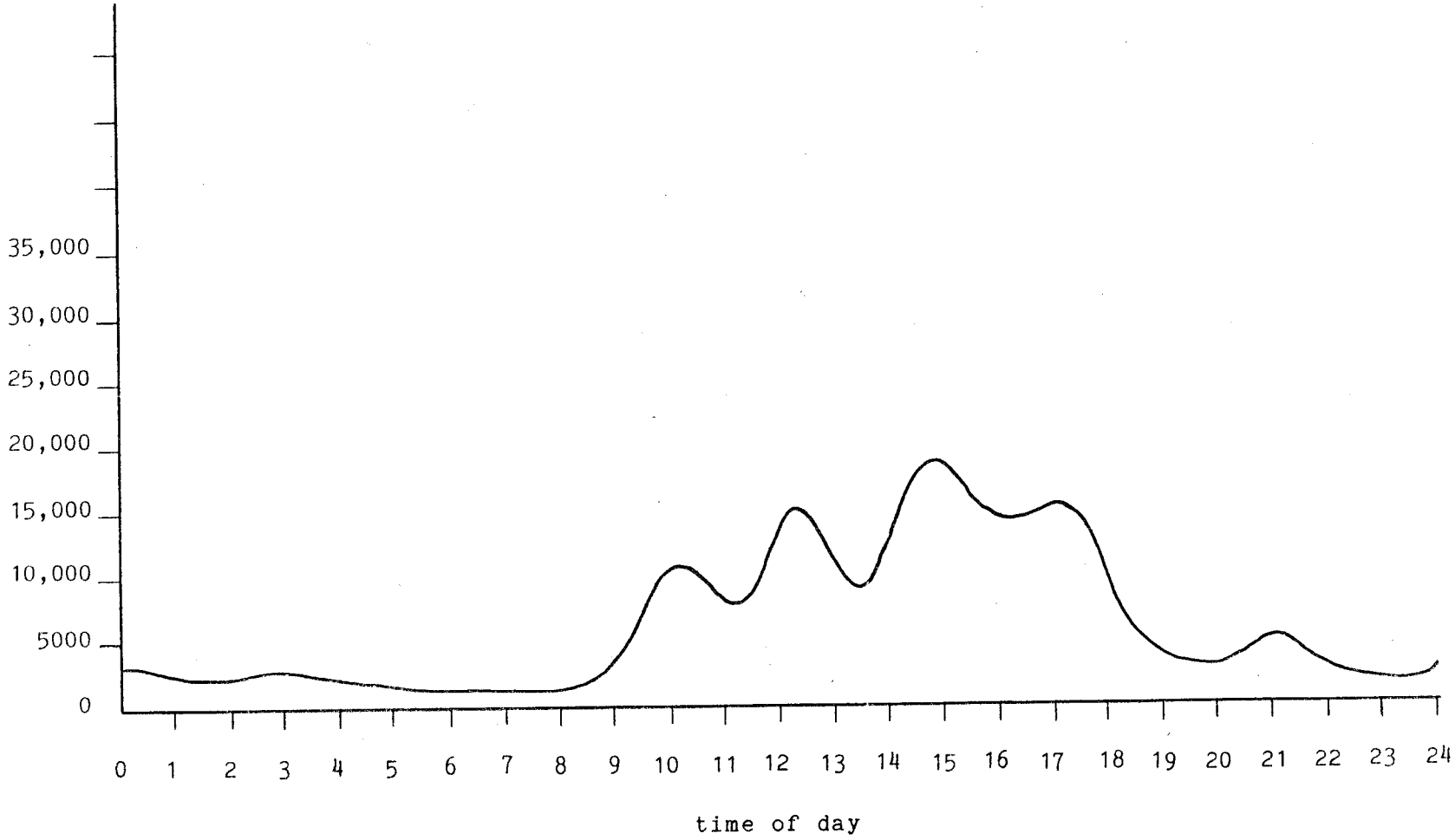
V - resource use per share

V is a measure of the value a project is getting for its shares. It is an informational quantity only, and does not appear in the scheduling formula. V is incremented by the resources used (not multiplied by price) for every job run, normalized by the number of shares. It decays in the same way as U. A project which is used only in prime shift will have a fairly low V, whereas an identical project with the same value of U used only at night may have a much higher value. The value of V gives an indication to the management whether a project is being used sensibly for the work for which it was allocated. The values of S, U, W and V for the relevant project are printed in the output for every job, and may also be inspected from a Phoenix terminal.

In addition to the quantities described above, the system maintains a value for the current demand for processing resources and a vector containing the expected demand over a 24 hour period. This dimensionless quantity is called N, and it has a granularity of 15 minutes. Linear interpolation is used to obtain values of N within the 15 minute intervals. The expected demand is based on recent history, and is modified according to the actual demand encountered. A graph of typical N values is shown in figure 2. From this it can be seen that most users' habits fall within the social norm. They start work at about 9.30 a.m., take a coffee break at 11 a.m., lunch from 1-2 p.m., really get into their stride in the afternoon, but start going home soon after 5 p.m. This is to some extent due to the fact that dedicated periods for system development occur fairly regularly at 5.30 p.m. Some users return after dinner to work in the evening.

In periods of very high overall demand the N values increase, but the general shape of the curve remains the same. Average daytime values of N in term time are around 20,000 to 30,000, peaking at times to 60,000. Night time values are normally in the low thousands. The algorithm for maintaining N takes into account work scheduled for the future, and the amount of work previously delivered at given times of day. This historical

Figure 2: Variation of N over a 24-hour period



data consists essentially of the amount of CPU time delivered to jobs in each 15 minute period.

The unit price for a job is computed by the formula

$$P = \frac{(U + W + k*S) * Np}{100,000 * S}$$

For jobs requesting immediate turnround, Np is the least value predicted for N in a period starting from the time of submission, and lasting the expected running time of the job, estimated as twelve times the CPU limit. For other jobs, Np is the predicted value of N for the time at which the job is scheduled.

If the formula yields a value of P which is greater than the unit price limit (Pl), either as specified in the job description or defaulted, the job is not accepted for the requested turnround, and an alternative scheduling strategy is adopted, as described in the following section. If the formula yields a value of P which is less than one, P is set to one.

The quantity k is a constant which currently has the value 50. Without the k*S term a project that is very underloaded (small U+W) is unable to attain anything other than immediate turnround (except by the use of TURNROUND AT, which is discouraged) as the formula yields a unit price of one for all times of day. Immediate turnround is not always wanted; a user may wish to submit a job for overnight running which uses data files to be created later in the day, for example.

Conceptually, the value of W is incremented to include the current job prior to the computation of P. However, as the increment for W depends on P, the scheduling formula is more correctly written as

$$P = \frac{(U + W + P*Re + k*S) * Np}{100,000 * S}$$

where W is now the previous value of W and Re is the estimated number of resource units for the current job. Solving for P yields

$$P = \frac{(U + W + k*S) * Np}{100,000*S - Np*Re}$$

Note the singularity in P for sufficiently large jobs. Its meaning is that there are some jobs which should never be accepted; nevertheless they <u>are</u> accepted, and given a very long turnround, as described below.

The revised formula for P also shows that a project can be in a state where small jobs are accepted for running at a given time of day, whereas large jobs are not. Another important feature of the formula is that jobs for different projects running at the same time are not necessarily charged the same unit price, because of the dependence on U+W and on S. Both the general user demand for resources (N) and the project's own recent usage relative to its shares (U/S) are taken into account.

For a project in continuous use, U is typically very much greater than W and k*S, and Np*Re is very much less than 100,000*S. The formula then approximates to

$$P = \frac{U * Np}{100{,}000 * S}$$

Since U decays at a constant rate (approximately 10% per day), the rate of charging that can be sustained by a project while keeping U constant in the long term is proportional to U. Hence the rate of work, which is the rate of charging divided by the unit price, is independent of U in a steady state, but proportional to S/Np.

Turnround Alternatives

"If at first you don't succeed, try, try, again."

(W.E. Hickson)

If the result of applying the scheduling formula described above is a value of P greater than the unit price limit (which itself cannot be greater than 255), then an alternative attempt at scheduling is tried. The default turnround request for this is 3 HOURS, but the user may specify different values, for example

```
TURNROUND NOW OR OVERNIGHT
```

The default value of 3 hours is chosen because

(a) It is not a fixed time of day, such as OVERNIGHT (which was originally used).

(b) At most times of day there is a descending part of the N graph within 3 hours, and so there is the expectation that N will fall below its present value.

(c) It seems sensible from a human point of view.

If both attempts at scheduling fail, then the job is scheduled for the earliest possible time at its price limit, that is, as soon as the value of N is expected to fall low enough. Thus the statement

```
TURNROUND NOW OR NOW
```

can be used to get the best possible turnround at the given price limit. If the job cannot be scheduled at all in the current 24-hour period, it is given a multi-day turnround. Such turnrounds are not catered for directly in the scheduling algorithm, and hence special mechanisms are provided to deal with these cases, which are in general a small proportion of the load. The maximum permitted turnround is 58 days, which coincidentally is the length of a Cambridge Term.

One further turnround request is REJECT, which can be used to cancel a job if the first turnround alternative cannot be met, for example

TURNROUND 45 MINS OR REJECT

This is the only circumstance in which jobs are not accepted by the scheduler.

Summary of the Scheduling System for Jobs

For each project:

S = number of shares
U = current usage, decaying with time
W = outstanding work in job queue
V = resources used per share, decaying with time

For each job:

Re = estimated resources
R = actual resources
P = unit price (1 <= P <= 255)
Pl = unit price limit

The user specifies Re implicitly, via a CPU time limit (which defaults to five seconds). Pl may be specified explicitly; if not it defaults according to job size such that smaller jobs have a higher default unit price limit. The user may also specify a scheduling request in the form

TURNROUND a OR b

where a and b are of the form

NOW
OVERNIGHT
n HOURS/MINS
BY time-of-day
AT time-of-day
REJECT

The defaults, which are independent of each other, are

TURNROUND NOW OR 3 HOURS

At All Times:

The current demand on the machine (N) is computed and used to update the expected demand in the future. The amount of work delivered by the machine is used to update the history data. Details of this process are given in appendix F.

Projects' U and V values decay at the rate of about 10% per day. This is how it appears to the user; in fact the values are only updated when they are accessed.

At Job Submission:

A unit price P for the job is computed according to the scheduling algorithm described above, and the job is scheduled for the appropriate turnround. Re*P, the estimated charge for the job, is added to the value of W for the project.

At Job Termination:

The number of resource units used by the job (R) is computed according to the formula in appendix B. The charge for the job (R*P) is added to the project's U value; Re*P is subtracted from its W value, and R/(20*S) is added to its V value.

Scheduling Time-sharing Sessions

"The beast with many heads butts me away."

(Shakespeare)

A time-sharing session is, broadly speaking, treated as a job with immediate turnround for scheduling and charging purposes. The same share allocation is used, and work done affects the turnround and/or unit price of jobs. Equally, work done by jobs affects the unit price of time-sharing sessions.

When a user attempts to logon to the system, the scheduler computes the unit price that would be required for the project to run a job with a CPU limit of ten seconds and immediate turnround. If this is less than or equal the maximum permitted unit price (255), the user is allowed to log on at that unit price. Otherwise the logon attempt is rejected. The user is permitted to specify a unit price limit less than 255 if he wishes; this facility is rarely used in practice.

As the session proceeds, whenever ten seconds of CPU time have been used, or 30 minutes of real time have elapsed, the resources used since the last such occurrence are multiplied by the current unit price and added to the project's U value. The session is then re-assessed by the scheduler in the same way as at logon. This may result in an increase or a decrease in the unit price, depending on the demand on the machine (and the project's U of course). If the unit price has risen above the limit, the user is given a warning message, and allowed a further three seconds of CPU time or five minutes of real time before being logged off. Resources used in this state are charged at the maximum unit price.

The reason for re-assessing the session every 30 minutes if less than ten seconds of CPU time have been used in that period is to ensure that the unit price for the session is kept approximately current. At some times of day N increases or decreases substantially over this time scale (see figure 2).

When a user logs off, the resources used since the last assessment are charged to the project. If the logoff was forced by the unit price reaching its limit, one last attempt is made to see if the session may continue. At certain times of day, when N is decreasing rapidly, or after a temporary peak in the value of N, this may succeed. In addition, the user may have cancelled outstanding batch work on the job queue, and so decreased W. The user is then told that his session has been reprieved, and it continues at the new unit price.

The time-sharing user need not be aware of the scheduling activity which occurs periodically during the session. However he may choose to be told whenever the unit price changes to a value above a given limit. Setting this limit to zero causes all changes of unit price to be notified; the limit is initialized at the start of a session to a value of 100.

A user who has access to more than one project may request a change of project in the middle of a session. If the usage on the new project is too high to allow immediate turnround, the change of project is rejected, and the session continues on the old project, possibly at a new unit price. Otherwise, the resources used since last assessment are charged to the old project, and a re-assessment takes place using the new project.

The current values of the unit price, CPU and channel time used, resources used and total charge for the session may be inspected by the user at any time.

Sustaining Scheduled Turnround

The turnround times allotted to jobs when they are submitted are not guarantees, but predictions. Like all predictions they are more accurate if the future behaviour of the parties involved does not differ radically from past behaviour. In the case of the job scheduler there are two factors involved:

. the rate of submission of work
. the rate of processing work

The former is dependent on user behaviour, which has proved to be a reasonably consistent function of time of day. Long term fluctuations, related for example to terms and vacations, and variations within the week, are less consistent. The rate of processing is more accurately predictable (though not constant; it is higher at night) except in unforseen circumstances such as a system failure.

The system monitors its own performance with regard to the accuracy of turnround predictions, and makes adjustments to deal with unexpected circumstances. A small overload or underload is handled by varying the rate at which the predicted turnround times decrease. Thus a job which is initially scheduled to run three hours after submission may have a turnround estimate of two and a quarter hours one hour later if there has been an unexpectedly heavy load in the meantime.

If the system finds that its predictions for immediate turnround have become seriously wrong, then all jobs other than those in the fast queue are re-scheduled. Jobs in the fast queue are those that were scheduled for immediate turnround when they were submitted, together with those whose allotted turnround time has expired. Allotted turnrounds for the remaining jobs are thus lengthened, and some may be moved from the day time into the night time, while others already scheduled for night time running may be deferred for 24 hours. Such action is inevitable after prolonged system failure; frequent occurrences when the system is running normally would indicate a lack of tuning in the scheduler. In practice, re-scheduling during normal operation is a rare event.

The scheduler takes special action to deal with certain expected dramatic changes in the pattern of demand, such as occur at around 9.30 a.m. each day.

Scheduling Overheads

The scheduling system would not be practical if the cost of running the algorithms were anything other than negligible. There are no figures available for the cost of the scheduler itself, but the resources used by the entire HASP system, which includes the scheduler, job management and unit record spooling, are less than 5% of the total. It is estimated that the scheduling overhead on the 370/165 is less than 1%.

Scheduling Problems

While the scheduling system in its current form has on the whole succeeded in the task of sharing out processing resources, one or two problems remain to be solved.

The major human factors problem is the rapidity with which U can increase once the project becomes seriously overused. Time-sharing users in particular tend to continue to log on while they can, even when paying the maximum unit price, so that by the time they are thrown off their U is very large and takes a long time to decay to a normal level. It is however possible for the management to reset U manually. This is done from time to time when the circumstances warrant it.

From the scheduler's point of view the problems are concerned with maintaining accurate predictions of turnround times. The system is sometimes rather slow to react to large fluctuations in user demand such as occur at the start of an academic year or immediately before Christmas, when there is a sudden drop in demand. In the former case unit prices are too low and turnround predictions are too short, while in the latter case the opposite is true. Users complain equally vociferously, whether the predictions are over-pessimistic or over-optimistic. The system always runs jobs from the head of the queue, whatever their predicted turnround.

A particular problem occurred when the Computing Service started running a Saturday daytime shift. The scheduler keeps N on a 24-hour basis only, so its predictions for Saturday are not accurate, since the pattern

of usage is not the same as on weekdays and there is no evening or night shift to follow. It might appear that the way to solve this problem is to keep N values for each separate day of the week. However, this would mean that changes in the overall load would take much longer to feed back into the history, and errors such as occur at the start of term would last for weeks rather than days. The most ambitious proposal for improving this area involves keeping a two-dimensional history of N in the form

H(day,time) = H1(day) * H2(time)

with H2 changing rapidly with variations in load, but H1 changing only slowly. This scheme is not likely to be implemented in the near future.

DISC SPACE CONTROL

"Annual income twenty pounds, annual expenditure nineteen nineteen six, result happiness. Annual income twenty pounds, annual expenditure twenty pounds ought and six, result misery."

(Charles Dickens)

The mechanism for sharing permanent disc storage amongst the user population is entirely separate and unrelated to the scheduling system described above. Separate allocations are made by the management, and shares cannot be traded for disc space or vice versa. The system is a flexible one which works on the principle of controlling average disc usage, and providing an incentive for the removal of unwanted material from the discs.

Under the OS/MVT operating system, disc space may be permanent or temporary. Files in the former category remain in existence when the job or session that created them terminates, whereas temporary files are automatically deleted at the end of a job or session. The amount of temporary disc space used by a job or session is neither limited nor accounted, except insofar as the use of channel time to access the disc will add to the processing resources used by the project. The system described below applies therefore to permanent disc space only, for files in the public filing system. A very few individuals are permitted the use of private disc packs (200 megabytes each) for particularly large amounts of data. Such disc space is excluded from the normal control mechanism.

Allocations of disc space are made to notional entities called filespaces. All the files in a given filespace have names beginning with the unique filespace name. In many cases filespace names are the same as user identifiers, but there are also many filespaces that do not correspond to an individual user, though every filespace has an associated manager who is himself a user. A group of users may request the allocation of a filespace for their communal use, as well as individual private filespaces, and there are of course a number of library filespaces containing files for the use of the public at large.

The OS/MVT system contains no suitable facilities for file protection among a large number of independent users. A local facility has been implemented which prevents write access to any file in a given filespace by users other than the filespace manager or those authorized by him. No protection against unauthorized reading of files is provided by the system.

The smallest file that can be created is of length one track, which can contain up to about 13,000 characters. File sizes may be specified in tracks or cylinders (1 cylinder = 19 tracks); however, all accounting is done in tracks.

Each filespace has associated with it two quantities determined by the management, the Quota (Q) and the Limit (L). Both are in units of tracks.

The Quota is an allocation of an amount of disc space which the filespace may not exceed on average. The Limit is an absolute limit on the amount of disc space in use by the filespace at any time. For most filespaces L is set equal to five times Q; however, a fixed allocation of space can be achieved by setting L equal to Q. This is appropriate for some system libraries.

For each filespace the system maintains the following quantities:

C - current disc use

This is the amount of disc currently occupied by permanent files in the filespace. It is updated whenever files are created, deleted, extended or compressed.

Rc - current disc use ratio

This quantity is defined as C/Q, and is the proportion of the quota currently in use.

A - average disc use

This figure is approximately the amount of disc space occupied on average during the last 200 hours of running, that is, about 10 working days. It is a function of time, defined formally by the integral

$$A(t) = 0.005 * \int_{-\infty}^{t} C(s)*EXP(-(t-s)*0.005)\ ds$$

where t measures Computing Service running time in hours. The value of A is updated by the system whenever files are created, deleted or extended in the filespace, and whenever the filespace parameters are inspected.

Ra - average disc use ratio

This quantity is defined as A/Q, and is the proportion of the quota used on average during the last 200 hours of running.

Control is exercised over filespaces in two ways:

1. Any attempt to create or extend a file which would cause C to exceed L is faulted. This prevents a filespace from monopolizing the system.
2. If the average disc use ratio (Ra) becomes greater than 1.0, that is, if A becomes greater than Q, the filespace is said to be inhibited. In this state the creation, extension, or updating of files in the filespace is prohibited. Files may still be read, but the only permitted changes to the filespace are deletion and compression.

The second control provides a continuous incentive to a user to remove unwanted files from the disc as soon as possible, in order to keep down

the value of Ra. The lower the value of Ra, the longer the filespace may exceed its quota without inhibition.

Once a filespace has become inhibited, files must be deleted or compressed until the current use ratio (Rc) is less than one. Then a recovery period is necessary while the average use ratio (Ra) decreases. When it has fallen to a value of 1.0 the filespace ceases to be inhibited. The lower the value of Rc, the more quickly the value of Ra will fall. It is possible for the management to reset Ra manually when special circumstances require it.

Summary of Disc Space Control

Q = Quota, a limit for average disc usage
L = Limit, an absolute limit
C = Current use
Rc = Current use ratio, C/Q
A = Average use over last 200 hours
Ra = Average use ratio, A/Q

C is never allowed to exceed L; the filespace is inhibited at any time when Ra is greater than one.

RESOURCE ALLOCATION

"Take care of the pence, for the pounds will take care of themselves."
(The Earl of Chesterfield)

In this section we discuss the means by which resources are allocated to individual users. The resources in question are processing resources, allocated as a number of shares, and disc space, allocated as a quota of tracks. There are, of course, other resources such as magnetic tapes, punched cards and special output forms which must be allocated, but these are not included in this discussion. Many of these are not scarce, and so their use is not formally controlled.

The two resource control mechanisms were developed because of the need to share computing resources amongst a large number of people in an environment where the use of real money was not favoured. Much of the work run on the 370/165 forms part of research projects for which it is very difficult to specify future computing needs. A system which allocates, say, a fixed number of CPU minutes per month, can be very unstable in such circumstances, as many users would tend to use resources very sparingly at the beginning of the allocation period and very profligately towards the end. Unless a complicated scheme of overlapping allocation periods were used, the demand for resources would become very uneven.

At Cambridge, processing resources are allocated as a rate of work relative to the rest of the active user community, while disc space is allocated as an average quantity. Controlling users' rates of work relative to each other has other advantages over a fixed allocation scheme in addition to stability. It means that a user can never run out of his allocation; if he is over-using it he is slowed down, but not stopped. It also means that unused allocations are not wasted.

Given the two control mechanisms, there must be administrative procedures for allocating shares to projects and quotas and limits to filespaces. In many universities resource allocation takes place in a top-down manner, with large proportions being allocated to faculties, who then subdivide into departments, and so on down. There are a number of disadvantages in this approach:

(1) Someone has to do the allocation at each level; at the higher levels it is likely to be a committee. Thus resource allocation can take on a political aspect.

(2) The individual user is at the bottom of a hierarchy of allocators, leading to potential delays.

(3) Individual users in one department may not receive the same treatment as comparable users in another department.

(4) Since the high-level allocations tend to be made on the basis of past use, computationally sophisticated departments continue to get the major slice of resources, while departments for which computing is a

relatively new activity have difficulty in obtaining allocations.

(5) Allocations at the highest levels tend to be made at relatively infrequent intervals, sometimes as long as a year. This means that changes in requirements on shorter timescales cannot easily be accommodated, and in addition there are undesirable discontinuities when the high level allocations are adjusted.

In summary, errors made at a high level cannot easily be rectified; there is a possibility of wastage, and the users can feel that they are at the mercy of a bureaucracy.

A contrasting approach, which has been adopted at Cambridge, is to allocate resources directly to the individual user, or in some cases to small groups of users. This scheme has the following advantages:

(1) All users are subject to the same control, which is administered by the Computing Service. Although this makes more work for the Service, it has the benefit of keeping Service staff much more in touch with what users are doing. It also does away with replication of effort in many departments.

(2) Errors made in individual allocations are small in relation to the total and can easily be adjusted. Users are encouraged to approach the Service whenever they feel their allocation is inadequate. The general policy is to begin by allocating small amounts of resources and then to increase them later as necessary.

(3) Users from departments with no previous computing requirements are accommodated without any special treatment.

We shall now describe some of the practical aspects of how bottom-up resource allocation actually works. While in principle any potential computer user may apply directly to the Computing Service for resources, in practice he is expected to consult the User Representative in his department (or other user group) first. User Representatives are experienced computer users in the relevant academic disiplines, and can advise as to whether what is being proposed is reasonable from the point of view of that discipline. Some departments with a lot of users have more than one User Representative, while on the other hand users from non-computational departments may have to discuss their requirements directly with the Computing Service.

A user's application for resources, having been approved by the User Representative, is handled by the User Services staff of the Computing Service. Many cases are straightforward, and the resources are allocated forthwith; in other cases the user is invited to discuss his requirements with Service staff, and technical advice from software experts is sought if necessary. The ultimate authority for resource allocation is the Director of the Computing Service, acting on behalf of the Computer Syndicate, a management committee responsible to the University Authorities. A user in dispute with the Service could in principle appeal to the Syndicate, though this has never occurred in practice.

Most of the projects are connected with research, and at the start it is not possible to predict what the ultimate requirements will be. The approach taken is to allocate a small amount of resources, and encourage the user to return if and when he finds his allocation inadequate.

Users do normally communicate with the Computing Service when they feel they have inadequate resources. On the other hand, they are unlikely to do so if they have received a larger allocation than necessary. This is not however a serious problem, for two reasons. The first is that individual allocations are small in relation to the total usage, and the second (which applies to processing resources only) is that there is in practice a steady inflation in the value of shares. This is due to the fact that both the number of computer users and the total usage are continually increasing, and that under-allocated users continue to request increases. Thus the total number of shares in existence is increasing, lowering the value of an individual share as time passes. Provided the rate is low (which it is), share inflation is seen as beneficial to the community, as it ensures that allocations that are in heavy use are regularly reviewed, while those that are not in use slowly diminish in value. Note, however, that the value of a share is not absolute, but relative to the resources available. If the system is enhanced to provide more computing power, all existing projects automatically benefit.

When a user does request an increase in his allocation he is required to justify the request. For small increases there is usually no problem; however, large requests are scrutinized to ensure that the user is making the best use of the machine that he reasonably can. This does not carry any implication of blame for the user. Most of the users are not computer professionals, and cannot be expected to be fully aware of all the facets of computer programming. In a number of cases Computing Service staff have been able to assist users to modify their programs so as to use substantially fewer resources; one particular example is recounted in appendix D. This benefits the community as a whole by freeing resources for other users. Assisting users to improve the performance of their programs can naturally be done with any allocation system. However, the policy of allocating a small amount initially, coupled with the gradual inflation in the value of shares, encourages regular reviews of large projects.

Summary

Computing resources are allocated directly to individuals or small groups, taking the advice of an experienced user in the academic department, and of software personnel if relevant. Users are encouraged to request increases if they feel their resources are inadequate, and this triggers a review of their computer use. This iterative aproach is appropriate in an environment where the actual resources required by many projects are impossible to forecast in advance, and where the users are not computer professionals. There is no attempt to allocate resources at department or faculty level.

This bottom-up approach to resource allocation contrasts strongly with the top-down strategy adopted in many other universities. The main points of comparison are as follows:

> A top-down approach, while apparently politically fair, has the danger of becoming an issue amongst the politicians and being influenced by extraneous considerations. At the higher levels the decisions are big ones, and so require appropriate committees and safeguards. Errors can have a large impact. The approach is also inflexible.
>
> A bottom-up approach is flexible for the user, and since all decisions are small ones, they can be made relatively easily. Errors do not have a large impact. This approach can only work, however, where political considerations allow it.

It must not be assumed from the above that the bottom-up method of resource allocation is without problems. With over 3,000 users to cater for there are bound to be mistakes, and personality differences between users mean that one may delay requesting an increase in his resources until he is desperate, while another may request a large increase as soon as he feels at all restricted. In particular, users familiar with top-down allocation methods are often reluctant to request a review of their resource allocation when it is needed. A tendency to accept the Computing Service's initial allocation as unchangeable can give rise to the feeling that the Service is inflexible, or that the system is more overloaded than is in fact the case. Continued publicity is necessary to overcome this problem.

No allocation system can ever be perfect, but it is felt that the bottom-up method is fairer and more flexible, leading to a high level of social acceptability.

CONCLUSION

"No society can surely be flourishing and happy, of which the far greater part of the members are poor and miserable."

(Adam Smith)

We have described control mechanisms and an associated allocation mechanism for sharing computer resources amongst a community of several thousand university users. The important features which characterise this environment are the lack of deadlined jobs and the unpredictability of future demand on individual projects.

The control mechanisms do not use fixed allocations. Instead, control of a project's rate of work in relation to other active projects is maintained, while disc space is controlled on an average basis. It is felt that these flexible systems, while appearing more complicated than traditional schemes, are nevertheless more appropriate in the university and similar environments. An important feature of the control mechanisms is that they provide incentives and opportunities to the users to manage their resources, for example by removing unwanted material from the discs as soon as possible. This is very necessary in a community where most users work as individuals and are self-motivated, and where a formal management structure is to a large extent absent.

The allocation of computer resources is handled directly by the Computing Service at the individual user level. This contrasts with similar installations where a top-down, hierarchical approach is often taken. Because of the inherent unpredictability of a project's computing requirements, allocation is essentially iterative, with the users interacting with Service staff as their requirements change or become better known. This provides an opportunity for professional computer staff to give advice on the use of computing facilities.

Any schemes for resource allocation and control can only work in practice if they prove acceptable to the majority of users. At Cambridge we believe that social acceptance has been achieved to a high degree. Frequent consultation with representative users has played a large part in this achievement.

The results of two recent questionnaires confirm the acceptability of the system. The first originated from the Government, and was concerned with the possibility of introducing partial real money charging. This was sent to all departments, who were also asked to give their opinions on the existing arrangements for resource allocation. The Council of the School of Physical Sciences, a body responsible for the major computer-using faculties, reported that its departments 'considered the share system superior to any departmental allocation scheme'.

The second questionnaire was circulated by the Computing Service to all departments as part of an evaluation of future needs. One of the questions asked the respondents to list those aspects of the present system they would like to be retained in the future. A significant number of departments included the job scheduler and the disc space control system among their requirements.

ACKNOWLEDGEMENTS

I am very grateful to Judy Bailey, David Hartley, Barry Landy, Anthony Stoneley and Chris Thompson for reading and re-reading early drafts of this report. As well as pointing out factual errors and misconceptions, their critical attention to the text enabled me to make numerous improvements.

REFERENCES

[1] J. Larmouth, 'Scheduling for a share of the machine', Software - Practice and Experience, **5**, 29-49 (1975).

[2] J. Larmouth, 'Scheduling for immediate turnround', Software - Practice and Experience, **8**, 559-578 (1978).

APPENDIX A: HISTORICAL SUMMARY

"All the ancient histories ... are just fables that have been agreed upon."

(Voltaire)

The resource control mechanisms described in the main body of this document are the result of a number of years of evolution and experimentation. The following is a brief summary of the sequence of events which relate to resource provision and control on the 370/165.

Jan 1972: 370/165 commissioned; one megabyte IBM core store, one byte multiplexer channel, two block multiplexer channels, 3330 discs.

Mar 1972: User service begun, batch only, continous turnround increments, usage decayed once per day. File space control based on Titan system (income, limit, bound) specified but not enforced. Resource unit based on CPU time and store estimates only.

Apr 1972: File space control enforced, daily accounting.

May 1972: PDP 11/20 installed as front-end for communications, both synchronous and asynchronous; software to be developed locally.

Jul 1972: Job store limit enforced.

Jan 1973: Experimental interactive service begun, Phoenix control language, no resource charging or control, users restricted to Service supplied programs (EDIT, SUBMIT, etc.). Initially invited users only, limited schedule (day shift only). Maximum simultaneous sessions 8.

Mar 1973: Actual resources used by jobs incorporated in scheduler instead of estimated resources.

Jun 1973: Phoenix generally available, 100 connected lines, still a limited schedule.

Sep 1973: Second megabyte of memory (AMS semiconductor) installed.

Oct 1973: Titan computer closed down. Phoenix schedule 08.30 - 23.59 daily. IBM 2305 fixed head disc installed.

Dec 1973: 34 simultaneous Phoenix sessions, Phoenix available whenever Computing Service running. Third block multiplexer channel installed, dedicated to the fixed head disc.

Feb 1974: Batch Monitor using Phoenix language in use for student teaching jobs. Flat rate charge per Batch Monitor job.

Apr 1974: TURNROUND statement introduced.

May 1974: Optional passwords in job descriptions.

Jun 1974: Batch Monitor available to all users, renamed Phoenix Monitor. Restricted set of Phoenix commands, LIST, WATFIV, EDIT, etc. Limited to jobs less than 120K and 2 sec CPU. Still a flat rate charge per Phoenix Monitor job.

Aug 1974: I/O timing implemented. Flat rate charge for tape deck use added to resource unit formula.

Oct 1974: 40 simultaneous Phoenix sessions. Standard Phoenix jobs introduced as an alternative to JCL jobs.

Nov 1974: PDP 11/40 installed to enhance communications front end.

Feb 1975: 48 simultaneous Phoenix sessions.

Mar 1975: I/O time limit of 10 seconds imposed on Phoenix Monitor jobs.

Apr 1975: Automatic logging off of Phoenix terminals which are unattended for more than 10 minutes.

May 1975: Scheduling variable V implemented. Phoenix monitor CPU limit raised to 3 sec. Restrictions on commands lifted - any job using less than 120K, 3 seconds CPU and 10 seconds disc i/o time can request processing via the Monitor.

Jun 1975: Experimental fast turnround job class, available on user request but charged at high rate.

Aug 1975: Phoenix Monitor limits raised to 5 seconds CPU and 15 seconds i/o time. Phoenix jobs within these limits automatically scheduled for Monitor processing.

Sep 1975: 64 simultaneous Phoenix sessions.

Dec 1975: 80 simultaneous Phoenix sessions, achieved by replacement of TCAM by Parrot.

Jan 1976: Revised job scheduler based on maximizing immediate turnround. Third megabyte of memory (AMS semiconductor) installed.

Feb 1976: Phoenix Monitor jobs charged for resources they actually use instead of a flat rate. CPU limits less than the maximum (5 sec) enforced for such jobs.

Aug 1976: Introduction of k*S into the scheduling formula.

Dec 1976: 90 simultaneous Phoenix sessions. File write protection system introduced.

Jan 1977: Charging for resources used in Phoenix sessions and logon control as part of the share system. Online passwords.

Feb 1977: Removal of restrictions on programs run in Phoenix sessions, 44K swapped region available for command system data (typically 8-10K) and arbitrary user programs.

Apr 1977: SAVE/COLLECT facilities introduced for moving jobs' output from the spool disc to Phoenix sessions. New resource unit formula incorporating a charge for I/O time etc. Store term f(x)=(x**3)/20.

Nov 1977: Two-region working for non-interactive steps run in a Phoenix session. Restricted to 5 sec CPU time, 15 sec I/O time and 130K.

Dec 1977: Phoenix Monitor disc limit raised to 25 sec.

Mar 1978: Continuous decay of U and V instead of once per day. Phoenix Monitor store limit raised to 200K.

Apr 1978: Two-region working for Phoenix jobs.

May 1978: Second region online limit raised to 200K.

Sep 1978: IBM core memory removed; two additional megabytes of AMS (now Intersil) memory added, bringing total to four megabytes.

Oct 1978: Processor speed-up (PSU) fitted giving approximately 20% increase in CPU power. New filestore controls based on average usage with continuous decay.

Jan 1979: Store-to-store swapping, 100 simultaneous Phoenix sessions.

Feb 1979: CPU limit for second region steps run in a Phoenix session raised to 60 seconds, I/O limit removed.

Jun 1979: Second region online limit raised to 250K.

Oct 1979: 110 simultaneous Phoenix sessions, store term in resource unit formula changed so that f(x)=x/10+(x**3)/40. Phoenix Monitor obsolete - all Phoenix jobs now bypass JCL interpreter, but special scheduling retained for jobs less than 5 sec CPU and 200K, and I/O limit of 25 seconds retained for such jobs. Passwords made mandatory for all access to the system.

Nov 1979: 120 simultaneous Phoenix sessions.

APPENDIX B: RESOURCE UNIT FORMULA

"What would life be without arithmetic, but a scene of horrors."
(Sydney Smith)

The formula used to compute the number of resource units used by a job or a Phoenix session is

$$R = 0.1 * \sum_i (10*Ci + 4*Di + 2*Ti + Ei*f(Si))$$
$$+ 0.1 * (H/100 + M*E + 3*P*E)$$
$$+ 0.1 * X * (5*N + J + A)$$

where the sum over i is a sum over all the steps of a job or session, and

Ci = CPU seconds used by step i

Di = channel seconds used by step i for disc access

Ti = channel seconds used by step i for tape access

Ei = 'execution time' for step i [1]

$$E = \sum_i Ei$$

Si = main store used by step i in units of 100K [2]

f(x) = x/10 + (x**3)/40 [2]

H = number of records spooled by HASP (input + output) [3]

M = (number of tape decks used)/(total number of tape decks) [4]

P = (number of private disc drives used)/(total number of private disc drives) [5]

X = 1 if a JCL job, 0 otherwise [6]

N = number of JCL steps

J = JCL statement count, including procedure expansions, but excluding comments

A = number of JCL DD allocations

Notes:

[1] 'Execution time' is notionally the real time a step would occupy store in an otherwise empty machine. This is not measurable on the 370/165, and the value Ci+Di+Ti is used to approximate it. This assumes no overlap of CPU and channel use within one job, which is

true for a large part of the load.

[2] Store becomes more expensive as larger amounts of it are used, since this cuts down the amount available to other jobs and may delay them because of fragmentation problems. (The 370/165 is not a virtual storage machine.)

[3] Includes card and paper tape input, and card, paper tape, plotter and printer output, whether to or from local devices or transmitted over a network.

[4] The total number of tape decks is currently ten.

[5] The total number of private disc drives is currently two.

[6] The resources used in JCL translation and interpretation are not included in the CPU and channel time measurements for jobs. This term is included to make the cost of running a program under JCL realistic compared with running it in a session or as a Phoenix job. Phoenix jobs bypass the JCL translation mechanism and use the Phoenix control language, whose resource usage <u>is</u> included in the normal measurements. Interactive sessions likewise make use of the Phoenix language.

APPENDIX C: STATISTICS

"There are three kinds of lies: lies, damned lies, and statistics."
(Benjamin Disraeli)

This appendix contains a number of miscellaneous statistics concerning the 370/165 system at the end of 1979. They pertain to a busy week near the end of term, and are presented in order to give some idea of the number of users involved and of the scale of computer operations. All the figures are therefore rounded.

Users

Number of registered users	3,500
Number of users active in one week	1,500

Projects

Number of projects	3,250
Number of projects active in one week	1,300
Number of shares allocated	120,000

The average number of shares per project is a misleading figure, as the distribution is very skewed. Some typical share allocations are

Hobbyist undergraduate	1-2
Undergraduate doing a computing course	5-10
Computer Science undergraduate	25
Research student	10-250
Research project	100-300
Computing Service staff	100-250
Major research project with many users	up to 1000

The distribution of shares to projects is as follows:

size	number of projects with S >= size	percentage
500	10	0.30
400	23	0.70
300	49	1.49
250	65	1.98
200	125	3.81
150	174	5.30
125	182	5.54
100	337	10.26
75	389	11.84
60	462	14.06
50	696	21.19
40	971	29.56
30	1266	38.54
25	1355	41.25
20	1828	55.65
15	1919	58.42
10	2887	87.88
5	2934	89.32
2	3098	94.31
1	3285	100.00

File Store

Number of filespaces	2,900
Disc space occupied (tracks)	100,000
Allocated quotas	150,000
Allocated limits	500,000

These figures include both user and system filespaces. As in the case of share allocations, the average quota has little meaning. Allocations vary from 2-5 tracks for small undergraduate projects to several thousand tracks for system libraries.

The distribution of file store quotas is as follows:

size	number of filespaces with Q >= size	percentage
3000	2 (system libraries)	0.07
2000	6	0.20
1000	16	0.54
500	44	1.48
250	82	2.76
200	122	4.11
150	148	4.99
100	267	9.00
75	316	10.65
50	517	17.42
40	612	20.62
30	804	27.09
25	922	31.06
20	1408	47.44
15	1599	52.53
10	2252	75.88
5	2759	92.96
1	2967	100.00

50% of all files are one track long, but they account for only 10% of the total disc usage. On the other hand, only 4% of all files are greater than or equal to one cylinder (19 tracks), but they account for about 50% of the total disc usage.

Phoenix Sessions

Sessions per week	12,000
Maximum simultaneous sessions	120
Resource units used in sessions per week	175,000

Jobs

Jobs per week	25,000
Submitted from Phoenix sessions	90%
Scheduled for immediate turnround	85%
Of those scheduled for immediate turnround,	
started within 1 minute of submission	80%
started within 5 minutes of submission	90%
started within 30 minutes of submission	99.9%
Resource units used in jobs per week	500,000
Resource units used by fast jobs	30%

APPENDIX D: IMPROVING USERS' PROGRAMS

"The object of teaching a child is to enable him to get along without teachers."

(Elbert Hubbard)

The following extract from the Computing Service Newsletter number 51 (October 1977) describes in light hearted manner an actual instance of a project's use of computing resources being drastically reduced as a result of technical advice from software staff.

A Cautionary Tale

Although the program did no wrong
It embarrassed us by running long.
All afternoon it sat in core
The overdue grew more and more.
When by even 'twas not done
After software 'twas re-run.
Five hours and more it did reside,
400K it took beside;
Full twenty mins. it did compute
With three hours disc I/O to boot.

And then, amid the wails and woes,
A mighty Service chief arose;
Full well for him the disc did turn,
At his behest the cores did churn,
Projects dear did he forsake,
This Augean task to undertake.
All day long the fray continued,
One by one the programs renewed;
Of disc I/O the hours were slain,
All sorry some two mins. remain,
In company with c.p.u.
Also chopped to minutes two.

How was this mighty feat composed?
What monstrous follies thus exposed?
O fie on manuals - such a way
The innocent to lead astray!
This user who in youth and keen
Some basic notions failed to glean.
Know ye then when numbers frisk
Twixt disc and core, and core and disc,
They go not freely on their own.
When one number flies alone
Enormous costs its pleasures blight,

But when it makes one charter flight
A thousand numbers are collected,
Great savings come to be expected.

The men who wrote the Fortran Guide
One great truth conspired to hide:
A Direct Access READ or WRITE
Incurs precisely one such flight.
DEFINE-ing FILE is also grim -
A thousand transfers reckon him!

Oh bitter irony to end on!
When purged of notions wholly wanton,
The user in the end conceded:
Direct Access was not needed.

APPENDIX E: JOB DESCRIPTIONS

"Give us the tools, and we will finish the job."

(Winston Churchill)

Every job sumitted to the Cambridge 370/165 must begin with a job description. This consists of a series of statements which list the resources required by the job, and identify the user and project for which it is to be run. Job description statements are written in free format, and consist of a keyword followed in most (but not all) cases by one or more arguments. Except in the job title, upper and lower case letters are synonymous. Job description statements are terminated by end of line or comma.

In the description which follows, upper case words stand for themselves, whereas lower case words indicate item types. Square brackets are used to enclose optional items; a vertical bar separates alternatives with underlining indicating a default. Numerical arguments which are followed by an optional letter K may either be specified in units of one or in units of 1024.

A job description begins with a JOB statement, which takes the form

JOB user-identifier project-number [title]

and ends with one of the terminating statements JCL or PHOENIX (or PHX) which take no arguments and which must be the last statement on a line. All other job description statements are optional, and need only be given when the defaults do not suffice. Thus an example of a minimal job description for a Phoenix job is

```
JOB SPQR 1234, PHX
```

This specifies a null string for the job title. The project number may be replaced by an asterisk to indicate the default project for the given user. An example of a more complicated job description is

```
JOB ABCD * Large Production Job
LIMSTORE 400K
COMP 6 MINS, TAPE9 ABCD99
TURNROUND OVERNIGHT
PRINTER 4000, ROUTE WESTCAM
JCL
```

We shall now describe the optional job description statements in alphabetical order. In job descriptions they may appear in any order. In all cases if a statement which requires arguments appears with none, it is ignored. This facilitates the construction of job descriptions by programs and Phoenix command sequences. Null job description statements are also ignored.

BIGDISC

This statement requests the mounting of a scratch disc pack for use by the

job. The 3330 packs used on the 370/165 can contain up to 200 megabytes of data. The user must have the necessary privilege in order to use this facility.

BIGPHX

Phoenix jobs are run in a small control region of 22K bytes which holds data for the Phoenix command processor; for each job step a second region of the requested size is obtained. In a very few cases, where deeply nested sets of Phoenix commands are used, the control region may be too small. This statement requests a larger one; 44K bytes is currently given.

CHAIN

This statement marks the job "chained". Jobs for a particular user that have the chain flag set will not be run simultaneously.

COMP n [MIN[S]|MINUTE[S]|SEC[S]|SECOND[S]]

This specifies a limit for the computation (CPU) time for the job. The default is five seconds, and the maximum permitted value is 20 minutes. TIME is a synonym for COMP.

COND (value,comparator)

This statement is useful only for JCL jobs; it sets a COND parameter for the job as described in IBM's JCL reference manual.

COPIES n

This statement causes n copies of the job's printer output to be produced. The default value is one.

DECKS [7|9] n

This specifies the number of seven or nine track tape decks required by the job. If omitted, the numbers are taken as equal to the number of seven or nine track tapes requested respectively. Use of the DECKS statement is mandatory if more tapes are to be used than there are decks permitted. Its use is recommended whenever fewer decks than tapes are required, as it enables the machine dependent scheduler to run tape jobs more efficiently.

DISC disc-name

This requests the mounting of a private disc pack for use by the job. The user must have the necessary privilege in order to use this facility.

DISCIO

Phoenix jobs which request no more than five seconds of CPU time and 200K bytes of store, and which use no private discs or tapes, are given preferential treatment if they are scheduled as fast jobs. However, they are limited to 25 seconds of channel time. (This state of affairs is a result of a previous state where such jobs were processed serially by a Batch Monitor system.) The DISCIO statement specifies that the job uses more than 25 seconds of channel time, and hence is not eligible for

special treatment. Its use is rare.

LIMSTORE n [K]

This specifies the maximum region size of any step in the job, in units of 1024 bytes. Any user may run jobs up to 500K; with special authorization up to one megabyte may be used.

MSGLEVEL (n,m)

This statement is only useful in JCL jobs. It sets the message level parameters for the job as described in IBM's JCL reference manual.

NOTIFY [user-identifier]

This statement requests that a message be sent to the terminal of the given user if he is logged on when the job finishes execution. The default identifier is that under which the job is run.

PAGE n

This specifies the number of lines on the page for lineprinter output. The default is 60. If zero is specified, the output spooling software inserts no pagination.

PASSWORD password

This statement supplies the password for the user given in the JOB statement. It is mandatory for jobs submitted externally through the card or paper-tape readers or from remote job entry links. For jobs submitted from other jobs, or from Phoenix sessions, this statement may be omitted if the user identifier is the same as that of the submitting job or session, because the password has already been quoted and checked.

PLOTTER n[K] [RECORDS]

This specifies the maximum number of records of pen plotter output that the job will produce. The default is zero, and the maximum permitted value is 10K. Jobs which exceed this limit are cancelled by the system.

POST user-identifier

This statement is used to cause the output from a job to be delivered to a user other than that for which the job is run. Several pseudo identifiers are also used, for example POST MAIL is a request for the output to be mailed to the user by reception staff.

PRINTER n[K] [LINES]

This specifies the maximum number of lines of lineprinter output that the job will produce. The default is 500, and the maximum permitted value is 80K. Jobs which exceed this limit are cancelled by the system.

PRIORITY n

This statement, whose keyword is historical, specifies a maximum unit price for the job. The value must lie in the range 1-255.

PUNCH n[K] [CARDS]

This specifies the maximum number of records or punch output that the job will produce. The default is zero, and the maximum permitted value is 10K. Jobs which exceed this limit are cancelled by the system.

ROUTE [device] destination

Output from jobs is normally routed to the output device associated with the reader where the job was read. For jobs submitted from Phoenix sessions the default routing is CENTRAL. This statement is used to change the routing. The device may be one of

ALL
PRINTER
PUNCH
PLOTTER

and the destination one of the current routes. One special destination is DUMMY, which causes the output to be discarded.

SAVE

This causes the output from the job to be saved on the spool disc instead of being printed or punched. From there it may be copied to a user's file by means of the Phoenix command COLLECT, enabling it to be inspected from a Phoenix session. The user may also inspect a copy of the output, and may release the saved job to be output as normal.

TAPE[7|9] name1[/access] [name2[/access]] ...

This specifies the tapes which are to be used by the job. It also specifies the number of tape decks required implicitly, unless the DECKS statement is also present. The access may be specified as R or W, for read or write; this determines whether the tape is mounted with or without a write permit ring. If the access is omitted, the tape is mounted in its default state, which is fixed for each tape. An error occurs if W is specified for a tape whose default state is 'never write'.

TURNROUND a [OR b]

The turnround requests a and b may be one of

NOW
OVERNIGHT
n [HOUR[S]|MIN[S]|DAY[S]]
BY time-of-day
AT time-of-day
REJECT

The defaults, which are independent, are

TURNROUND NOW OR 3 HOURS