P. A. Franaszek
B. T. Bennett

Adaptive Variation of the Transfer Unit in a Storage Hierarchy

Abstract: Consider a paged storage hierarchy with at least two levels L_1 and L_2, where L_1 denotes main storage and L_2 secondary storage. Suppose that the unit of replacement for L_1 is a single page, and that the L_2-to-L_1 transfer unit, given a page fault, is an integer number of pages. Then, given a suitable replacement policy for L_1, increasing the unit of transfer often results in a lower miss ratio at the expense of increased paging traffic. This paper explores the possibility of adaptively varying the L_2-to-L_1 transfer unit as a function of the reference history of the data to be fetched. Experiments on traces drawn from two large data base systems suggest that such adaptation can result in improved tradeoffs between miss ratios and paging traffic.

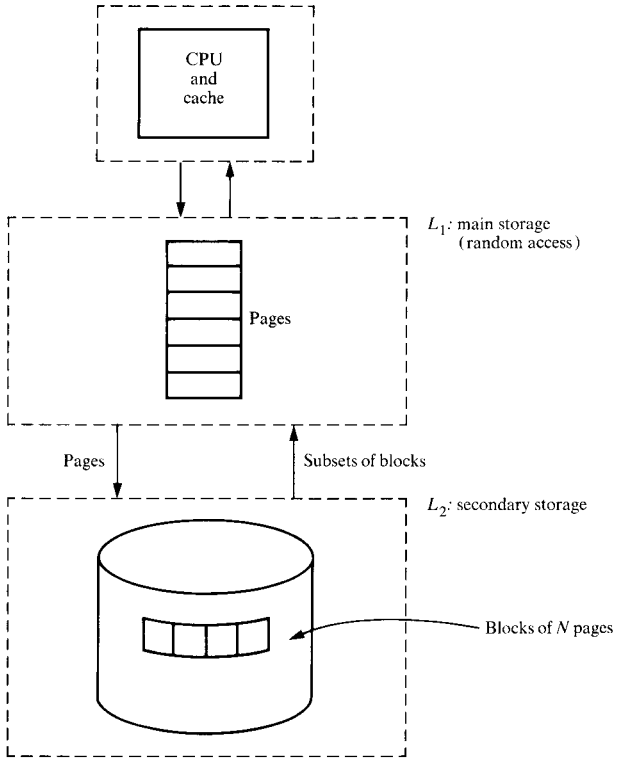
I. Introduction

The question of alternative methods for managing data in a computing system is assuming greater importance because of the increasing emphasis on data base applications. One proposed method is to handle data by means of paging, analogously to the way programs are managed in virtual memory systems. Such paged storage hierarchies might include an automatic mechanism for controlling data transfers and placements to suit patterns of usage. This paper investigates problems related to transfers of data between main and secondary storage. Specifically, the notion is introduced of varying the unit of transfer as a function of the reference history of the data in question.

Figure 1 illustrates a storage hierarchy which for simplicity has only two levels: L_1 and L_2. Level L_1 represents main or random access storage; L_2 is the backing store, implemented by direct access devices such as drums or disks; and L_1 is allocated in units termed *page frames*, each of which can hold one *page*. It is assumed that each page has an L_2 *home address*, a location in L_2 where a copy resides, and that the set of pages is partitioned into *blocks* of N pages, where the home addresses of the pages in each block are contiguous.

Included in the set of policies for managing L_1 is the *fetching policy*, which determines which pages are transferred into L_1 and the times of such transfers. An example of a class of fetching policies is *demand paging*. Here

Figure 1 A two-level storage hierarchy. L_1: main storage (random access); L_2: secondary storage.



fetches are initiated only by faults, and each fetch includes only the faulted page. This paper treats a class of policies termed *demand blocking* or *block prefetching*. As with demand paging, fetches are initiated only on faults. Each fetch, however, may cause the transfer into L_1 of not just the faulted page, but any subset of pages from the block containing it. Since a block of pages is stored contiguously on L_2, fetching more than one page causes few additional device delays. If the grouping of pages into blocks reflects the likely reference patterns, such prefetching may save an appreciable number of faults. The number of pages fetched from a block given a fault will be termed the L_2-to-L_1 *transfer unit* or *TU*. Main storage is managed on a page basis. Thus if a block of N pages is fetched, and N pages are to be removed from L_1, the latter may be members of more than one block.

It is well known that block prefetching may result in a significant reduction in the fault rate [1–3]. However, a large transfer unit incurs a number of costs, which include

1. An increase in channel traffic. This results in greater memory interference between the channels and central processing units, as well as longer queues for I/O.
2. Wastage of storage space due to the transfer of pages that will not be referenced.
3. Greater processing overhead in the L_1 page replacement algorithm.

A suitable fetching policy thus involves a number of tradeoffs. These, however, are a function of the system load and data usage patterns, which in general are time-varying and/or unknown. This suggests the use of an adaptive policy that changes as a function of a number of observed parameters.

One approach to such adaptation is to concentrate on individual process behavior. For example, occurrences of sequential page references in a given process could be monitored and the units of transfer adjusted accordingly. A potential disadvantage of this method is that if the number of processes is large, as is typically the case in a data base system, monitoring the reference patterns for each process could result in a substantial computational overhead. Moreover, the presence of shared data might require that all references be monitored, not just the faults.

This paper introduces a second alternative: adaptation based on observations of how the data in question are referenced, as discussed briefly in Ref. 4. Note that this alternative entails an assumption about reference patterns. Programs in execution might be expected to have fairly regular patterns of reference, at least sufficiently so for adaptive prefetching to be advantageous. However, it is not clear whether patterns of references to data bases are sufficiently regular to be exploitable. Thus much of the emphasis of this investigation is on experimental results and methodology. Also included is some discussion of expected performance benefits and approaches to implementation.

Section II describes a class of fetching policies. These give the option of fetching either the faulted page or all missing pages from a block. Section III gives an approximate analysis of the prefetching process which explores the relation between gains associated with prefetching and the average number of prefetched pages that are referenced. Section IV presents simple heuristics for varying the transfer unit size based on estimates of the ratio of prefetched pages referenced to overall traffic. This involves approximations, required as the result of the nonobservability of relevant variables when only a faulted page is fetched. Experiments on traces drawn from two large data base systems are described in Section V.

The results indicate that block prefetching yields a lower miss ratio than does demand paging, even when the page size for the latter policy is tailored to the application. Adaptive variation of the transfer unit size substantially lowers the paging traffic required to maintain a given miss ratio in comparison with a fixed policy of block prefetching. Reference patterns in the two data bases thus appear to be sufficiently stable as to be exploitable by learning procedures. Section VI summarizes the results.

II. Main memory management and the fetching policy

Systems with demand paging policies generally have page replacement algorithms that favor for retention in L_1 those pages most recently referenced [1]. The least-recently-used (LRU) algorithm is a well known idealization [1]. Pages managed by LRU may be viewed as occupying a stack, with the most recently referenced page on the top and the least recently referenced on the bottom. A page occupying an intermediate position is, if referenced, moved to the top, and those which had been above it are moved down by one position. Replacements are made from the bottom of the stack.

Suppose main memory management incorporates an LRU replacement policy. Consider the case where the unit of transfer, given a page fault, consists of several pages. These will include, as well as the faulted page, prefetched pages which have not been referenced since their last residence in main memory. How should these *unreferenced pages* be treated on arrival in L_1? One way would be to determine the average probability of reference to such pages, and to insert them into a corresponding level of the LRU stack. In general, however, these pages may have a reference probability whose time dependency is substantially different from that of those pages which have been referenced during their current sojourn in L_1. This suggests that prefetched (unreferenced) pages be treated separately. This is the approach taken here and in several studies of nonadaptive prefetching [1, 2].

As shown in Fig. 2, main storage is partitioned into two sections Q_1 and Q_2. Section Q_1 holds referenced pages, and is managed by a replacement algorithm such as LRU. Section Q_2 is a first-in-first-out (FIFO) stack, holding unreferenced pages.

It is convenient to introduce notation in which B_j, where $j = 1, 2, \cdots$, is the set of blocks in the storage hierarchy. Each B_j contains a set of pages P_{ij}, where $i = 1, 2, \cdots, N$.

Consider a set of pages belonging to B_j, which are transferred to L_1 as the result of a fault to page P_{ij}; P_{ij} is immediately placed in Q_1. The other (prefetched) pages are placed in the Q_2 FIFO stack. A prefetched page referenced before deletion from Q_2 is transferred to Q_1. For simplicity, it is assumed that Q_1 and Q_2 are each allotted a fixed percentage of the L_1 page frames. Replacements are from Q_2 unless Q_1 exceeds its allocation.

As mentioned above, given a page fault, the fetching policy determines the unit of transfer. The policy may include a set of parameters, the setting of which is performed by another mechanism. The policy to be considered is

A1: Associated with each block B_j is a *transfer number* $TN(j)$, which determines, given a fault to a page P_{ij}, whether just this page or all missing pages from this block should be fetched.

An implementation of A1 may be based on a system directory with entries in the following format:

Block ID	*Block Address*	*TN*

The transfer number may be regarded as a summary of the reference statistics for a block. The form of the *TN* is a function of the specific implementation. The fetching policy may be either fixed or adaptive. In the adaptive case, the block transfer numbers are modified dynamically as a function of observed page references.

III. Analysis

Let $R = \{r_1, r_2, \cdots\}$ denote the reference string, where r_m represents a reference to P_{ij} at time t_m. Consider a fault to page $P_{ij} \in B_j$ at time t_m. Let $B_j^d(m)$ denote those pages in B_j absent from L_1 at t_m, and let $B_j^c(m)$ be $B_j^d(m)$ minus the faulted page. For simplicity, it will be assumed that all pages transferred to Q_2 at a time t_m will be removed simultaneously at some time $(t_m + \Delta_m)$ if unreferenced. Let ψ denote the expected number of pages from $B_j^c(m)$ referenced in the interval $(t_m, t_m + \Delta_m)$. Suppose that the fetching policy A1 is stationary with respect to B_j in the interval $(t_m, t_m + \Delta_m)$. That is, either (a) $B_j^d(m)$ is fetched as the result of each fault to B_j in $(t_m, t_m + \Delta_m)$ or (b) there is a policy of demand paging with respect to the block B_j in the interval $(t_m, t_m + \Delta_m)$. Fetching $B_j^c(m)$ will then reduce the expected number of page faults by ψ.

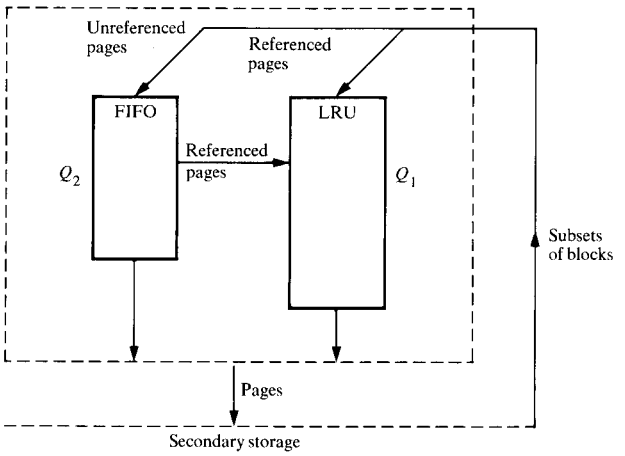


Figure 2 Memory management for demand blocking.

Fetching $B_j^c(m)$ will push down the pages in the FIFO stack, reducing their potential residence times. Let V be the expected reduction due to prefetching $B_j^c(m)$ in the number of pages resident in Q_2 that are referenced before removal. The value V provides an upper bound to the increased number of faults to pages other than $B_j^c(m)$, since an ejected page may be prefetched before the fault, or if faulted may result in the fetching of a page that would otherwise be faulted.

Let C_f and C_p denote respectively the cost of a page fault and the cost of prefetching a page. The above suggests that $B_j^c(m)$ should be fetched when

$$C_p < \frac{C_f(\psi - V)}{N[B_j^c(m)]}, \tag{1}$$

where $N[B_j^c(m)]$ is the number of pages in $B_j^c(m)$. Note that as $\psi/N[B_j^c(m)]$ increases, $V/N[B_j^c(m)]$ tends to decrease, since the sooner a prefetched page is referenced, the less effect it has on the pages below it in the FIFO stack. In other words, $(\psi - V)/N[B_j^c(m)]$ tends to grow monotonically with $\psi/N[B_j^c(m)]$. One might then conclude that a fetch is desirable when

$$R_j = \frac{\psi}{N[B_j^c(m)]} \tag{2}$$

is sufficiently large. The value R_j is simply the expected fraction of prefetched pages which are actually referenced. The fetching policies considered here operate on the principle of fetching all blocks B_j whose estimated R_j is greater than some threshold value.

It is interesting to consider what happens if the stationary assumption does not hold. Let $P_{kj} \in B_j^c(m)$ be a page that is actually referenced in $(t_m, t_m + \Delta_m)$. Prefetching this page at time t_m may not save a fault because

1. Another fault at time $t_n > t_m$ may occur, resulting in the prefetching of P_{kj}.
2. A fault to P_{kj}, if it occurs, may cause the fetching of some other page that would have been faulted otherwise.

Moreover, prefetching a page P_{kj} may cause an additional fault, since

3. If P_{kj} is not fetched at time t_m, and not referenced in time $(t_m, t_m + \Delta_m)$, a reference to it at time $t_n > t_m + \Delta_m$ may cause a fault. If, however, P_{kj} had not been prefetched at t_m, it might have been prefetched at a time t_r, such that $t_r < t_n < t_r + \Delta_r$.

Observations 1-3 illustrate features of demand blocking that make it difficult to obtain optimal algorithms analogous to Belady's MIN policy [5] for demand paging.

The primary potential advantage of adaptive prefetching in the system considered here is a lower paging rate required to obtain a given miss ratio as compared to a fixed policy of block prefetching. It can be expected that there is little potential for lowering the miss ratio. To see why this is so, consider a fixed prefetching policy with a block size of N pages and suppose N is such as to yield the minimum number of page faults for a given Q_1 and Q_2. If adaptive policy A1 is adopted and is successful, the result is to lower the number of useless pages transferred. Suppose that the overall traffic is reduced by a factor of two. To a first degree of approximation, this implies that Q_2 may be halved with little effect on the page fault rate. For a fixed memory size, this means that Q_1 may be increased correspondingly. However, since Q_2 comprises (as shown subsequently) on the order of ten to twenty percent of L_1, the resulting effect on the page fault rate is small.

IV. Heuristics

This section discusses methods for varying the unit of transfer associated with each block B_j through estimation of R_j, the expected ratio of referenced to prefetched pages that would be expected if a fixed policy of block prefetching were in effect. Note that, given such a policy, a simple estimator of R_j is the fraction of prefetched pages from B_j that are referenced before removal from the FIFO stack. Since the actual policy is adaptive, this ratio, if it is to be used, must first be estimated. The approach taken below is to perform this estimation by means of an approximate simulation of the effects of a fixed prefetching policy. Let $\bar{R}_j$ denote the estimated ratio.

Let $\{V_i\}$, $i = 1, 2, \cdots$, be the sequence of references to pages not contained in Q_1. The contents of Q_1 are independent of the prefetching policy, so that the sequence of faults under either fixed or adaptive block prefetching will be a subsequence of $\{V_i\}$. Let $\{V'_i\}$ denote that subsequence which corresponds to the faults indicated by the simulation of fixed block prefetching. These will be termed the *simulated faults*. Let F'_i represent the number of such faults up to but not including the time of V'_i.

Two methods of simulation were tried.

- *Method 1*

Associated with the main memory directory for each block B_j is a number D_j, which is equal to $F_{i(j)}$, the number of simulated faults up to but not including the time $t_{i(j)}$ of the most recent simulated fault to B_j, $V'_{i(j)}$. If no pages from B_j are in Q_1, then let $D_j = -\infty$.

Suppose V'_i, $i > i(j)$, is a reference to B_j; then V'_i is a simulated fault if

$$(F_i - D_j) \geq \left(\frac{M_2}{N - \beta - 1}\right), \tag{3}$$

where

M_2 represents the number of page frames allocated to Q_2.

N is the number of pages per block.

$(N - \beta)$ represents the average simulated number of pages transferred to L_1 as the result of a fault. This includes the faulted as well as the prefetched pages.

Note that transfers between Q_2 and Q_1 are ignored. This tends to result in an underestimation of the Q_2 residence times for prefetched pages that are not referenced before removal, leading to overestimation of the number of faults and thus to an underestimation of the value of the $\{R_j\}$. However, this effect is not expected to be large. Only a minority of the $\{V_i\}$ represent references to pages resident in Q_2, and each fault generally results in the transfer of several pages. Thus the residence times are largely determined by the number of faults and the average transfer unit size.

- *Method 2*

A reference V_k represents a simulated fault if and only if it is to a page P_{ij} whose block B_j has no contents in Q_1.

This approach resulted from the observation that, under a fixed policy of block prefetching, the average number of pages transferred due to a fault is close to N, the block size, so that most faults are to blocks with no pages in Q_1. The advantages are elimination of the parameters $\{D_j\}$ from the directory and the computation associated with [2]. The disadvantage appears to be a substantially greater error than with Method 1 in estimating the R_j. Suppose the residence time for a page Q_1 is, as expected, always greater than for a page in Q_2. Then if V_k is a reference to a block B_j with no pages in Q_1, V_k would be an actual fault. However, the presence of some pages from B_j in Q_1 does not necessarily imply that the remainder are in Q_2. Thus Method 2 tends to underestimate the number of faults, and overestimate the $\{R_j\}$.

The decision on the transfer unit size for B_j, given a fault to this block, is based on whether $\bar{R}_j$, the estimated value of R_j, is greater than a given threshold. Let this threshold be denoted by α. Information related to the current value of $\bar{R}_j$ is stored in the directory as the transfer number $TN(j)$. Storing the actual value of R_j is inconvenient, since updating requires knowledge of the number of faults. A simple representation for $TN(j)$ was chosen for the experiments, based on the observation that the average transfer unit size, if all missing pages are prefetched, is close to N.

Suppose α is such that

$$\alpha = \frac{X_1}{X_2(N-1)} \tag{4}$$

for integer values X_1, X_2. Let $TN(j)$ be as follows:

1. Initially $TN(j) = X_0$.

Each time a page P_{ij} is transferred to Q_1 (either from L_2 or Q_2), i.e., for each V_i,

2. If this transfer represents a simulated fault, $TN(j) \leftarrow TN(j) - X_1$;

3. Otherwise, $TN(j) \leftarrow TN(j) + X_2$.

If, given a fixed policy of block prefetching, the number of pages transferred due to a fault is always N, then $TN(j) \geq 0$ whenever $\bar{R}_j \geq \alpha$. In an actual system, it might be advantageous to introduce modifications that would ensure that the recent reference history of a block is weighted more heavily. A simple way of doing this is to introduce bounds for TN.

V. Experimental results

A number of experiments were performed on traces of references to two large data bases; the IBM Advanced Administrative System (AAS) data base [6] and one data base of an IBM IMS [7] system.

The AAS trace represents approximately three days of transactions and consists of 6×10^6 references to a data base of about two million 1693-byte physical records. For the purpose of these experiments, each physical record was taken to be one page. Blocks represent N consecutive physical records from a particular AAS file. That is, each file was partitioned into blocks of size N starting with records $1, N+1, 2N+1, \cdots$.

The reference string for IMS represents approximately one day of references to one data base and index of an IMS system. It consists of 520 000 references. The reference string was created by J. H. Mommens of IBM Research, Yorktown, from a trace of DL/1 calls and a map of the data base. It represents references to the data base

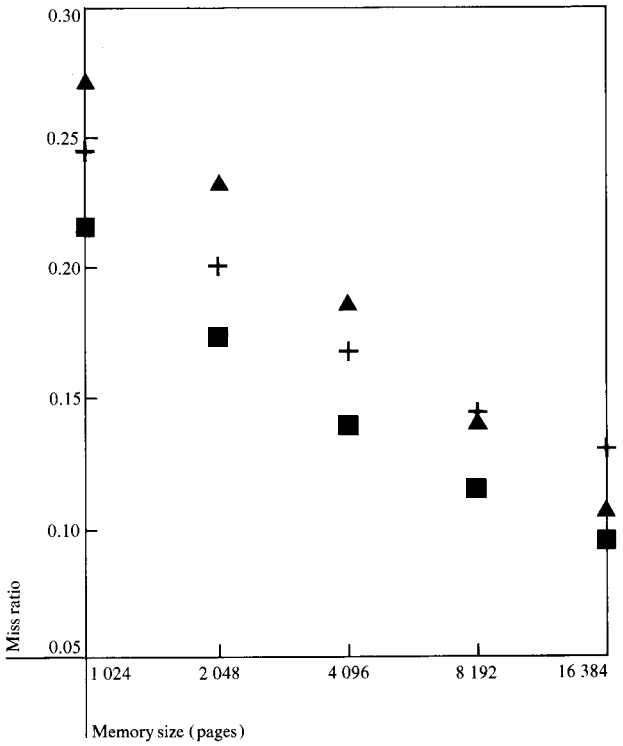
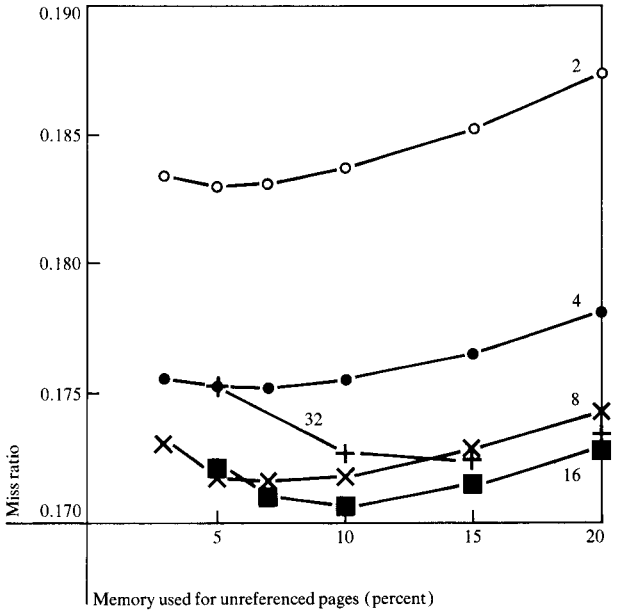


Figure 3 Miss ratios for various memory management policies for six million AAS references: ▲, LRU page replacement; +, LRU replacement of a block of eight pages; and ■, block prefetching a block of eight pages with 20 percent of memory reserved for unreferenced pages.

Figure 4 Miss ratios for block prefetching of six million AAS references with a 2048-page memory. Results for various block sizes (2, 4, 8, 16, 32) are presented for various percentages of memory allocated to the unreferenced pages.



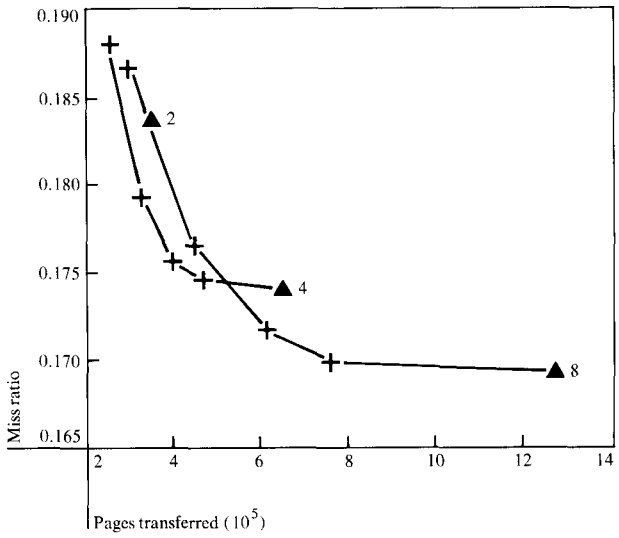


Figure 5 Miss ratio vs pages transferred comparison of prefetching (▲) and adaptive prefetching method 1 (+). Lines connect results for block sizes 4 and 8. Results are for the five million to six million reference interval in AAS. Memory size is 2048 pages with five percent reserved for unreferenced pages.

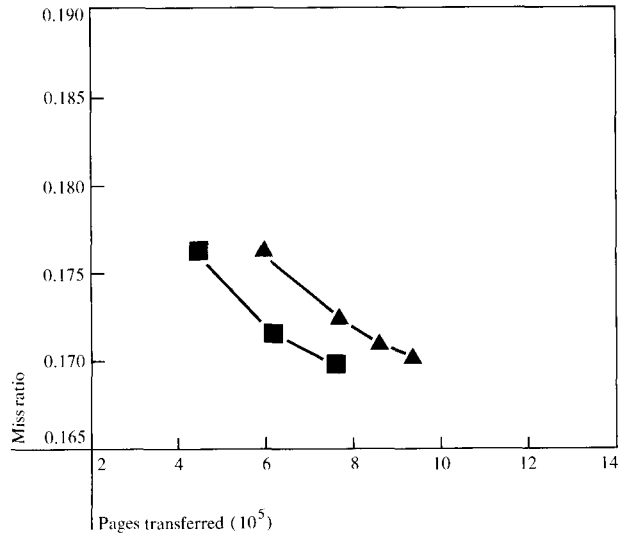


Figure 6 Miss ratio vs pages transferred for AAS references five million to six million for a memory size of 2048 pages, with five percent reserved for unreferenced pages and eight pages per block: ■, adaptive prefetching method 1; ▲, adaptive prefetching method 2.

mapped into a linear space. This linear space is regarded as being divided into pages. Blocks are sets of N consecutive pages.

Figures 3 to 6 represent the results of experiments on the AAS trace. Figure 3 presents miss ratios vs memory size for three management policies:

- Demand paging with LRU replacement.
- Demand paging with LRU replacement, but with a page size eight times that for the first policy.
- A fixed policy of block prefetching with $N = 8$. Q_1 is managed by LRU, and Q_2 by a FIFO policy. Q_2 was allotted 20 percent of L_1.

Figure 4 shows that the miss ratio for block prefetching is relatively insensitive to the percentage of main storage devoted to holding unreferenced pages, at least in the range of 5 to 15 percent. Significant improvements in the miss ratio may be obtained over demand paging by block prefetching with $N = 2$, and also by increasing the block size to $N = 4$ and $N = 8$. However, each doubling of the block size approximately doubles the number of page transfers. Thus increasing the block size involves a tradeoff between traffic and page fault rates.

Figure 5 illustrates the effect of adaptive prefetching with adaptation on a per-block basis. Miss ratios and traffic are given for a segment of 10^6 references after a learning period of 5×10^6 references. Method 1 (Section IV) was used to vary the block transfer numbers for a range of parameters (X_1, X_2) to generate miss ratio vs traffic curves for $N = 4$ and $N = 8$. The initial value for TN, X_0, was chosen so that a policy of block prefetching was in effect at the beginning of the simulation. It can be seen that adaption can result in a substantial reduction in traffic with little effect on the miss ratio. The results show the effect of continuous adaptation. Experiments were also tried using the first 5×10^6 references to set the block transfer numbers, which were then fixed for the remaining 10^6 references. The results were quite similar to those shown in Fig. 5, with only a slight increase in traffic and miss ratios in comparison with continuous adaptation.

Figure 6 compares the two methods for varying the transfer numbers described in Section IV. Method 1, which gives a more accurate simulation, yields better results.

Figures 7 to 9 represent results of experiments on the IMS trace. Figure 7 gives miss ratios vs memory size for three management policies:

- Demand paging with LRU replacement and a page size of 512 bytes.
- Demand paging with LRU replacement and a page size of 2048 bytes.
- A fixed policy of block prefetching with blocks of $N = 4$ 512-byte pages and 10 percent of main storage allotted to Q_2.

Figure 8 shows the effect of varying the percentage of main storage devoted to Q_2.

Figure 9 illustrates the effect of adaptive prefetching using policy A1 with adaptation on a per-block basis, with Method 1 used to vary the block transfer numbers. Re-

sults are for the 4×10^5 to 5×10^5 reference interval, allowing an initial learning period of 4×10^5 references. The curves were obtained by varying the parameters X_1 and X_2. As X_1 and X_2 are modified to obtain less traffic, there is an initial point with a substantial decrease in traffic and slightly lower miss ratio than fixed prefetching. Experiments using Method 2 rather than Method 1 obtain essentially the same results for this trace.

In observing the IMS results, a possible conjecture is that the adaptive algorithm selects data blocks for prefetching and disqualifies index blocks, and therefore that a data base administrator could do as well. However, half the blocks disqualified are index and half data (the ratio of index to data base blocks observed in the trace was 1:5). A policy of fixed demand paging for index pages coupled with block prefetching for data was simulated for the 4×10^5 to 5×10^5 reference interval. A miss ratio of 0.0356 was obtained, with approximately 15 000 page transfers. The adaptive method results in a similar number of page transfers and a 15 percent lower miss ratio.

VI. Discussion and conclusion

Experiments on traces drawn from the AAS data base and an IMS system produced lower fault rates for block prefetching than for demand paging, even with a page size tailored to the application. A possible drawback of block prefetching, i.e., that it might be difficult to implement a practical page replacement policy that guarantees a given percentage of allocated page frames to prefetched as distinct from referenced pages from a given data base, appears not to be a problem. The results indicate that prefetching performance is insensitive to the ratio of frames allocated to Q_1 vs Q_2. Thus a replacement policy that does not separate referenced from prefetched pages (as, for example, inserting prefetched pages into some intermediate level of the LRU stack) may yield similar results.

Increasing the transfer unit to more than a single page appears to offer substantial benefits in terms of lowering the L_1 miss ratio. However, doubling the transfer unit tends to almost double the number of page transfers. Adaptive prefetching appears to yield substantial traffic reductions compared to such a fixed policy, with little adverse effect on the page fault rate. The results were obtained with a policy that provides only the options of fetching one page or the entire block. An interesting question is what additional benefits could be obtained from a more flexible policy, such as one which would view each block as comprising a hierarchy of sub-blocks.

Adaptive prefetching requires some computational overhead, as well as additional space in the L_2 directories. However, computation is only required on transfers of pages to Q_1, at a time at which L_1 storage management is already invoked. Moreover, this overhead should be weighted against that required for computation associated with space allocation for pages that otherwise would be fetched. In the case where a dynamic variation of the transfer numbers is impractical, a possible alternative is to log the transfer of pages to Q_1, and to update the units

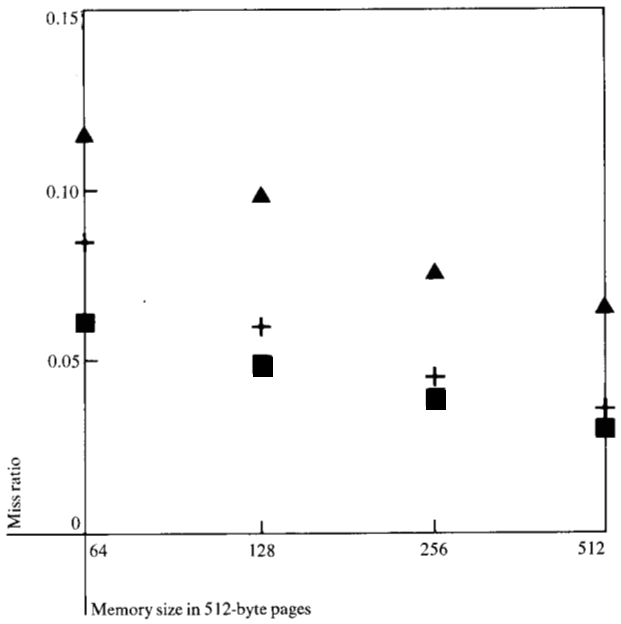


Figure 7 Miss ratios for various memory replacement policies for 520 000 IMS references: ▲, LRU 512-byte page replacement; +, LRU 2048-byte page replacement; and ■, block prefetch of four 512-byte pages with ten percent of memory used for unreferenced pages.

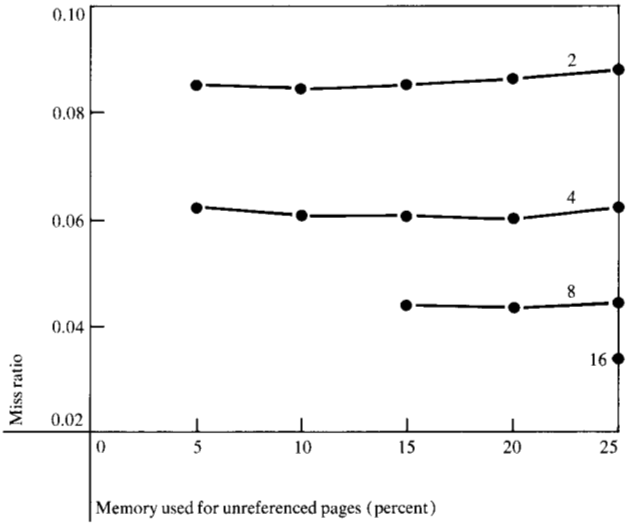


Figure 8 Miss ratio for 520 000 IMS references for block prefetching with various numbers of pages per block and percentages of memory reserved for unreferenced pages; page size: 512 bytes, memory size: 64 pages.

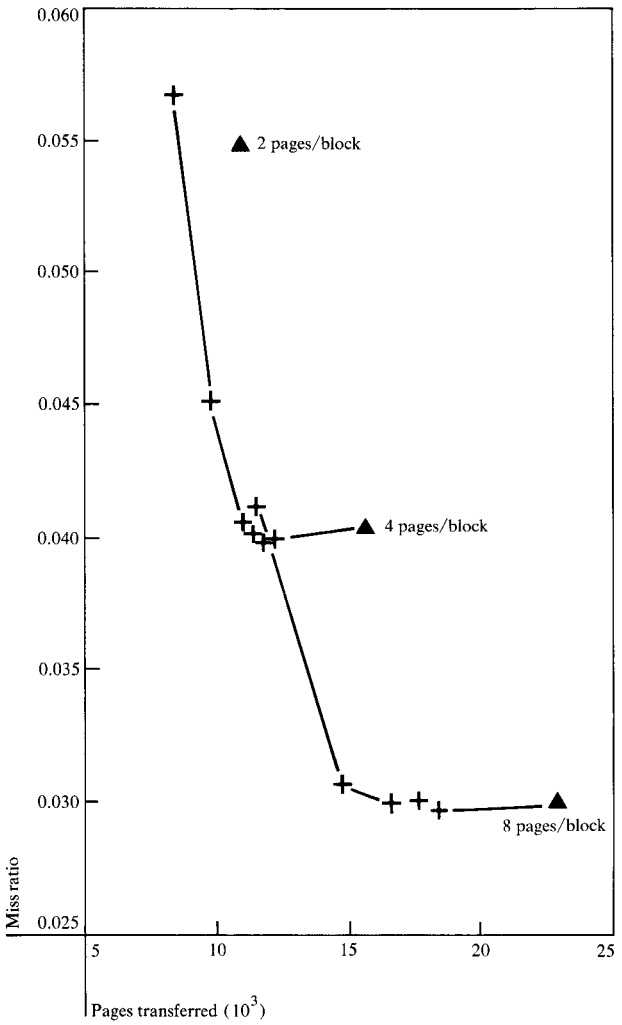


Figure 9 Miss ratio vs pages transferred comparison of prefetching (▲) and adaptive prefetching method 1 (+) for various block sizes. Results are for the 400 000–500 000 reference interval for IMS. Memory size: 64 pages, each of 512 bytes, with 15 percent reserved for unreferenced pages.

of transfer periodically. Experiments performed on the AAS trace suggest that the reference patterns are sufficiently stationary that this approach may also yield advantageous tradeoffs between traffic and miss ratios.

In summary, a number of performance issues associated with block prefetching were explored. Heuristics were constructed for dynamically varying the L_2-to-L_1 unit of transfer, and experimental results were obtained which show that this approach may yield useful benefits in the form of decreased traffic.

The overall question of whether block prefetching rather than demand paging should be chosen for a data base system with paged main storage involves a number of issues not considered in this paper. An example is the costs associated with maintaining L_2 home addresses. Consider a storage system where L_2 consists of several units of drum storage. The requirement that a page, when written out, must be placed in a specific location complicates the problem of I/O load balancing. It is not clear to what extent techniques such as random allocation of blocks to drums would mitigate this problem. This and other issues remain as areas for further investigation.

References

1. M. Joseph, "An Analysis of Paging and Program Behavior," *Comput. J.* **13,** 48 (1970).
2. B. T. Bennett and P. A. Franaszek, "Permutation Clustering: An Approach to On-Line Storage Reorganization," *IBM J. Res. Develop.* **21,** 528 (1977).
3. J. Rodriguez-Rosell, "Empirical Data Reference Behavior in Data Base Systems," *Computer* **9,** 9 (1976).
4. P. A. Franaszek, "Adaptive Transfer Unit Size Variation," *IBM Tech. Disclosure Bull.* **18,** 2348 (1975).
5. L. A. Belady, "A Study of Replacement Algorithms for Virtual Storage Computers," *IBM Syst. J.* **2,** 78 (1966).
6. J. H. Wimbrow, "A Large Scale Interactive Administrative System," *IBM Syst. J.* **4,** 260 (1971).
7. "IMS/VS General Information Manual," Order no. SH20-1260, IBM Corporation, White Plains, NY 10604.

Received December 16, 1976; revised Feb. 10, 1978

The authors are located at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598.