The Pitt time-sharing system for the IBM system 360: Two year's experience

by GEORGE F. BADGER, JR., E. ANDREW JOHNSON
and RICHARD W. PHILIPS*

University of Pittsburgh
Pittsburgh, Pennsylvania

INTRODUCTION

Overview of the system

The University of Pittsburgh has developed a console-based Time-Sharing System,[1] and has had it in service since March of 1966. This paper is to serve as a report on the utilization of such a system and on certain conclusions we have come to regarding Time-Sharing Systems.

First, we would like to describe the services available under this system. The services range from a highly conversational and interactive language called the Pitt Interpretive Language (PIL) to languages on which no substantial advantage is gained by use of a console. PIL is fully interpretitive and has placed heavy emphasis on making things easy for the user. A great deal of care is placed on giving meaningful diagnostics and easy error correction. Decisions in the initial implementation between coding efficiency and ease of use were always made in favor of the user. In a second version of the interpreter some emphasis is being put on efficiency while retaining the error handling facility. The ability to service a number of programs such as PIL, with the required speed of interaction, is our justification for calling this a Time-Sharing System.

The second class of services are those which are partially interactive or in which the user gains some advantage by working from a console. These include an OS compatible Assembly Language, and a portion of the OS macro compiler capability, a FORTRAN IV compiler with a pre-editor, a conversational contex editor, and a file maintenance package. Again in the case of the shared file system, great care was exercised in preserving ease of use.

During his use of the system the user may take advantage of any or all of the facilities. He may switch processors as often as desired, and may intermix certain command language statements while working within a processor, e.g., list part of a file while preparing a program which will work with it.

This entire set of services runs on an IBM System/360 Model 50, utilizing a 2361 large capacity storage (LCS) unit. The configuration and some idea of the storage utilization are presented in Figure 1. The most important components of this system are the large capacity storage and the 2314 disk system. The system runs with the user storage being contained in LCS, both while the user is idle and while he is in execution.[2] The high speed storage of the machine is utilized for one processor and the operating system as can be seen from Figure 1. The primary processor resident in high speed storage is the interactive language processor. Because input to the assembler forms a large part of the service of the system, the assembler also is resident, but in LCS. Assembly is an I/O bound process compared to PIL which is compute bound.

---

*Currently with White, Weld & Company

[1]This system was completely written by Computer Center Staff and has no dependencies on other operating systems. Parts of the system are compatible with OS/360, and several language processors are fully compatible.

[2]When necessary, the system will resort to memory swapping onto an IBM 2314 disk. This happens with approximately 30 users signed on, otherwise memory is allocated in contiguous blocks from a free pool when the user signs on.

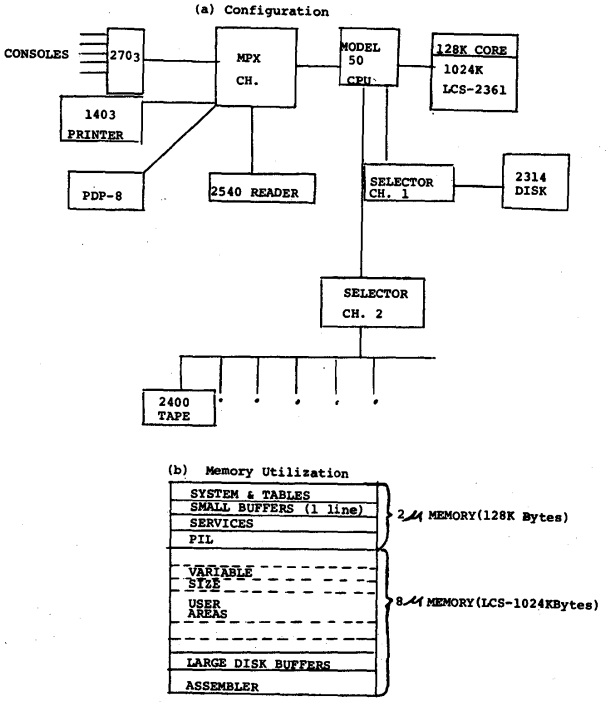

FIGURE 1

The system services something identified as a user who consists of a principal input file end a principal output file. Of the 32 users on the system, as of April, 1968, 31 have a remote console for these principal files. We currently support IBM 2741 and 1050, Teletype 33 or 35, Friden 7100 or 7102, and Sanders 720 scopes.

Because of the structure of the system, it is also possible to introduce one or several pseudo-batch background users whose principal input and output files could be any devices on the system. As of April, 1968, one such user runs in the system from a 2540 card reader to a 1403 printer. The services available to this batch user are identical to those available from the remote console. The only difference in his treatment is that a fatal error results in going to the next job rather than allowing further input which, in the case of a console, could be attempts at correction.

The system favors those who are working in a conversational mode, and has a simple queueing system to allow very rapid response to minimal requests. Because of the emphasis on interaction, the completion of any I/O results in the user being dispatched to a high priority queue which will get service as quickly as possible. Upon being put into this queue, the user will follow all those who arrived in the queue beforehand, and then receives one-quarter second of central processing time. During the one-quarter second, he may either run out of things to do (for example, because of issuing requests for further I/O and waiting for their completion), or he may use the entire one-quarter second. In the latter case, he goes into a lower priority queue which receives service on a time available basis. The quantum of time given in this lower priority queue is increased to two seconds. This then is a general description of the system and its services.

Patterns of system utilization

The use of the system has increased very rapidly during the two years that the system has been in use. As of April, 1968, we run approximately 5000 console sessions per month, and approximately 3000 of the batch jobs. The console sessions use all of the services, but the batch user typically restricts himself to the use of either the Assembly Language or the Fortran Processor. In the conclusions of this paper we will make some comment on the choice by the user as to whether he runs in the interactive or pseudo-batch mode.

As far as the time utilization within the system, PIL accounts for approximately 23% of the Central Processor time available. That is, during a ten-hour day, 2½ hours are actually devoted to the execution of PIL programs.[3] User execution of other processors and generated code accounts for an additional 20%. The processing of supervisor calls and general system overhead accounts for 15%, and the Central Processor is in the idle state 17% of the time. The servicing of I/O interrupts, and disk unblocking with data transfer, accounts for the remaining 24%.[4,5] (The CPU is busy 83% of the time.)

Within this pattern of utilization we are interested in the demands put upon the system in

[3]PIL, being fully interpretive, is very slow in execution. On a matrix inversion, for example, a PIL program will execute at 1/75 the speed of a Fortran program excluding its compilation. A second implementation of PIL will significantly modify this ratio.

[4]All statistics in this paper are representative, but do not necessarily hold for long periods, e.g., a month.

[5]Because of the extremely heavy dependence of the System on the IBM 2314 disk with an associated file system, the CPU is devoted to blocking variable length records. This is explained later in the paper.

more basic units. In particular, there is the question of how much CPU time is required as a time slice, since this partially determines the ability of a given system to support conversational users. Table 2 and Graph 1 present this data in some detail.

These figures show an average time slot of $(20\pm5)/300$ seconds. The data, however, do not show the reason for termination of user execution. Since the user is primarily interested in being terminated because his work is complete, more important data are presented in Table 3. The data present the system as a finite Markov process[6] with the transition probabilities for the set of all users (the system) being given.

The data presented in Tables 3 and 4 represent a composite of two periods, one relatively light and the other extremely busy.[7] Values with the superscript (1) varied considerably with the probability of going from user execution to SVC varying from .7 to .3, the probability to I/O from .3 to .7, and the average time spent in user execution taking the values 3.89 and .65 respectively. All other values of interest remain relatively stable.

Since the user is frequently taken out of execution due to an I/O interrupt, we are currently attempting to study the impact of masking these interrupts for a short period of time (approximately 25/300 seconds—the level at which Graph 1 flattens out). By doing this, approximately 95% of all requests for service would be serviced in a single slice and without interruption. This should preserve the response time of the system for short requests, as well as utilizing peripherals effectively. This would make response time only slightly dependent on the system loading. On the other hand, it might allow quicker preparation of work which will finally require computation of longer duration.

We could consider the strategy above as minimizing the maximum response time for a request up to some limited size. Since a stable level of response time appears to be at least as important as fast response, we are also considering the reverse. That is, never respond in a period less than the period the system is capable of sustaining. Such a level, if it fell between one and two seconds would give very good performance from the conversational users point of view. Further, it would provide increased flexibility in choice of scheduling. This last might be of great importance if a swapping or paging mechanism were in use since you could reduce channel activity by choosing opportune times for the transfer.

The data presented above are representative but are not system parameters. They are intended only to give a picture of the relative activities within a time-shared environment.

System states for Tables 1–4

User execution—The system is running a user program including use of compilers, assemblers or interpreters or the file system. PIL execution plus other user executions.

SVC—A request has been made for supervisor services via the SVC instruction—mainly I/O requests.

I/O—An I/O interrupt is being processed.

Overhead—A clock trap is being processed, the scheduler is running, or certain overhead routines are in use (queueing etc.).

Idle—The CPU is in wait status, i.e., not running.

TABLE 1—CPU Utilization

STATE	PERCENT OF TIME
PIL Execution	23
Other User Exec.	20
SVC Processing	5
I/O Processing	24
System Overhead	10
Idle (CPU waiting)	17

Some conclusions

The utilization of the LCS[8] unit has proved to be extremely successful on the Model 50. Because of the overlapping of the rewrite cycle on core with Central Processor execution, many programs run almost as fast out of LCS as they did in Cen-

[6]Treating this as a Markov chain, finding the steady state vector, and using this to predict percentages of time spent in the various states yields surprisingly accurate results. This will be studied further in a later paper.

[7]The system seems to be representable as a Markov chain, i.e., the transition matrix is constant for extended periods of time. It does seem to vary over time, but the variation is basically related to level of activity, or time in the idle state.

[8]LCS is an 8μ memory with a 4 byte fetch when used by a Model 50. In all other respects it is identical to main memory.

TABLE 2—Frequency of time slots actually given prior to seeking another user to execute

Time in 1/60 Second	Percent of Slots Exactly This Size	Percent of Slots This Size or Smaller
1 or less	48	48
2	20	68
3	6	74
4	3	77
5	2	79
6 to 14	8	87
15	10	97
16 to 120	3	100

GRAPH 1

Frequency of Time Slots Actually Given Prior to Seeking Another User to Execute versus Size of Slot in Percent

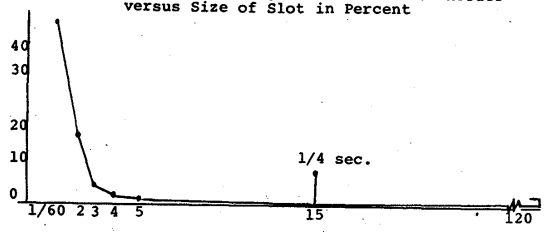

TABLE 3—Transition probabilities of the system states

		Next State: User Exec	SVC	I/O	Overhead	Idle
	User			(1)	(1)	
	Executing	0	.32	.66	.01	0
	SVC	.45	.45	0	.10	0
Current	I/O	.90	0	0	.10	0
State	Overhead	.82	.0	0	.10	.08
	Idle	0	0	95	0.1	.04

TABLE 4—Average time spent in current state before moving to next state in 1/300 second

		Next State: User Exec	SVC	I/O	Overhead	Idle	Mean
	User			(1)			
	Executing	0	1.77	1.17	49.29	0	1.49
Current	SVC	.75	3.08	0	.91	0	1.81
State	I/O	.92	0	0	1.25	0	97
	Overhead	.20	0	0	1.0	1.54	.38
	Idle	0	0	14.65	237.39	8.73	15.74

tral Memory. In the case of PIL, the interpreter itself is resident in the main storage while its data, or the text of the user's program, is resident in LCS. Timing tests have indicated that while there is some degradation of PIL's speed due to this (20% or less), it probably is not significant when compared to either swap or batch mechanisms for other Time-Sharing Systems.[9] While the LCS proves to be very economical on the Model 50, its effectiveness would be somewhat reduced on a Model 65, and probably greatly reduced on a Model 75 because of their greatly increased internal speeds. If we were to move the system to a Model 75, it is quite likely that rather than executing out of LCS, we would use the LCS as a swap device analogous to a drum. One of the reasons that the LCS has proved very effective is that many of the requests within a Time-Sharing System are for very minimal amounts of Central Processor time. Of all the times when a user is put into execution, the majority of these times he has requested execution for the purpose of entering a single line to a processor such as the assembler or the interpreter. These requests can be serviced in a matter of several thousandths of a second, and the relative lack of speed on the large capacity storage has no significant effect on performance.

The second area in which we have drawn some conclusions with regard to the Time-Sharing System is that the file system is probably the most important single part of the system. This is true both in the sense of being a major concern of users, and of being a major potential bottleneck if handled poorly. Consistent with data presented in earlier papers by other authors, each request for service by a user generates a demand for several thousand characters of data to be transmitted from the disk system.[10] Our early implementation of the file system was a very straightforward, unblocked and unbuffered tape simulation. The file system was sequentially structured. It soon became evident that the pattern of requests by users was such that a tremendous amount of head movement was necessary for this unblocked

[9]Considerable work has been done on the use of LCS as the swap/paging mechanism, and this provides great promise for future development. It is currently not possible to page on the Model 50, and this limits the value of this alternative.

[10]This is user owned data and does not include any "overhead" transmission.

mechanism. To improve the performance we have come to quite a different concept on the disk system. The basic unit of physical activity on the disk system is now the reading or writing of 2,048 character records. The logical records, as seen by the user, are all treated as being variable length and are blocked automatically by the system. Here also LCS is of great value since 50-100K of buffers make system performance more efficient.

A descriptor, providing a record of the blocking, is associated with each 2,048 byte record and the system can pull these records apart very rapidly upon request by the users. On a single physical read or write, the user now makes available to himself 25 card images, for example, thereby causing 1/25th of the number of seeks and interrupts to be processed. This has had a tremendous impact on the performance of the system both in the amount of time that the system has kept idle because of having to access disk records, and the amount of elapsed time while a user is doing something such as assembling, loading or compiling. The improvement was on the order of 30 to 1. Given a high speed file system such as this, and a very easy mechanism for using it, the users rapidly learn to take advantage of the facility. This is important not only to the user whose time is saved, but it is also extremely important to the system since it can accomplish much more working from disks than it could working from other devices. Further, it expands the size of jobs which are feasible under the system. Also, it greatly decreases the probability that an I/O request will require physical activity, entailing possible changing of users which may also require swapping or paging. It will also reduce the number of inter rupts.

A third area in which we have drawn some conclusions regards the integrity of performance of a Time-Sharing System. We have learned over the course of two years, not always in a painless manner, that the user of a Time-Sharing System is quite different in his demands than the user of any other system. He expects the system to be on when it is scheduled to be on, and to perform in a rather stable manner while it is on. In particular, he expects the file system to perform perfectly at all times so that any work he does will never have to be redone. The user of such a system typically has a scheduled time at which he and the system are to work together and if the system is not available at the time, he cares very little why or what kind of excuses you can present. The user is typically quite reasonable if you give him adequate warning that the system will not be available at any given time.

Maintaining the software programs of such a system is of considerable difficulty and should not be further complicated by the introduction of machine performance problems. The area in which we have incurred the most difficulty is that manufacturers do not understand systems which are to run reliabily for extended periods of time. There seems to be no comfortable middle ground between extreme safety built into systems such as the FAA and other military systems, and those systems of the second generation which ran adequately for a batch processing environment. This lack of understanding is found on many levels but perhaps most significant is the lack of any facility for informing the user when malfunctions have occurred and the system is off the air. In general, there are many small rough edges in present equipment. An area of difficulty is that some part of the configuration usually is in marginal condition. The installation does not wish to terminate service completely for an extended period. The maintenance of the machine typically requires this. The machine normally runs in a partially crippled condition or not at all. An overall change in the maintenance methodology must occur soon.

A fourth area of experience relates to the use of the console system versus the use of its batch equivalent. Since the user has accessible identical facilities under each environment, we could expect him to move to that manner of operation which performed best for his job. This experience must be regarded carefully since the batch system often requires a long walk whereas, console service is available locally.

Clearly, in the area of interactive languages with emphasis on conversation, you would expect the user to remain on the console and this has been our experience. It is very unusual to find any user submitting a batch job which is a PIL or context editor run. One other use of batch is the creation of tape and disk files for use from the console.

The Assembler and Fortran are not overly conversational and do not take great advantage of the fact that the user is available. In the case of Fortran, the work is about equally divided between jobs submitted under the batch system, and jobs executed from the console. The Assembler, which

gets its main use by students in beginning programming courses, is probably used more under the batch system than under the console system.[11] After programs are running in either of these languages, the user typically will leave his object program on disk and will revert back to using the console for execution or for final debugging. It should be noted here that in addition to the Time-Sharing System on the Model 50, the Center makes available a batch processing system on an IBM 7090. There seems to be a rather clear dividing line rapidly discovered by the users between the jobs which should be done on the 7090, and jobs which should be done on the Model 50. The two machines are of approximately the same capabilities as far as speed and storage space are concerned. Those jobs which are heavily compute bound remain on the 7090 as do any jobs which gain little or nothing by interaction.

Because of the lack of success of other installations in using a Time-Sharing System for all of their computing and because of our own success with the two independent systems, we plan to provide a 360 batch system (OS/360) on a second Model 50 to maintain systems which are fitted to the differing demands by our users. Running the two facilities in parallel should give a clearer picture in the future of the relative advantages of time-sharing and batch processing.

Future directions

The system is still under development and we are continuing to add facilities if we find them useful as manpower permits. We are in the process of adding other processors including Algol, PL/1 and further extensions of Fortran. We also plan to implement some kind of a random access and index sequential file structure although it is not clear that these will be implemented within the shared file structure we currently use.

Another useful service that we plan to implement is the ability to submit jobs to either Model 50 system from the remote consoles. In order to implement this, we will put a channel-to-channel attachment between the two Model 50's as well as making parts of the 2314 disk available to both systems. How far we will push this effort is undetermined at present. Finally, in an effort to reduce the number of interrupts presented to the system, and in order to maintain effective utilization of unit record equipment, we are putting together a spooling package for the printer, punch and card reader. This will allow blocking to be done by the channel program with one interrupt per block. These spoolers will work in such a way as to present a job stream which runs from one disk file to another in the same manner as the batch job stream currently runs from the card reader to the printer.

[11] This is at least partly due to the fact that some understanding is required to be able to effectively debug machine language at a console.

BIBLIOGRAPHY

1 *The Pitt time-sharing system for the IBM system/360*
Computer Center University of Pittsburgh March 1968
2 ———*The Pitt interpretive language for the IBM System/360*
March 1968
3 D W FIFE
An optimization model for time-sharing
Proceedings SJCC 1966 pp 97–104
4 J D FOLEY
A Markovian model of the University of Michigan executive system
CACM Vol 10 No 9 September 1967
5 T J O'CONNOR
Analysis of a time-sharing system
Stanford University Unpublished PhD Dissertation
6 E PARZEN
Stochastic processes
Holden-Day San Francisco 1965 Chapter 6
7 A L SCHERR
An analysis of time-shared computer systems
The MIT Press 1967