HOW TO EVALUATE PAGE REPLACEMENT ALGORITHMS

Richard Y. Kain
University of Minnesota

The designer of a virtual memory system can obtain accurate estimates of the average memory requirements of programs running in the system by weighting the average allocation during execution intervals with the average allocation during page waiting intervals. We show how to combine the averages, how to use the measure to determine the size of primary memory while achieving system balance between memory and processor demands, and how to partially order the performance of paging algorithms.

Key Words and Phrases: Memory Management, Operating Systems, Paging, Performance Evaluation

CR Categories: 4.3, 4.32

Introduction

Systems designers introduce virtual memory to decrease software costs and to reduce the main memory requirements [6,8]. Previous analysts have considered a process's memory requirement as the (virtual-time) average of the space used during the execution intervals of that process. They trade this space requirement against the corresponding page fault frequency. The virtual-time average does not match the real-time average when the allocation varies dynamically. The real-time average is the average over the entire time that the process is resident in the memory. The designer can use the real-time average to rank page replacement algorithms and to determine the size of main memory such that the processor and memory demands will be matched.

Other investigators [9,10] have used queueing networks for performance studies. Queueing models require restrictions on the service time distributions (at least they must have rational Laplace transforms); these require approximations to the actual situation. Models with different classes of customers[3] should be better approximations, but to exactly model the paging environment many states would be needed, and they would have to be aggregated before computing the solutions. The analyst would have to use simulations to verify the solutions.

Time Requirements

A process in a paging system cannot use one contiguous block of processor time because the process requires input/output and page swapping activity. For this analysis we ignore input/output activity.

Conventional paging algorithm performance analyses have determined the page fault frequency f: the fractional number of memory references that name a page not currently stored in primary memory. If there is one memory reference per time unit, the execution of a process will cause fT page faults, where T is the total execution time of the process.

Let R be the (assumed constant) time required to handle a page fault, including swapping the new page. Then the total system residence time* is T(1 + fR); during that interval the process will have utilized the processor for time T, so its fractional use of the processor's capacity is

$$\rho = \frac{1}{1+fR} \qquad (1)$$

Note that ρ depends on many factors, including the memory management algorithm since that determines f. Note also that scheduling waits may cause the process to remain longer in the system, using a smaller fraction of the processor's capacity. Because analysts may approximate the latter effect by increasing the value of R, the following developments ignore scheduling waits.

Space Requirements

Some paging algorithms (e.g., fixed-partition LRU) assign "fixed" partitions of memory to each process when it is created. If the partition size** is m, the process uses m pages during its entire system residence time, and therefore its space requirement is m.

---

* The real time interval during which memory space is allocated to the process.

** All space measures are in units of page frames.

Most paging algorithms (e.g., working set) allow dynamic allocations; the instantaneous memory allocation to each process is determined by complex interactions among many factors including the referencing pattern and the management algorithm. In most, but not all, management algorithms the allocation changes only at page fault times.

Therefore, each interval of constant allocation begins (Fig. 1) with a page fetch of length R followed by a wait for CPU time (assumed to be zero), followed by an execution interval terminating in a page fault, a scheduler interrupt, or job completion (since we ignored input/output activity). Since we assume that the space allocation changes only at page faults, scheduler interrupts do not affect the space allocation for this process.

Conventional analyses of paging algorithms determine the average allocation in virtual time. These virtual-time averages are not the same as the real-time averages because they do not include the average allocation during page waits, which requires considering the correlation between the size of the memory allocation and the length of the execution interval before the page fault.

To analyze this effect, consider a single process in the system. Let p(i) be the fractional amount of virtual time that the memory allocation for that process is i pages, and let h(i) be the fraction of references that cause page faults among all reference attempts while the process has been allocated i pages. The (virtual-time) average allocation during execution intervals is

$$\bar{m}_e = \Sigma_i ip(i) \tag{2}$$

To compute the allocation averaged over all page fault waiting intervals, count the number of page faults and the numbers of times that the allocation is i when the fault occurs. The total number of page faults with allocation i is Tp(i)h(i) since h(i) is conditioned upon the allocation being i, and Tp(i) is the number of reference attempts while the allocation is i. Summing over all i, we obtain the total number of faults $F = \sum_i Tp(i)h(i)$. Now the cumulative allocation at page faults times is $T\Sigma ip(i)h(i)$, assuming that the allocation during fault processing is the same as the allocation during the next execution interval. By associating the waiting period preceding each execution interval with the fault at the end of the interval, we obtain the expression for the average allocation at page fault times:

$$\bar{m}_f = \frac{\sum_i ip(i)h(i)}{\sum_i p(i)h(i)} \tag{3}$$

The allocation averaged in real time is

$$\bar{m} = \frac{\bar{m}_e + fR\bar{m}_f}{1 + fR} \tag{4}$$

Conventional performance analyses derive $\bar{m}_e$, but $\bar{m}$ is the measure of the memory requirement for each process. The error depends on the speed ratio R and the difference between $\bar{m}_e$ and $\bar{m}_f$. Whenever the page fault frequency is a monotonically decreasing function of the memory allocation (which is true for all stack algorithms [6]), page faults tend to occur more frequently when the allocation is smaller. This observation is the basis for a formal proof (omitted here) that $\bar{m}_f < \bar{m}_e$. Since $\bar{m}$ is a weighted average of $\bar{m}_f$ and $\bar{m}_e$, then $\bar{m} < \bar{m}_e$. An approximation to this effect is given by the following argument.

Assume an exponential behavior for the page fault frequency as a function of the memory allocation:

$$h(i) = \lambda e^{-\lambda i} \tag{5}$$

Assume the distribution of p(i) to be the weighted sum of normal distributions [4]:

$$p(i) = \frac{1}{\sqrt{2\pi}} \sum_{j=1}^{J} \frac{p_j}{\sigma_j} e^{-\frac{(i-m_j)^2}{2\sigma_j^2}}, \text{ with } \sum_{j=1}^{J} p_j = 1 \tag{6}$$

Now approximate the summations in (3) by integrals:

$$\bar{m}_f = \frac{\sum_{j=1}^{J} \frac{p_j\lambda}{\sigma_j\sqrt{2\pi}} \int_{-\infty}^{\infty} xe^{-\lambda x} e^{-\frac{(x-m_j)^2}{2\sigma_j^2}} dx}{\sum_{j=1}^{J} \frac{p_j\lambda}{\sigma_j\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-\lambda x} e^{-\frac{(x-m_j)^2}{2\sigma_j^2}} dx} \tag{7}$$

where we have taken liberties with the lower limits to obtain approximate solutions*. Integrating, we have, for J = 1,

$$\bar{m}_f = \bar{m}_e - \lambda\sigma_e^2 \tag{8}$$

Table 1 show this approximation compared with the actual behavior of the working set algorithm [1,2] and with the modeled behavior of the page fault frequency algorithm [11]. Columns 4 and 5 show

* For realistic parameters, $m_j > 2\sigma_j$, and the areas under the negative regions will not be significant.

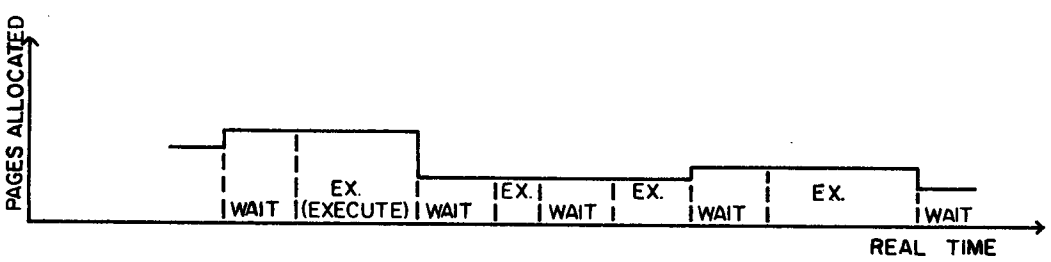

Figure 1: Page and Processor Use History

Program	Algorithm		$\bar{m}_e$	$\bar{m}_f$	$\bar{m}_f$ from (8)	R = 3000		R = 10000	
	Type*	Parameter				ρ	d	ρ	d
	A	10	2.53	2.26	1.93	.0039	.580	.0012	1.928
	A	50	3.82	2.69	2.58	.0057	.472	.0017	1.566
	A	100	4.91	3.18	3.14	.0078	.408	.0024	1.348
	A	300	7.07	4.84	3.74	.0159	.306	.0048	1.005
1	A	500	7.99	5.01	2.74	.0174	.291	.0053	.952
	B	10	5.46	4.86	5.03	.0148	.329	.0045	1.085
	B	50	7.28	6.88	6.72	.0272	.253	.0083	.827
	B	100	8.76	8.22	8.20	.0354	.233	.0109	.755
	B	500	15.61	14.28	14.12	.0877	.164	.0280	.510
	A	10	2.48	1.93	1.99	.0034	.570	.0010	1.893
	A	50	3.46	2.32	2.22	.0067	.349	.0020	1.154
	A	100	4.00	2.57	1.94	.0088	.293	.0026	.968
	A	300	5.09	3.00	1.37	.0130	.232	.0040	.761
2	A	500	5.71	3.02	0.68	.0142	.216	.0043	.705
	B	10	5.63	5.03	5.04	.0349	.145	.0107	.470
	B	50	6.88	6.24	6.27	.0593	.106	.0185	.337
	B	100	7.75	7.09	7.11	.0819	.087	.0261	.273
	B	500	10.68	9.98	9.88	.1880	.054	.0649	.154

*A = Working Set (actual behavior) - Parameter = window size
B = Page Fault Frequency (modeled behavior) - Parameter = width of critical interval for page deletion.

Table 1: Representative Values for Demands with M = 1000

that the unimodal approximation is quite good in most cases.

For J > 1, the result can be expressed in closed form:

$$\bar{m} = \frac{\sum_{j=1}^{J} \frac{p_j}{\sigma_j}(m_j - \lambda\sigma_j^2)e^{m_j\lambda - \lambda^2\sigma_j^2}}{\sum_{j=1}^{J} \frac{p_j}{\sigma_j} e^{m_j\lambda - \lambda^2\sigma_j^2}} \quad (9)$$

Bryant's results [4] suggest that a bimodal approximation will be more exact for certain programs; that approximation has not been tested against actual data.

Balancing the System

Let M be the total memory size. Then the process requires the memory fraction

$$\mu = \frac{\bar{m}}{M} \quad (10)$$

It uses the processor fraction

$$\rho = \frac{1}{1+f\bar{R}} \quad (11)$$

Under system balance the addition of either processor capacity or memory capacity will not increase the system throughout, except by second-order effects due to statistical variations. For the system to be balanced [5], we desire

$$\mu = \rho \quad (12)$$

or

$$M = \bar{m}_e + f\bar{R}\bar{m}_f \quad (13)$$

If all processes in the system are statistically identical, the system will be balanced when the degree of multiprogramming D is [12]*.

$$D = \frac{1}{\mu} = \frac{1}{\rho} = 1 + f\bar{R} \quad (14)$$

When the processes are not statistically identical, we must modify the previous argument, as follows.

Let ρ_j and μ_j be the fractional processor demand and the fractional memory demand by the j^{th} process in a set of k processes. Define the demand ratio d_j for process j to be

$$d_j = \frac{\mu_j}{\rho_j}, \quad 1 \leq j \leq k \quad (15)$$

Selecting an Algorithm

Real job mixes contain processes with differing demand ratios. Furthermore, these ratios depend upon the memory management algorithm in use. The system designer can select a memory management algorithm based on the distributions of demand ratios it produces from a given distribution of program types. To select an algorithm, plot the histogram of the demand ratios produced by the distribution of processes running under

* This argument ignores queueing effects, because it considers only the average behavior. See [9,10] for detailed discussions of related queueing effects.

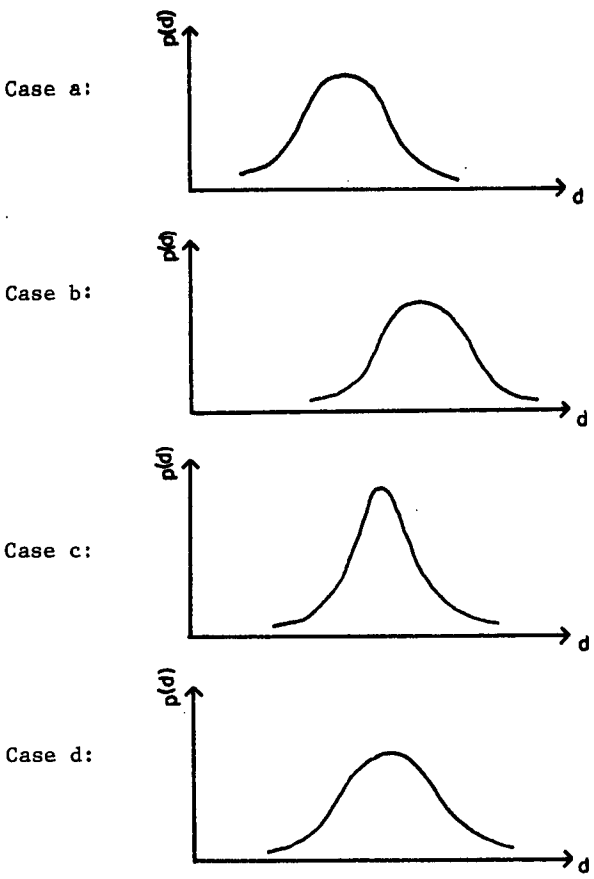

Figure 2: Representative Demand Histograms

a potential management algorithm. Some possible demand histogram shapes are shown in Figure 2. Let $\overline{d(j)}$ and $\sigma_d^2(j)$ be the mean and variance of the demand histogram for management algorithm j.

Now, impose a partial ordering on the management algorithms as follows: Let $i \geq j$ denote that algorithm i is better than algorithm j. Then

$$i \geq j \text{ if a) } \overline{d(i)} \leq \overline{d(j)} \text{ and } \sigma_d^2(i) = \sigma_d^2(j)$$
$$\text{or b) } \overline{d(i)} = \overline{d(j)} \text{ and } \sigma_d^2(i) \leq \sigma_d^2(j)$$
$$\text{or c) } \overline{d(i)} \leq \overline{d(j)} \text{ and } \sigma_d^2(i) \leq \sigma_d^2(j)$$

In case a) the two algorithms have the same variance but one is better because its mean is lower. Thus the algorithm producing the histogram in Fig. 2a is better than the algorithm producing the histogram in Fig. 2b. In case b) the means are the same but one algorithm gives a lower variance, which is more desirable because it will be less likely that the system is badly unbalanced due to the instantaneous job mix. Thus the algorithm producing the histogram in Fig. 2c is better than the algorithm producing the histogram in Fig. 2d. In case c) one algorithm is clearly better than the other on both criteria.

The major advantage of the partial ordering proposed is that the paging algorithm selection is based on the interactions between individual processes and the management algorithm; interactions among processes need not be considered. Unfortunately these interactions must be considered (by queueing approximations [9,10], for example) when the partial ordering criteria do not select an algorithm.

Given an algorithm (or set of algorithm parameters), the designer can determine the required paging memory size either from the average demand $\overline{d}$ or from that demand which covers a specified percentage of the processes in the labe mix. (See [7] for a discussion of balancing considering the detailed demand distributions). For example, if the histogram of d were approximately normal, then the choice $M' = (\overline{d} + \sigma_d)M$ would allow most mixes to execute without encountering a memory space bottleneck too often.

Summary

Conventional measures of paging algorithm performance do not account for the actual demands on the processor and memory. The demand ratio not only reflects the actual demands but also permits system balancing. The designer can select a memory management algorithm on the basis of the demand distribution it produces in conjunction with the assumed job mix statistics. A detailed analysis of trace tapes is needed to determine typical demand distributions.

Acknowledgments

This work was supported in part by the Office of Computing Activities of the National Science Foundation through grant GJ-32504. I wish to acknowledge helpful discussions with Arvind, D. Grit, and M. Riad in connection with our paging studies, and with E. Sadeh particularly in reference to the content of this paper. P. J. Denning and the referees provided comments that improved the presentation.

References

1. Arvind, "Experimental Data from Four Trace Tapes from CDC 6600", Tech. Report. NSF-OCA-GJ-32504-4, Elec. Eng. Dept., Univ. of Minn., Minneapolis, Jan. 1974.
2. Arvind, "Models for the Comparison of Memory Management Algorithms", Ph.D. Thesis, Univ. of Minn., October 1973.
3. Baskett, F., et al, "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers", JACM 22, 2, 248-260, April, 1975.
4. Bryant, P., "Predicting Working Set Sizes", IBM Jour. R and D 19, 3, 221-229, May 1975.
5. Buzen, J. P., "Optimizing the Degree of Multiprogramming in Demand Paging Systems", Proc. IEEE-CS Conf. IEEE Publ. no. 71-C41-C, pp. 139-140, Sept. 1971.
6. Coffman, E. G., Jr., and Denning, P. J., Operating Systems Theory, Prentice-Hall, 1973.
7. Denning, P. J., "Equipment Configuration in Balanced Computer Systems", IEEE Trans. on Computers C-18, 11, 1008-1012, Nov. 1969.
8. Denning, P. J., "Virtual Memory", Computing Surveys 2, 3, 153-190, Sept. 1970.
9. Denning, P. J., and Graham, G. S., "Multiprogrammed Memory Management", Proc. IEEE 63, 6, 924-939, June 1975.
10. Muntz, R. R., "Analytic Modeling of Interactive Systems", Proc. IEEE 63, 946-953, June 1975.

11. Sadeh, E., "An Analysis of the Page Fault Frequency Algorithm", Ph.D. Thesis, Univ. of Minn., April 1975.
12. Sager, G. R., "Symbiotic Scheduling", Proc. Second Texas Conf. on Computer Systems, pp. 9-1 - 9-6, Nov. 1973.