VSWS: THE VARIABLE-INTERVAL SAMPLED WORKING SET POLICY

by

Domenico Ferrari and Yiu-Yo Yih

Memorandum No. UCB/ERL M82/16

17 March 1982

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

VSWS: The Variable-Interval Sampled Working Set Policy[°]

Domenico Ferrari and Yiu-Yo Yih[°°]

Computer Science Division
Department of Electrical Engineering and Computer Sciences
and Electronics Research Laboratory
University of California, Berkeley

Abstract

A local variable-size memory policy called the variable-interval sampled working set (VSWS) policy is described. The results of trace-driven simulation experiments reported here show that VSWS has a static performance comparable to those of the working set (WS) and sampled working set (SWS) policies, a dynamic performance better than those of WS, SWS, and the page fault frequency (PFF) policy, and similar to that of the damped working set (DWS) policy. Furthermore, VSWS generally causes substantially less process suspensions than SWS, and is less expensive to implement than WS or DWS, since it requires the same hardware support as SWS and PFF. The sampling overhead of VSWS is comparable to that of SWS and lower than that of PFF.

Keywords: Damped working set policy, Local replacement policy, Memory management, Page fault frequency policy, Program behavior, Replacement algorithm, Sampled working set policy, Variable-size policy, Virtual memory, Working set policy.

[°] The work reported here was supported in part by the Computer Systems Design Program of the National Science Foundation under Grant MCS80-12900.

[°°] Author's current address: Bell Laboratories, Holmdel, NJ 07733.

1. Introduction

Among the realizable policies proposed for the management of memory in a virtual memory system, none has proved substantially and consistently better than the *working set (WS) policy* [5][7]. This policy assumes that the size and the membership of the current locality of each running process coincide with the size and the membership of its current *working set*. In a paging environment, the working set of a process at virtual time t is the set of pages referenced by the process in the interval $(t-T,t)$, where T is the *working set parameter* or *window size*. Not only is the working set of a process to be kept in main memory in order for the process to be part of the multiprogramming set (i.e., to be allowed to run), but the loading of a process is made conditional on there being sufficient room in memory for its working set. Furthermore, lack of space to accommodate the demands of a process whose working set size is growing will cause one of the processes in main memory to be deactivated and unloaded.

Thus, the WS policy is a local variable-size replacement algorithm that incorporates memory scheduling and swapping policies as well as a policy for the automatic control of the multiprogramming level. Its being a variable-size policy, that is, its allowing the resident set size of a process to grow and shrink dynamically during execution, makes it capable of utilizing memory more efficiently than any constant-size local policy [9]. The built-in control scheme for the multiprogramming level prevents thrashing without the addition of external mechanisms like those required by global replacement policies [6]. The incorporated scheduling and swapping policies make the design of a crucial part of the operating system cleaner and easier to understand: the interactions among all the policies involved are better known and controlled than when each of these resource management problems is solved in a separate, *ad-hoc* fashion.

These advantages are unfortunately accompanied by a few drawbacks, the most notable of which are the cost of implementation and the inability of the policy to deal with abrupt inter-locality transitions efficiently.

To implement the WS policy in its pure form, the system should keep track of the virtual time at which each page in main memory has been last referenced, and eliminate from the resident set (i.e., return to the free pool the page frames they occupy) those pages whose virtual time since the last reference becomes longer than T. Even with the advent of VLSI processors, these operations, which should be performed at each reference, are probably too expensive, or time-consuming, or both, for a practical implementation of the pure WS policy to be justified, especially if the current trend towards larger memory demands and smaller page sizes continues. Only one implementation approximating quite closely the pure WS policy has been reported in the literature [14].

This problem may be solved by adopting an approximation called the *sampled working set (SWS) policy* [4][16][17][19]; historically, SWS preceded WS, which was introduced as an idealization of SWS [7]. The SWS policy evaluates the working set only at sampling instants which are equidistant in virtual time (note that virtual time may be measured in memory references or instructions executed rather than in seconds). The most common choice is to make the duration I of the sampling interval equal to the window size T. Each page frame in memory has a *use bit* which is turned on by the hardware whenever an information item stored in that frame is referenced. At the beginning of a sampling interval, the use bits of all the frames allocated to the running process are reset;

at the end, only the frames containing pages which have been referenced during the interval will have their use bit set; these pages are retained in the resident set of the process throughout the next interval, while the others are expelled (i.e., their frames are returned to the free frame pool). The resident set size of a process can only decrease at the end of an interval and not during each interval; the faulted pages do not replace any of the pages in the resident set, but are stored into frames from the free pool.

The SWS policy is clearly less expensive to implement than the WS policy, and in general exhibits a performance quite close to that of the latter [15]. Two inconvenient aspects of SWS are the periodic suspensions of a process' execution and the need for the operating system to scan the use bits of all the pages in the running process' resident set at every such suspension. It should be noted, however, that this overhead is inconvenient in absolute, not in relative terms, since there are no use-bit-based local variable-size policies with an overhead lower than that of SWS.

A variable-size policy which is also much cheaper to implement than WS is the *page fault frequency (PFF) policy* [3]. PFF eliminates pages from the resident set by examining the use bits only at the time of a page fault; at that time, pages are expelled only if the previous page fault occurred more than τ time units (or references) before. When a page fault occurs at a distance in time from the previous fault less than or equal to τ, which is the PFF policy parameter, the resident set size is increased by one frame and no page is expelled. In spite of its inexpensive implementability and of its not requiring in general any suspensions of process execution besides those due to page faults, the PFF policy has been found to exhibit more anomalies [8] and to be more sensitive to the value of the policy parameter [10][11] than WS. It should be noted that, unlike WS, PFF does not have the inclusion property [13]; in other words, given two values τ_1 and τ_2 of the policy parameter, with $\tau_1 < \tau_2$, the resident set of a process under PFF with parameter τ_1 is not guaranteed to be included at every virtual time instant in the resident set with parameter τ_2. Note also that, though it is a good approximation of the WS policy from a performance viewpoint, SWS does not exhibit the inclusion property for all values $T_2 > T_1$, but only for $T_2 \geq 2T_1$.

The other major problem with the WS policy is due to the fact that no page ever drops out of the working set (hence, of the resident set) before T virtual time units have elapsed since it was last referenced. During inter-locality transitions, the rapid succession of page faults causes the resident set of a process to swell before the pages of the old locality are expelled, and the sudden peaks of memory demand may produce unnecessary process deactivations and reactivations, with the corresponding undesirable switching and swapping overheads.

From the dynamic viewpoint, neither SWS nor PFF outperforms WS. Actually, SWS is slightly worse than WS, due to the additional delay it introduces in eliminating unreferenced pages from the resident set; PFF is substantially worse, since it does not get rid of any unused pages as long as page faults keep occurring at intervals shorter than τ. Unused pages tend to be kept in memory by PFF also when a smaller-locality phase of execution is entered, where no page faults are generated for a relatively long time; to reduce the negative effects of this problem, a second parameter may be introduced, which establishes the maximum interval during which a process may run without its use bits being scanned.

The desire to improve the performance of the WS policy during abrupt transitions suggested the introduction of the *damped working set (DWS) policy* [18]. DWS can be informally described as an algorithm which reduces by a fixed factor the window size (hence, the resident set size) whenever the page fault rate gets larger than a given threshold. This two-parameter policy has proved to be effective in reducing the peaks of resident set sizes, but is unfortunately as expensive to implement as the WS policy.

Ideally, one would like to have a policy characterized by the WS policy's performance and controllability (i.e., the possibility of finding values for the policy parameter which will produce near-optimal performance for a large fraction of the processes executed by an installation), the implementation costs of SWS (or lower), and the dynamic performance of DWS. This paper presents a new policy which may be considered a step in the direction of the ideal one we have just described.

2. The Variable-Interval Sampled Working Set Policy

The memory management policy to be presented in this section is an SWS-like policy with three parameters, two of which, however, act as bounds and should be expected to require very infrequent, if any, tuning modifications. These two parameters, to be denoted by M and L, respectively, are the minimum and the maximum durations of the sampling interval. The third parameter is the number Q of page faults after which the use bits are to be scanned.

The VSWS (Variable-Interval Sampled Working Set) policy can be more precisely described as follows.

1. If the virtual time since the last scanning of the use bits reaches L, then suspend the process and scan the use bits.

2. At the Q-th page fault which occurs after the last scanning of the use bits, (a) if the virtual time elapsed since that scanning operation is less than M, then wait until the elapsed virtual time reaches M to suspend the process and scan the use bits; (b) if the elapsed virtual time is greater than or equal to M, then scan the use bits while processing the Q-th page fault.

Note that, as in SWS, all pages whose use bit is found to be 0 are expelled from the resident set, and then all use bits are reset.

As mentioned above, the parameter values are to be selected so that most of the scannings of the use bits of a process will normally be triggered by the occurrence of the Q-th page fault after the last scan (case (2b)). Like DWS, the VSWS policy tries to reduce the peak memory demands caused by abrupt inter-locality transitions under the WS and SWS policies by increasing the sampling frequency, hence the rate at which unused pages drop out of the resident set, when the paging rate increases. It is interesting to note that the PFF policy is based on the opposite philosophy: it only expels unused pages from the resident set when the virtual time since the last page fault is greater than τ, that is, when the page fault frequency is lower than the threshold corresponding to τ. PFF tries to adjust the size of the resident set according to the fluctuations of a process' memory demand, assuming that a locality grows and shrinks in size while retaining an always active core of essential pages. The model of program behavior assumed by VSWS is different: according to this model, when a process references new pages, even with a very low frequency, it is likely to stop referencing a large fraction of the old pages. Thus, the model underlying VSWS assumes more dynamic and serial referencing patterns than those assumed by the PFF model. The VSWS model is certainly a more accurate description of abrupt inter-locality transitions, where most of the faults usually occur in the execution of many processes [12], and where some of the worst memory scheduling and allocation problems arise. Note also that, with $M = \tau$ and $Q = 1$, VSWS reduces to a policy very similar (but not identical) to PFF; however, the correct range of M, a parameter which establishes an upper limit for the sampling frequency, will in general contain much lower values than those normally considered for τ.

The implementation of VSWS, like those of SWS and of PFF, only requires a use bit for each page frame (note that use bits may be simulated in software, introducing a small amount of additional overhead [1]) and a virtual-time interval timer for each process. Every time a sample is taken, all the use bits of the frames allotted to the running process are to be examined, as is the case with SWS. However, the number of process suspensions for use bit scanning purposes is generally much lower than that which characterizes the SWS policy; if the two bounds M and L were never invoked, this number would be exactly zero, since all scannings would take place at page fault times. When page faults are infrequent, the sampling rate of VSWS is lower than that of PFF; when, on the other hand, this is not the case, PFF is likely to require a larger amount of memory space for a given process.

It should also be noted that VSWS exhibits a partial inclusion property similar to the one which, as mentioned in Section 1, holds for the SWS policy. A sufficient condition for the resident set produced by VSWS with parameter Q_2 to include at all virtual time instants the resident set with parameter Q_1 is that $Q_2 \geq 2Q_1 - 1$. This result can be proved by observing that the worst case arises when the $(Q_1 - 1)$-th page fault of a sampling interval under VSWS with parameter Q_1 coincides with the Q_2-th page fault of a sampling interval under VSWS with parameter Q_2. Immediately after this page fault, the resident set of VSWS with parameter Q_1 contains all the pages referenced since the $(2Q_1 - 1)$-th most recent page fault, while the resident set with parameter Q_2 contains all the pages referenced since the Q_2-th most recent page fault; thus, if the above inequality is satisfied, the latter resident set will certainly include the former. In this argument, we have assumed that the two executions (with parameters Q_1 and Q_2 respectively) produce exactly the same page faults. In general, if the above inequality is satisfied, the lower value of Q (i.e., Q_1) will generate more page faults; furthermore, due to the inclusion property, all the page faults caused by Q_2 will be page faults also for parameter Q_1; thus, the $(2Q_1 - 1)$-th most recent fault will occur later than the Q_2-th most recent fault, thereby making the inclusion condition even more easily satisfied. This remark explains why the above condition is not necessary, but only sufficient. Clearly, the derivation of this result has been based on the assumption that neither of the two bounds M and L ever causes any additional process suspension. When this assumption is not satisfied, the above condition is not even sufficient anymore.

The discussion in this section could be summarized by stating that the VSWS policy is a variable-size local replacement algorithm which

(a) has about the same performance as WS or SWS;

(b) has dynamic properties better than those of WS, SWS, and PFF, and similar to those of DWS;

(c) is less expensive to implement than WS or DWS in their pure forms, having about the same cost as SWS and PFF; and

(d) causes many less process suspensions than SWS.

Of course, claims (a), (b), and (d) require that appropriate criteria for comparison be selected, and must be confirmed by suitable experiments. The criteria, the experiments, and their results are described in the next section.

3. Experimenting with VSWS

The minimum set of policies to be compared with VSWS in order to verify the validity of the claims made at the end of the previous section clearly consists of SWS and DWS. Since the three policies involved were local and of the WS type, it was felt that uniprogramming experiments would be sufficient for their comparison. Three program traces were selected for our trace-driven simulation experiments: APL, the execution of a program originally written in APL, and containing 1,642,000 references; WATEX, the execution of a FORTRAN program translated by a WATFIV compiler, a trace of 1,642,200 references; and WATFIV, generated by a WATFIV compiler, and consisting of 1,048,599 references. The page size used to transform these traces into page reference strings was 512 words. Extensive data on the performance of these traces under the WS policy has been published by Smith [18].

To compare the static performances of the three policies (claim (a)), we resorted to the three types of diagrams most widely used for this purpose: those of the mean fault rate vs. the mean memory occupancy, of the mean memory occupancy vs. the policy parameter, and of the space-time product vs. the policy parameter. Since the parameter of the VSWS policy to be modified (i.e., Q) is not homogeneous with those of the other two policies, that are window sizes, for VSWS we plotted the memory occupancy and the space-time product vs. the mean sampling interval (i.e., the mean time between successive scannings of the use bits). The memory occupancy of a process at a certain time was considered to be equal to the size of its resident set at that time, ignoring the fact that a page dropped from the resident set is still in main memory, and can be reclaimed, until it is replaced. The dynamic performances of the three policies (claim (b)) were compared by visual inspection of the memory occupancy vs. virtual time curves. Some dynamic information was also provided by the diagrams of the mean page fault rate vs. the maximum memory occupancy. As far as claim (d) was concerned, the numbers of process suspensions required by SWS and VSWS were plotted as functions of the mean sampling interval.

Some preliminary experiments were run to determine reasonable values for the M and L parameters of VSWS. For all the traces, we set M = 500 references and L = 15,000 references. The choice of the value of M was dictated by our estimate of the number of instructions the operating system may have to execute in order to examine all the use bits of a process and to make the appropriate changes in the process' page table. To select the value of L, we observed that, for very low page fault rates, VSWS degenerates into SWS with a window size (and sampling interval) equal to L. Thus, it seems reasonable to choose for L a value not too far from those values of T beyond which the space-time product of a process starts increasing. This guarantees that, even during possibly long periods of very low fault rates, the space-time product will still be in the neighborhood of its minimum. In all experiments, the multiplicative parameter of DWS was chosen equal to 0.5.

The three traces produced roughly similar results. Figure 1 displays the mean fault rate vs. mean and maximum memory occupancy diagrams for WATEX. While the mean curves are practically indistinguishable, the maximum curves show that VSWS and DWS are better than SWS, though the differences are rather small. The results produced by the other traces, especially those for $Q >$ 5, were about the same. Note that some of the values of Q are reported on the curves for VSWS. The memory occupancy vs. T curves for WATFIV are displayed in Fig.2; those for the other two traces are similar to them. In Fig.2, DWS performs better than SWS in all respects, while VSWS exhibits an intermediate performance, closer to that of DWS for small values of T and to that of SWS for larger values of T (when T approaches 15,000 references, VSWS degenerates into SWS). It should be noted that for DWS the actual values of the mean window size are smaller than those reported in Figs.2 and 3, which correspond to the maximum window sizes used in the corresponding runs under DWS.

The diagrams of the space-time product vs. T for WATFIV in Fig.3 show that DWS achieves smaller memory occupancies at the price of higher fault rates. The result is a space-(virtual)time product larger than those achieved by the other two policies, of which VSWS is, by a small margin, the better.

The validity of claim (b), that is, that VSWS has a dynamic performance comparable to the one of DWS and substantially better than the performance of SWS, was tested in a large number of experiments. Figures 4, 5, and 6 present some sample results. In Fig.4, we see the memory occupancy of the APL trace vs. virtual time for the first 450,000 references, both under SWS and under VSWS. The window size for SWS has been set equal to the mean sampling interval of VSWS, whose parameter Q equals 12. As expected, the VSWS curve has lower peaks and milder ascending slopes. This tends to be the case also for the depressions, where the memory occupancy of VSWS is often higher than that of SWS, and for some of the descending slopes, which VSWS often follows with some delay with respect to SWS.

The behaviors of the same portion of the APL trace under the three policies to be compared in our experiments, but with a different value of T (the one given by VSWS with $Q = 7$), are displayed in Fig.5. VSWS seems in most cases even more effective than DWS in clipping the memory occupancy waveform, but is often slower in reducing the resident set size, and tends to keep it at slightly higher values during the depressions, when page faults are rare. This phenomenon is more clearly visible in Fig.6, where the portion of the WATEX trace between reference 420,000 and reference 1,020,000 is used to compare DWS with VSWS for $Q = 5$ ($T = 13,138$ references).

To substantiate claim (d), the numbers s of process suspensions required by VSWS and by SWS have been measured in all experiments. In Fig.7, we have plotted s vs. T for the two policies and the WATFIV trace. The difference between the two curves is still substantial (i.e., a factor of 2) at $T = 10,000$, which, for WATFIV, corresponds to $Q = 7$.

If we take all of our results into account, the most convenient range of values for Q seems to be the one between 5 and 8 for all the traces. Even though the problem of determining the sensitivity of the policy's performance to the value of Q has not been directly addressed in our experiments, this observation suggests that in most installations a value for Q can probably be found such that most of the processes in the workload will run with nearly-optimum static and dynamic performance, and a number of suspensions substantially smaller than under SWS.

4. Conclusions

A new local variable size memory management policy, the variable-interval sampled working set (VSWS) policy, has been presented. VSWS is a working-set-like algorithm that, in the experiments we have performed, exhibited a static performance, as measured by all the commonly used indices, very close to that of the SWS policy, and a dynamic performance noticeably better than that of SWS. Its implementation requires the type of hardware support needed by SWS and PFF, which is substantially simpler than that required by WS or by DWS in their original versions. Another advantage of VSWS over SWS is its generally lower number of process suspensions for use bit scanning purposes, due to the fact that most of the scannings take place at page fault times; in the limit, the number of such suspensions under VSWS could approach that of PFF, which is zero or (if a bound on the duration of a faultless interval is added as a second parameter to PFF) very small. Having three parameters instead of one is a drawback of VSWS, but two of the parameters are actually sampling frequency bounds and should be expected to be modified much less frequently (especially M) than the third. Furthermore, our limited experimental investigations seem to suggest that it may be possible to choose a value for the third parameter, Q, which will make the performances of most processes close to their optimum values. In other words, if further, more extensive experimentation will confirm the validity of these observations, the controllability of VSWS may be found to be closer to that of WS than to that of PFF.

Most of the performance comparisons between SWS and VSWS in Section 3 have been made for equal values of T, that is, for equal total sampling overheads. Since the values of the static performance indices produced by the two policies are quite close (see Figs.2 and 3), their actual total overheads (as measured, for example, by the respective numbers of times the use bits are scanned during the execution of a process) should be expected to be roughly the same if the performance levels achieved are about the same. This ovehead is substantially lower than that caused by the PFF policy, which scans all the use bits at every page fault, at least in order to reset them, if not also to determine the pages to be expelled from the resident set. Thus, SWS and VSWS are the local variable-size policies, among those known today, with the lowest overhead. The only existing working-set-like policy with a substantially lower overhead is WSClock [2], which, however, is a global policy.

Due to the increasing memory demands in a number of important applications, and to the technology-driven trend toward smaller page sizes, the use bit scanning overhead is bound to become a major characteristic of those memory policies which base their estimation of the current locality on the information contained in the use bits. Schemes for reducing the overhead of VSWS or SWS will therefore have to be studied and experimented with before local variable-size policies become really viable alternatives to the global policies adopted in most current virtual memory systems.

Acknowledgements

The authors are grateful to Peter Denning for an enlightening conversation on the subject of this paper and to the members of the PROGRES group for the several improvements that resulted from many long and heated discussions on VSWS.

References

[1] O.Babaoglu, Virtual Storage Management in the Absence of Reference Bits, Ph.D. dissertation, University of California, Berkeley (November 1981); also, PROGRES Report No.81.8, Computer Science Division, University of California, Berkeley (December 1981).

[2] R.W.Carr and J.L.Hennessy, WSClock - A simple and effective algorithm for virtual memory management, Proc. ACM-SIGOPS 8th Symp. on Operating Systems Principles (December 1981), 87-95.

[3] W.W.Chu and H.Opderbeck, The page fault frequency replacement algorithm, AFIPS Conf. Proc. 41 (FJCC 1972), 597-609.

[4] P.J.Denning, Memory Allocation in Multiprogrammed Computer Systems, MIT Project MAC, Computation Structures Group Memo 24 (March 1966).

[5] P.J.Denning, The working set model for program behavior, Comm. ACM 11 (May 1968), 323-333.

[6] P.J.Denning, Thrashing: Its causes and prevention, AFIPS Conf. Proc. 33 (FJCC 1968), 915-922.

[7] P.J.Denning, Working sets past and present, IEEE Trans. Software Engineering SE-6,1 (January 1980), 64-84.

[8] M.A.Franklin, G.S.Graham, and R.K.Gupta, Anomalies with variable partition paging algorithms, Comm. ACM 21 (March 1978), 232-236.

[9] G.S.Graham, A Study of Program and Memory Policy Behavior, Ph.D. dissertation, Purdue University (December 1976).

[10] G.S.Graham and P.J.Denning, On the Relative Controllability of Memory Policies, in: K.M.Chandy and M.Reiser, eds., Computer Performance (North-Holland Publ. Comp., Amsterdam, 1977), 411-428.

[11] R.K.Gupta and M.A.Franklin, Working set and page fault frequency replacement algorithms: A performance comparison, IEEE Trans. Computers C-27 (Au-

gust 1978), 706-712.

[12] K.C.Kahn, Program Behavior and Load Dependent System Performance, Ph.D. dissertation, Purdue University (August 1976).

[13] R.L.Mattson, J.Gecsei, D.R.Slutz, and I.L.Traiger, Evaluation techniques for storage hierarchies, IBM Sys. J. 9 (1970), 78-117.

[14] J.B.Morris, Demand paging through the use of working sets on the MANIAC II, Comm. ACM 15 (October 1972), 867-872.

[15] J.-F.Paris, Personal communication (February 1981).

[16] B.G.Prieve, A Page Partition Replacement Algorithm, Ph.D. dissertation, University of California, Berkeley (1974).

[17] J.Rodriguez-Rosell and J.P.Dupuy, The design, implementation, and evaluation of a working set dispatcher, Comm. ACM 16 (April 1973), 247-253.

[18] A.J.Smith, A modified working set paging algorithm, IEEE Trans. Computers C-25 (September 1976), 907-914.

[19] J.R.Spirn, Program Locality and Dynamic Memory Management, Ph.D. dissertation, Princeton University (March 1973).

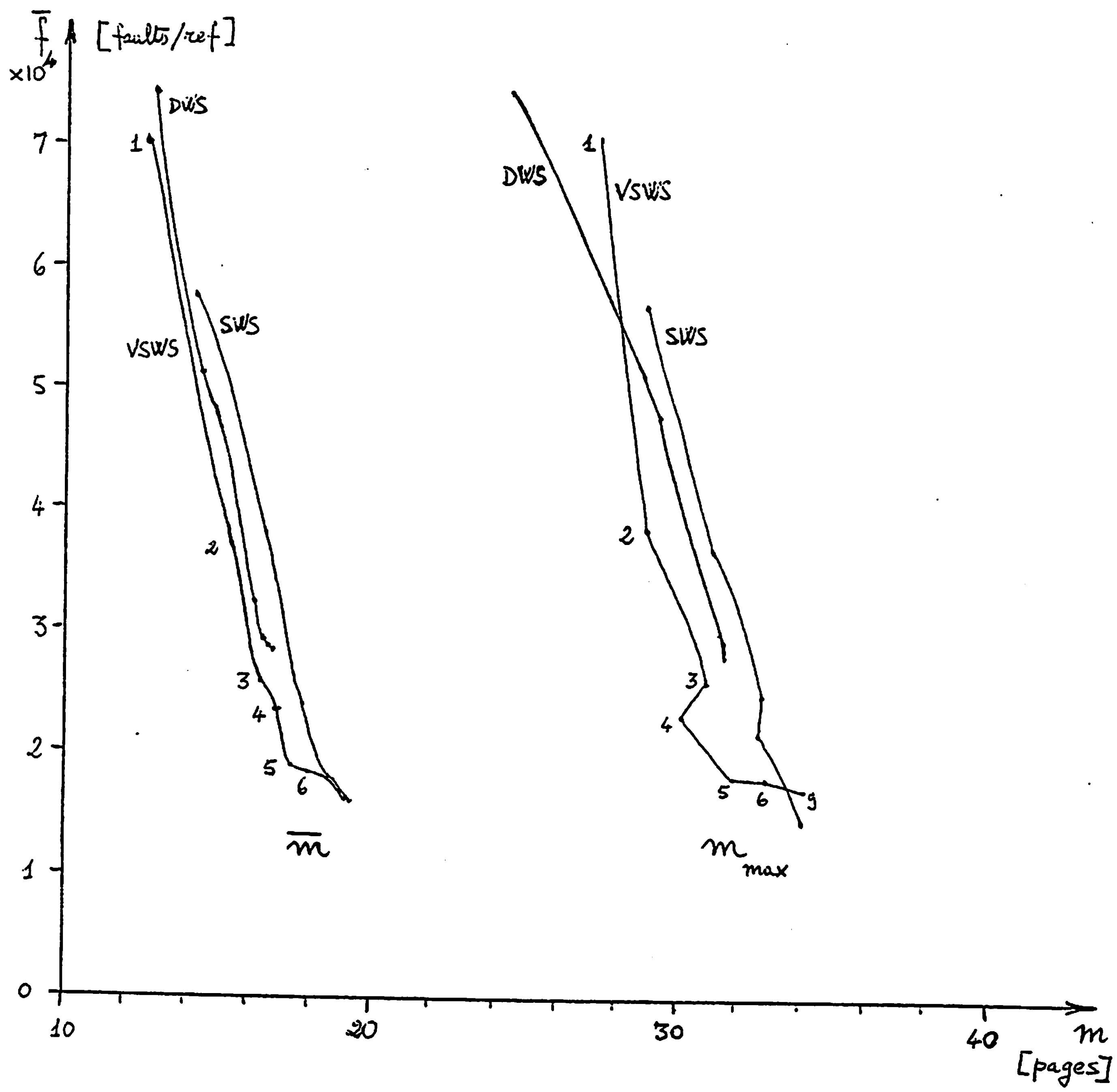

Fig.1. Mean fault rate ($\bar{f}$) vs. mean ($\bar{m}$) and maximum (m_{max}) memory occupancy for WATEX under SWS, DWS, and VSWS.

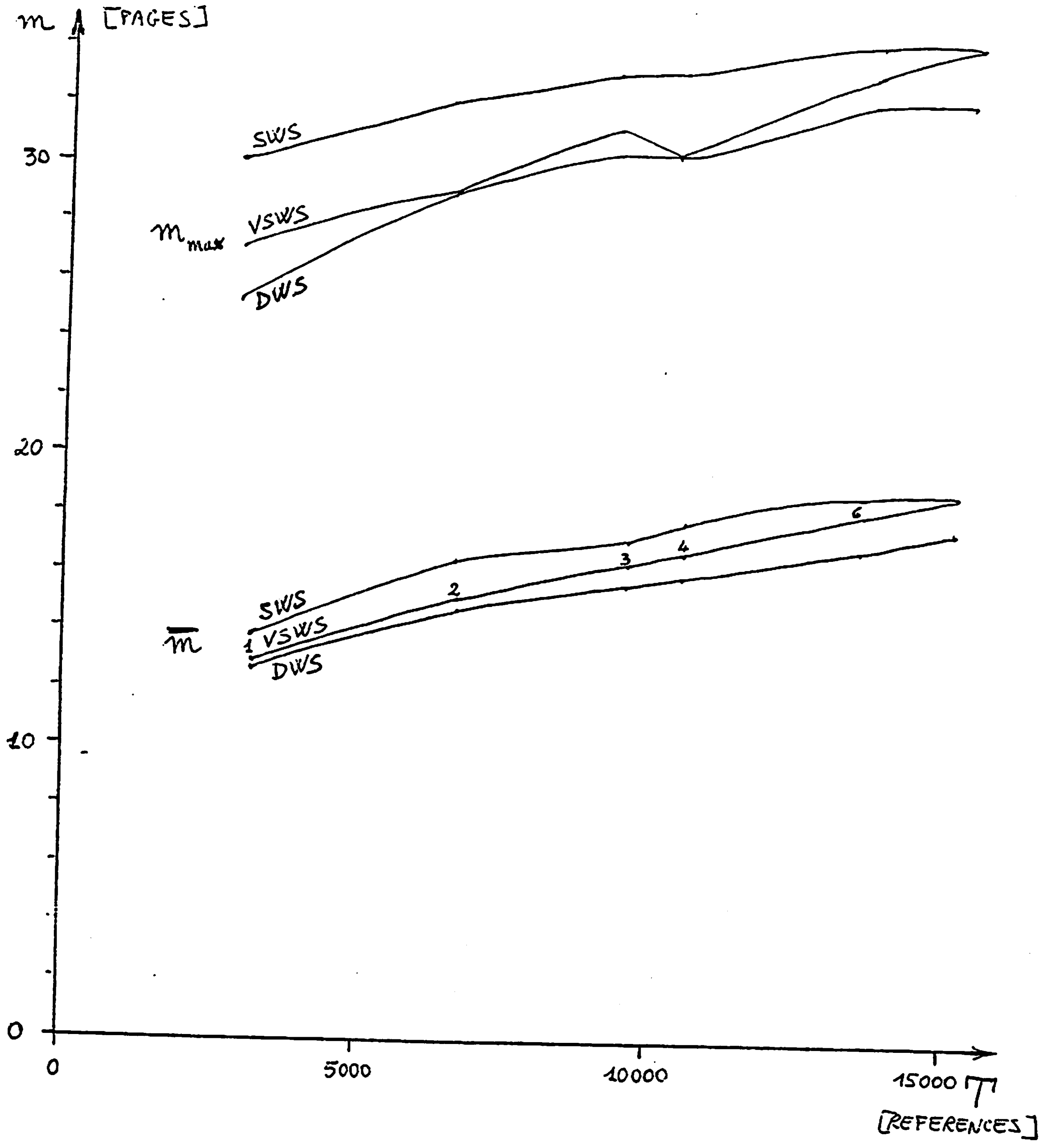

Fig.2. Mean ($\overline{m}$) and maximum (m_{max}) memory occupancy vs. T (the sampling interval and window size for SWS, the window size for DWS, the mean sampling interval and mean window size for VSWS) for WATEX.

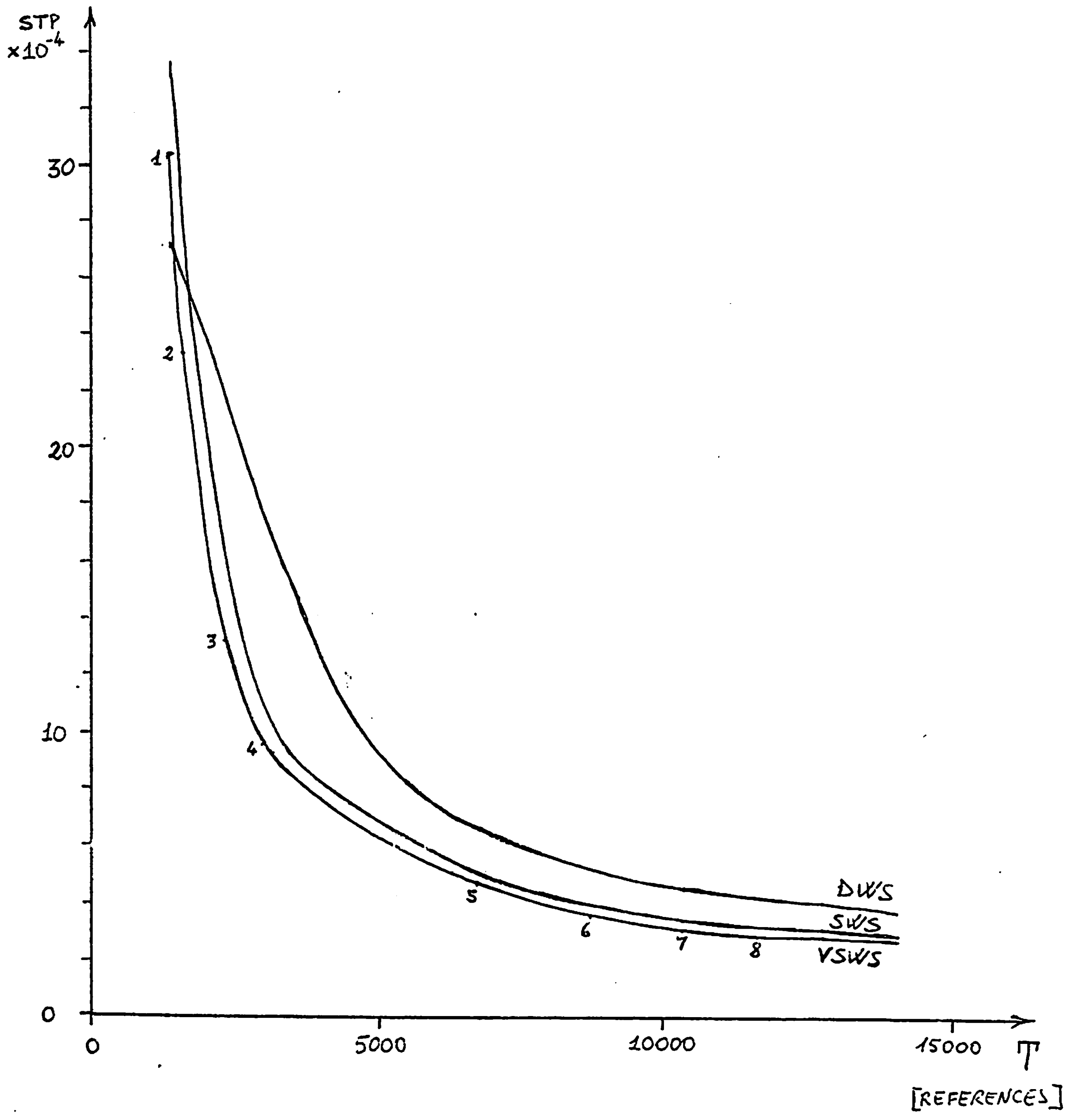

Fig.3. Space-time product vs. *T* for WATFIV under SWS, DWS, and VSWS.

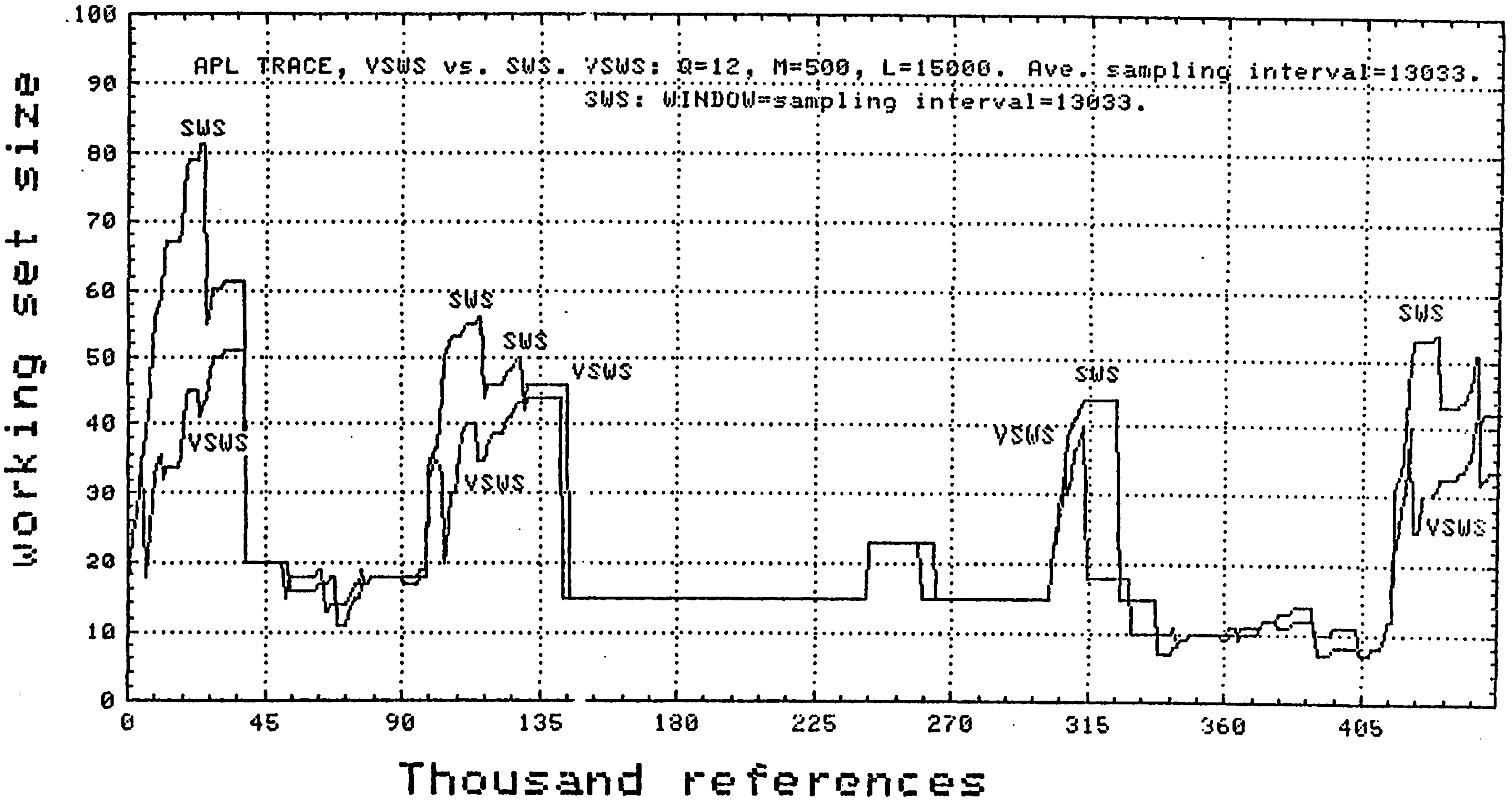

Fig.4. Memory occupancy vs. virtual time for APL (first 450,000 references) under SWS with $T = 13,033$ and VSWS with $Q = 12$.

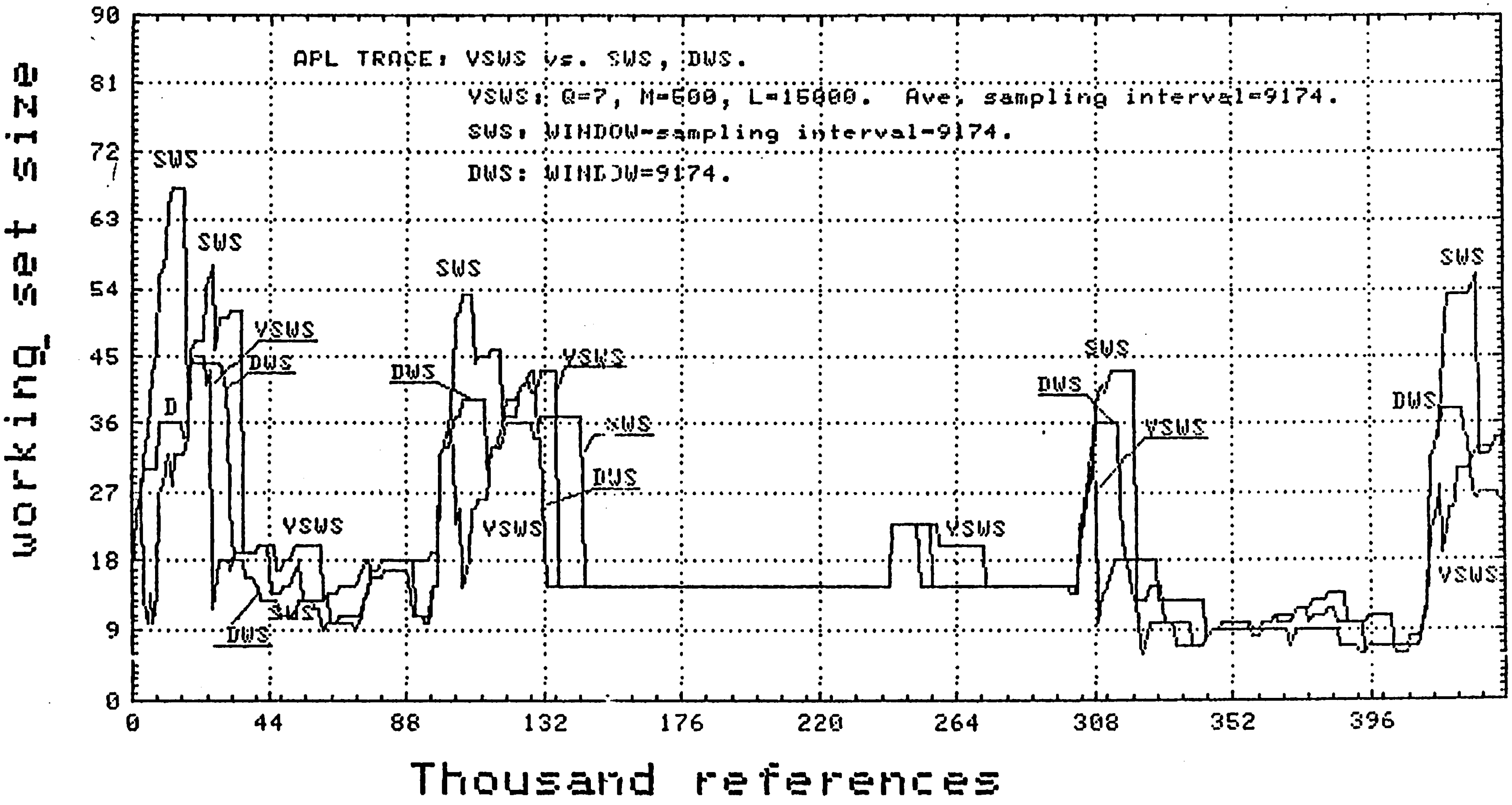

Fig.5. Memory occupancy vs. virtual time for APL (first 440,000 references) under SWS and DWS with $T = 9174$, and VSWS with $Q = 7$.

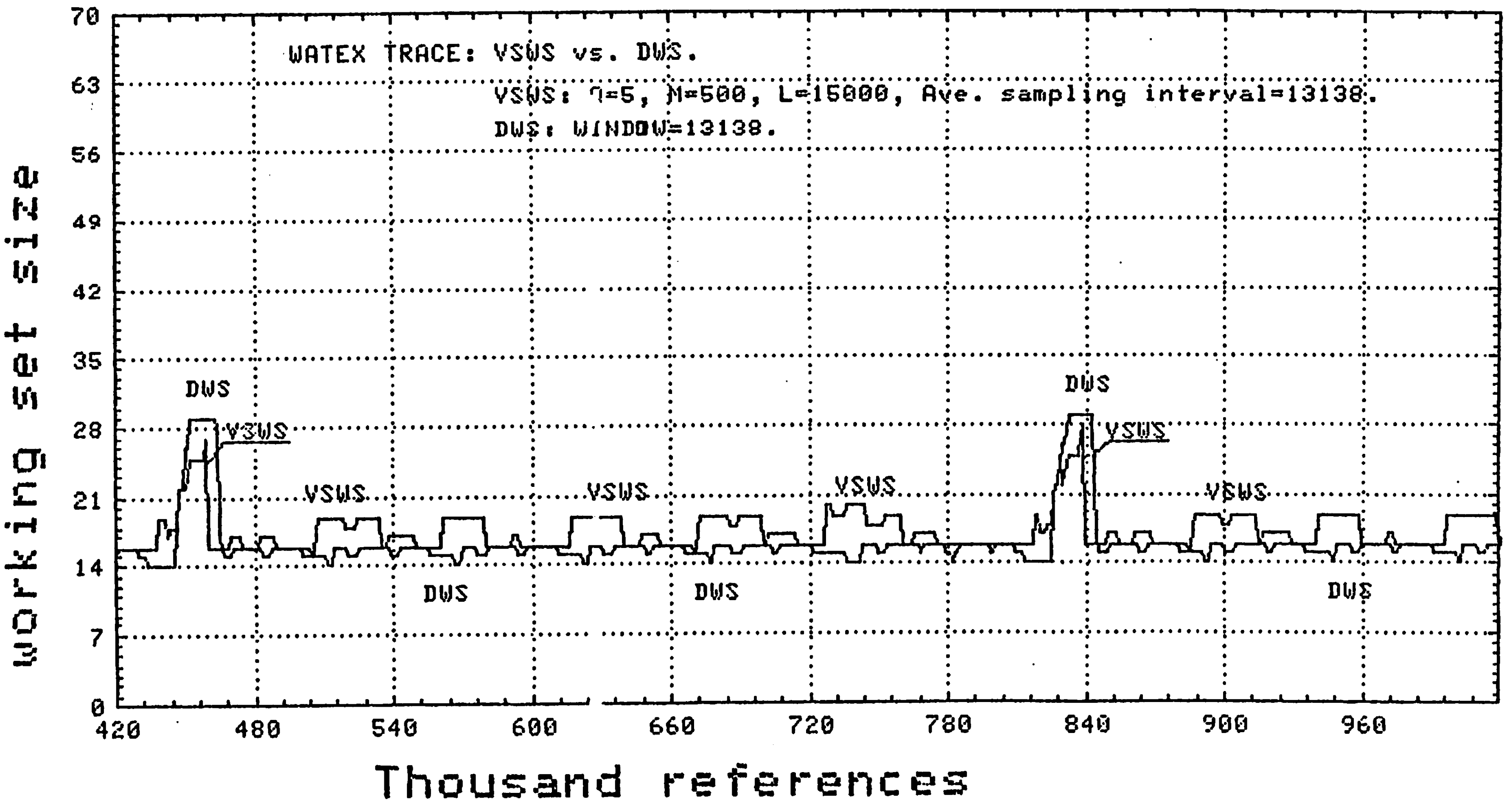

Fig.6. Memory occupancy vs. virtual time for WATEX (references between 420,000 and 1,020,000) under DWS with $T = 13,138$ and VSWS with $Q = 5$.

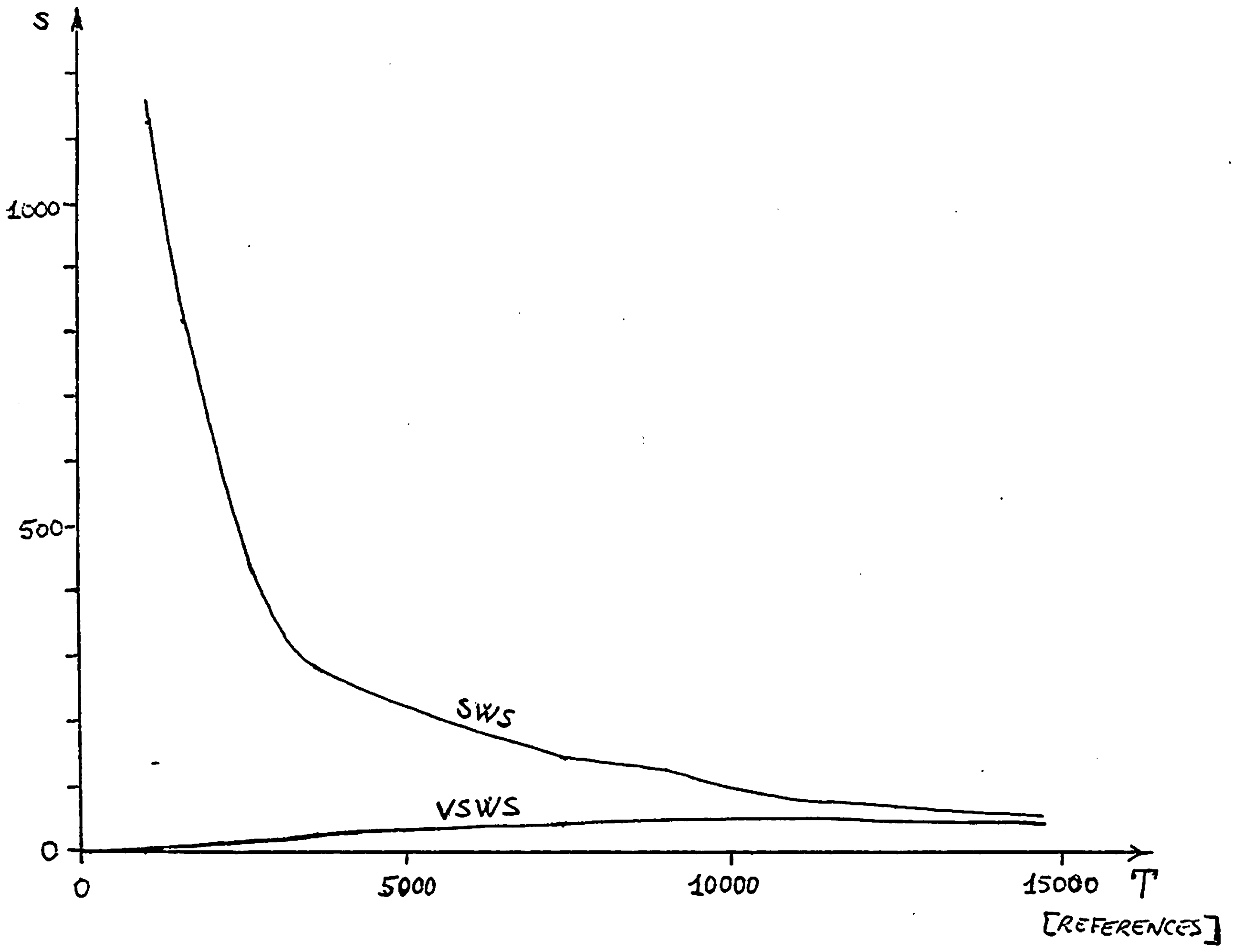

Fig.7. Number of process suspensions (*s*) vs. *T* for WATFIV under SWS and VSWS.