A REGRESSION MODEL OF A SWAPPING SYSTEM

by

Samuel T. Chanson

Technical Report 82-5

July 1982

Department of Computer Science,
University of British Columbia,
Vancouver, B.C., V6T 1W5.

Abstract

This paper describes a measurement experiment performed on a PDP 11/45 system running under UNIX (version six) which employs swapping rather than paging in managing memory. Regression equations relating the system's responsiveness to certain system and workload parameters are obtained. Sample applications such as predicting the system's performance due to workload and system changes, load control as well as representing the swapping behaviour in simulation and analytic models are presented. The similarities between the paging and swapping dynamics are discussed. The paper also includes a brief discussion of the accuracy of the model as well as the advantages and disadvantages of the regression technique.

1. Introduction

Paging dynamics have been studied extensively and various models (most noteably the lifetime function) exist which describe the relationships between program behaviour, system parameters and the paging rate [1,5,13]. By contrast, swapping,[1] commonly used in operating systems running on micro and minicomputers (such as UNIX) has not received much attention. As the interest in micro and minicomputers grows, the need to better understand the behaviour of swapping systems becomes apparent.

A project was undertaken on a PDP 11/45 running under UNIX (version six). The basic goal is to discover the major influences in the system and their relationships, and to try to find if there is a parallel between the paging and the swapping dynamics.

The approach chosen is, first of all, to determine probable workload parameters and pertinent hardware and internal system parameters and to develop a suitable performance parameter. Then a controlled measurement experiment, using a synthetic workload, is to be conducted varying the parameters and recording the values of the performance index. The results will then be analyzed using regression to try and establish some

---

[1] In this paper, swapping means that the entire program code of the process is moved in and out of main storage as a unit.

strong relationships among the parameters.

2. Setup of the experiment

A. The hardware

The host machine for this project was a PDP 11/45. It had 112K bytes of main storage, of which about 48K bytes were used by the operating system. On-line secondary storage consisted of two RK05 disk cartridges. These are small capacity (2.4 Mega bytes each) disks with slow transfer time (180,000 bytes/sec.). They operated under the same controller and did not have the capability to overlap seeks. There were also several tape drives, a printer, a plotter, a card reader as well as a few terminals. It was suspected that the system was disk-constrained.

B. Choice of performance index

A performance index should be sensitive to workload and system variations and provide a good measure of the level of performance for a given system configuration and workload. Since the primary concern from the user's viewpoint is the responsiveness of the system, it was decided that the index should reflect this. Because of the lack of an external

hardware driver to simulate terminal inputs, the response time was not an appropriate parameter. It was eventually decided to use a type of reaction time which we labelled "priority based reaction time". The event which is timed is the period from when a process wakes up until it is selected as the next process to run. A process sleeps or is suspended in UNIX when for instance it must wait for some I/O to finish. The reaction time is not measured for all processes however, but only for those whose priority is that of terminal input or lower (which includes terminal output). (UNIX has a complex way of updating the priority of a process depending on its past and current activities).

The choice was made after initial experiments showed that it is more sensitive to workload variation than other definitions of reaction time. (For example, the same definition but measured for all processes, or, the popular definition of the time from the input of a command until the CPU starts to act on that command).

C. Other parameters

The workload parameters used in the experiments were:

1. The number of processes (PN),
2. Mean process size (in units of 64-byte blocks) (PS),

3. File system requirements of a process, and,
4. CPU requirements of a process.

Items 3. and 4. are related and a single parameter PC (= mean CPU time between successive disk I/O's) is used to characterize them.

The system hardware parameters studied were:

1. Main memory capacity (in units of 64-byte blocks), and,
2. Disk configuration.

(Note that tapes were not considered as they were not involved in swapping and were not frequently used by the typical user). The internal system parameters (i.e., those quantities derived from the internal structure of the operating system) were:

1. Swap rate,
2. Swap wait,
3. Disk queue length,
4. Disk waits, and,
5. CPU intervals.

Swap wait includes the waiting time in the queue as well as the actual I/O times. It is included along with the swap rate since the swap rate does not differentiate between the size of

swaps, which for a slow disk, could have a substantial effect on responsiveness. The swap wait does capture this effect.

As with disk queue length, disk waits indicate the number of file system accesses. However, it is much more heavily influenced by the swap I/O (which is queued onto the same disk queue). This is because all file system I/O consists of a block of 512 bytes, whereas the size of swap I/O blocks often is several Kbytes. The swap rate, however is at least an order of magnitude less than the disk I/O rate in our system.

The CPU interval is the time between context switches (not to be confused with the time between I/O's).

These parameters are included for the possible insights they may give in interpreting the results of the experiment. There are basically two views that can be taken towards these quantities. One can view them as indicators of how well the system is handling the workload, that is, as a reaction of the system to the workload. In this project, the internal system parameters are viewed as variables that can be related to the performance parameter.

D. Workload used for the experiment

Since we need a controlled environment to establish the

relationships of the parameters mentioned above, a synthetic workload [2] was used. The composition of the prototype program is an infinite loop consisting of:

a) compute loops,
b) disk I/O operations,
c) sleep calls, (there is a system call in UNIX which allows a process to suspend itself for the number of seconds passed as a parameter), and,
d) terminal I/O's , (terminal outputs only, as terminal inputs were difficult to produce for lack of hardware and manpower).

Each parameter is driven by a different distribution which is adjustable. There is also a dummy array in the prototype program which can be statically varied to give us the desired program size.

The experiment performed was a factorial experiment [3] involing 83 twenty-minute runs. Some of the runs were used in validating the model. The length of the sessions was determined experimentally. It is assumed that the series of reaction times are ergodic, that is, the accuracy of the mean reaction time increases as the number of observations in the series is increased. On this basis a sample workload was run several times increasing the time length until the variation over several runs of the same time length was acceptably small. The 20-minute runs were found to produce differences of 5% or less

in the values of the performance parameter. (There are statistical methods to determine the session length, see for example Chapter 2 of [6]).

3. Results and interpretation

Perhaps the most striking result is the dominant factor of a quantity which we call the percentage of remaining main memory (denoted hereafter by PCTRMM) on reaction time.

PCTRMM = 100 * (swappable main memory capacity - total memory requirements of swappable active processes) /(swappable main memory capacity)

= (MM -PN * PS) / MM * 100

This quantity, which can be either positive or negative depending on whether all the processes can fit into main memory or not, can account for over 80% of the variation in reaction time.

Note that PCTRMM is a good indicator of the rate of swapping. A brief description of the UNIX (version six) swapping policy is included in the Appendix. It is apparent that as the number of processes that cannot be fitted into main storage increases (and hence the negative value of PCTRMM increases) swapping activity will intensify. The measured real

time intervals between successive system swaps plotted against -PCTRMM support this (Figure 1).

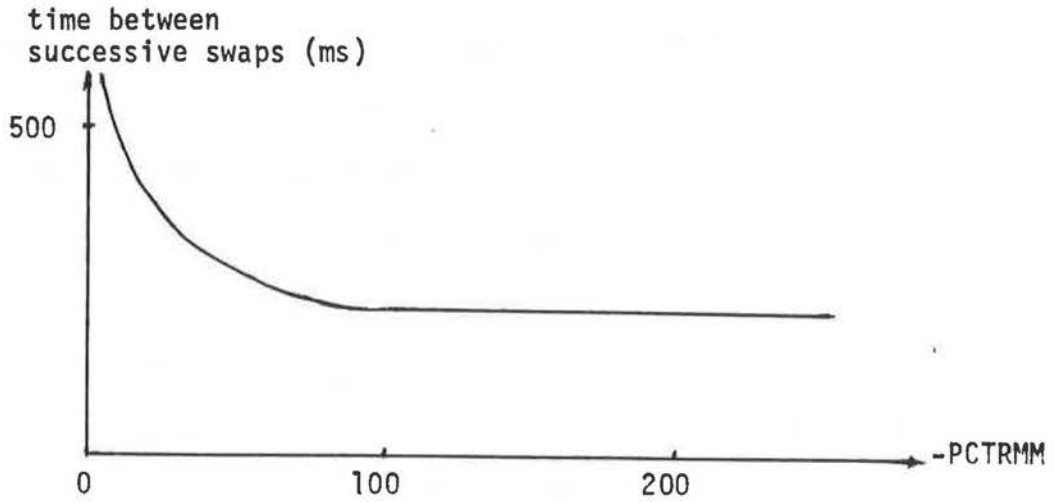

Figure 1. -PCTRMM vs real time between successive system swaps

Note the similarity between PCTRMM and the resident set size in a paging system. Both indicate the portion of the current information that is contained in main storage. However, the resident set size and the lifetime are usually defined for each process whereas PCTRMM is a per system quantity. This difference comes about because in our definition of a swapping system, the entire process must be in main storage before it can start execution. Thus the 'resident set' of a process is not under system control and it is meaningless to talk about per process behaviour with respect to different 'resident set' size. A direct consequence is that whereas lifetime is normally measured in terms of virtual time (i.e., CPU time), we measure the interval between successive system swaps in real (wall-clock) time.

3.1 Complexity of the PC factor.

The nature of the PC factor appears to be quite complicated. On an independent basis, an increase in the CPU requirements would tend to increase reaction time since, as in this experiment, for all CPU requests under one second, the CPU will not do a context switch. Thus any other process competing for the CPU will have to wait longer. With the file system access rate however, things are not so simple and cannot even be considered independent of the CPU requests in the context of this experiment. For in the prototype process, increasing the rate of file system accesses is always accompanied by a decrease in the CPU requirements between accesses.

Generally when a file system access occurs a context switch will be made while that process waits for its I/O to complete. Thus any process waiting for the CPU will have a greater chance of getting its request serviced and thus the reaction time will generally be less. This trend would start to reverse however if most of the processes were doing a large amount of disk I/O, since disk congestion would start to take its toll by increasing the time to complete any individual request. As well, there are other things to consider. If user processes are trying to access the disk then they will be interfering with the swapping if there is any. This will tend to increase the reaction time. Furthermore, processes which have just completed doing a file system access retain for a short period in this UNIX version,

the priority of a disk request (which is much higher than the normal user priority). In this case then, as the number of file system accesses increases, there is an increasing chance of such processes getting the CPU before a process, which because it is at a lower priority and thus will produce a reaction time, will obtain the CPU. This would tend to increase reaction time values. Regardless, there will generally be a longer wait, since the CPU must attend to disk I/O first, which will add slightly to overhead.

Finally, the swapping algorithm needs to be examined for its indirect contribution in this matter. It should be noted that when swapping must occur, the swapper initially looks for blocked processes to swap out. Hence whenever a process does I/O, it is quite likely that it will get swapped out before the I/O is completed. This probability increases with the number of processes and with shorter CPU requests. The result is that as the PC factor decreases, less and less work is getting done by a process before being swapped out. This will be reflected in significantly larger reaction times and is due to the simplistic nature of the swapping algorithm.

In summary, it is sufficient to say that a change in the value of the PC factor may have a significant and complex effect on the reaction time. Examining the data, it appears that the overriding effect is that as the PC value is increases, the mean reaction time drops, indicating the file system access rate has

the dominant influence in our system. This holds true for most, but not all, of the cases in this experiment. The complexity of the issue, though, helps explain the difficulty in developing a good regression equation.

A hypothesis can be made that in the general case the PC factor affects the size of the CPU and disk queues. For small PC values it is felt that the length of the disk queue on the average will be longer than that of the CPU queue but that this will reverse as PC increases. When the disk queue is substantially longer, the file access rate will increase the value of the mean reaction time. As PC increases, however, this influence will drop and eventually when the CPU queue becomes substantially larger than the disk queue, the mean reaction time will again be increased except that this time it will be due to the longer CPU waits. Clearly if this is the case, then it would be desirable to find the transition point, where neither influence has much effect.

A separate simulation experiment was carried out using a general purpose computer system simulator [4]. The mean reaction time was measured for different values of PC (the only variable in the experiment). It was observed that the mean reaction time was always close to the minimal value when the mean CPU and the disk queue wait times were approximately equal. Though this situation does not necessarily imply a "balanced" system [7], it is generally true that no serious bottleneck is

likely to occur either. Thus balancing the CPU and the disk loads in a swapping system appears to be desirable.

3.2 The regression equation.

A regression equation of reaction time for our system was obtained. The equation expresses reaction time as a function of only PCTRMM, PN, PS and PC. However the equation is not a very well-fitted one as its r**2 value (the square of the correlation coefficient) is only about 0.9. The equation that resulted was:

```
RT = 703 - 15 * PCTRMM + 2.3 * PC
     - .004 * PC *PN *PS
```

This seems to support the idea of the dual nature of the PC factor, with the positive contribution of the PC term and the negative contribution of the PC*PN*PS term. The latter quantity will have a more influential contribution when PN or PS is large, specifically when PN*PS > 575.

By removing the factor PC (i.e., keeping it constant), it is possible to obtain a regression equation with much higher r**2 value.

PC level	r**2 value	Equation
12	.96	RT = 588 - 16.9 * PCTRMM ..... (1)
186	.96	RT = 595 - 8.9 * PCTRMM
354	.91	RT = -241 -4.5 * PCTRMM + 204 * PN

From a 90-minute measurement of the production workload, the mean value of PC was found to be 12 ms. for conversational type workload.

Validation of the model consisted of comparing predicted performance to that obtained by measurement. The mean absolute error is about 20%. Thus its use lies mainly in the trends it demonstrates and the rough approximations it gives to predictions, rather than in an ability to give exact results.

4. Sample applications

1. Prediction of the effect of the size of main memory on performance.

   Note that because memory is a passive device, it is difficult to model it explicitly in the traditional queueing model.

2. Prediction of load increase on performance.

The following example will illustrate the above two applications.

Suppose there was a UNIX installation which had 700 blocks of main memory available for users and that they were concerned about their heavy demand periods when there are an average of eight user processes with an average size of 200 blocks each. To find their current reaction time we use equation 1:

$$RT = 588 - 16.9 * \frac{(700 - 6 * 200)}{700} * 100$$

This gives a reaction time of around 2.8 seconds which is, indeed a sluggish system.

(a) One situation might be that they had the option of buying 16K bytes (256 blocks) of main memory but were unsure of the effect it would have. Using equation 1, it could easily be determined that it would reduce reaction time to a little over 1.7 seconds. While it is a definite improvement, the responsiveness is still not that good.

(b) Suppose instead that they wanted to know how much more main memory they would need to reduce the reaction time to 1 second. To find this, let X represents the amount of additional memory needed, then it becomes a simple matter of solving the following equation:

$$1000 = 588 - 1690 * \frac{((700 + X) - 1600)}{(700 + X)}$$

This yields a value for X of 587 blocks for a little over 36K bytes.

(c) Let us assume that the installation was able to purchase the extra 587 blocks of memory and thus reduce their reaction time to under one second. According to the natural laws of increased capacity, let us presume that soon after the purchase, management wished to add more terminal lines into the system and wanted to know what effect it would have. If the computing centre staff could estimate on average how many extra processes it would introduce into the system, say for example three with the same average size of 200 blocks, then it can easily be calculated that the reaction time would increase to 1.8 seconds.

3. Load control.

Version six UNIX has no explicit load control mechanism. If an installation wanted to guarantee its users a certain response level, then a valuable and easy step would be to change the routine which logs users onto the system ("/etc/init") such that it first examines the current responsiveness of the system. If it exceeds a given threshold, then the user would be denied access.

Load control has been extensively studied (see for example [8-12]). However, with the exception of [8], most adaptive schemes (i.e., those that will adjust to changing workload

condition) are applicable only to paging systems. The static ones, on the other hand, are less efficient. The proposed method is adaptive to workload variation but has the advantage of being very simple and requires very little overhead.

4. The swapping lifetime function may be used in simulation or analytic models of swapping systems in much the same way the paging lifetime functions are being used.

5. Conclusions.

A regression model of the reaction time of a PDP 11/45 system running under UNIX (version six) was derived and sample applications given. It must be remembered, however, that the reaction time values calculated from the model are simply rough estimates. As well, they will generally form upper bounds since the synthetic workload did not take advantage of the UNIX concept of 'text' which would reduce the swapping load somewhat in real systems. From validation, it was seen that the model tends to overestimate the actual values.

Another important point to realize is that regression models based on measurement are generally not portable. They should be used only for the system from which data were gathered. This is because a regression model does not attempt to explain why the system behaves in a particular way. They

reflect all the pecularities of the given system as well as the usage pattern of the given user community. The advantage is that it is possible to model any system, however complex, by simple equations to obtain some quick but rough indications of performance. Furthermore, the modelling procedure is straightforward and a result (though not necessarily very accurate) is guaranteed. This may be useful in case the system cannot be adequately modelled by the traditional queueing model. It is also usually much less costly than simulation, particularly when the system under study is complex.

Acknowledgement

I would like to thank Rod Downing for useful discussions and programming help. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada under grant A3554.

References

[1] Chamberlin, D., Fuller, S., Liu, L., "An analysis of page allocation strategies for virtual memory systems", IBM Journal of Research and Development, Vol.17, No.5, 1973,404-412.

[2] Ferrari, D.,"Workload characterization and selection in computer performance measurement", Computer, Vol.5, No.4, 1972, 717-721.

[3] Tsao, R. and Margolin, R., "A multi-factor paging experiment - II. Statistical methodology", in: Freiberger, 1972, 135-158.

[4] Chanson, S. and Bishop, C., "A simulation study of adaptive scheduling policies in interactive computer systems", Proceedings of Winter Simulation Conference, Gaithersburg, Maryland, 1977, 634-641.

[5] Belady,L. and Kuehner, C., "Dynamic space sharing in computer system", Comm. of ACM, Vol.12, No.5, 1969, 282-288.

[6] Ferrari, D., Computer Systems Performance Evaluation, Prentice Hall, New Jersey, 1978.

[7] Buzen, J., "Analysis of system bottlenecks using a queueing network model", Proc., Workshop on System Performance Evaluation, Harvard University, April 1971, 82-103.

[8] Chanson, S. and Sinha, P., "Adaptive load control in batch-interactive computer systems", Proc. of 16th Computer Performance Users Group, Oct. 1980, 207-213.

[9] Badel, M., Gelenbe, E., Leroudier, J. and Potier, D., "Adaptive optimization of a time-sharing system's performance", IEEE Proc., Vol.63, 1975, 958-965.

[10] Badel, M. and Leroudier, J., "Adaptive multiprogramming systems can exist,", Performance of Computer Installations, D. Ferrari (ed.), North-Holland, 1978, 115-135.

[11] Denning, P., Kahn, K., Leroudier, J., Potier, D. and Suri, R., "Optimal multiprogramming", Acta Informatica, Vol.7, No.2, 1976, 197-216.

[12] Gelenbe, E. and Kurinckx, A., "Random injection control of multiprogramming in virtual memory", IEEE Trans. on Software Engineering, Vol.4, No.1, 1978, 2-17.

[13] Denning, P. and Kahn , K., "A study of program locality and lifetime functions", Proc. Fifth SIGOPS, Nov. 1975, 207-216.

APPENDIX. Swapping Policy in UNIX (Version Six)

1. Routine and variables used:

sched - routine called to swap in all processes that it can from disk.

runout - a global flag which is set and slept on by sched when there are no more READY processes out on disk. Thus other routines can test runout and if appropriate, wakeup the runin - a global flag which is set and slept on by sched when it was unable to swap in all the READY processes. As well as being accessible to other routines, runin is tested every second by the clock routine.

2. Swapping-in policy.

i) when: sched is only called when a READY process is out on disk, and thus wants into main memory. This can occur due to two situations:

   1) In the previous execution of sched, it was unable to load all the READY process out on disk. Thus it set and slept on runin. In this case, sched will be awakened every second by the clock routine until there are no more READY processes left on disk.

2) In the previous execution of sched, it did load all the READY processes into core (and thus set and slept on runout). Later, a process (which was not READY, e.g.,sleeping due to I/O wait) out on disk became READY (i.e., was awakened). In this case, sched will be executed whenever such a situation arises.

ii) who: The policy is based solely on the length of time a process has been out on disk. Sched starts with the READY process out the longest and tries to load all of them into core. If it fills all of main memory and there are still READY processes on disk, it will still try and load them in as long as (1) there are processes in core which are not READY (e.g., sleeping on low priority), - they will be successively swapped out to make room; or failing that, (2) the READY process(es) on disk has been there for more than 2 seconds and there is an in-core process (which is READY or sleeping on high priority) which has been in core for more than 1 second.

3. Swapping out policy.

i) when: only when necessary as determined by the swap-in algorithm, i.e., not all the READY processes out on disk will fit into core, etc..

ii) who: any process sleeping on low priority or being traced, and failing that, if the process on disk has been out there for more than 2 seconds, then any process (READY or sleeping

on high priority) that has been in-core for more than 1 second (starting with the one who's been in the longest, i.e., based solely on elapsed time in core).