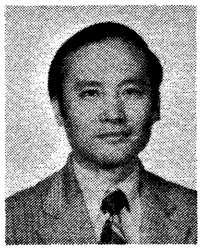
J., Extra Issue 26, Nippon Telegraph and Telephone Public Corp., Tokyo, Japan, 1968.
[16] J. Culliney, "On the synthesis by integer programming of optimal NOR gate networks for four-variable switching functions," Dep. Comput. Sci., Univ. of Illinois, Urbana, IL, Rep. 480, 1971.
[17] H. C. Lai, T. Nakagawa, and S. Muroga, "Redundancy check technique in designing optimal networks by branch-and-bound method," *Int. J. Comput. and Inform. Sci.*, vol. 3, pp. 251–271, Sept. 1974.
[18] H. Hart and A. Slob, "Integrated injection logic: A new approach to LSI," *IEEE J. Solid-State Circuits*, pp. 346–351, Oct. 1972.
[19] H. H. Berger and S. K. Wiedman, "Merged-transistor logic—A low cost bipolar logic concept," *IEEE J. Solid-State Circuits*, pp. 340–346, Oct. 1972.
[20] American Micro-Systems, Inc., *MOS Integrated Circuits*. Princeton, NJ: Van Nostrand, 1972.
[21] *Electronics*, pp. 91–96, Feb. 21, 1974.

Saburo Muroga (SM'60) was born in Numazu, Japan, on March 15, 1925. He graduated from the Department of Electrical Engineering, Tokyo University, Tokyo, Japan, in 1947.

After working with the Railway Technical Laboratories and then the Radio Regulatory Commission, he joined the staff of the Electrical Communication Laboratories, Nippon Telegraph and Telephone Public Corporation, in 1951. There he was engaged in research of information theory. He later was in charge of design, construction and operation of MUSASINO-1, the first universal digital computer with parametrons in Japan. In 1960, he joined the research staff of IBM Thomas J. Watson Research Center, Yorktown Heights, NY, and since 1964 he has been Professor of Computer Science and Electrical Engineering, at the Department of Computer Science, University of Illinois, Urbana, IL. He has published several scores of papers (many in Japanese) and a few Japanese books on pulse modulation theory, narrow-band voice transmission system, information theory, computer organization, threshold logic, file-memory addressing, integer programming, switching theory and logical design of integrated circuits. He has more than a score of patents. In 1971 he authored a book titled *Threshold Logic and Its Applications* (New York: Wiley, 1971).

Dr. Muroga is a member of the Association for Computing Machinery, the Information Processing Society of Japan, and the Institute of Electrical Communication Engineers of Japan.



Hung Chi Lai received the B.Eng. and M.-Eng. degrees in electronic engineering in 1968 and 1970, respectively, from the University of Tokyo, Tokyo, Japan, and the Ph.D. degree in computer science in 1975 from the University of Illinois at Urbana-Champaign.

During 1970–1975, he was a Research Assistant in the Department of Computer Science, University of Illinois. He is now with the Laboratory Division, Fujitsu California, Inc., Sunnyvale, CA.

Dr. Lai is a member of Sigma Xi, Phi Kappa Phi, and the Association for Computing Machinery.

A Modified Working Set Paging Algorithm

ALAN J. SMITH, MEMBER, IEEE

Abstract—**The working set paging algorithm is known to be highly efficient, yet has the disadvantage that during changes of locality large numbers of pages are accumulated in memory unnecessarily. The author proposes a modification of the working set algorithm called the damped working set (DWS) algorithm which resists sudden expansion of working set size and exhibits far greater stability in the number of page frames allocated to an active process. Program address traces are analyzed to determine the effectiveness of the DWS algorithm.**

Index Terms—**Paging algorithms, page replacement algorithm, virtual memory, working set paging algorithm.**

I. Introduction

A PAGED virtual memory, implemented as early as 1958 in the Atlas computer [1], has become a common feature of large, modern operating systems. Paging is highly advantageous in that it permits and simplifies the management of virtual address spaces that are often larger than the physical memory available. Paging, however, introduces an overhead in the operation of a computer system in several ways; it requires additional time for address formation and page faults require processing time to execute the page replacement and page placement algorithms. Even in multiprogrammed computer systems, the processor is often forced to remain idle while pages are transferred from secondary storage to main memory.

The costs associated with paging have led to a great deal of research on the subject of paging algorithms. The implicit or explicit objective of this research has been to maximize the "throughput" which is itself an ill-defined term. Because of the complexities inherent in designing a paging algorithm to cope with both uni- and multiprogramming, fixed or variable partitions, drums, disks or slow core as the secondary storage medium and differing processors, some researchers (e.g., [2]) have chosen as their objective to minimize either the *real* or *virtual space-time product*. The space-time product is the amount of the memory resource used (in bit seconds or word seconds) by the process under the given paging algorithm. The virtual space-time product is the space-time product measured during process or virtual time. The real space-time product is commonly defined as the space-time product as com-

Manuscript received September 30, 1974; revised December 19, 1975. This research was supported under a fellowship from the National Science Foundation. This research was also supported in part by the National Science Foundation under Grant DCR74-18375. Computer time was provided by the Atomic Energy Commission under Contract AT(04-3)-515.

The author is with the Computer Science Division, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720.

puted in a uniprogramming system with a secondary storage device of some specified characteristics, in which the processor remains idle during page faults. The real space-time product obviously includes the resource used while the page is brought into memory. Since the real space-time product can be determined (approximately) once the virtual space-time product is known (see Section II), we chose to restrict ourselves to the simpler case of virtual space-time.

We shall be concerned in this paper with describing a demand paging algorithm proposed by the author (see also [3]) and in experimentally examining its performance. For evaluation, the operation of the least recently used (LRU), MIN, working set, and VMIN algorithms will be compared with our algorithm; we briefly review for the reader the definition of these algorithms below and then we indicate the bases for comparison.

1) LRU [4]: At the time of a page fault, remove the page in memory that has not been referenced for the longest period of time and use the page frame made available.

2) MIN [4]–[7]: Remove the page from memory that *will* not be used for the longest period of time. This algorithm requires knowledge of the future and is, therefore, not implementable in practice, but it is known to be optimal [4] in terms of minimizing the number of page replacements in a fixed-size memory and, therefore, provides a useful standard for comparison.

3) Working Set [8]–[15]: The working set algorithm retains in memory exactly those pages of each process that have been referenced in the preceding T seconds of process (virtual) time. If an insufficient number of page frames are available, then a process is deactivated in order to provide additional page frames.

4) VMIN [16]: Remove from main memory any page that *will* not be referenced sooner than T seconds. This algorithm minimizes the virtual space-time product (see [16]) for a given fault rate but is unrealizable in practice. It provides, as does MIN, a useful standard of comparison.

Of the four paging algorithms discussed above, LRU and MIN assume that the amount of memory available to the process(es) is fixed. (These algorithms may also be applied globally over a set of programs in a multiprogramming environment, in which case the number of pages available to each program is permitted to vary. While there is some evidence that this produces improved performance [17], we will concern ourselves with measurements taken in a uniprogramming environment for simplicity.) Working set and VMIN both assume that the number of page frames available to any one process is variable according to the needs of the process; therefore, they implicitly function only in a multiprogramming environment. Comparisons between these algorithms exhibit the inherent advantages of variable space algorithms and argue for multiprogramming. The operation of variable space algorithms also requires the existence of a buffer pool of available page frames and our measurements of the average number of page frames used does not take this into account. There seems to be no simple way to avoid this problem.

Some analysis of the above algorithms is reported in the literature [5], [14], [18]–[21]. Some comparisons, based on measured memory address strings or models of program behavior are also reported in the literature [5], [22], [23]. As a general rule, those algorithms using past history (e.g., LRU) do better than those not using past history and those which allow the space occupied by a program to vary (e.g., working set) do better than those which do not (e.g., LRU).

We are concerned in this discussion solely with demand paging algorithms; algorithms which attempt to predict which pages will be needed and fetch them in advance do not concern us here. The working set algorithm, being perhaps the most efficient (although difficult to implement) and having a simple definition, was a promising starting place for obtaining a still better algorithm.

II. The Working Set Algorithm

The working set algorithm keeps in memory exactly the members of the working set ($W(t,T)$) which is defined for time t as the set of pages referenced in the process time interval $(t - T,t)$. The working set size $w(t,T)$ is the number of pages in $W(t,T)$. $w(T)$ is the mean working set size. We note that working set is very similar to LRU; the working set algorithm specifies removal of the LRU page when that page has not been used for the preceding T time units, whereas the LRU algorithm specifies removing the Kth most recently used page when a page fault occurs in a (full) memory of capacity K.

The success of the working set algorithm is based on the observed fact that a process executes in a succession of localities [9]; that is, for some period of time the process uses only a subset of its pages and that with this set of pages in memory the program will execute efficiently. Because at various times the number of pages needed to execute efficiently will change, the pages used in the preceding T seconds (for some appropriate T) are considered to be a better predictor than simply the set of K (for some K) pages most recently used. Thus, for example, a compiler may need only 25 pages to execute efficiently during parsing, but may need 50 during code generation; working set with the correct choice of the parameter T would adapt well to this situation, whereas a constant K over both phases of the compiler would either use excess space in the syntax phase or insufficient space in the code generation phase. The problem of choosing the appropriate value of T has been the subject of some research [8], [13], [24], but will not be considered here.

Available to the author are the memory address traces of a number of different programs as executed on the IBM System 360, including WATFIV, a Fortran compiler; WATEX, the execution of a typical program written in Fortran and compiled using the WATFIV compiler; APL, the execution of a program written in APL; and two fast Fourier transform programs, FFT1 and FFT2. Interpretive simulation of different paging algorithms on these address traces has been used to examine the behavior of different paging algorithms. Efficient methods for such interpretative simulations are discussed by Belady [5], [6] who provides an efficient implementation of the MIN algorithm and by Mattson *et al.* [4]. By collecting the distribution of interreference times to individual pages, it is possible to calculate the number of faults that occur when the working set algorithm is used [10], [15]. In Fig. 1 we show such a plot for each of the above programs, for a page size of 256 words. In Fig. 2 we show the mean working set size in virtual time ($w(T)$) and the maximum observed working set size (labeled "max"), as a function of the working set parameter T, for three of the programs. We note that the virtual space-time product when using the working set algorithm can be computed for any value of the working set parameter T by multiplying the mean working set size in virtual time by the length of the memory address trace. This also holds true for the data presented in later illustrations. The observed behavior is similar to that observed and/or predicted by other researchers [2], [10], [14]. The WATFIV program executed for 1 050 001 memory cycles; the others were examined for the first 1 500 000 cycles.

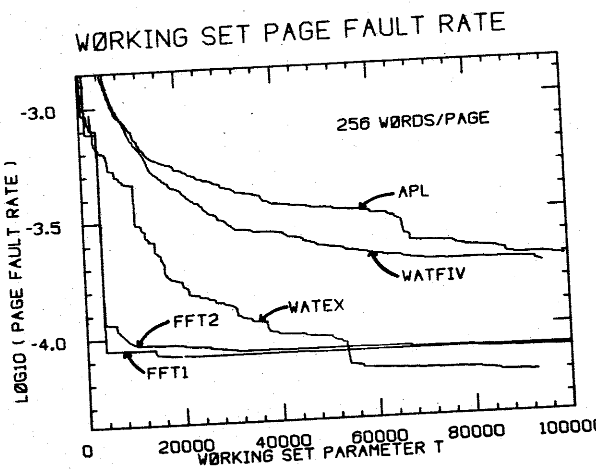


Fig. 1. Working set page fault rate.

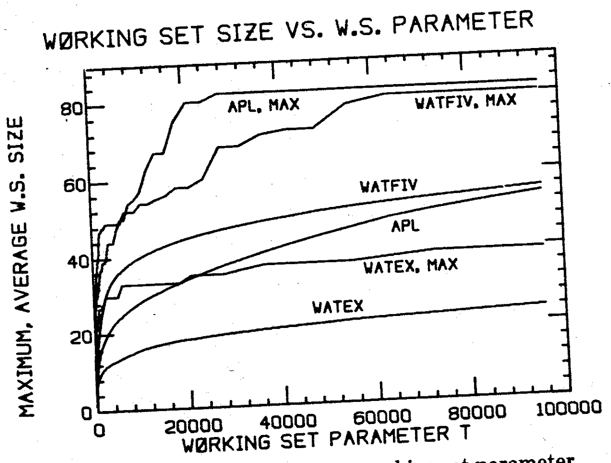


Fig. 2. Working set size versus working set parameter.

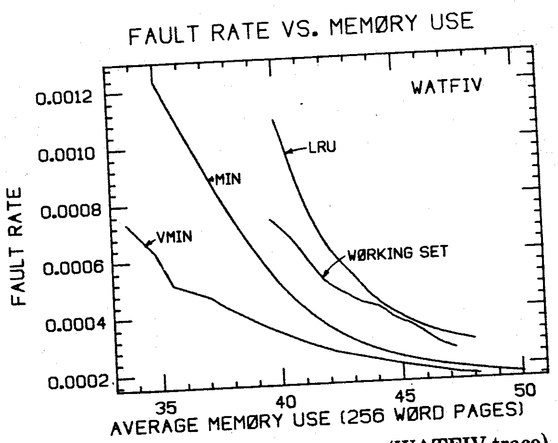


Fig. 3. Fault rate versus memory use (WATFIV trace).

Figs. 3 and 4 are plots of the fault rate in virtual time for two of the programs, WATFIV and WATEX, as a function of the average memory use, also in virtual time. We note that there is a fairly consistent ordering which has also been observed by the author on other traces and elsewhere (e.g., [24]) by other researchers: VMIN out-performs MIN and working set out-performs LRU. Both of these results show variable space algorithms out-performing fixed space algorithms. We note that in Fig. 4 working set has out-performed MIN for a portion of the scale. This type of behavior has been observed on other occasions [24] and results from the fact that working set is able to adapt to the wide variations in the number of pages necessary for efficient operation whereas MIN is not able to adapt.

We note from Fig. 1 that the number of faults for relatively large values of the working set parameter (e.g., greater than 20 000 memory cycles) is relatively insensitive to the value of the parameter T. That is, large changes in T produce small changes in $w(T)$ and in the observed fault rate, which is a valuable property. Although all of the algorithms examined display comparable wild increases in the fault rate as the (mean) number of page frames allocated decreases, the result is highly sensitive to the independent parameter K, the number of page frames, in the MIN and LRU cases whereas with VMIN and working set the independent parameter is T and the changes are far less drastic.

In Fig. 5 the expected time until the next reference to a page is plotted as a function of the time since the last reference to that page (all pages never referenced again were ignored) and it can be seen that the expected time until the next reference is a generally increasing function of the time since the last reference. This confirms the expectation from which working set and LRU originated: that the most recently used pages are those most likely to be used again soon.

The working set size was also measured at the times of page faults and no statistically significant and consistent difference was found to exist between the mean value measured over all time (actually at 1000 reference intervals) and the value measured at fault times. This indicates that there seems to be no simple and obvious modification that we should make to the number of pages kept in core as a function of $w(t,T)$. Thus, modifications of the type "when $w(t,T)$ is large keep extra pages in core and con-

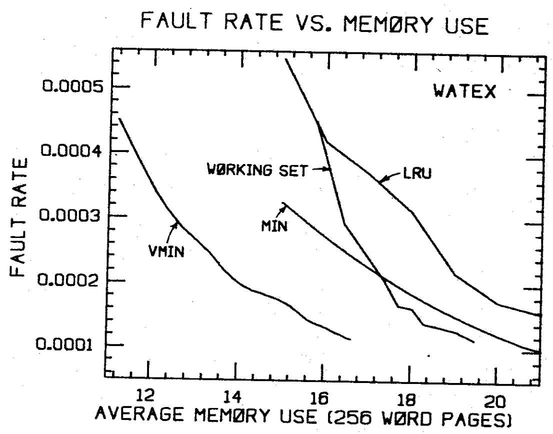


Fig. 4. Fault rate versus memory use (WATEX trace).

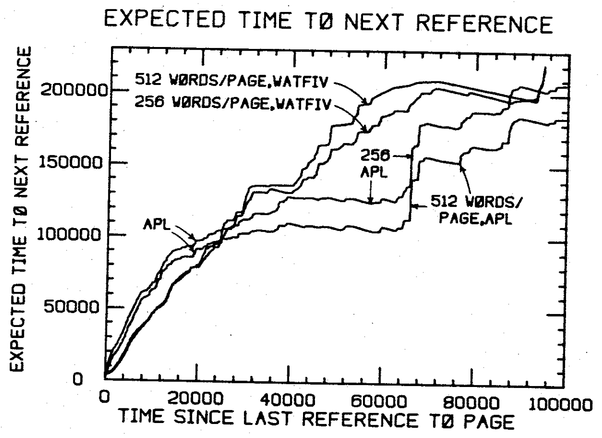


Fig. 5. Expected time to next reference versus time since last reference to page.

versely" seem to be inappropriate. This is similar to the observation that because of the concave nature of the faults versus T plot (Fig. 1), varying T around an average value of $\bar{T}$ would yield more faults than keeping T constant. The latter observation is based on the assumption that variations in T are independent of the value of $w(t,T)$. The lack of a correlation between the working set size and the fault rate indicates that variations of T that depend on $w(t,T)$ would be no more effective.

This last observation, that the working set size and fault rates are uncorrelated, allows us to calculate (approximately) the real space–time product. The page faults increment the virtual space-time product by an amount equal to the product of the mean working set size, the number of page faults and the mean time to process a page fault. Therefore, we have (approximately)

$$\begin{aligned}&\text{real space-time product} \approx\\&(\text{trace length}) * (\text{mean working set size})\\&+ (\text{time to process page fault})\\&* (\text{mean working set size})\\&* (\text{number of page faults}).\end{aligned}$$

The interested reader may make this calculation from the values given in Figs. 1, 3 and 4; we abstain for reasons of brevity.

III. The Search for a New Algorithm

There is one difficulty with the working set algorithm which up until now has not been mentioned. Occasionally, or even frequently during the execution of a program, a sudden change of locality will occur. For example, a compiler will often have a syntax analysis phase (parsing) followed by a code generation phase. The pages used during these two phases will be substantially different. Much the same sudden change of locality will occur when a large subroutine which may do substantial computation and use considerable amounts of memory is called. The problem is that when such a change of locality occurs, a time of T will elapse before the pages belonging to the old locality may be completely removed from memory. In the meantime, a number of pages belonging to the new locality will be brought into memory. Such a situation is illustrated in Fig. 6, where dimensionless graphs of the expected type of behavior are shown. (There may also be pages in common to the old and new localities.) Figs. 7–9 are plots of the working set size $w(t,T)$ (labeled with "mult = 1"; the uppermost trace) for three of the above-mentioned programs as a function of time. It can easily be seen that there are sudden peaks in $w(t,T)$ which represent just such changes of locality. (The meaning of "mult" and of the other curves will be discussed in Section IV.)

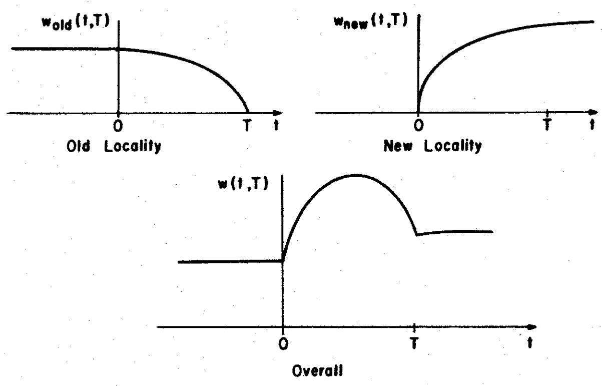


Fig. 6. Working set size during change of locality.

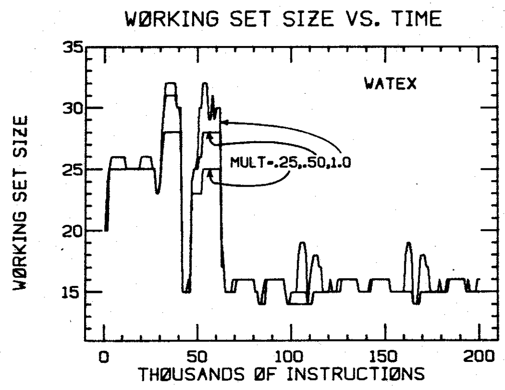


Fig. 7. Working set size versus time (WATEX trace).

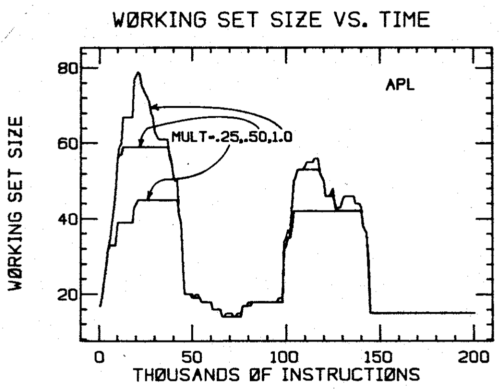


Fig. 8. Working set size versus time (APL trace).

In an inexhaustible memory where the only constraint on the memory use is its cost (average memory size used) the peaks in the working set size are unimportant, as they contribute very little to $w(T)$. Unfortunately, all current memories are finite and the use of working set implies a multiprogramming environment. Such peaks in $w(t,T)$ represent page frames that are either taken from the common pool of unassigned frames, or are frames belonging to another process. In the latter case, it may even be necessary to deactivate a process in order to accommodate this peak demand. It seems at least as important to seek to minimize the maximum memory demand of a process (consistent with efficient operation) as to minimize the average memory demand.

What is desired then in a paging algorithm is to retain in main memory exactly those pages in the current locality without accumulating excess pages during changes in locality. The effect of this policy, if applied to a working set type algorithm, would be to cut off the peaks of the $w(t,T)$ versus t plot. It would be similar to applying a low-pass filter to the $w(t,T)$ "signal" (see Section IV).

In his dissertation, Prieve [24] considered the problem of choosing an appropriate working set parameter and suggested using a different value of T for each page in the address space of the process. By observing the previous periods of activity and inactivity of each page, he attempted to compute a value of T that will indicate when the process has indeed finished using the page. A by-product of his approach is to reduce the maximum working set size, but he does not indicate the behavior of his algorithm during changes in locality.

Before deciding on the damped working set (DWS) algorithm, discussed in the next section, the author investigated a number of other approaches. These are discussed in [25].

IV. The DWS Algorithm

The algorithm chosen by the author to remedy the defects of the working set algorithm, the DWS algorithm, functions as follows.

1) Any page not referenced within the preceding T seconds is removed from main memory.
2) At the time of a page fault, if the time since the last reference to the least recently used page in core is greater than mult * T (mult less than 1); then replace the least recently used page by the new page, otherwise increase the space allocation by one page frame.

The DWS algorithm has two parameters; T, the working set parameter, and mult, a multiplier less than or equal to 1. Alternately, the two parameters may be referred to as

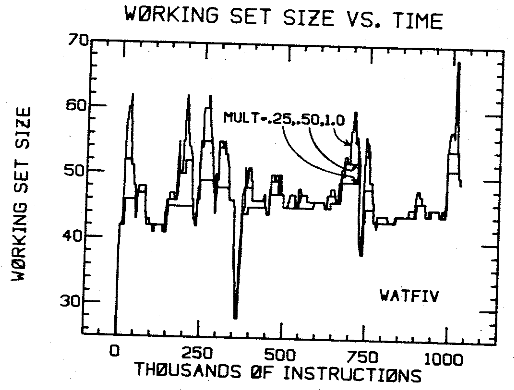


Fig. 9. Working set size versus time (WATFIV trace).

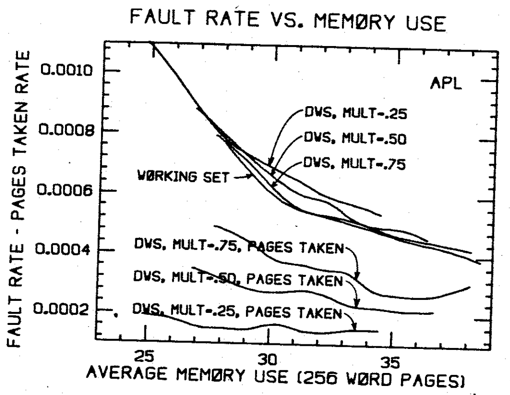


Fig. 10. Fault rate versus memory use (APL trace).

T and T′ where T′ = mult * T. The working set algorithm is itself a special case of DWS in which mult = 1. An algorithm similar to DWS is reported by Fogel [3] and is discussed in Section V; his algorithm was created for different reasons.

The behavior of the DWS algorithm is illustrated in Figs. 7–9. In Figs. 10–12 we plot the number of faults as a function of the average memory space used, for three different programs and different values of mult ("pages taken" is discussed below). It can be seen that the number of faults with DWS is sometimes slightly higher (APL, WATFIV) than working set for the same average memory allocation and sometimes lower (WATEX). The occasional increase in the fault rate by DWS over working set is not unexpected; DWS resists true changes in the working set size and in programs in which these are a frequent occurrence, the fault rate can be expected to increase slightly. Conversely, the WATEX trace displays large fluctuations in $w(t,T)$ which do not reflect true increases in the number of pages necessary to run efficiently; thus the fault rate remains (almost) unchanged as a function of $w(T)$ as $w(T)$ decreases for DWS. In all cases, it can be seen that the increase or decrease in fault rate as a function of the mean working set size is small and that the analysis of many more traces would be necessary to establish any consistent pattern.

We see in Figs. 7–9 that DWS has produced the desired type of behavior. The peaks in the $w(t,T)$ versus t curve have been cut off, more and more effectively for smaller and smaller values of mult. This is further indicated in Figs. 13 and 14 where the maximum number of pages used is plotted as a function of the working set parameter T for two of the traces. It can be seen that the maximum is considerably less than for traditional working set. In Figs. 10–12 the number of "pages taken" is shown, that is the number of times that the space allocated to the process under damped working set was increased. This number is considerably less than the number of faults and represents the number of times that a page frame must be obtained either from the common pool or from another process. With working set, every fault requires that a new page frame be obtained.

We pause here slightly to note that the problem of

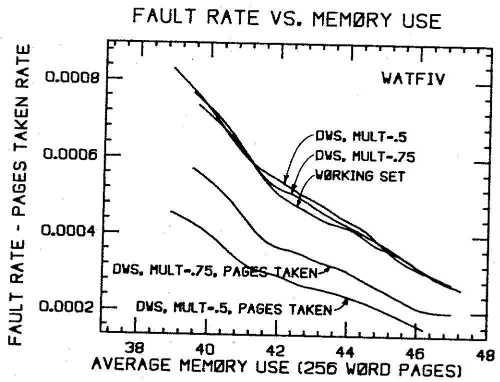


Fig. 11. Fault rate versus memory use (WATFIV trace).

controlling the large peaks in the working set size that occur when the locality changes is really a problem in time-series analysis. That is, the time function of the working set size is a time series [26]–[28]. Throughout this paper we have looked at this problem in the time domain only, but as one of the most popular approaches to time series is spectral analysis, we examine briefly the effect of the DWS algorithm on the frequency spectrum of the working set size. In Fig. 15 we show the discrete Fourier transform of $w(t,T)$ and $w(t,T,\text{mult})$ where $w(t,T,\text{mult})$ is the DWS size. The transform was generated by applying the fast Fourier transform algorithm [29] to the $w(t,T)$ function sampled at 1000 reference intervals. 1024 sample points were used and the result for clarity was smoothed using a ten point rectangular window. It can be seen in Fig. 15 that there is a smaller (proportionately) high-frequency component in $w(t,T,\text{mult})$ than in $w(t,T)$; thus our comment about introducing a "high frequency filter" seems appropriate. (We note that the log scale permits easy comparison of relative magnitudes.)

V. Implementation

The problem of implementing the working set paging algorithm has always been a difficult one. Denning [8] suggests modifying the hardware with capacitors to keep track of the time since last reference. Morris [20] describes an actual implementation on the Maniac II of the working set paging algorithm using modified hardware. Prieve [24]

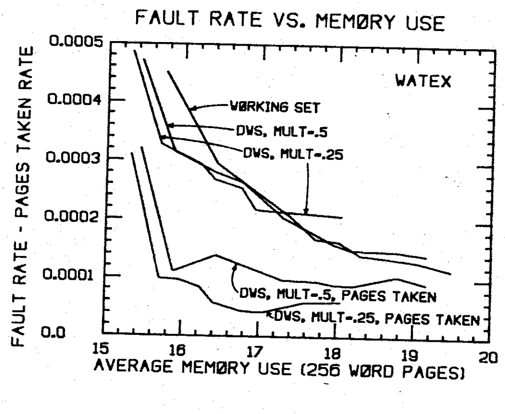


Fig. 12. Fault rate versus memory use (WATEX trace).

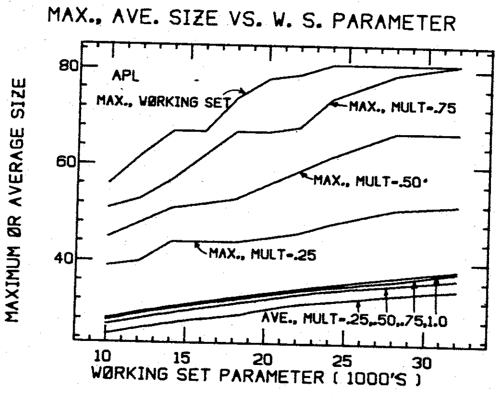


Fig. 13. Maximum and average working set size versus working set parameter (APL trace).

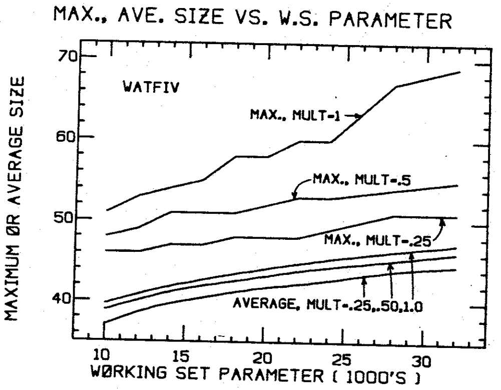


Fig. 14. Maximum and average working set size versus working set parameter (WATFIV trace).

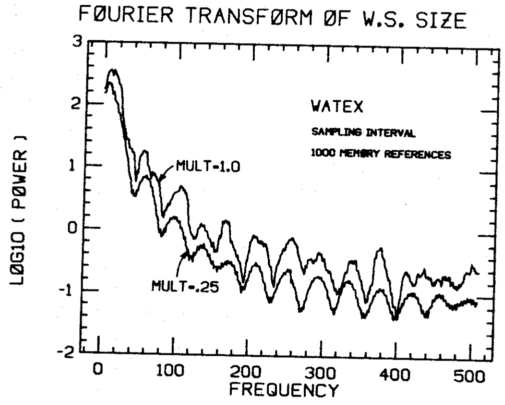


Fig. 15. Discrete Fourier transform of working set size versus time (1024 sample points, taken at intervals of 1000 memory references, 10-point rectangular window for smoothing).

suggests the use of reference bits which are examined at intervals, to approximate the working set algorithm as does Fogel [3]. Reference bits are common on many computers and are set when a page is referenced. The CDC Star [17] uses an LRU paging algorithm, with the LRU stack kept locally for each process. It could easily be modified to implement any version of the working set paging algorithm (either damped working set or working set) by associating with each entry in the LRU stack a separate field which gives the time of last reference. Any of the above suggestions are feasible, with some modification, for implementation of the DWS algorithm. The simplest would be the CDC Star with the additional "last referenced" field. We note that although DWS requires more information to be kept than working set and, therefore, presumably involves more overhead, this is almost certainly more than compensated for by fewer process deactivations.

An implementation of a similar, but not identical, algorithm to the DWS algorithm is reported by Fogel [3]. He describes the paging algorithm currently used by the UNIVAC virtual memory operating system (VMOS) running on the UNIVAC series 70 systems. VMOS keeps track of the time since last reference to a page by periodically scanning the access bits of the page frames. An estimate of the working set size is kept for each process by counting the number of pages used during its previous activation. The pages belonging to each process are grouped in two sets; a) those used within the preceding mult * T instructions, and b) those used within the preceding T instructions, but not within the preceding mult * T instructions where mult is less than one. (The mult notation is not that of the paper by Fogel [3].) When a page fault occurs, the need for a page is satisfied:

1) If the process is using all pages allocated to it under its estimate of its working set size (see above), then a page frame from set (b) is chosen; if none is available, then a frame from the pool of free frames is chosen. If none is available either way, the process is deactivated.
2) If the process has not yet reached its estimated size, then a page is taken from the pool of free frames [if any; if none are available, then from its set (b)]. If none are available either way, the process is deactivated.

Whenever a page from set (b) is chosen, the least recently used one is taken.

Fogel presents no analysis of the operation of this algorithm, other than to say that performance improvements were experienced in comparison to the previously used paging algorithm. Some suggested values for the parameters mult and T are given, but they represent experience with only one set of benchmark programs. By using the estimated working set size for the previous activation, there is the possibility that "stale" information, that is, outdated estimates of the working set size, will cause suboptimal behavior. It seems more efficient to simply prepage the entire working set from the end of the previous activation when a new activation is begun. The working set size could then be kept track of dynamically with no special adjustments for activation and deactivation. This could also avoid the overhead of processing page faults individually with the accompanying long latency. By placing the pages from the preceding activation on adjoining sectors on the paging drum (a frequently used paging device), a great deal of time could be saved in page fetch. If this were done, the algorithm described by Fogel would be essentially the same as DWS.

VI. Conclusions

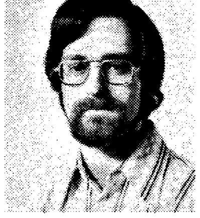
Working set is known to be an effective page replacement algorithm yielding a relatively low number of page faults as a function of the average memory space occupied. It has the problem that during changes of locality excessively large numbers of pages are collected in memory, with consequent harm to other processes in a multiprogramming environment. The proposed DWS algorithm alleviates this problem and is comparable in efficiency to working set.

Acknowledgment

The author would like to thank Prof. F. Baskett of Stanford University for his help and encouragement in this research and for pointing out the paper by Fogel [3]. The author also thanks L. Shustek for generating the program address traces analyzed in this paper, R. Chafee and R. Beach of the Stanford Linear Accelerator Center for the graphics systems that made possible the illustrations in this paper, and T. Kenyon and R. Suzuki for their skillful typing of successive drafts of this manuscript.

References

[1] T. Kilburn, D. B. G. Edwards, M. J. Lanigan, and F. H. Sumner, "One-level storage system," *IRE Trans. Electron. Comput.*, vol. EC-11, pp. 223–235, Apr. 1962.
[2] W. W. Chu and H. Opderbeck, "The page fault frequency replacement algorithm," in *1972 Fall Joint Comput. Conf., AFIPS Conf. Proc.*, vol. 41. Montvale, NJ: AFIPS Press, 1972, pp. 597–609.
[3] M. H. Fogel, "The VMOS paging algorithm, a practical implementation of the working set model," *Operating Syst. Rev.*, vol. 8, pp. 8–17, Jan. 1974.
[4] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation techniques for storage hierarchies," *IBM Syst. J.*, vol. 9, pp. 78–117, 1970.
[5] L. A. Belady, "A study of replacement algorithms for a virtual storage computer," *IBM Syst. J.*, vol. 5, pp. 78–101, 1966.
[6] L. A. Belady and F. P. Palermo, "On-line measurement of paging behavior by the multivalued MIN algorithm," *IBM J. Res. Develop.*, vol. 18, pp. 2–19, Jan. 1974.
[7] E. G. Coffman, Jr. and L. C. Varian, "Further experimental data on behavior of programs in a paging environment," *Commun. Ass. Comput. Mach.*, vol. 11, pp. 471–474, July 1968.
[8] P. J. Denning, "The working set model for program behavior," *Commun. Ass. Comput. Mach.*, vol. 11, pp. 323–333, May 1968.
[9] ——, "On modeling program behavior," in *1972 Spring Joint Comput. Conf., AFIPS Conf. Proc.*, vol. 40. Montvale, NJ: AFIPS Press, 1972, pp. 937–944.
[10] P. J. Denning and S. C. Schwartz, "Properties of the working set model," *Commun. Ass. Comput. Mach.*, vol. 15, pp. 191–198, Mar. 1972.
[11] D. P. Fenton, "B6700 'working set' memory allocation," in *Operating Systems Techniques*, C. A. Hoare and R. H. Perrot, Eds. New York: Academic, 1972, pp. 321–327.
[12] D. J. Hatfield and J. Gerald, "Locality, working set, request string, distance function, and replacement stack," in *Statistical Computer Performance Evaluation*, Frieberger, Ed. New York: Academic, 1972, pp. 407–422.
[13] B. G. Prieve, "Using page residency to select the working set parameter," *Commun. Ass. Comput. Mach.*, vol. 16, pp. 619–620, Oct. 1973.
[14] J. R. Spirn and P. J. Denning, "Experiments with program locality," in *1972 Fall Joint Comput. Conf., AFIPS Conf. Proc.*, vol. 41. Montvale, NJ: AFIPS Press, 1972, pp. 611–621.
[15] D. R. Slutz and I. L. Traiger, "A note on the calculation of average working set size," *Commun. Ass. Comput. Mach.*, vol. 17, pp. 563–565, Oct. 1974.
[16] B. G. Prieve and R. S. Fabry, "An optimal variable space page replacement algorithm," in *Proc. Fifth Symp. Operating Systems Principles* (Supp.), Austin, TX, Nov. 1–3, 1975; also in *Commun. Ass. Comput. Mach.*, vol. 19, pp. 295–297, June 1976.
[17] N. A. Oliver, "Experimental data on page replacement algorithm," in *Proc. Nat. Comput. Conf.*, 1974, pp. 179–184.
[18] D. J. Hatfield, "Experiments on page size, program access patterns and virtual memory performance," *IBM J. Res. Develop.*, vol. 16, pp. 58–66, Jan. 1972.
[19] P. A. W. Lewis and G. S. Shedler, "Empirically derived micromodels for sequences of page exceptions," *IBM J. Res. Develop.*, vol. 17, pp. 86–100, Mar. 1973.
[20] J. B. Morris, "Demand paging through utilization of working sets on the MANIAC II," *Commun. Ass. Comput. Mach.*, vol. 15, pp. 867–872, Oct. 1972.
[21] A. J. Smith, "Analysis of the optimal, look ahead, demand paging algorithms," *SIAM J. Comput.*, to be published.
[22] J. L. Baer and G. R. Sager, "Measurement and improvement of program behavior under paging systems," in *Statistical Computer Performance Evaluation*, Freiberger, Ed. New York: Academic, 1972, pp. 241–264.
[23] B. S. Brawn and F. G. Gustavson, "Program behavior in a paging environment," in *1968 Fall Joint Comput. Conf., AFIPS Conf. Proc.*, vol. 33. Montvale, NJ: AFIPS Press, 1968, pp. 1021–1032.
[24] B. G. Prieve, "A page partition replacement algorithm," Ph.D. dissertation, Univ. California, Berkeley, 1974.
[25] A. J. Smith, "Performance analysis of computer system components," Ph.D. dissertation, Dep. Comput. Sci., Stanford Univ., Stanford, CA, 1974.
[26] M. A. Kendall, *Time Series*. London, England: Griffin, 1973.
[27] G. E. P. Box and G. M. Jenkins, *Time Series Analysis: Forecasting and Control*. San Francisco, CA: Holden-Day, 1970.
[28] J. Makhoul, "Linear prediction: A tutorial review," *Proc. IEEE*, vol. 63, pp. 561–580, Apr. 1975.
[29] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, pp. 297–301.



Alan J. Smith (S'73–M'74) was born in New Rochelle, NY. He received the S.B. degree in electrical engineering from the Massachusetts Institute of Technology, Cambridge, and the M.S. and Ph.D. degrees in computer science from Stanford University, Stanford, CA, the latter in 1974.

He is currently an Assistant Professor in the Computer Science Division of the Electrical Engineering and Computer Sciences Department, University of California, Berkeley, a position he has held since 1974. His research interests include the analysis and modeling of computer systems and devices, operating systems, and data compression.

Dr. Smith is a member of the Association for Computing Machinery, the Society for Industrial and Applied Mathematics, Eta Kappa Nu, Tau Beta Pi, and Sigma Xi.