Because the ORing operation introduces ambiguity into the search process, the optimal tree for this implementation may differ from that which is optimal in the ideal case.

A useful algorithm, primarily a modification of the routine for designing an ideal tree, has been developed for organizing trees based on ORed records. Experimental retrieval performance on small static files (hundreds of records and queries) has been encouraging when the same queries are used for both design and test of the tree. A more realistic evaluation has not yet been undertaken.

Optimizing the tree structure on the basis of actual queries may be a rash procedure in applications where the frequency distribution of fields and values interrogated tends to fluctuate rapidly and markedly. An alternative design approach for such problems is to assume a uniform probability distribution of queries. The method can be economically implemented within the framework described in this paper. The number of queries satisfied by a given a.v. need only be calculated analytically rather than obtained by testing samples. A similar expedient is to employ a past sample of queries only to estimate the parameters of an assumed probability distribution, and to optimize the retrieval tree with respect to this distribution. Such an approach is simpler than the direct one when there are many queries in the design set. It can also be effected so as to reduce the risk due to radical changes in query characteristics.

The time-varying behavior of this system is not known. Its performance under fluctuations in query characteristics and an appreciable update load has yet to be evaluated.

The tree search described may possibly be a forerunner of retrieval systems whose organization is closely tuned to actual operating experience. Since design is based on sample queries and records, the retrieval tree is potentially adaptive, like a feedback control system, continuously reorganizing itself without manual intervention in order to adjust to changes in query and data distributions.

Received September 1972; revised January 1973

References

1. Sussenguth, E.H. Jr. Use of tree structures for processing files. *Comm. ACM 6,* 5 (May 1963), 272–279.
2. Scidmore, A.K., and Weinberg, B.L. Storage and search properties of a tree-organized memory system. *Comm. ACM 6,* 1 (Jan. 1963), 28–31.
3. Arora, S.R., and Dent, W.T. Randomized binary search technique. *Comm. ACM 12,* 2 (Feb. 1969), 77–80.
4. Patt, Y.N. Variable length tree structures having minimum average search time. *Comm. ACM 12,* 2 (Feb. 1969), 72–76.
5. Frazer, W.D. A proposed system for multiple descriptor data retrieval. In *Some Problems in Information Science*, M. Kochen (Ed.) Scarecrow Press, New York, 1965, pp. 187–205.
6. Casey, R., and Nagy, G. An autonomous reading machine. *IEEE Trans. Comp. 17,* 5 (May 1968), 492–503.

Operating Systems C. Weissman Editor

Empirical Working Set Behavior

Juan Rodriguez-Rosell
The Royal Institute of Technology
Stockholm

The working set model for program behavior has been proposed in recent years as a basis for the design of scheduling and paging algorithms. Although the words "working set" are now commonly encountered in the literature dealing with resource allocation, there is a dearth of published data on program working set behavior. It is the purpose of this paper to present empirical data from actual program measurements, in the hope that workers in the field might find experimental evidence upon which to substantiate and base theoretical work.

Key Words and Phrases: virtual memory, paging, working set, software measurement, program behavior

CR Categories: 4.3

This work has been supported by a research grant from IBM Svenska A.B., and a scholarship from the French government. Author's address: Division of Applied Mathematics, Brown University, Providence, RI 02912.

Definitions

The working set $W(t,T)$ of a process is the collection of information referenced by the process in the time span $(t - T,t)$. The domain of definition of t and T is time, expressed in some adequate unit, and since we are interested in paging machines, we will take the domain of definition of W to be the set of virtual pages allowed to be referenced. Another variable, the working set size, $w(t,T)$, is simply the number of pages in the working set. Its expected value, $E_t(w(t, T))$, is denoted by $\bar{w}(T)$. As the process executes, it will reference pages not present in the working set. The reentry rate, $l(t,T)$, is the number of pages reentering the working set per unit time. It is defined as

$$l(t,T) = \frac{| W(t+T,T) - W(t,T) |}{T}.$$

The notation means that if A and B are sets, $| A |$ is the number of elements in the set A, and $A - B$ is a set composed of elements x such that x is an element of A and x is not an element of B. The expected value of the reentry rate can be written as $\bar{l}(T)$, a function of T alone. It is proved in [1] that

$$\bar{l}(T) = \frac{\mathbf{d}\, \bar{w}(T)}{\mathbf{d}\, T}.$$

The distribution of working set sizes is an interesting quantity to consider. Let $[G(s,T) =$ Probability $(w(t, T) \leq s)$, and $[g(s,T) =$ Probability $(w(t,T) = s)$.

A complete discussion of the working set model for program behavior is found in [1].

Design of the Experiment

To investigate program working set behavior, a wholly interpretive simulator designed to monitor any IBM System/360 program was used. A set of routines gathers information on which pages a program references at every instruction. The pages are then annotated in a Boolean matrix kept in core. When the program requests an I/O operation, the pages it references are also noted down by a routine that examines the channel program. All data reduction is done after the monitor relinquishes control. To calculate $W(t,T)$ it is sufficient to know $W(t,1)$, since

$$W(t,T) = W(t-1,T-1) \cup W(t,1), \qquad \text{for } T > 1.$$

The data are later presented on an IBM 2250 Graphic Display Unit, and a conversational system permits the choice of parameters and of functions to be displayed. The data thus presented can be sent upon request to an off-line plotter.

To run the experiments, the monitoring system is loaded in a virtual machine running under control of the Cambridge Monitor System, CMS [2]. The results correspond to the monitoring of widely used system components, like assemblers and compilers, as well as user programs. All programs were run using the operating system CMS, which was itself also monitored and worked under the illusion of having 256K 8-bit bytes of memory available in the virtual machine. This address space was divided in pages of 1K, 2K, 4K, and 8K bytes, giving respectively 256, 128, 64, and 32 pages. This structure permits observation of the influence of page size upon the working set of programs.

It must be pointed out that any program running under CMS can be monitored. We do not attempt to discuss "typical" user behavior in this paper. Our aim is to show the type of data that can be obtained and some of the conclusions that can be drawn about program behavior. The programs available to us under the CMS system have not been written with a paging mechanism in mind. On the other hand, care has been taken when developing CMS to reorganize the system nucleus to produce fewer page exceptions.

Presentation of Results

It would be impossible to present all the data and all the interesting combinations of parameters. More data are found in [10]. Data are presented for the University of Waterloo's G-level assembler, as modified by Grenoble University to run under CMS, which assembled a program about 300 instructions long. It was the same assembler program used in [3]. All data are for a page size of 4096 8-bit bytes, the page size actually employed by the IBM System 360/67, unless otherwise indicated.

Figure 1 gives $w(t,T_0)$, with T_0 fixed at 5000 instructions, for the assembler. The assembler was written without a paging mechanism in mind, and it does overlaying. Some of the peaks (viz. those at 25000, 1100000, 1300000, 1550000, and 2000000 instructions) as are caused by this overlaying. The other peaks are probably caused by changes in the working set the assembler enters and exits loops. As T increases, the peaks remain, becoming wider in the graph. The near-equal-amplitude oscillation about a mean value (i.e. from 250000 instructions to 1000000 instructions in the case of the assembler) disappears, leaving only the maximum value. When the page size is decreased, the relative importance of the peaks decreases.

Figures 2 and 3 are, respectively, the assembler's average working set size as a function of T, together with the standard deviation, and the average reentry rate. It can be seen that the standard deviation reaches a maximum, as predicted by the theory [1]. The coefficient of variation is rather small in all cases, decreasing as T increases. After $T = 500000$ instructions, both the standard deviation and the reentry rate are very small. Computing working set sizes at intervals of this order of magnitude allows a process to accumulate about 75 percent of the pages it will need, and the paging rate will be extremely low. The working set size will essentially be determined by the heights of the peaks.

Figures 4 and 5 detail the region between 5000 and 50000 instructions, judged to be most interesting for scheduler design [3]. In this range, the standard deviation is nearly constant even though there are sizable differences in working set sizes. The reentry rate curves are invariably concave upward.

Figure 6 presents the frequency function and the cumulative distribution function for the working set size of the assembler. It seems that programs have definite preferences for some working set sizes. There has been some debate on whether the distributions should be normal [5, 6]. Actually the question that arises is, in which range of parameters are normal distributions good approximations to the distribution functions of working set sizes? How good an approximation is depends, of course, on the criterion used in judging. Among the salient characteristics of normal distributions is symmetry about the mean value, where the only maximum of the function occurs. It is desirable that the function to be approximated share as many characteristics as possible with the normal distribution function. Now choose a value for T, say T_0, and let

$$S_{\max} = \max_t w(t, T_0), \qquad S_{\min} = \min_t w(t, T_0),$$

$$S_0 = \frac{S_{\max} + S_{\min}}{2},$$

so that the frequency function of the working set sizes is

$$0 \leq g(s, T_0) \leq 1, \qquad \text{for } S_{\min} \leq s \leq S_{\max},$$
$$g(s, T_0) = 0, \qquad \text{otherwise.}$$

For the normal to be a good approximation to $g(s, T_0)$, it is desirable that $\bar{w}(T_0) = S_0$, and that $g(s, T_0)$ be symmetrical about S_0 exhibiting only one maximum located at S_0. Now consider the IBM System/360 [9], and fix $T_0 = 1$ instruction. Every instruction in this system can reference from a minimum of one page, as in the case of register-to-register instructions, to a maximum of eight pages, in the case of an execute instruction distributed over two pages whose subject instruction is also distributed over two pages and references two main memory arguments, both of which must equally be distributed over two different pages. To anyone familiar with IBM System/360 programs, it is clear that the number of register-to-register instructions far exceeds the number of execute instructions, particularly those so taxing that their execution requires eight pages in memory. In fact, it is probable that the majority of instructions refers only to one page. An analogous argument holds at the other end of the spectrum of T values. Thus we find that the frequency function will lack the desired symmetry, even if the referenced pages are mutually

Fig. 1.

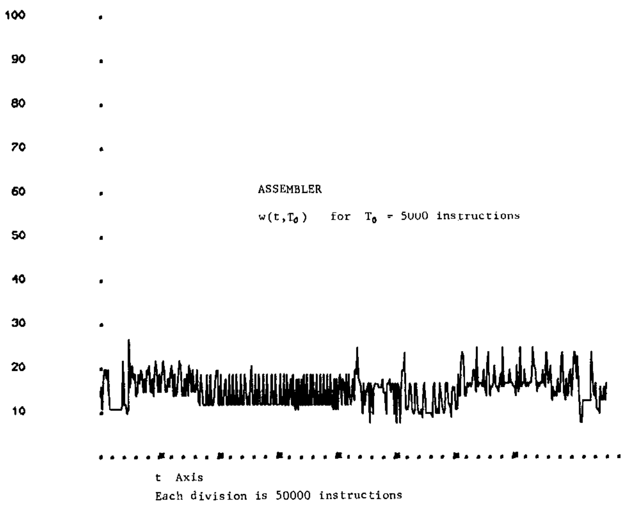


Fig. 2.

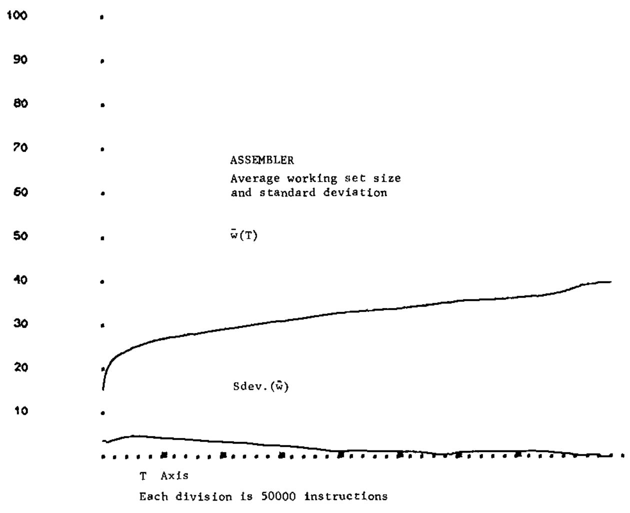


Fig. 3.

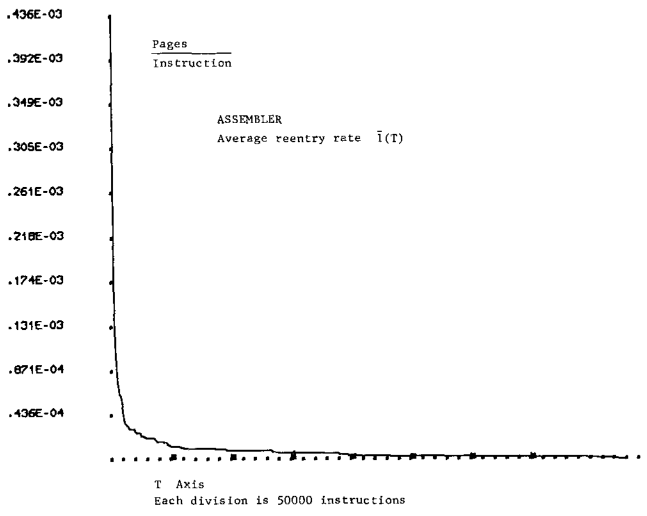


Fig. 4.

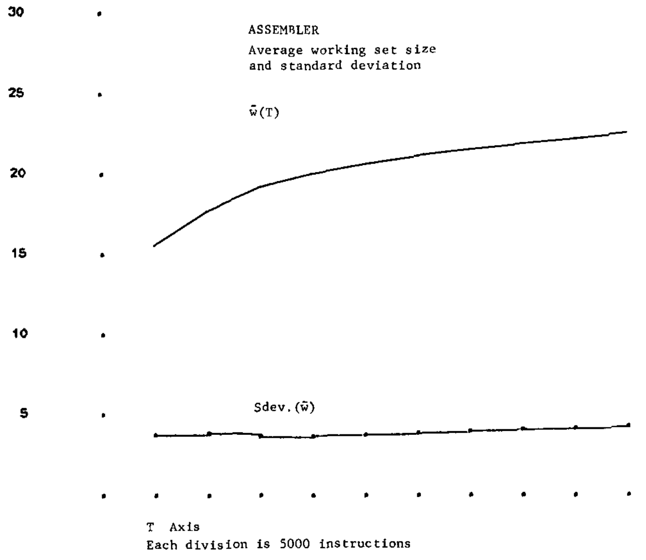


Fig. 5.

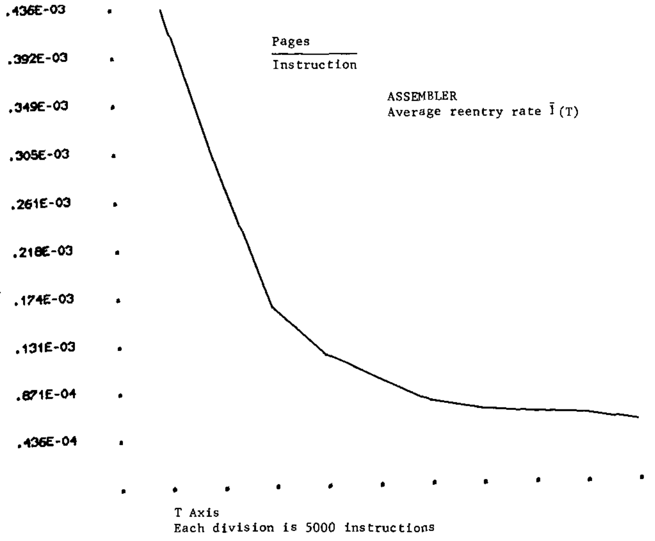
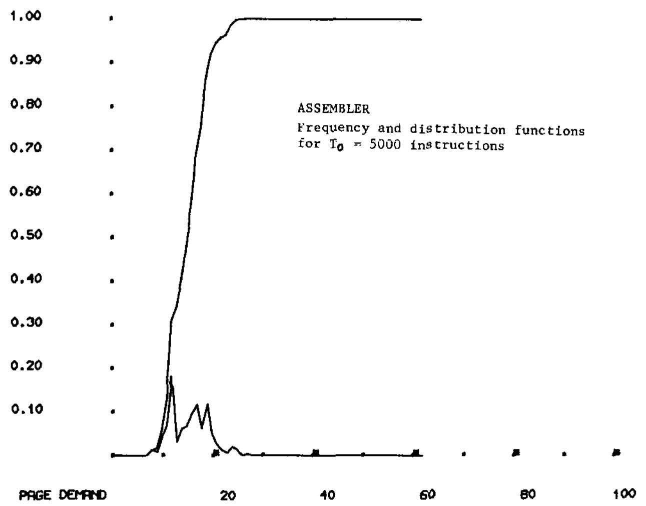


Fig. 6.



independent. Asymptotic uncorrelation does not seem to be a sufficient condition to insure nearly normal distribution functions.

These arguments correspond to T values that are outside of the range of interest for system design, but it is not clear in which range normal approximations are applicable. Figure 6 evidences the same lack of symmetry, and the mean value differs from the most probable value, in this case, for T values that can be considered acceptable for system design.

Figures 7 and 8 show the variation in working set size and in reentry rate as a function of T, using the page size as parameter, for the assembler. It can be seen that halving the page size less than doubles the average working set size. More memory will thus be available for multiprogramming, the drawback being the increased page fault rate. For example, decreasing the page size from 4K to 1K roughly quadruples the page fault rate. This could mean that paging channels which are adequate to handle the page traffic resulting from 4K page sizes might risk saturation.

It is rightly pointed out in [7] that the optimal page size is primarily influenced by fragmentation and efficiency of page transport operations. It is suggested that, from the point of view of fragmentation, the optimal page size should be close to 50 words. That this optimal page size depends largely on the operating system can be seen easily. Consider CP-67 [8]. In this operating system every virtual machine can be considered as a process, and the whole page table of the virtual machine is always present in core. Furthermore, in addition to the information contained in the page table and intended for the dynamic address translation mechanism, it is necessary to have information intended for the operating system. An extension to the page table is needed. A total of 10 bytes, constantly present in main memory, is needed for each page of every virtual machine that is logged-into the system. It is easy to calculate, based on the formula given in [7], that from the point of view of fragmentation and for CP-67, the optimal page size ranges from $512\sqrt{20}$ bytes for a 256K bytes virtual machine, to $512\sqrt{80}$ bytes for a 1024K bytes virtual machine. This is obviously a chicken-versus-egg question, in the sense that one may wonder what changes in operating system design would have been made had other page sizes been available. The fact remains, however, that careful consideration must be given to the operating system's structure before attempting to calculate optimal system parameters. Large page sizes are not only due to considerations about efficiency of page transport operations. It is not clear whether for all operating systems there is a great discrepancy between the page size for maximum storage utilization and the page size for maximizing page transport efficiency. Also, evidence in [4] seems to indicate that better restructuring of programs is achieved when large page sizes are employed.

Data as shown in Figures 7 and 8 may serve as

Fig. 7.

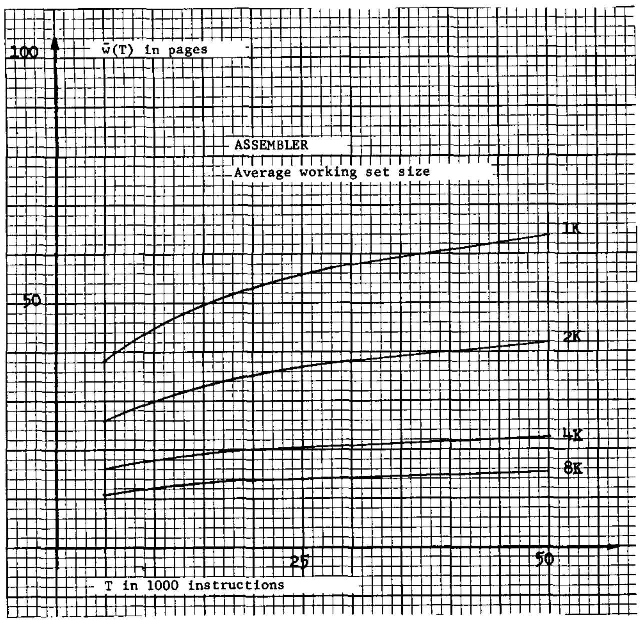
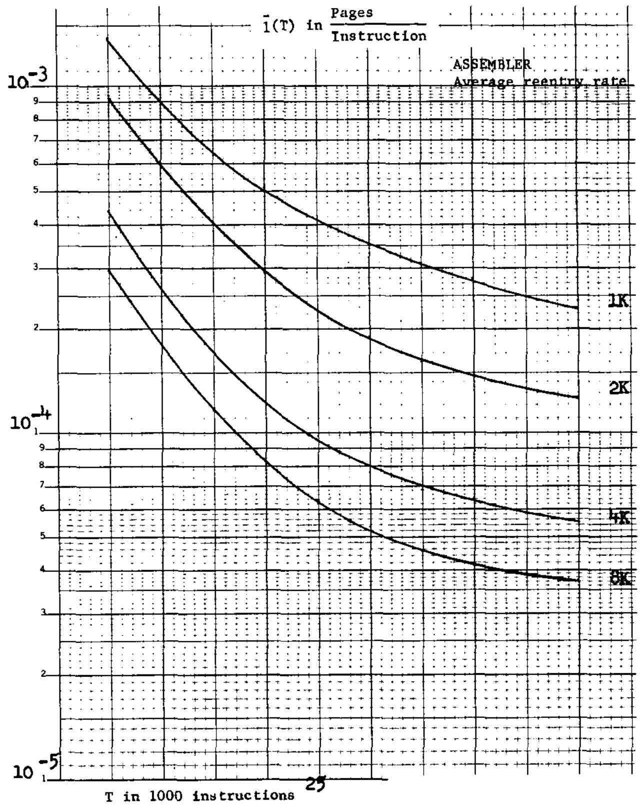


Fig. 8.



guidelines when trying to determine optimal page size from the point of view of page transport efficiency. However, many factors come into play when considering this problem. The multiprogramming level of the system affects cpu utilization. Both the choice of page size and the choice of T affect the paging rate. Care must be taken to insure that paging channels do not become bottlenecks. The hardware characteristics of the devices involved and the system's configuration must also be taken into consideration.

Conclusion

It has been the object of this paper to present data about program working set behavior. It is hoped that experimental computer scientists will provide similar measurements to compare and ascertain patterns in program behavior.

Acknowledgments: These experiments have been conducted at the computing center of the Institute of Applied Mathematics of the University of Grenoble. M. Peltier of IBM France Scientific Center has constantly encouraged us in our work, and Professor B. Arden, has been kind enough to revise the report. Thanks are also due to the referees and to the editor for their perceptive criticism.

Received February 1972, revised November 1972

References

1. Denning, P.J. Resource allocation in multiprocess computer systems. Tech. Rep. MAC-TR-50, MIT MAC, Cambridge, Mass., 1968.
2. CP-67/CMS User's guide. Form GH20-0859-0. IBM Corp. Tech. Pub. Dep.
3. Rodriguez-Rosell, J. Experimental data on how program behavior affects the choice of scheduler parameters. Third ACM Symposium on Operating Systems Principles, Stanford U., 1971.
4. Hatfield, D.J., and Gerald, J. "Program restructuring for virtual memory." *IBM Syst. J. 10*, 3 (1971).
5. Coffman, E.G., and Ryan, T.A. A study of storage partitioning using a mathematical model of locality. *Comm. ACM 15*, 3 (Mar. 1972), 185–190.
6. Denning, P.J., and Schwartz, S.C. Properties of the working set model. *Comm. ACM 15*, 3 (Mar. 1972), 191–198.
7. Denning, P.J. Virtual memory. *Computing Surveys, 2*, 3, 1970.
8. CP program logic manual. Form GY20-0590-0. IBM Corp. Tech. Pub. Dep.
9. IBM System/360 principles of operation. Form A22-6821-4. IBM Corp. Tech. Pub. Dep.
10. Rodriguez-Rosell, J. The working set behavior of some programs. Tech. Rept. NA 72-51, Dep. of Inf. Proc., The Royal Institute of Technology, Stockholm, Sweden, 1972.