ISSN 0381-9019

AUTOMATIC JOB SCHEDULING IN HASP

BY

UDAY G. GUJAR

AND

DAVID M. FELLOWS

TR77-014, JANUARY 1977

ABSTRACT

Multiprogramming systems require that a fair, equitable algorithm be used for the scheduling of jobs. This paper discusses some of the problems associated with this and proposes an automatic job scheduling algorithm. The major parts of the algorithm have been implemented and have been in use for over one year. The user interface is simplified and the operational complexities are minimized. The parameters used for the algorithm are the estimates of the central processor time and the memory required by the job. All types of jobs including those requiring operator attention during execution are covered under the scheme. Operational data and the reactions from the users indicate that the results have been as expected.

AUTOMATIC JOB SCHEDULING IN HASP

BY

Uday G. Gujar

Computing Centre
and
School of Computer Science
University of New Brunswick
Fredericton, N.B.

David M. Fellows

Computing Centre
and
School of Computer Science
University of New Brunswick
Fredericton, N.B.

TR77-014, January 1977

1. INTRODUCTION:

This paper briefly reviews the job scheduling scheme employed by HASP (Version 4) in the OS/VS2 environment (see Appendix I for the list of abbreviations used in this paper). This scheme is essentially the same as the one used in the previous versions of HASP (Version 3.0, 3.1) used for MFT and MVT. For details about HASP and various IBM operating systems, namely MVT and VS2, the reader is referred to the various sources cited in the references. The paper then focuses on some of the problems encountered in using this scheme and suggests a method which eliminates these and other drawbacks. This method has been designed and implemented and is in use at the authors' installation; it has proved to be very satisfactory.

Even though the method is implemented for HASP 4.0 (i.e. for VS2), the same technique can be used for previous versions of HASP for MVT.

2. EXISTING SCHEME:

The job selection algorithm used by HASP [1, 2] and the problems associated with it are discussed in this section. Finally, some particular problems at the authors' installation are discussed.

2.1. The Scheme:

HASP 4.0, as it comes from IBM, uses the following method for selecting a job for execution.

The user is required to supply, among other things, the values for the following parameters:

e_t - estimate for execution time for the job

e_l - estimate for the number of printed lines to be produced

e_c - estimate for the number of punched cards to be produced

c - the job class

A separate job execution queue is maintained for each job class. The jobs within each queue for a particular class are arranged according to the priority which is assigned by HASP based upon the values of e_t, e_1 and e_c supplied by the user, either explicitly or by default. The priority, p, is determined by:

$$p = \lfloor \frac{P_T(e_t) + P_0(e_1+e_c)}{2}$$

where $\lfloor$ denotes the floor operation and the functions P_T and P_0 are defined by tables which are created when HASP is generated. As an example, the default values generated for these functions are given in Table 1.

If the priorities for the jobs within a class are the same, they are placed in the queue in the order of arrival.

Previous versions of HASP [3] used for MVT use a somewhat different job scheduling algorithm. However, the priority, p, is still a function of $P_T(e_t)$ and $P_0(e_1)$.

Each initiator has a list, either provided by the operator explicitly or supplied by HASP by default, specifying the classes of jobs it is to service. When any initiator becomes idle, the following step is iterated for all integer values of i from one to the number of eligible classes:

```
        if the job class queue for the ith eligible class is not empty,
        select the first job from that job class queue for execution.
```

If the end of the list is reached without a successful selection of a job, the initiator waits until an appropriate job arrives.

2.2. Problems:

There are several problems inherent in the above job scheduling process.

One major drawback is that one of the important resources, namely the amount of memory required by the job, is totally ignored.

In the MVT system [4, 5, 6] main memory is allocated dynamically on a job step basis. Normally, the memory required by a job step must be supplied in a contiguous block. If the memory required for a job step is not available, the initiator will wait until sufficient memory has been released by other job steps. The dynamic nature of the allocation process gives rise to memory fragmentation. If the memory available for dynamic allocation is small, fragmentation may cause very low utilization of this resource.

A badly fragmented memory can be tolerated for short periods of time but if it persists it can substantially reduce the capacity of a system.

The MVT system does not incorporate any control of memory fragmentation.

The VS2 Release 1 [10] operating system is basically MVT modified to exploit the paging hardware of the /370 machines. The addition of a paging supervisor permits the operation of a slightly enhanced MVT system in a single 16×10^6 byte address space (virtual memory). Virtual memory is allocated to job steps in exactly the same way real memory is in MVT. Fragmentation of the address space does occur, but is usually not a problem since the total address space is so large. Nevertheless, VS2 in common with all virtual memory operating systems, can suffer from excessive paging if the executing programs are using too much virtual memory relative to the real memory available to support it. This problem has been studied extensively by Denning [7] and many others. A discussion of the solution of this problem is beyond the scope of this paper. Suffice it to say that the VS2 Release 1 system paging algorithm does not provide adequate control over the paging rate. However, a crude control can be exercised by limiting the total virtual memory allocated to executing job steps to a "safe" value.

A second problem which is not satisfactorily solved in the MVT or VS2 Release 1 revolves around the process of allocation of input/output devices (tape devices, etc.) to job steps. This process is serialized to prevent dead-locks from occuring. Furthermore, once it is begun it must be completed. If a device requested is not available the initiator waits for it, no further job steps can be started. If the device required is held by, say, a job step that requires two hours to execute, no other job steps can start in that time. One solution to this problem is to ensure that jobs requiring specific devices are not allowed to start executing until it is known that the devices will be available. It is possible to do this in the job scheduling algorithm. It is also an area where a human operator's judgment may be useful.

Another aspect of the job scheduling scheme used by HASP is that it is based upon the estimate of the real time to execute the job. In a multiprogramming environment, this time is unpredictable, unreliable, beyond the control of the user and does not truely represent the resources used by a job.

2.3. Job Processing:

There were other aspects of the job processing that created problems at the authors' installation.

One was the handling of jobs requiring operator invention such as tape mounts, disk mounts, etc. The users were required to "scribble" the special requirements on the job card before submitting the decks for processing. These decks, along with others, were held in an input rack until the operator decided to read them in. Just before reading a deck, the operator was required to copy down the user requests on a sheet of paper. If a request for any intervention appeared on the operator's console on behalf of a job during its execution, the operators were supposed to consult their sheets and act accoridngly to the user instructions.

Transcribing of the information from the job card to a separate sheet was necessitated by the fact that a job could encounter any amount of delay before its turn to execute. This ad hoc procedure gave erratic service to the system's users. In addition, every now and then operators were forced with a request not in thier list and had to determine its validity which invariably resulted in system tie ups and execution delays.

The second problem was caused by an increasing number of jobs being submitted either from HASP remote work stations or through the remote job entry facility in the APL system [8]. Outputs of such jobs were being held in an attempt to match these with the corresponding card decks. Furthermore, such users were u·able to scribble operator messages on their "decks".

3. OBJECTIVES:

The following objectives were set while designing the job scheduling scheme:

a) The scheme used should reflect as well as possible the amount of resources used by the job during its execution. Jobs requiring fewer resources should be scheduled to run before those requiring more resources.

b) The parameters entering into the job scheduling scheme should be easily estimâble and give consistent resúlts.

c) No redundent information be required from the users.

d) The parameters supplied by the users should be policed in the sense that the amount of a resource used may not exceed the estimate.

e) "Short" jobs should have a turn-around time short enough to justify users waiting for their output. Unfortunately "short" is inevitably a function of the system load.

f) No job queue should be blocked forever. Thus, even though the short jobs should run earlier, a steady stream of short jobs should not be able to delay longer jobs for an indefinite period of time.

g) The scheme should be easy to use by both the users and the operators.

h) The scheme should be within the resources of the authors' organization to implement and also maintain in spite of future modifications to the operating system.

i) The human intervention required for job processing should be minimized, ideally no human intervention should be required.

j) The jobs requiring operator attention during execution should be handled in a uniform manner and should not suffer undue penalties or erratic favours. Most requests for operator attention are requests for disks or tapes to be mounted. The satisfaction of such requests requires that both the required volume and a device on which to mount it be available. Thus, the availability of devices must be considered in job scheduling.

k) Finally, with all this automation, there must be a way to run some special jobs outside the above automatic scheme. This is required for two reasons. The first is that essential real time services, like a terminal system, etc., cannot tolerate the delays inherent in the automatic scheme. Since these real time services are usually I/O bound and almost always stay in the system for a long time, usually all the time when the system is up, this exception will not affect the job scheduling scheme adversely. The second reason is that sometimes the management may want to run some special jobs in a hurry. By treating these jobs separately on an individual basis, one can identify them easily, minimize their occurrence and still process them without unduly affecting the other jobs.

4. ALGORITHM:

The form of the solution to our problem was heavily influenced by objective (h). The maintainability aspect ruled out solutions which required extensive modifications to the operating system. HASP was chosen as the implementation vehicle for a number of reasons:

a) It is a well documented program which is relatively easy to modify.

b) The source code was readily available.

c) One of the authors had considerable experience in modifying HASP.

d) One of its functions was job scheduling.

This choice narrowed the solution to our general problem to one of designing a suitable scheduling algorithm.

The algorithm is based on the assumption that the resources used by a job are adequately measured by CPU time, memory and the number of I/O requests. The I/O requests, referred to as the EXCP counts, should include the activity on all peripheral devices. There is a fourth factor, namely, the amount of operator intervention required during the execution of a job; however, this is treated somewhat differently.

The selection of a job for execution is based upon two factors: job class and the priority. It would be ideal if all jobs were assigned the same class; then there would be only one queue and the jobs would be scheduled according to priority alone. However, there are two considerations which do not favour this one class concept:

a) The range of priorities allowed by HASP is from 0 to 15 which is rather too restrictive for this purpose.

b) More importantly, at any given time, a situation could arise when all the initiators may be executing relatively long jobs. Then, if a short job is submitted, it will have to wait until one of the initiators becomes free which may not happen for a relatively long period of time (e.g. 30 minutes). This situation was regarded as intolerable during prime time at least.

Even though the authors felt that the estimate of the EXCP count should enter into the scheduling algorithm, it was decided, for reasons beyond their control, to leave it out of the initial stages of the implementation.

4.1. Implementation:

For ease of modularity and to remove interdependencies as far as possible, the implementation is broken into several stages.

4.1.1. Stage 1:

In this stage, an immediate partial relief is found for the jobs requiring operator interventions. All such jobs are required to provide a /*SETUP control card stating the kind of operator attention required while the job is executing. The effect of this control card,which is a standard HASP feature, is that the job containing it is placed in hold status and the message is written on the operator's console when the job is read. The job remains in hold status until manually released by the operator; then and only then it could be selected for execution. Effective use of this procedure presumes that a hard copy device is available as an operator's console and, further, that the number and type of messages that could appear on this device can be controlled. This is the case with the multiple console support feature of the VS2 system.

If the operator is faced with a "surprise" request which was not conveyed via the /*SETUP card, the job is cancelled.

Once this policy is instituted, there is no reason for sorting the jobs manually at the input racks or for any scribbling on the job card. Thus, the time spent by these (and other) jobs at the input racks gathering dust, is totally eliminated. It also permits any type of job to be submitted from any input device.

4.1.2. Stage 2:

In this stage, two HASP scheduling parameters whose values are to be supplied by the user (via the JOB or the /*JOBPARM card) are implemented. These are the estimated CPU time, t, in seconds, and the maximum amount of memory, m, in k-bytes. The specification of these parameters is policed in the sense that a job is cancelled as soon as it uses up its estimated CPU time limit, and the region allocated to each job step is the lesser of the estimate and the request. If this size is adequate, the job may still run normally, otherwise it will be terminated abnormally.

4.1.3. Stage 3:

In this stage the concept of job class is eliminated as far as the user is concerned. If a job class parameter is supplied by the user, it is ignored.

HASP determines the job class and priority from the user supplied values for the estimates of the CPU time, t, in seconds, and the maximum amount of memory, m, in k-bytes. Since the VS2 operating system allocates region in 64k-byte segments, m is rounded up to be an exact multiple of 64; i.e. a value of r is used for the estimate of the maximum region, where

$$r = 64 \times \lfloor (m+63) \div 64$$

and $\lfloor$ is used to denote the "floor" operation.

The job class, c, and the priority, p, are assigned according to the following algorithm:

```
if ($t > TH_n$) or ($r > RH_n$)
  then
     $c = C_n$
     $p = 0$
     place job in hold status
  else
     do $i = 1$ to $n$
        if ($t \le TH_i$) and ($r \le RH_i$)
           then
              $c = C_i$
              $p = PMAX_i - \lfloor(t \div TF_i) - \lfloor(r \div RF_i)$
              break
     end
```

The various quantities used in the above algorithm, created as tables at the HASP generation time, for values of i ranging from one to the number of classes, n, are:

C_i - i^{th} class

TH_i - largest CPU time, in seconds, allowed for the i^{th} class

RH_i - largest region, in k-bytes, allowed for the i^{th} class

TF_i - a factor giving the contribution of the CPU time towards the priority assignment for the i^{th} class

RF_i - a factor giving the contribution of the region towards the priority assignment for the i^{th} class

$PMAX_i$ - highest priority allowed for the i^{th} class.

The factors TF_i and RF_i are calculated at the HASPGEN time from the relative weights for time, TW_i, and region, RW_i, according to the following formulae:

$$TF_i = \lfloor(TH_i + TW_i - 1) \div TW_i$$
$$RF_i = \lfloor(RH_i + RW_i - 1) \div RW_i$$

The relation of all these factors is depicted pictorially in Fig. 1. Note that the inequality

$$0 \le TW_i + RW_i \le PMAX_i \le 15$$

must hold for all the values of i. This constraint is imposed by the range allowed by HASP for the value of priority; the generation macro ensures that this condition is not violated to guard against erroneous specifications.

It is important to note at this point that distortions between class boundaries can be minimized by keeping the number of classes as small as possible.

It was decided to have three classes:

Class A - short jobs that are guaranteed to run with a minimum of delay.

Class B - jobs requiring medium amount of resources which could be run during the prime shift but have no guaranteed fast turnaround time.

Class C - jobs with time and memory requirements that cannot be run during the prime shift. Such jobs will be run only during off-prime shifts.

The values used for the tables and the factors generated are given in Table 2. Fig. 2 is a graphical representation of the class/priority function based on the factors given in Table 2.

Assuming that four batch initiators are available, as is the case at the authors' installation, they are defined as shown in Table 3. It can be observed from this table that, during prime shift, class A jobs are guaranteed to have two initiators, get a preferential treatment from the third initiator and will be serviced by the fourth initiator only if there are no class B jobs. Conversely, class B jobs get a preferential treatment from one initiator and can have access to another initiator only if there are no class A jobs. Similar comments can be made about the definition of initiators outside the prime shift. This arrangement assures that no class will entirely block processing of any other class.

In order to assure that no job within a class will be blocked off indefinitely because of a steady arrival of the higher priority jobs within that class, the HASP priority aging scheme [9] is used. Under this scheme, the priorities of all the jobs, that fall within the range of this scheme, are increased by 1/16 at a regular fixed time interval specified at the HASPGEN time. A job is said to be within the range of priority aging scheme if its initial priority, p, satisfies the inequality

$$\text{PRILOW} \le p \le \text{PRIHIGH}$$

where the values of PRILOW and PRIHIGH are specified at the HASPGEN time. Thus, if a job remains in the system for sixteen such time intervals, its priority is incremented by one. This process continues on until the job reaches the top allowable priority (which is PRIHIGH).

The range of the priority aging scheme was set at from 0 to 14 and the time interval for which the priority is incremented by one was set at two hours.

Certain job class specifications may be honoured outside the scheduling scheme. In this case the job is placed in the execution queue for the class specified. This facility is intended for jobs submitted under strict supervision. It is policed because the initiators servicing these classes are either busy or stopped.

4.1.4. Stage 4:

The automation of the scheduling of jobs that require operator intervention by way of disk mounts, tape mounts, replies, etc. during execution are handled in this stage. These jobs are treated in exactly the same way as all other jobs except that they are held automatically. When HASP is ready to run such a job, the operator is informed of the special needs and HASP selects next eligible job, if any, for execution. It is now the operator's responsibility to release the job manually as soon as he can satisfy these special needs.

If the job is subsequently passed over in the job selection process, the operator will be reminded of the fact that it is due for execution. The reminder will prevent a job from getting "lost" but will allow the operator to delay executing a job if necessary. For example, there may not be sufficient tape drives currently available to satisfy the job requirements.

The big advantage of the above procedure is that the only penalty suffered by such jobs is the time taken by the operator to react positively - and this is the maximum penalty these jobs should suffer.

The various factors involved are:

a) The User's responsibility: The user is required to add the '/*MOUNT' and/or the '/*REPLY' control cards (which are described below) to his job.

b) The Operator's responsibility: The requests specifying the kind of interventions required for a job appear on a console when that job's turn to execute comes. The operator will release that job immediately or as soon as he can satisfy these demands.

c) HASP modifications: Provide two new control cards, namely /*MOUNT and /*REPLY to do the following:

   i) hold the job

   ii) store the specified requests in a HASP control block

   iii) set a flag

   iv) Whenever the job is eligible for execution, write a message to the operator describing the intervention request.

   The format of the above control cards could be:

   /*MOUNT volume name, volume name, ...

   /*REPLY additional information

4.1.5. Stage 5:

In this stage some specialized HASP operation commands which enable the operator to display the status of jobs according to /*MOUNT and/or /*REPLY requirements will be supplied.

5. RESULTS:

This section discusses the results of implementing stages 1 - 3 at the authors' computing centre. The parameters actually used are those presented in section 4. The defaults for the estimated CPU time and region were set at one minute and 256 k-bytes respectively.

The IBM automatic priority group option [10] was used to share the CPU equitably between the initiators.

5.1. Hardware Description:

At the time the proposed algorithm was implemented, the hardware comprised an IBM Model 158 cpu with 1.57 x 10^6 bytes of memory, connected to 16 disk drives, 4 tape drives, 2 high speed printers and 2 card readers. One reader/printer work station was located in the machine room, the other was located in an adjacent public room where staff and students could submit and retrieve their own jobs. In addition, there were 10 HASP Remote Work Stations connected via telecommunication lines. These remote work stations were located in other provincial universities and on the premises of some commercial users.

5.2. Workload:

The workload to be handled by the scheduling algorithm consists of a mix of "scientific" and "commercial" jobs. It can be characterized as a base load due to "production" programs from the university's data processing department, researchers, and external users. These jobs tend to require more resources than the average. Superimposed on this is a load due to students working on course and research projects. This load is cyclic and has two pronounced peaks, one in November and a higher one in March. The work load has been growing at a compound annual rate of about 15 percent for some years.

In addition to the above, the central processor is also used to supply APL timesharing service and a "superfast" service [11] for compiling and executing very short student programs. This requires about 15 percent of the total cpu time and is not considered further here.

5.3. Intermediate System:

The VS2 system in use prior to adopting the scheduling algorithm discussed here was itself a modification of that distributed by IBM and incorporated a partial implementation of stage 2 described in Section 4.1.2. The principal modifications to HASP, pertinent to job scheduling, were:

a) The estimate of the CPU time could be supplied by the user and was policed.

b) A job's priority in the execution queue was determined solely by the estimate of CPU time (see Table 4).

c) The job class"Q"was treated specially in that it forced the limits of 15 seconds of CPU time and 1000 lines of output. Further, it was advertised that such jobs should not use more than 110 k-bytes of memory.

During prime shift, there were four initiators defined as

Q, JQ, KJQ, LMNKJQ

to serve the various jobs. Class Q was intended to provide a "while-you-wait" service.

The class names were used to indicate the amount of memory required for the job. However, the user community quickly found out that there was no reason to abide by the advertised guidelines. As a result, most jobs used class J (and some even used class Q) irrespective of the memory requirements.

A typical day's statistics are shown in Table 5.

Stage 1, discussed in Section 4.1.1, had been implemented at this time. No reliable statistical data is available prior to the implementation of stage 1 because the job queue was maintained manually in card racks.

It can be clearly seen from Table 5 that short jobs are most numerous and very few jobs require more than 15 minutes of CPU time. Also, this particular day was one with only a moderate load.

5.4. New System:

Stages 1, 2 and 3 given in Section 4 became fully effective on October 6, 1975. The parameters actually used in the scheduling algorithm are those shown in Table 2 with initiator definitions as shown in Table 3.

The statistics for a day, shortly after the new algorithm was implemented, are given in Table 6. The long term effects of the scheduling system can be seen in Fig. 3. Table 7 clearly shows that the scheduling algorithm does in fact discriminate smoothly against jobs with increasing resource requests.

5.5. Discussion:

The user reaction to the new algorithm has generally been favourable due to the additional facilities which became available especially through the self-service reader. The unfavourable reaction has come from some of the users with a large number of moderately long jobs requiring operator intervention. They complain that these jobs suffer unduly long delays.

A comparison of this algorithm with the one supplied by IBM is impossible since the later was used in the too distant a past and as such no data is available on its performance.

A fair comparison between the "intermediate" and "new" system proved to be almost impossible with the data available. The data is derived from accounting records provided by the operating system [12]. The measure used was the distribution of the differences between the time a job was placed in the execution queue and the time it was removed from that queue. The 90th percentile of this distribution is taken as a convenient single measure. This measure is sensitive to a number of factors other than the scheduling algorithm. These other parameters, such as hours of operation, machine failures, load on the machine, etc., were not, or could not be, controlled in the period around the switch. The data for class A jobs after the changeover is biased by one particular job which should have been scheduled outside the range of algorithm. This job was run frequently (2 to 10 times/day) and was purposely held before execution for long periods (typically 5 to 15 hours). The two week period immediately after the changeover was exceptionally bad because of outages caused by hardware. Subsequently, the load increased sufficiently so that comparisons with summer time data were not valid.

The data for two selected days, given in Tables 5 and 6 , has been arranged so that approximately comparable jobs would appear in corresponding rows. The days are close enough together that many users presumably had not changed their estimates to take advantage of the new scheduling strategy. Most of the class A, one minute jobs are probably old class Q jobs that did not explicitly specify a time estimate and hence had the one minute default value supplied.

There is too much variance in the data to draw any conclusions about the effect of the change between the "intermediate" and "new" systems on the execution delays experienced by jobs.

5. CONCLUSIONS

An automatic job scheduling system has been proposed and partially implemented. The scheme is based upon the estimates of the actual resources that a job uses. The users are provided with a mechanism which allows them to control the priority and class for their jobs.

The part that has been implemented works and provides the sort of results expected of it. The code has been written in such a manner that the parameters of the scheme, namely, number of classes, time and region boundaries, time and region weights and the assignment of classes, can be changed very easily. This facilitates the further adjustments of these parameters to optimize the turnaround for the users' jobs. Such a work is being undertaken.

In retrospect, it appears that the authors' suggestion of including the estimate of number of input/output operations was a sound one and it was a mistake to omit it. However, the scheme is general enough to include this in a future work.

Experience to date has been good. Indications are that the next stage, i.e. automatic scheduling of jobs requiring operator intervention, should be undertaken in order to reduce the delays for such jobs.

6. ACKNOWLEDGEMENTS

We would like to thank Mrs. A. Stocek and Mrs. A. Anderson for typing the entire manuscript without loosing their usual cheerful smiles. Thanks are due to the colleagues in the Computing Centre for useful discussions.

REFERENCES

1. IBM, OS/VS2 HASP II Version 4 System Programmer's Guide, Rockville, International Business Machines Corporation. Form No. GC27-6992-0, 1973.

2. IBM, OS/VS2 HASP II Version 4 Logic, Rockville, International Business Machines Corporation, Form No. GY27-7255-0, 1973.

3. IBM, The HASP System, White Plains, International Business Machines Corporation, 1971.

4. Mealy, G.H. et al, "The Functional Structure of OS/360", Parts I, II, III, IBM Systems Journal, 5 (1), p. 3-51, 1966.

5. Donovan, J.J., Systems Programming, New York, McGraw-Hill, 1972.

6. Flores, I., Operating System for Multiprogramming with Variable Number of Tasks, Boston, Allyn and Bacon, 1973.

7. Coffman, E.G., Jr. and P.J. Denning, Operating Systems Theory, Englewood Cliffs, Prentice-Hall, 1973.

8. Gujar, U.G., "Remote Job Entry and Output Through APL", APL75: Congress Proceedings, p. 148-157, Pisa, 1975.

9. IBM, "OS/VS2 HASP II Version 4 source listings".

10. IBM, OS/VS2 Planning and User Guide, 2nd ed., Poughkeepsie, International Business Machines Corporation, Form No. GC28-0600-2, 1972.

11. Emin, P.P., "A Partition Monitor for Fast-Batch-Processing with Limited Execution (FABLE)", TR76-012, School of Computer Science, University of New Brunswick, Fredericton, Canada, Mar. 1976.

12. IBM, OS/VS System Management Facilities (SMF), 4th ed., Boulder, International Business Machines Corp., Form No. GC35-0004-3, 1973.

APPENDIX I
List of Abbreviations

HASP	- Houston Automatic Spooling Program
OS/VS2	- Operating System/Virtual Storage 2
MVT	- Multiprogramming with Variable Number of Tasks
IBM	- International Business Machines Corporation
APL	- A Programming Language
I/O	- Input/Output
EXCP	- EXecute Channel Program
CPU	- Central Processing Unit
HASPGEN	- Houston Automatic Spooling Program GENeration
k-byte	- 1024 bytes

Table 1

Default HASP priority functions

estimated execution time (minutes)	Priority Component (P_T)	estimated total output records	Priority Component (P_0)
$e_t \le 2$	9	$e_1 + e_c \le 2000$	9
$2 < e_t \le 5$	8	$2000 < e_1 + e_c \le 5000$	8
$5 < e_t \le 15$	7	$5000 < e_1 + e_c \le 15000$	7
$15 < e_t$	6	$15000 < e_1 + e_c$	6

Table 2

HASP class-priority table

i	C_i	TH_i	RH_i	TW_i	RW_i	$PMAX_i$	TF_i	RF_i
1	A	60	1024	10	5	15	6	205
2	B	1200	3072	10	4	14	120	768
3	C	6000	5120	10	4	14	600	1280

Table 3

Definition of batch initiators

Batch initiator	Prime shift	Off prime shift
1	A	A
2	A	AB
3	AB	BCA
4	BA	CAB

Table 4

Priority function for "intermediate" system

Estimated CPU Time (minutes)	≤2	≤5	≤15	≤30	≤45	≤60	≤90	≤120
Priority	9	8	7	6	5	4	3	2

Table 5

Distribution of times spent in execution queue
for 24 hour period (October 2, 1975) - "intermediate" system

Class	Estimated CPU Time (minutes)	Limits of Region	No. of Jobs	50th percentile time (minutes)	90th percentile time (minutes)	97th percentile time (minutes)
Q	≤ 0.25	unknown	213	0.0	0.9	3.2
J,K,L,M	≤ 1	unknown	94	0.0	10.9	25.2
J,K,L,M	≤ 2	unknown	48	1.9	25.3	40.4
J,K,L,M	≤ 5	unknown	33	0.0	5.7	—
J,K,L,M	≤ 15	unknown	17	1.2	107.3	—
Any class any time setup			109	13.1	281.	619.
All jobs			514	0.0	22.3	136.6

Table 6

Distribution of times spent in execution queue
for 24 hour period (October 16, 1975) - "new" system

Class	Estimated CPU Time (minutes)	Limits of Region	No. of Jobs	50th percentile time (minutes)	90th percentile time (minutes)	97th percentile time (minutes)
A	≤ 0.25	1024	68	0.0	1.0	2.6
A	≤ 1	1024	234	0.0	13.4	50.6
B	≤ 2	3072	29	2.9	30.4	-
B	≤ 5	3072	39	1.1	26.2	48.9
B	≤ 15	3072	28	11.8	42.0	-
Any class any time setup			111	18.6	90.0	366.0
All jobs			509	0.6	42.1	112.4

Table 7

Execution delays - March 1976

		Jobs not requiring operator intervention			Jobs requiring operator intervention		
Class	Priority	Jobs/Day (average)	50th percentile time (minutes)	90th percentile time (minutes)	Jobs/Day (average)	50th percentile time (minutes)	90th percentile time (minutes)
A	15	39.5	0.2	1.7	1.0	8.8	49.8
	14	108.6	0.1	2.2	5.5	7.1	40.5
	12-13	177.6	0.4	6.7	9.0	6.3	50.7
	8-11	103.8	0.2	8.1	13.5	13.3	57.2
	0-7	150.4	0.1	27.1	36.8	18.1	87.8
B	14	10.2	7.9	61.2	1.1	28.6	201.
	13	39.3	19.1	139.7	20.4	52.8	213.
	11-12	20.5	66.3	290.5	35.9	163.	276.
	7-10	9.0	23.9	468.1	11.8	501.	721.
	0-6	2.5	65.9	1172.	7.6	744.	1280.
C	11-12	1.1	746.	1055.	2.5	744.	2119.
	7-10	0.4	443.	1537.	0.8	844.	3193.
	0-6	0.6	281.	3515.	0.6	845.	3660.

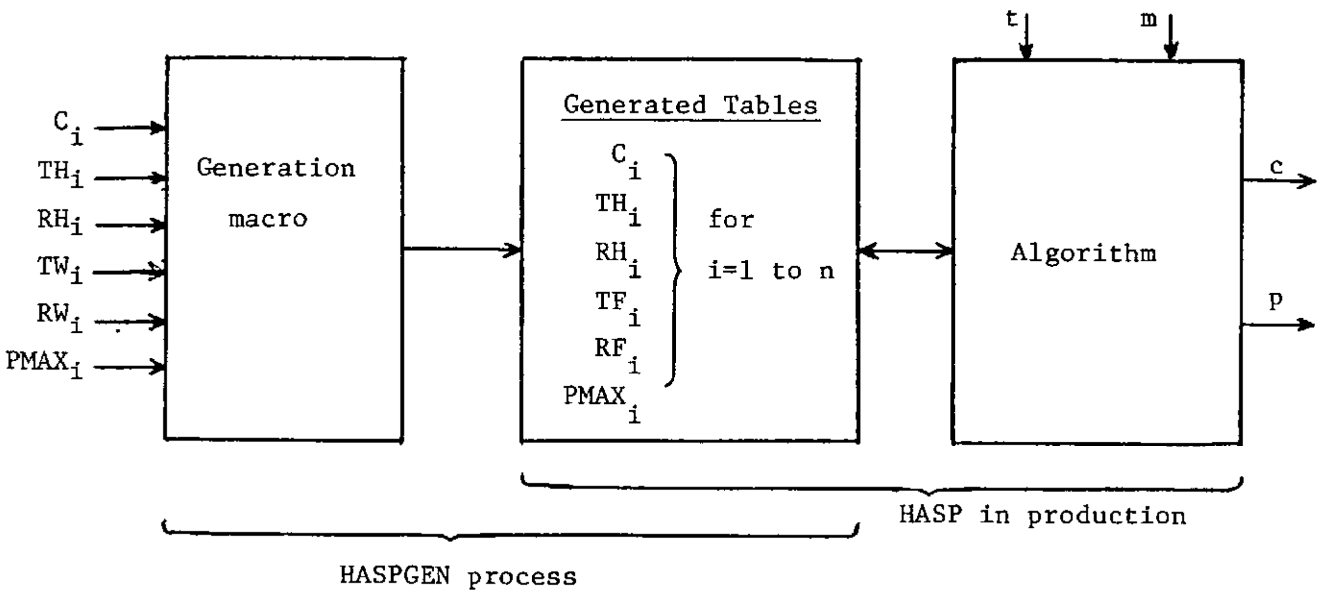


Fig. 1: Job scheduling parameters

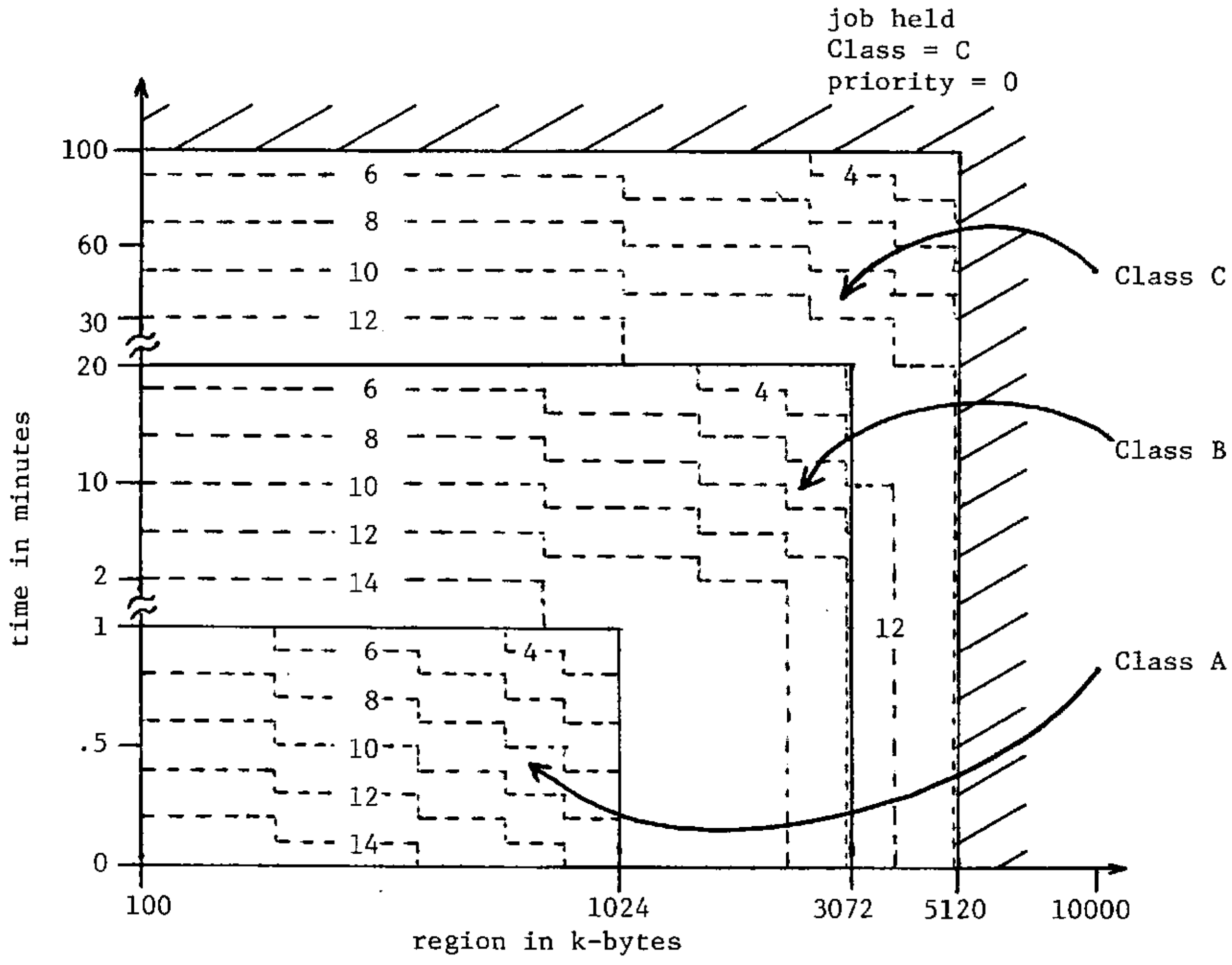
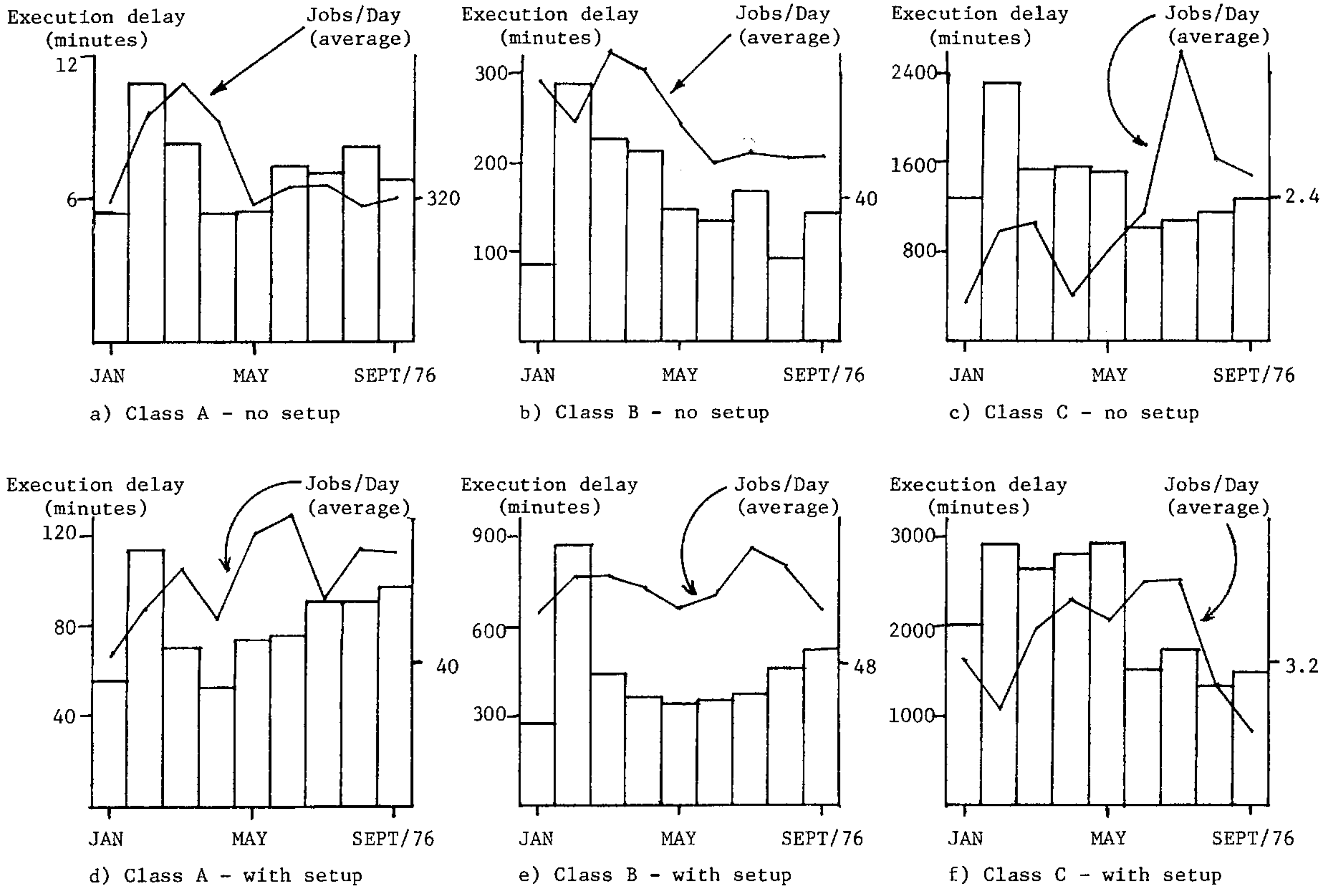


Fig. 2: Job classes and priority contours



a) Class A - no setup

b) Class B - no setup

c) Class C - no setup

d) Class A - with setup

e) Class B - with setup

f) Class C - with setup

Fig. 3: Long term effects