COMPARING USER RESPONSE TIMES ON PAGED AND SWAPPED UNIX

by

Luis Felipe Cabrera and Jehan-François Pâris

Memorandum No. UCB/ERL M81/39

1 June 1981

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Comparing User Response Times on Paged and Swapped UNIX by the Terminal Probe Method

Luis Felipe Cabrera

Department of Mathematics
and
Electronics Research Laboratory
University of California, Berkeley
Berkeley, CA 94720

Jehan-François Paris

Department of Computer Sciences
Purdue University
W. Lafayette, IN 47907

ABSTRACT

In this paper we present a comparison of user response times on paged and swapped versions of the operating system UNIX for the DEC VAX 11/780. The technique used was to construct a script that periodically evaluated the system's work load and measured the system's response times to a set of benchmark programs.

These measurements, collected at two different sites, show that differences of responsiveness observed between the two systems depended much more on the workloads and the configurations than on the operating systems themselves.

Since we only used standard UNIX tools to build our scripts, they are highly portable and can be installed on any standard UNIX system in a matter of minutes without bringing the system down.

1. INTRODUCTION

A change from one version of an operating system to another one always constitutes a difficult problem for the management of a computer installation. The problems of updating the software while attempting to keep the user's interfaces as unchanged as possible are well known. We will address here another problem, namely the performance implications of the change.

Numerous tools already exist with the purpose of predicting or measuring the performance of a computer system. In contrast to many of these tools, the method we present here requires no special hardware and very little effort for modeling the system or estimating its work load. The technique used, which applies to all interactive systems, essentially consists of constructing a script that periodically evaluates the system's work load and measures the system's response times to a set of benchmark programs. As we will see, the only drawback of the method is that systems with highly variable work loads will require longer data gathering periods.

Since standard UNIX tools were used to build these scripts, they are highly portable and can be installed on any standard UNIX system in a matter of minutes without bringing the system down. Moreover, the philosophy of the method is by no means specific to UNIX systems and could be applied to other interactive systems.

Section 2 of this paper presents those features of the UNIX operating system which are relevant to our problem. In Section 3 we introduce the data gathering method. In Section 4 we present and analyze diagrams which compare the performance of the installations under the different versions of the operating system. Section 5 discusses some general issues about comparing distinct operating systems. Section 6 has our conclusions.

2. THE UNIX SYSTEM

UNIX is a trademark for a family of time-sharing operating systems developed at Bell Laboratories during the last twelve years [Ritc74, Kern81]. The first version of UNIX was implemented in 1969 on a PDP-7. Since then, several versions of UNIX have been developed at Bell Laboratories. They were primarily aimed at various members of the PDP-11 family of minicomputers and, in 1978, Bell Laboratories released a version of UNIX aimed at the new VAX 11/780. Despite the fact that the VAX hardware was designed to support a paged virtual memory system (DEC's own VMS), the successive versions of UNIX developed at Bell Laboratories did not support paging. This motivated a group at the University of California, Berkeley, to implement a virtual memory extension to UNIX for the VAX [Baba79]. This extension used a Global LRU page replacement strategy with use bits simulated by software and is now an integral part of what is known as the "Berkeley UNIX."

One of the most remarkable features of UNIX is its user interface. On logging in, each user is assigned a special process containing a command interpreter that listens to the terminal. This command interpreter, known as the *shell* [Bour78], parses the input line decoding the command requested, its flags, and the arguments passed to it. The shell then *exec*'s the command. The shell also has a redirection mechanism that allows it to read command lines stored in files. Users can thus define sequences of shell commands, known as shell *scripts*, and store them in files awaiting later invocation. The versatility of these scripts is greatly enhanced by the fact that the shell language also contains control-flow primitives, string-valued variables and even simple arithmetic facilities. Thus, building a sequence of commands that will be repeatedly executed at fixed time intervals is on UNIX a nearly

trivial task.

The portability of our tools was also facilitated by the fact that UNIX handles automatically all file allocation decisions. UNIX files are accessed through a hierarchy of directories and the typical user is not aware of the physical location of the files he or she manipulates.

3. THE DATA GATHERING METHOD

Our strategy for monitoring each system's responsiveness was the same one used in [Cabr80]. It consists of running periodically a set of predefined benchmarks in a totally automatic way. This was achieved by writing a shell script that is essentially a loop containing the benchmarks together with commands that gather statistics about the work loads and measure the time it takes each benchmark to complete. Each time the script has cycled through the execution of these commands, it executes a *sleep* command that suspends its execution and then wakes it up after a predetermined number of seconds. This script is then run as a background job (with the same priority as any user process) during the operation time of the system.

This data gathering method can be categorized as a time-sampling method [Ferr78] and in fact is very similar to Karush's *terminal probe* method [Karu69]. By using it, we measure the work load of the system as well as the dependency of our performance indexes on the underlying equipment. We are thus evaluating the performance of an installation.

Although running a script affects the load of the system--and thus its responsiveness, it was felt that this would not affect the validity of our comparison study since each system would be presented with the same script. The main purpose of our comparison experiment was precisely to observe how each system reacted to this stimulus.

Our commitment to use only standard UNIX tools decided us upon the usage of the *time* command for measuring the completion time of each benchmark. The *time* command returns, upon completion of the command it prefixes, three measurements; response time, system time and user time. Response time is only accurate to the second (*time* truncates, does not round off). This low resolution of *time*, together with our desire that no individual measurement be off by more than 10%, led us to restrict ourselves to benchmarks that would never take less than five seconds to complete.

4. THE MEASUREMENTS

The design of a comparison study like ours is faced with many constraints and trade-offs. There are basically three main areas in which major decisions have to be made: the selection of the benchmarks, the length of the data gathering period and the representation of the data.

Since our method measures the performance of an installation, i.e., the work load is also included in the observations, a desirable goal is to use as benchmark a task representative of the user's tasks. At the same time, one must try to capture the work load through some characterization. Then, the response time of the task is plotted against the characterization of load.

We have chosen tasks which exercise the system in a "natural" way (i.e., we did not run tasks which would *a priori* perform better in a paged environment or in a swapping environment) and which were representative of the user's activities at the time. This led us to use, in both sites, the command *man man*, which retrieves and formats the entry for the manual page out of the on-line copy of the UNIX Programmer's manual. This task is interesting because the entry is retrieved from disk using the widely used formatting program *nroff*. To avoid screen output problems when running the script,

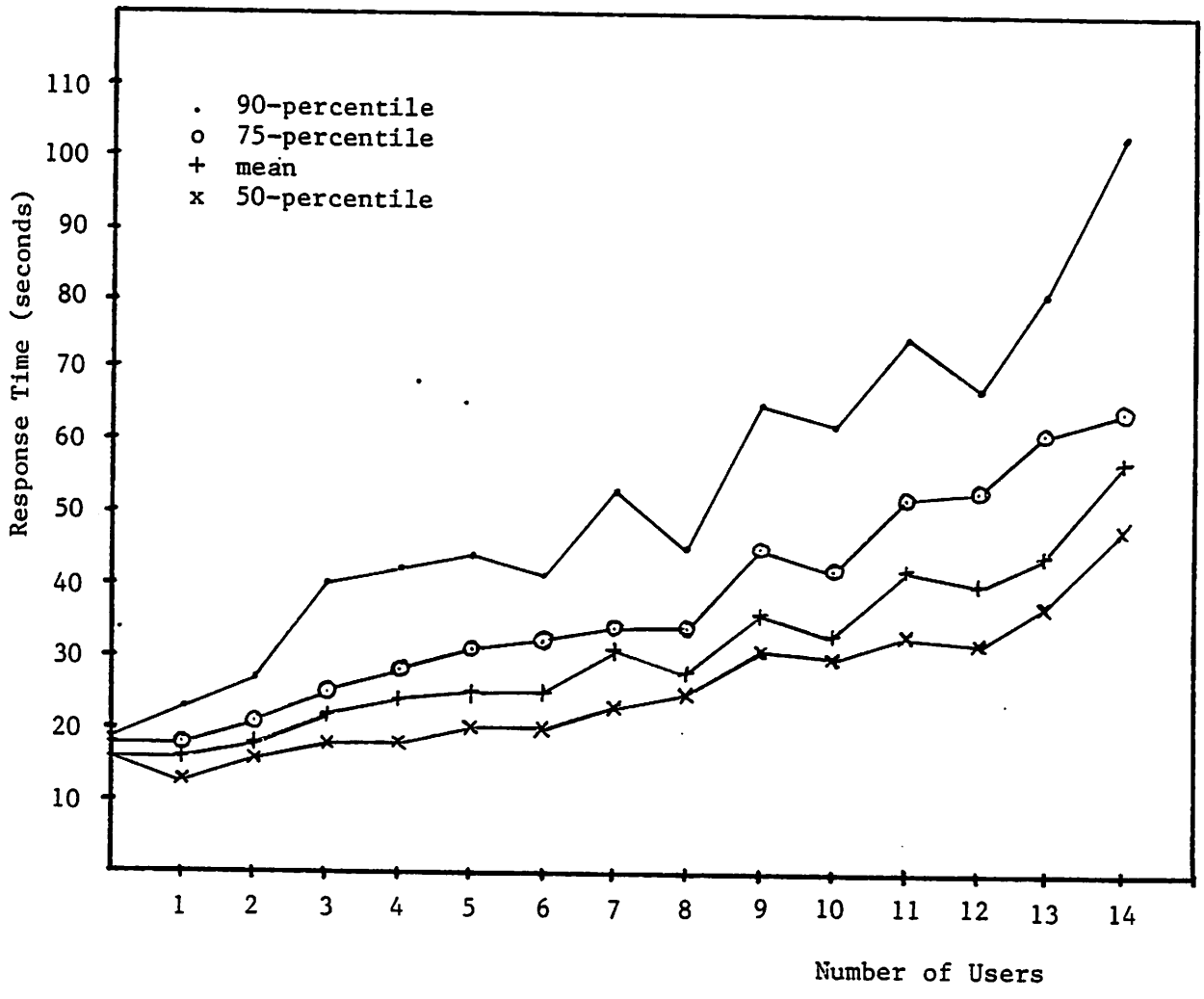

Figure 1 **Median, Mean, 75 and 90 percentiles for man man Swapped UNIX**

the output of *man man* was sent to /dev/null instead of sending it to a real terminal. This had the effect of discarding the already formated text of the retrieved page. The length of each data gathering period was determined at each site and will be discussed in the next two subsections. As for the display of the data, after an exploratory analysis of our measurements it was decided to use 75-percentile curves. In Figures 1 and 2 we display four curves for the task *man man* using the number of logged in users as characterization of load. The four curves are the median (50-percentile), mean, 75-percentile and 90-percentile of the distribution of response times per

value of our work load characterization.

Figure 1 displays these curves for the swapping system and Figure 2 for the paging system. In them we may appreciate how the influence of the outliers becomes apparent in the 90-percentile curve. We may also notice that the median of the response time samples are consistently lower than the mean of the samples. This happens in any distribution which is skewed towards the lower range values or, whose outliers lie in the higher range of the values. These two characteristics were common to all our data. To

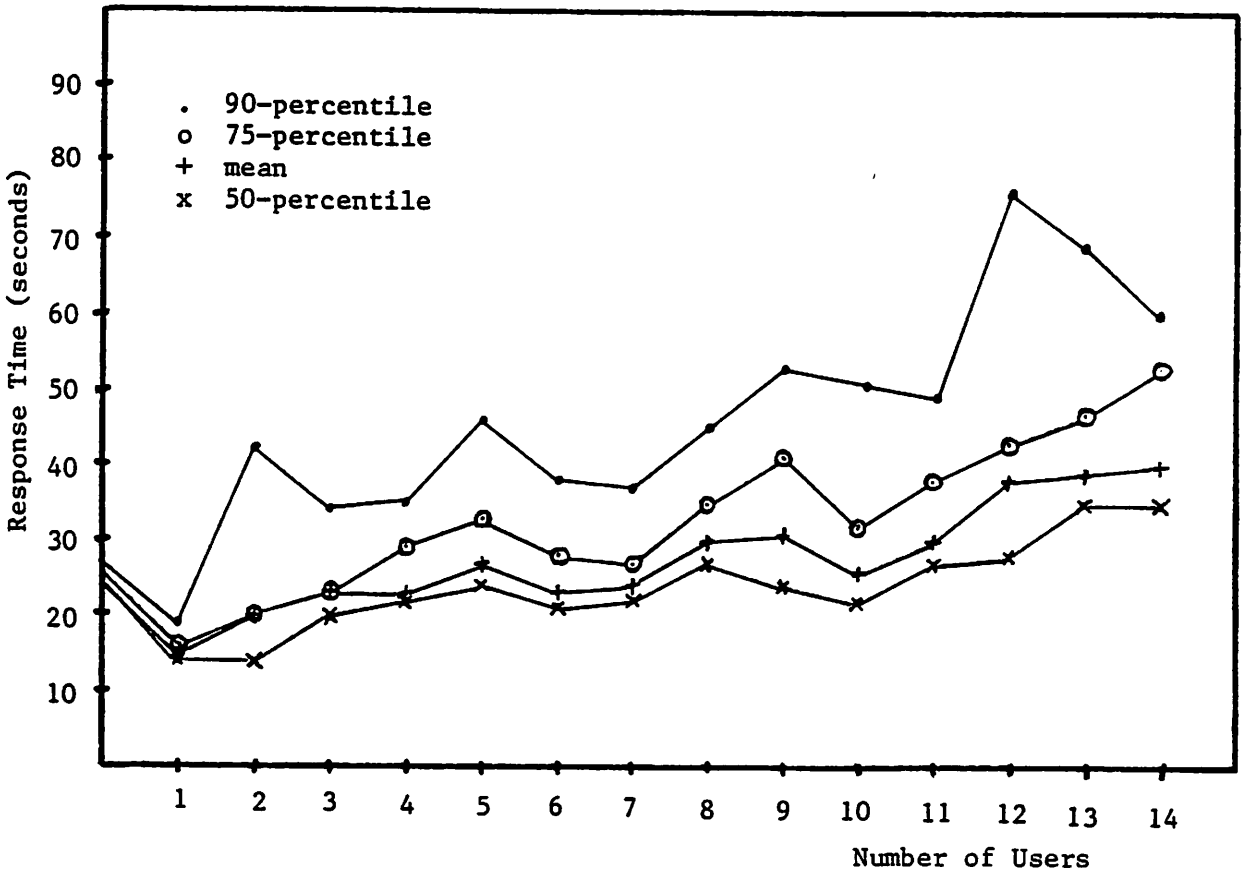

Figure 2 **Median, Mean, 75 and 90 percentiles for man man Paged UNIX**

deemphasize the effect of outliers we chose to use the 75-percentile curve to display our data. This decision was partially motivated by unavailability of abundant data from one of the sites.

We decided on two single-variable characterizations of load: number of users logged in and number of user processes. This latter index represents the number of processes which have been generated by user commands. It does not include any system generated processes.

4.1. The Berkeley Measurements

This analysis was based in 431 data points for the swapping system and 1455 points for the paging system generated during the second half of 1979. Throughout the data gathering period the system's hardware remained unaltered. The work load, however, underwent a substantial change (by very large programs such as the VAX version of the algebraic manipulation system MACSYMA [Fate79]) to thoroughly exercise the memory management capabilities of the new kernel.

Given the small amount of main memory the system had at the time these large address space programs did influence significantly the work load of the system. The hardware configuration of the system was a VAX 11/780 CPU, 512K bytes of main memory, 16 ports, two RP06 disk drives with a DEC disk controller and one TE 16 tape drive.

We shall present data for three tasks: the command *man man*, the execution of a cpu-bound job and the compilation of a short C program. The choice of the cpu-bound job was motivated by our desire to observe the effect paging had on a task that would almost never be swapped out: its size is 2K bytes.

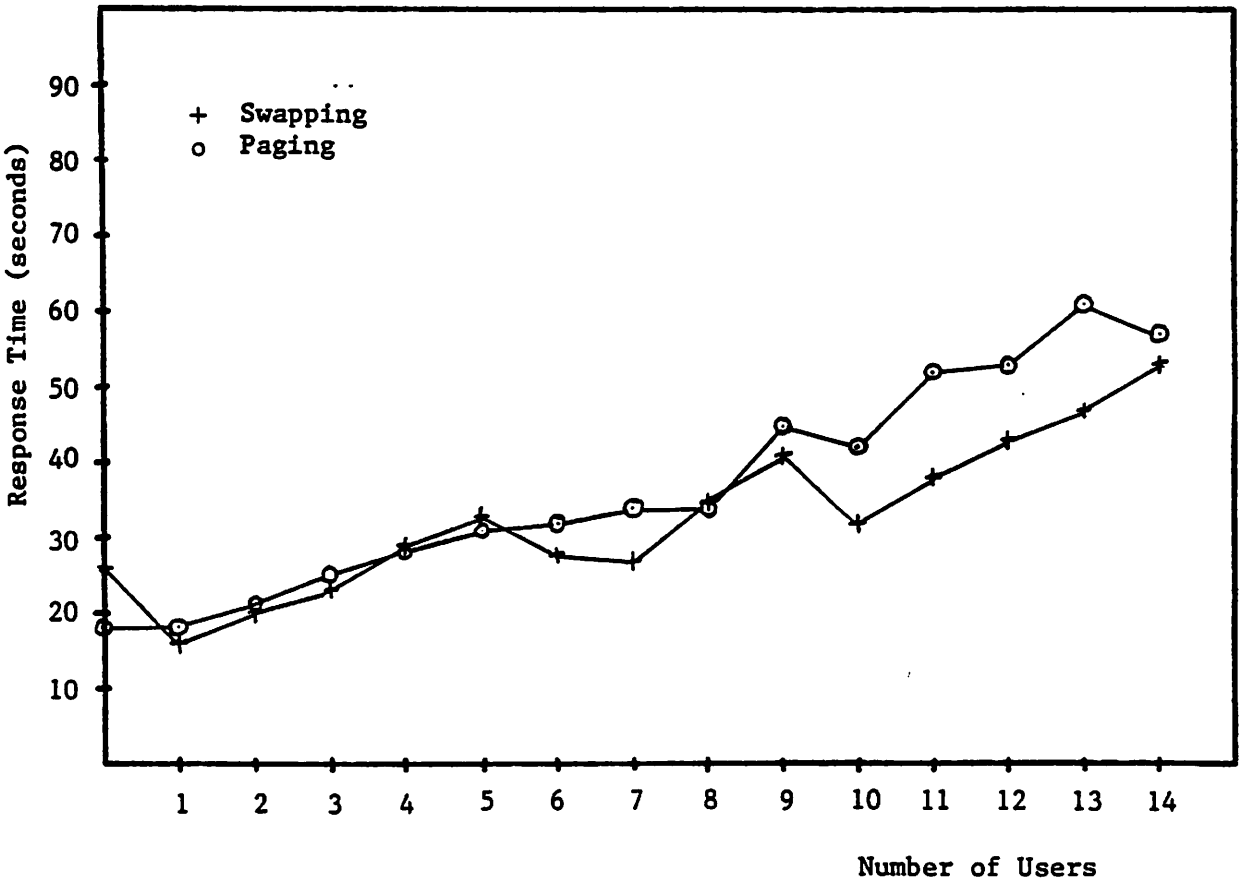

Figure 3 Response Time 75-percentile vs Number of Users for the man man command

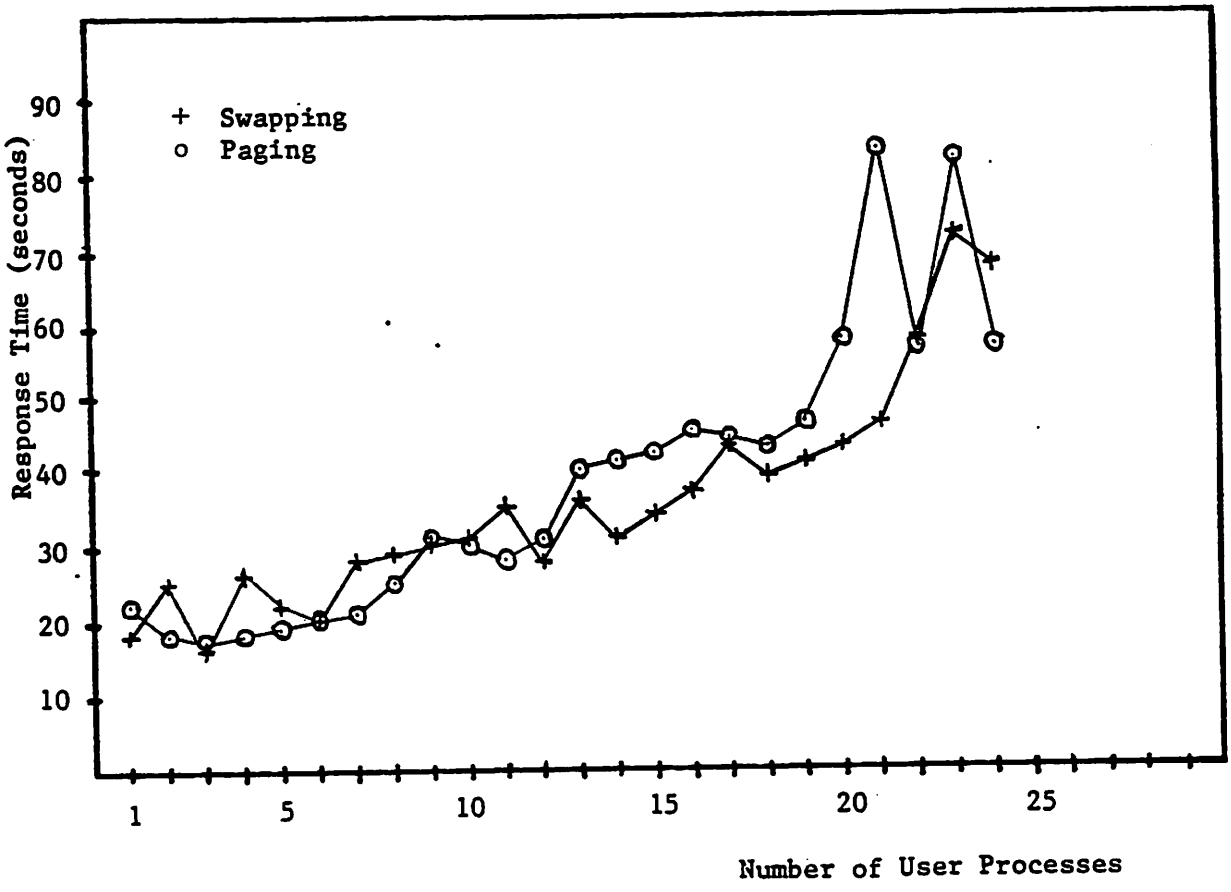

Figure 4 Response Time 75-percentile vs Number of User Processes for the man man command'

Figures 3 and 4 display the 75-percentile curves of the response time for the command *man man*. On both figures we see that the swapping system performs slightly better than the paging system at higher load levels. This difference can be justified by the dramatic change in the workload which occurred when the paging system was brought up. Since we do not have data from stand-alone measurements, we cannot rate the two systems solely on these data, but it becomes apparent that none outperforms the other one throughout all the values of load.

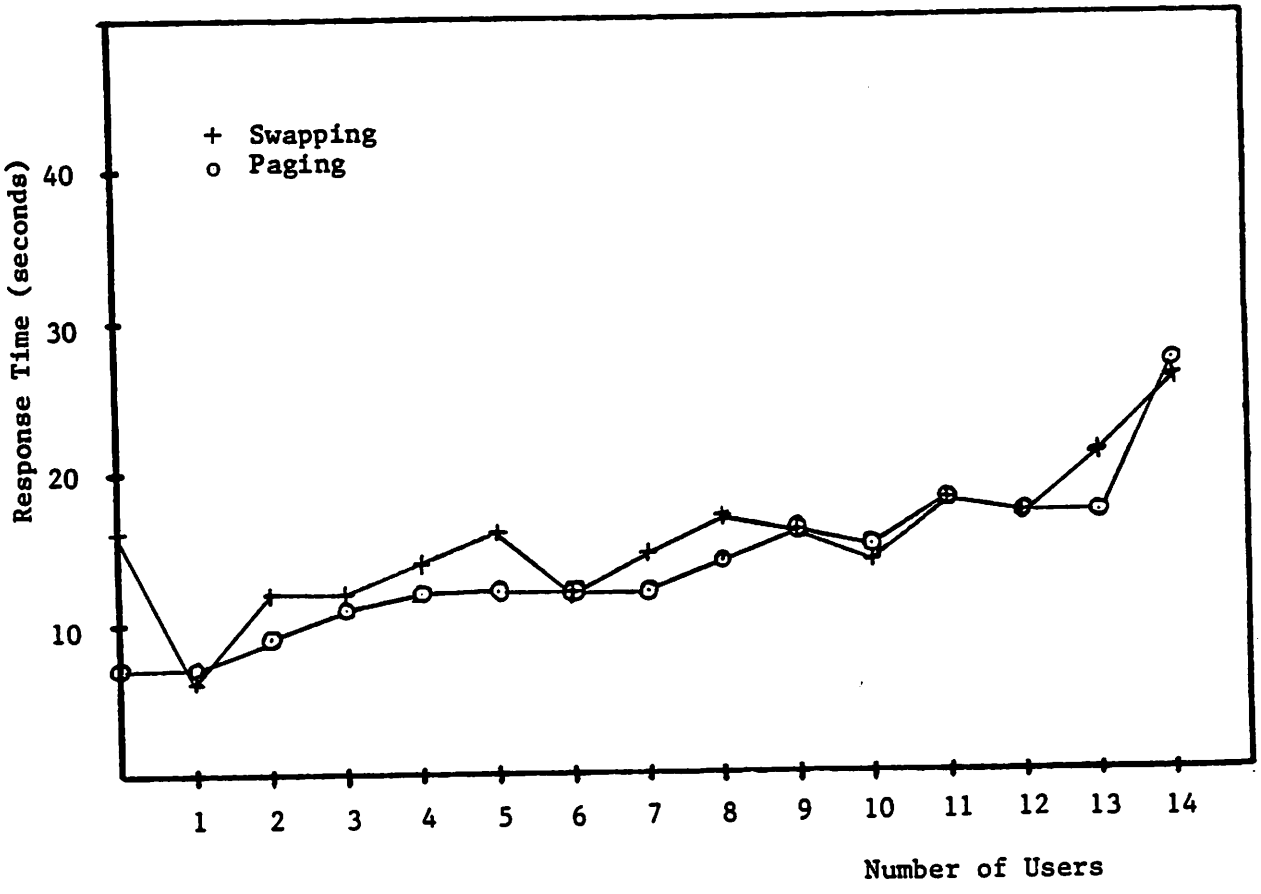

Figure 5 Response Time 75-percentile vs Number of Users for the CPU-bound job

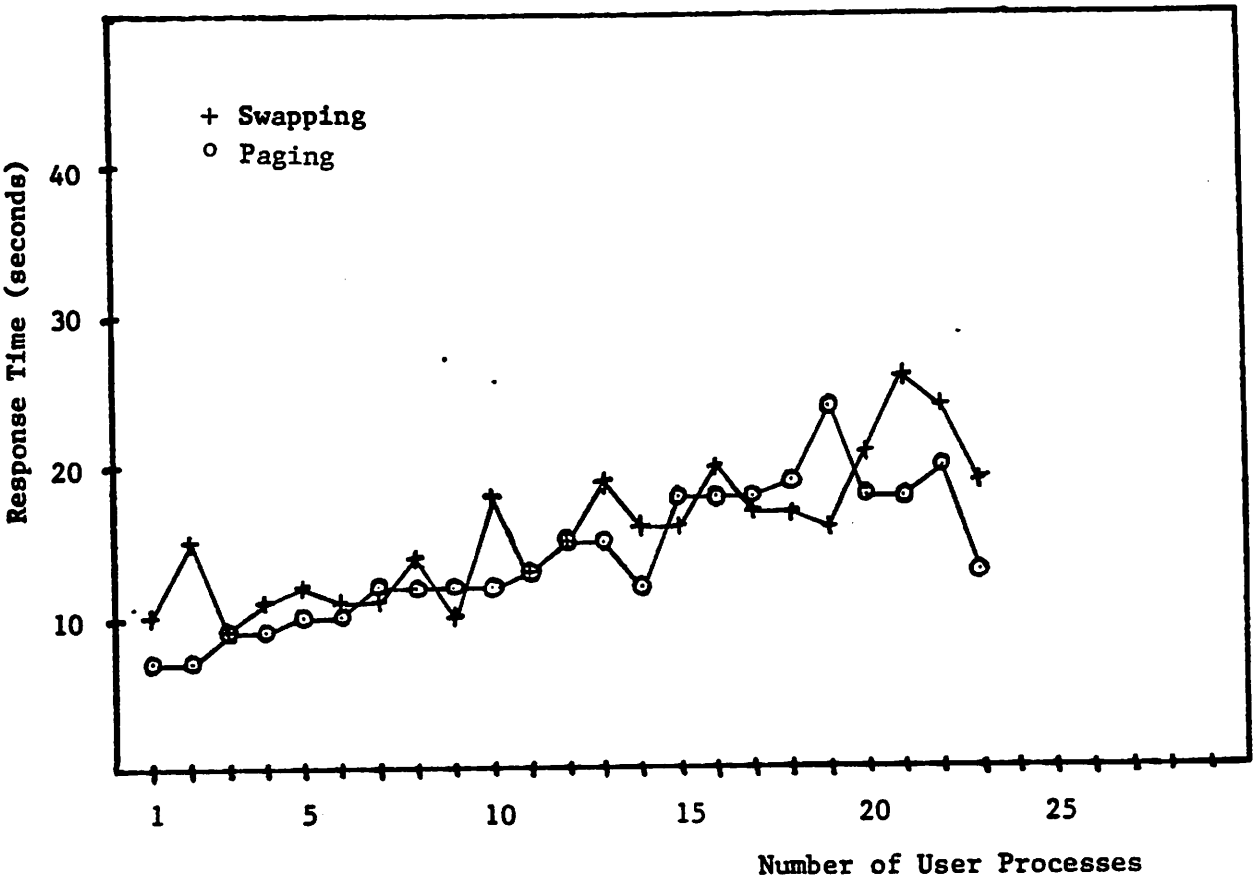

Figure 6 Response Time 75-percentile vs Number of User Processes for the CPU-bound job

Figures 5 and 6 display our data for the CPU-bound job. This job consists of two nested loops and an inner block of statements. A 9 statement sequence of integer arithmetic operations is executed 100,000 times. The size of the object code is only 2K bytes--i.e. four 512 byte pages-- and thus its probability of being swapped out is quite small. It is quite interesting to observe that the paging system outperforms the swapping system under both characterizations of the workload. This can be explained in terms of the additional I/O activity existing in the paging system. Indeed, since our job is very small in size and performs no I/O, the only impediments that may block

it to run to completion are time-quantum expirations and multiprogramming delays. Thus our cpu-bound job is either running or in the ready queue. In the paging system it gets more often the CPU because of other jobs being blocked by page faults. This fact makes us believe that trivial tasks should perform better in the paged version of UNIX.

Figures 7 and 8 display our data for the C compilation of a small C program. This is the only observed task where the swapping system appears superior. This is specially clear in Figure 8 where the number of user

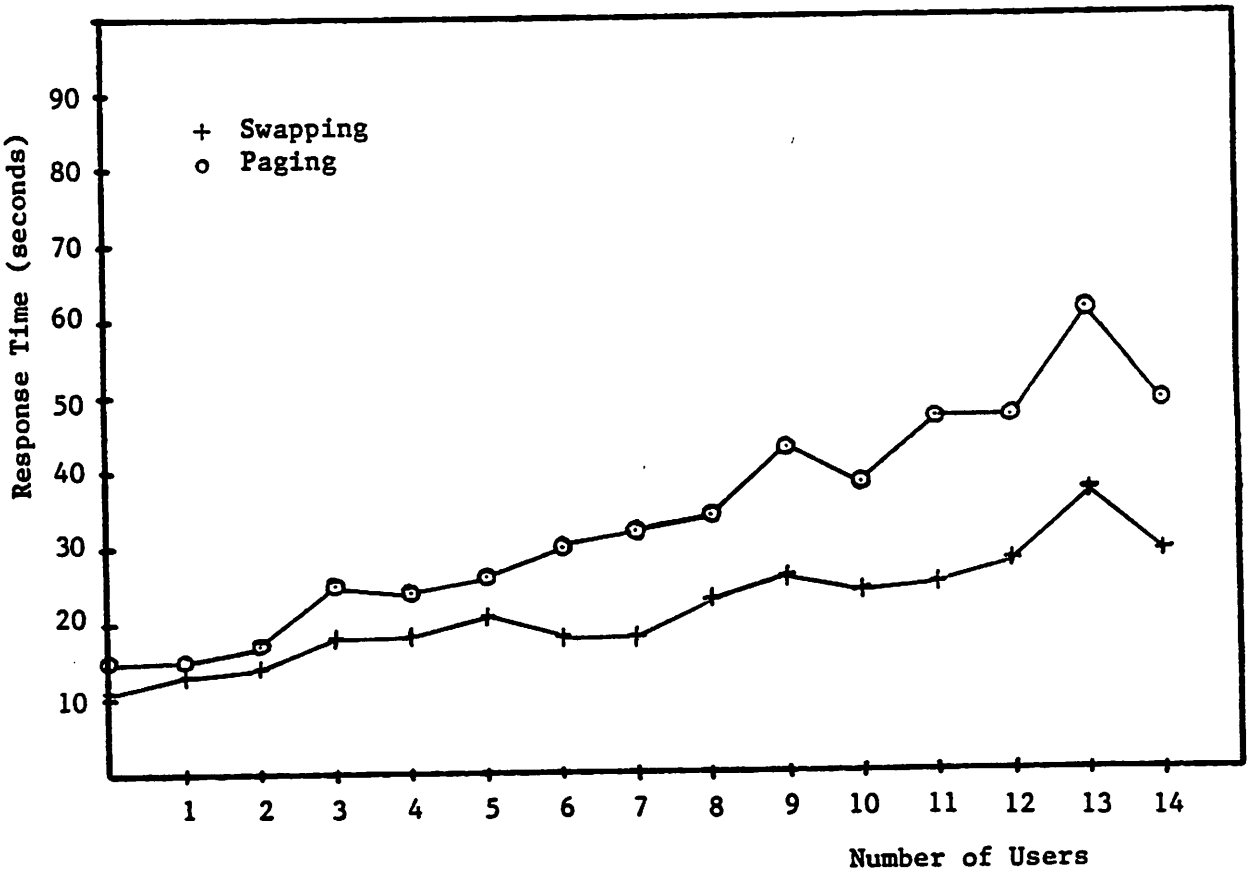

Figure 7 Response Time 75-percentile vs Number of Users for the C compilation

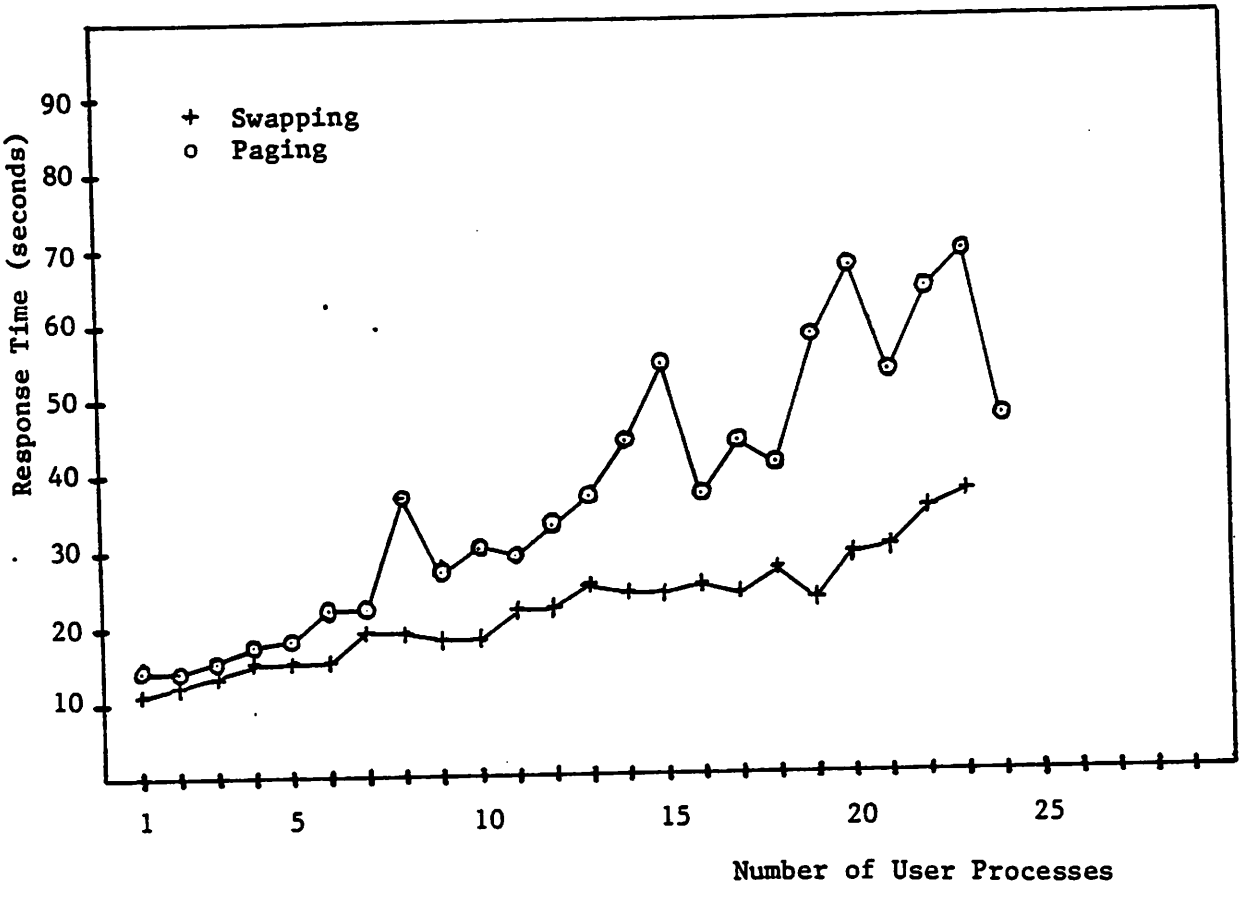

Figure 8 Response Time 75-percentile vs Number of User Processes for the C compilation

processes characterization of load is used. We believe that the difference in performance observed in this case is mostly due to the change in workload which occurred in the system. The execution of processes whose size was much larger than the available memory created high contention for memory. Thus medium size processes like the C compiler would always be losing their used pages and paging in those pieces of code needed for further processing. Their resident set would never be allowed to remain at any reasonable size. The effect of large processes was quite noticeable at the time, specially when processes like VAXIMA [Fate79] would do garbage collection through a 2

Megabyte address space. At those points, response time for all commands would degrade ostensibly.

4.2. The Purdue Measurements

These experiments were performed during the Summer of 1980 on the VAX 11/780 of the Department of Computer Sciences at Purdue University when the Department decided to switch from UNIX version 7 to Berkeley UNIX. At that time, the machine had 3 Megabytes of main memory, 56 ports, three RM03 disk drives on Massbus 0 and one TE16 tape drive on Massbus 1.

We chose to run as benchmarks the UNIX command *man man*, which formats and displays the entry of the UNIX manual describing the command itself as well as a small script containing "man man" and several small tasks. As said before, this choice was motivated by the fact that text processing constituted then the application concerning the greatest number of users. In UNIX version 7, the *man* command is implemented as a *shell script* while Berkeley UNIX uses directly executable code. Since this latter alternative is inherently much faster than a script, which must be interpreted by the *shell* at each execution, a direct comparison of the response times for the *man man* command would have been grossly unfair to the version 7. We thus decided to run the version 7 script in place of the original *man* command in our measurements with the Berkeley UNIX. As a result, the response times for the *man man* command measured at Purdue were much higher than those observed at Berkeley, where the same problem did not occur.

Because of the short interval of time left between our decision of running the script and the scheduled switch from swapped to paged UNIX, we were only able to collect 152 measurements with the swapped version of UNIX. This left us with about ten observations for each load level, as expressed

either by the number of users logged in or the total number of user processes. These values were obviously much lower than those required to obtain acceptable estimators of the 75 percentiles of the response time.

Faced with the same problem, but on a much smaller scale, one of the authors [Cabr80, Cabr81] decided to cluster neighboring load levels with insufficient numbers of observations. For instance, if 30 was the minimum acceptable sample size and there were 19 observations corresponding to 15 users logged in and 12 observations corresponding to 16 users, the two sets of observations would be merged into a single set of 31 observations and this set made to correspond to a work load of (15+16)/2 = 15.5 users. The same approach applied to our Purdue data would unfortunately resulted into too little load levels after clustering. We decided therefore to use a scheme in which the 75 percentile for the load level i would be computed taking in account all the measurements at load levels $i-1$, i and $i+1$. This filtering greatly reduced the influence of outliers on the 75 percentiles. It has, however, the unfortunate side-effect of introducing a positive correlation between neighboring values on the percentile curves and therefore should not be considered as a substitute for more measurements

Figure 9 and 10 display the 75-percentile curves of the response time for the script version of the *man man* command. As one can see on both figures, the response times for the swapping version of UNIX appear to be somewhat higher than those corresponding to the paging UNIX and exhibit also a more erratic behavior. These results apparently do not agree with those observed on the Berkeley VAX. One should however point out that the Purdue VAX had a much larger memory and that all our measurements relative to the paging version of UNIX were made during the month immediately following the

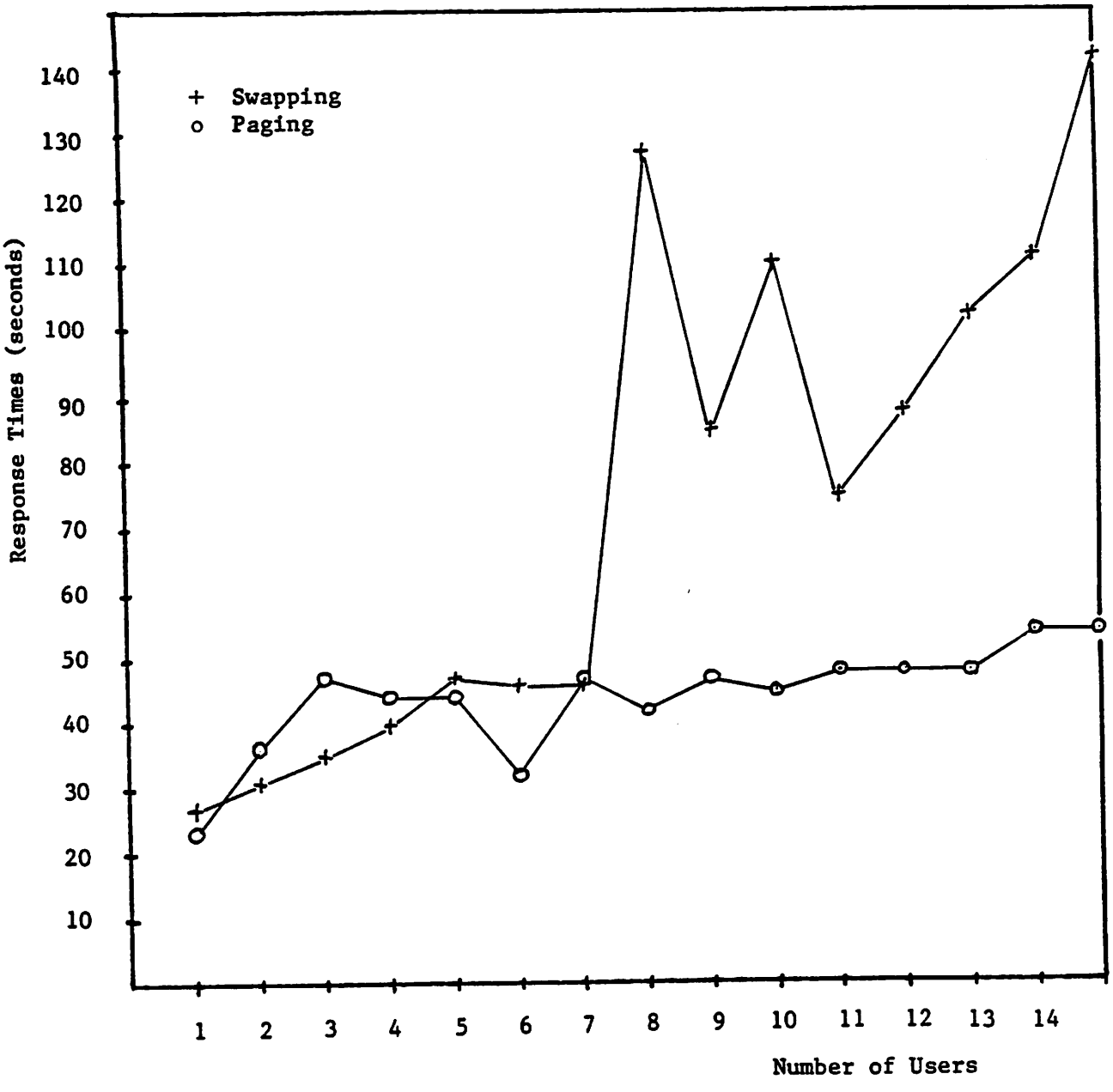

Figure 9 Response Time 75-percentile vs Number of Users for the script version of man man

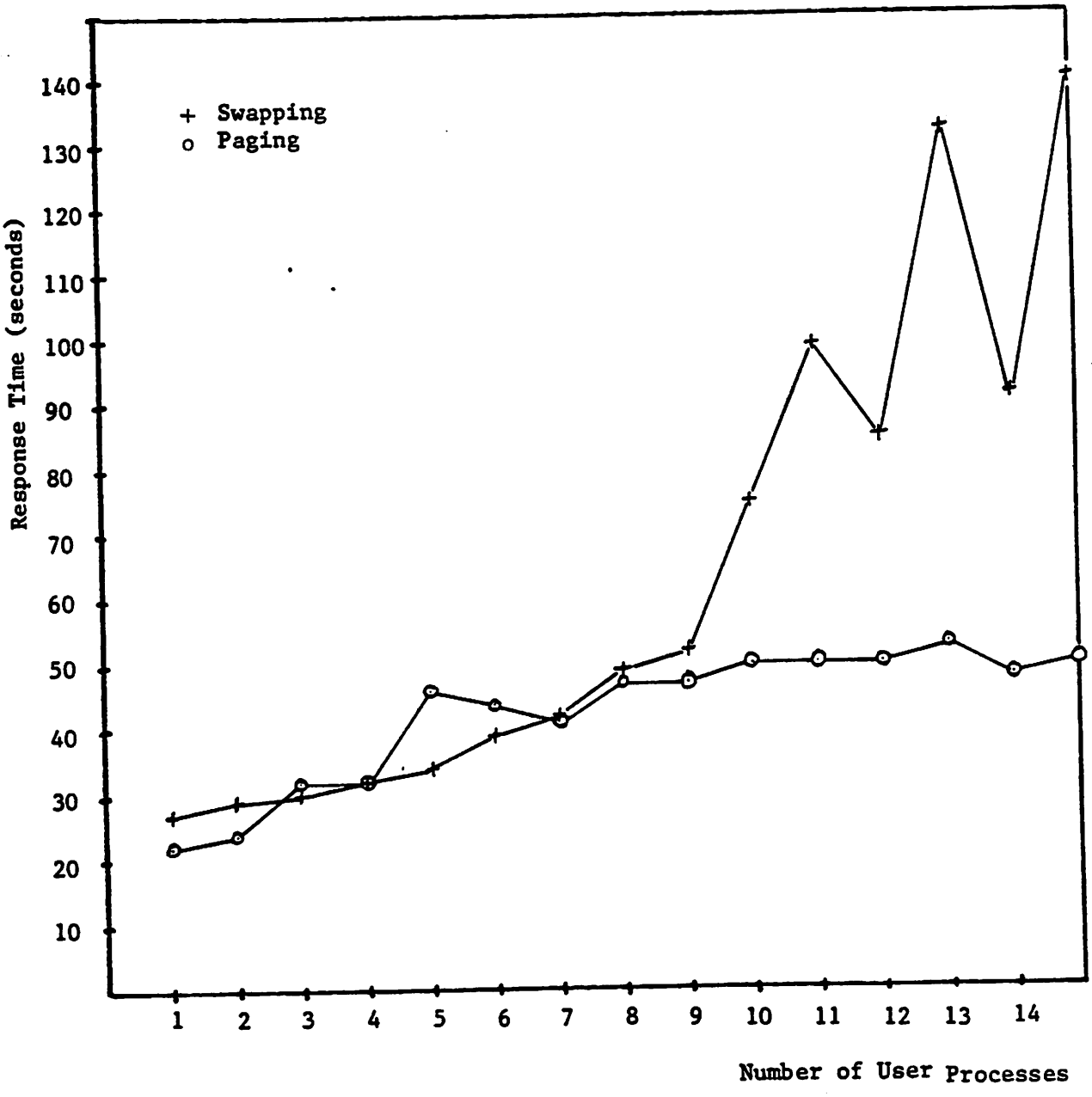

Figure 10 Response Time 75-percentile vs Number of User Processes for the script version of man man

conversion to Berkeley UNIX, thus *before* any change in the working habits of the users could have occurred.

Another point to mention is that the curves representing the 75-percentiles of the response time against the number of user processes are somewhat better behaved than those corresponding to the number of users. This suggests that the number of user processes is a better estimator of the system's workload.

5. DISCUSSION

One can conclude from our measurements that the switch from UNIX version 7 to Berkeley UNIX did not alter significantly user's response times on any of the two monitored installations. This conclusion, however, depends on the work loads observed on the two machines and is by no means an answer to the question: "Which one of the two systems is better?"

Paging was introduced, more than twenty years ago, in order to allow bigger jobs to run on machines then characterized by very small main memories. It became later a nearly universal feature of large-scale multi-access systems because it allowed to keep more jobs simultaneously residing in main memory, thus avoiding swapping delays. As it was quickly noticed, paging unfortunately never comes for free: it requires special hardware, introduces a fair amount of software overhead and performs very poorly with programs exhibiting scattered reference patterns.

One may then question the efficiency of paging at a time where main memory is so cheap that it becomes possible to keep residing in memory enough conversational users to saturate the CPU of a machine like the VAX. Would this be the normal case, jobs would typically run without having been ever swapped out during their execution. This would remove any incentive for implementing a paging scheme. Moreover, it would even make straight swapping more effective than paging since it is usually more efficient to

bring into memory the whole address space of a program in a single I/O operation than by successively fetching faulting pages.

This argument does not, however, stand against the well-known fact that larger address spaces have always resulted in larger programs. In fact, one of the strongest motivations of the Berkeley UNIX was to provide the larger address space required by algebraic manipulation programs. Our measurements on the Berkeley VAX show indeed that the switch from swapping to paging has significantly altered the system's work load by allowing bigger jobs to run on the machine. Although it did not appear in our data, the same phenomenon also occurred on the Purdue VAX, whose current work load is strikingly different from the one existing before the switch to Berkeley UNIX.

The main advantage of one operating system over the other one is thus more a question of increased capabilities than faster response times. As a cynical observer could point out, the switch to a better system might be accompanied by an increase of the average response time resulting from the increased demands placed on the system's hardware.

6. CONCLUSIONS

We have presented here a simple method for comparing user response times on two versions of an operating system. The method can be very easily implemented on UNIX and UNIX look-alike systems. It does not require the system to be brought down and does not affect the normal operation of the installation.

Results concerning a paged and a swapped versions of the VAX 11/780 UNIX system show that the observed differences of responsiveness between the systems depended more on the workloads and the configurations than on the operating systems themselves.

Thus our methods should not be construed as a technique for comparing the inherent merits of two operating systems, but rather as a tool giving prompt quantitative answers on the responsiveness of any particular installation.

Acknowledgements

The work reported here was supported in part by the NSF grant MCS 80-12900. The authors express their gratitude to their respective departments for having provided the facilities used in this study.

References

[Baba79]
Babaoglu, O., W. Joy and J. Porcar, "Design and Implementation of the Berkeley Virtual Memory Extension to the UNIX Operating System," Department of EECS--Computer Science Division, University of California, Berkeley, (1979).

[Bour78]
Bourne, S. R., "The UNIX Shell," *The Bell System Technical J.* 57, 6 Part 2 (Jul.-Aug. 1978), 1971-1990.

[Cabr80]
Cabrera, L. F., "A Performance Analysis Study of UNIX," *Proceedings of the 16th Meeting of the CPEUG*, Orlando, FL, October 1980, pp. 233-243.

[Cabr81]
Cabrera, L. F., "Benchmarking UNIX: A Comparative Study," in *Experimental Computer Performance Evaluation* (D. Ferrari and M. Spadoni eds.) North-Holland, Amsterdam, Netherlands,pp 205-215.

[DEC78]
Digital Equipment Corporation, *VAX 11/780 Technical Summary*. Maynard, Mass., 1978.

[Fate79]
Fateman, R. J., "Addendum to the Mathlab/MIT MACSYMA Reference Manual for VAX/UNIX VAXIMA," Department of EECS--Computer Science Division, University of California, Berkeley (Dec. 1979).

[Ferr78]
Ferrari, D. *Computer Systems Performance Evaluation*, Prentice-Hall, Englewood Cliffs, NJ, 1978.

[Karu69]
Karush, A. D., "The Benchmarking Method Applied to Time-Sharing Systems," Rept. SP-3347, System's Development Corporation, Santa Monica, CA, August 1969.

[Kern81]
Kernighan, B. W. and J. R. Mashey, "The UNIX Programming Environment," *Computer* 14, 4 (Apr. 1981), 12-24.

[Ritc74]
Ritchie, D. M. and K. L. Thompson, "The UNIX Time-Sharing System," *Comm. ACM* 17, 7 (Jul. 1974), 365-375. A revised version appeared in *The Bell System Technical J.* 57, 6 Part 2 (Jul.-Aug. 1978), 1295-1990.

[Ritc78]
Ritchie, D. M., S. C. Johnson, M.E. Lesk and B. W. Kernighan, "The C Programming Language," *The Bell System Technical J.* 57, 6 Part 2 (Jul.-Aug. 1978), 1991-2019.