Two-Level Replacement Decisions in Paging Stores

ÖZALP BABAOĞLU, MEMBER, IEEE, AND DOMENICO FERRARI, SENIOR MEMBER, IEEE

***Abstract*—One of the primary motivations for implementing virtual memory is its ability to automatically manage a hierarchy of storage systems with different characteristics. The composite system behaves as if it were a single-level system having the more desirable characteristics of each of its constituent levels. In this paper we extend the virtual memory concept to within the top level of a two-level hierarchy. Here, the top level is thought of as containing two additional levels within it. This hierarchy is not a physical one, but rather an artificial one arising from the employment of two different replacement algorithms. Given two replacement algorithms, one of which has good performance but high implementation cost and the other poor performance but low implementation cost, we propose and analyze schemes that result in an overall algorithm having the performance characteristics of the former and the cost characteristics of the latter. We discuss the suitability of such schemes in the management of storage hierarchies that lack page reference bits.**

***Index Terms*—FIFO, LRU, Markov chain, page replacement algorithms, UNIX, VAX VMS, virtual memory, working set.**

I. Introduction

ECONOMIC realities force the construction of large virtual address spaces in the form of hierarchies. A typical two-level hierarchy consists of a fast, expensive, and small (the latter two being a consequence of the first attribute) primary memory in addition to a slow, cheap, and large secondary memory. Obtaining the appearance of a single level that is fast and large by the automatic management of the hierarchy is probably one of the primary motivations for incorporating virtual memory in a computer system [15], [17].

Every level except the last (i.e., the slowest) in a hierarchy adopts a replacement policy to remove data down to the next level when necessary. The topic of replacement algorithms that are suitable for such environments has been the subject of a large number of investigations [23]. Specifically, the virtual memory system we are concerned with employs *demand paging* [12]. The replacement algorithms that have been proposed for this environment can be crudely classified as being either inexpensive to implement but having poor performance, or being expensive to implement but having better performance. Our informal notion of cost of implementation considers factors such as hardware support other than the basic address translation mechanism and the complexity of the associated software. Our criterion for performance, on the other hand, is based on the frequency of page faults. Examples of replacement algorithms of the cheap-to-implement-but-poor-performance type include the First-In-First-Out (FIFO) and the Random (RAND) algorithms [7]. The Least Recently Used (LRU) [19] and the Working Set (WS) [11] algorithms, on the other hand, are examples of the expensive-to-implement-but-good-performance class.

In the next few sections, we introduce a class of *hybrid* replacement policies that combine two algorithms, one of each of the above categories, within a single level of the physical storage hierarchy. After deriving expressions for their performance, we demonstrate that these algorithms, in fact, achieve performances close to those of LRU and WS while having implementation costs comparable to those of FIFO and RAND. The next section introduces the program model on which our analyses will be based.

II. The Independent Reference Model

The mathematical analysis of a replacement algorithm requires a model of the programs on which the policy operates. For our purposes, an execution of a program consisting of n pages labeled $\{1, 2, \cdots, n\}$ results in a page reference string, $r_1, r_2, r_3, \cdots, r_{t-1}, r_t, r_{t+1}, \cdots$ where $r_i = i$ if page i is referenced at time instant t (memory references are assumed to occur at equidistant time points, and we define their distance to be the unit of time). We will assume a particularly simple stochastic structure for the reference string, known as the Independent Reference Model (IRM) [1]. As the name implies, the string $\{r_i; i = 1, 2, \cdots\}$ is assumed to be a sequence of independent identically distributed random variables from the population $\{1, 2, \cdots, n\}$ where $\Pr(r_t = i) = \beta_i$ for all t and $\sum_{i=1}^{n} \beta_i = 1$.

Modeling an actual program with the IRM involves obtaining point estimates for the model parameters (β_i's) that are simply the frequency counts of pages in an actual reference string generated by the program. Baskett and Rafii have proposed an alternate method for obtaining estimates for the β_i's, called the A_0 *Inversion Model* [6]. The resulting model, although structurally identical to the IRM, has much better

Manuscript received August 27, 1982; revised February 25, 1983, and June 3, 1983. This work was supported in part by the National Science Foundation under Grant MCS78-24618. The work of Ö. Babaoğlu was supported by the Italian National Research Council through a Fellowship for Foreign Scholars.

Ö. Babaoğlu was with the University of California, Berkeley, CA. He is now with the Department of Computer Science, Cornell University, Ithaca, NY 14853.

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

predictive capabilities for the performance of real programs.

III. Hybrid Policies

In a demand-paged virtual memory system, referencing a page that is *invalid*—not in main memory—causes a trap, which is known as a *page fault.* We note that, even in the absence of any additional hardware support to maintain *reference bits*, this address translation mechanism can be put to use to detect references to pages that are already in memory. All that is required is that we be able to distinguish these faults from normal page faults and refrain from initiating the I/O operation. This special state of a page will be called the *reclaimable* state, and will be identified by one additional bit in each page table entry. Since this method of detecting references to pages comes at a cost (to be discussed later), we are interested in replacement algorithms that collect reference information only for a subset of the pages that a program has in memory. More formally, we have partitioned the set of pages in memory into two disjoint classes, $\{valid\}$ and $\{reclaimable\}$, such that

$$\{\text{memory}\} = \{\text{valid}\} \cup \{\text{reclaimable}\} \text{ and } \{\text{valid}\} \cap \{\text{reclaimable}\} = \phi.$$

To keep the cost of generating spurious faults to the reclaimable pages at reasonably low levels, we would like to have

$$|\{\text{reclaimable}\}| \ll |\{\text{valid}\}|.$$

We make these statements more precise in the following sections. For reasons which will become clear below, we shall refer to the set {valid} as **T** (for "top"), and to the set {reclaimable} as **B** (for "bottom"). Having partitioned the program's pages in memory into these two classes, we consider various policies for top-to-bottom and memory-to-secondary storage replacements. Since in our investigation we assume a single program to be executing, the analyses presented in the forthcoming sections consider *local* management policies. Extensions of these policies to *global* replacement schemes in multiprogramming environments are discussed in Section VI.

The two reasonable choices for the management of **T** are FIFO and RAND since for those pages we assume there does not exist hardware collection of reference information. Algorithms such as the Sampled Working Set [14], [20] and Clock [9], [13] that require this reference information are clearly out of consideration in this region. On the other hand, we can employ either the LRU or the WS algorithm for the management of **B** since the necessary information can be gathered at every occurrence of an artificial page fault. This results in four possible combinations that make up the hybrid class to be studied. Denoted $H_{\text{FIFO-LRU}}$, $H_{\text{RAND-LRU}}$, $H_{\text{FIFO-WS}}$ and $H_{\text{RAND-WS}}$, they are the FIFO-LRU, RAND-LRU, FIFO-WS, and the RAND-WS hybrid policies, respectively.

A. Fixed Partition Hybrids

Employment of the LRU policy for the management of **B** results in hybrid algorithms that operate in a fixed size memory partition. However, in a multiprogramming environment, the use of a common bottom among all the active programs results in a variable size partition for each, even though the tops are strictly local. We comment about such extensions in Section VI and restrict our study here to uniprogramming environments. For the following analysis, assume that **T** consists of k pages where k is the parameter of the policy, whereas the fixed partition size is m pages where $k \leq m \leq n$.

1) The FIFO-LRU Hybrid Policy: Given a page reference r_t at time t, the operation of the $H_{\text{FIFO-LRU}}(k)$ policy is as follows.

H1: If $r_t \in \mathbf{T}$, no control state change takes place. This is because this type of reference is transparent to our mechanism.

H2: (Reclaim) If $r_t \in \mathbf{B}$, then $\mathbf{T} \leftarrow \mathbf{T} + r_t - i$ where i is the FIFO (oldest) page in **T**, and $\mathbf{B} \leftarrow \mathbf{B} - r_t + i$ where "+" and "−" denote set membership operations.

H3: (Page fault) If $r_t \notin \{\text{memory}\}$, then $\mathbf{T} \leftarrow \mathbf{T} + r_t - i$ where page i is as in H2, and $\mathbf{B} \leftarrow \mathbf{B} + i - j$ such that page j is the one that has been (approximately) least recently used amongst all pages in **B**.

We cannot state that page j is exactly the LRU page because the ordering amongst the top is by time of entry and not by recency of use. Consequently, there may be pages in memory that have been referenced earlier than page j if, for example, page j was referenced just prior to its most recent departure from **T**. A more appropriate name for the replacement policy employed in **B** is *Least Recently Reclaimed.* In Section IV we present numerical results which suggest that, under a wide range of circumstances, the page replaced by these fixed partition hybrid policies from the bottom is very close to being the LRU page.

If we envision the control state associated with the algorithm as constituting a stack, the $H_{\text{FIFO-LRU}}$ policy can be regarded as a modification to the pure LRU policy where references to the top k positions of the LRU stack cause no control state change [19]. Note that, for the degenerate case $k = 1$, the page replaced from memory to secondary storage by the $H_{\text{FIFO-LRU}}$ policy is exactly the same page that would be replaced by the pure LRU policy. Furthermore, when $k = m$, the $H_{\text{FIFO-LRU}}$ policy degenerates into the pure FIFO policy. We point out that, as presented above, the $H_{\text{FIFO-LRU}}$ algorithm is identical to the *Segmented FIFO* (SFIFO) algorithm as described by Turner and Levy [24], and to the A_k^1 algorithm analyzed by Aven *et al.* [2].

The performance index that we will use to compare different policies is the *steady-state fault rate.* For a replacement algorithm A, the steady-state fault rate is defined as

$$F(A) = \lim_{t \to \infty} [\Pr(r_t \notin \{\text{memory}\})].$$

In other words, $F(A)$ is the limiting probability with which a reference to a page causes a page fault.

We are now in a position to derive an expression for $F(H_{\text{FIFO-LRU}}(k))$, based on the IRM. Such an expression has been previously obtained by Aven *et al.* [2]. Here, we sketch our derivation to illustrate the analysis technique which is similar to that used in [8] and which will be applied to the other hybrid algorithms as well. Let $\mathbf{s} = [j_1, j_2, \cdots, j_k, j_{k+1}, \cdots, j_m]$

by an m-tuple (without repetitions) corresponding to the memory control state of the policy. The first k entries of $\mathbf{s}$ contain the page names that constitute $\mathbf{T}$, whereas the remaining $m - k$ entries contain the names of the elements of $\mathbf{B}$. Define the Markov chain $\{X_t, t = 0, 1, \cdots\}$, with $X_t = \mathbf{s}$ if the memory control state at time t is given by $\mathbf{s}$. Let $\mathbf{Q} = \{\mathbf{s}\}$ denote the state space of this Markov chain.

The one-step transition probabilities denoted by

$$p(\mathbf{s}, \mathbf{s}') = \Pr(X_t = \mathbf{s}' | X_{t-1} = \mathbf{s}), \qquad t \geq 1$$

can be determined easily based on the IRM parameters and on the algorithm's description. Specifically, for $H_{\text{FIFO-LRU}}$,

$$p(\mathbf{s}, \mathbf{s}') = \begin{cases} \sum_{i=1}^{k} \beta_{j_i} & \text{if } \mathbf{s}' = \mathbf{s} \\ \beta_{j_l}, & \text{if } \mathbf{s}' = [j_l, j_1, j_2, \cdots, j_{l-1}, j_{l+1}, \cdots, j_m], k < l \leq m \\ \beta_j, & \text{if } \mathbf{s}' = [j, j_1, j_2, \cdots, j_{m-2}, j_{m-1}], j \notin \mathbf{s} \\ 0, & \text{otherwise.} \end{cases}$$

The three nonzero cases correspond to the $H1$, $H2$, and $H3$ events of the algorithm's description, respectively. As the above-defined chain is ergodic, the limiting state occupancy probabilities $\boldsymbol{\pi}$ exist and satisfy

$$\boldsymbol{\pi} = \boldsymbol{\pi} \cdot \mathbf{P} \tag{1}$$

where $\mathbf{P} = [p(\mathbf{s}, \mathbf{s}')]$ is the one-step transition matrix [21]. The limiting state occupancy probabilities $\boldsymbol{\pi}$, which are the eigenvalues of $\mathbf{P}$, have also to satisfy the normalization condition $\sum_{\mathbf{s} \in \mathbf{Q}} \pi_{\mathbf{s}} = 1$. For a particular state $\mathbf{s} = [j_1, j_2, j_3, \cdots, j_m]$, the matrix equation (1) can be written as

$$\pi_{\mathbf{s}} = \pi_{\mathbf{s}} \sum_{i=1}^{k} \beta_{j_i} + \beta_{j_1} \left[\sum_{j \notin \mathbf{s}} \pi_{\mathbf{u}_j} + \sum_{k < l \leq m} \pi_{\mathbf{v}_l} \right] \tag{2}$$

where $\mathbf{u}_j = [j_2, j_3, \cdots, j_m, j]$ and $\mathbf{v}_l = [j_2, j_3, \cdots, j_l, j_1, j_{l+1}, \cdots, j_m]$. Note that states $\mathbf{u}_j$ and $\mathbf{v}_l$ have been constructed so that a reference to page j_1 causes the algorithm to make a transition to state $\mathbf{s}$.

Lemma 1: For the $H_{\text{FIFO-LRU}}$ policy with parameter k, the equilibrium probability of state $\mathbf{s} = [j_1, j_2, j_3, \cdots, j_m]$ is given by

$$\pi_{\mathbf{s}} = \frac{\prod_{i=1}^{m} \beta_{j_i}}{G(k) \prod_{i=2}^{m-k+1} D_i(\mathbf{s})}$$

where

$$G(k) = \frac{(n-m)!}{(n-k)!} \sum_{\mathbf{s} \in \mathbf{Q}} \prod_{i=1}^{k} \beta_{j_i} \text{ and } D_i(\mathbf{s}) = 1 - \sum_{l=1}^{m-i+1} \beta_{j_l}.$$

The proof is by substitution and can be found in [5].

That the normalizing condition $\sum_{\mathbf{s} \in \mathbf{Q}} \pi_{\mathbf{s}} = 1$ is satisfied can be shown by an aggregation argument where first the $\pi_{\mathbf{s}}$'s are summed over the $(n-k)!/(n-m)!$ states that have the same $\mathbf{T}$ (the first k elements of $\mathbf{s}$) and then the resulting aggregates are summed to cover the entire state space $\mathbf{Q}$. The second summation can be shown to be the normalization condition for the equilibrium probabilities of the memory control states for a k-page memory managed by the pure FIFO policy which we know to hold.

Having the above lemma at hand, the following theorem can easily be proved.

Theorem 1: The steady-state fault rate generated by the $H_{\text{FIFO-LRU}}$ policy with parameter k and operating in a memory of m page frames is given by

$$F(H_{\text{FIFO-LRU}}(k)) = \sum_{\mathbf{s} \in \mathbf{Q}} \left[G^{-1}(k) D_1^2(\mathbf{s}) \frac{\prod_{i=1}^{m} \beta_{j_i}}{\prod_{i=1}^{m-k+1} D_i(\mathbf{s})} \right] \tag{3}$$

where $G(k)$ and $D_i(\mathbf{s})$ are as defined in Lemma 1.

Proof: Note that, given the current memory state $\mathbf{s}$, the probability of a page fault is simply $1 - \sum_{i=1}^{m} \beta_{j_i}$ which is by definition $D_1(\mathbf{s})$. Thus, conditioning on the state s, we can write

$$\Pr(\text{page fault}) = \sum_{\mathbf{s} \in \mathbf{Q}} \Pr(\text{page fault} | X_t = \mathbf{s}) \cdot \Pr(X_t = \mathbf{s}).$$

In steady state, we can replace the above probabilities with their limiting values and obtain the desired expression. □

Corollary 1: The steady-state fault rate for the pure LRU policy is given by

$$F(\text{LRU}) = \sum_{\mathbf{s} \in \mathbf{Q}} \left[D_1^2(\mathbf{s}) \frac{\prod_{i=1}^{m} \beta_{j_i}}{\prod_{i=1}^{m} D_i(\mathbf{s})} \right].$$

Proof: For $k = 1$, the $H_{\text{FIFO-LRU}}$ policy replaces the same page that the pure LRU policy replaces. Thus, $F(H_{\text{FIFO-LRU}}(1)) = F(\text{LRU})$. Substituting $k = 1$ into (3), we immediately have the desired result. □

Corollary 2: The steady-state fault rate for the pure FIFO policy is given by

$$F(\text{FIFO}) = \sum_{\mathbf{s} \in \mathbf{Q}} \left[G^{-1}(m) D_1(\mathbf{s}) \prod_{i=1}^{m} \beta_{j_i} \right].$$

Proof: For $k = m$ (i.e., $\mathbf{B} = \phi$), the $H_{\text{FIFO-LRU}}$ policy degenerates into the pure FIFO policy. Therefore, $F(H_{\text{FIFO-LRU}}(m)) = \text{F(FIFO)}$. The result follows trivially by setting $k = m$ in (3). □

2) *The RAND-LRU Hybrid Policy:* In this section, we consider the simple variant of the $H_{\text{FIFO-LRU}}$ policy where the page to be moved from $\mathbf{T}$ to $\mathbf{B}$ at the time of a replacement is selected at random, uniformly over the pages that currently constitute $\mathbf{T}$. More precisely, the algorithm is identical to the $H_{\text{FIFO-LRU}}$ policy except that the top-to-bottom replacement is performed according to the RAND policy.

We proceed with the analysis after giving the problem a formulation identical to that in the previous section. For this policy, the one-step transition probabilities are

$$p(s, s') = \begin{cases} \sum_{i=1}^{k} \beta_{j_i}, & \text{if } \mathbf{s}' = \mathbf{s} \\ \beta_j/k, & \text{if } \mathbf{s}' = [j\, j_1 j_2 \cdots j_{h-1} j_{h+1} \cdots j_k j_h j_{k+1} \cdots j_{m-1}] \\ & \text{where } j \notin \mathbf{s} \text{ and } 1 \le h \le k \\ \beta_l/k, & \text{if } \mathbf{s}' = [j_l j_1 j_2 \cdots j_{h-1} j_{h+1} \cdots j_k j_h k_{k+1} \cdots j_{l-1} j_{l+1} \cdots j_m] \\ & \text{where } 1 \le h \le k \text{ and } k < l \le m \\ 0, & \text{otherwise.} \end{cases}$$

It can easily be demonstrated that the resulting Markov chain is ergodic and therefore $\boldsymbol{\pi}$ exists and is a solution to (1) with the above one-step transition probabilities. For state $\mathbf{s} = [j_1, j_2, j_3, \cdots, j_m]$, (1) can be written as

$$\pi_{\mathbf{s}} = \pi_{\mathbf{s}} \sum_{i=1}^{k} \beta_{j_i} + \frac{\beta_{j_1}}{k} \left[\sum_{j \notin \mathbf{s}} \pi_{\mathbf{u}_j} + \sum_{h=1}^{k} \sum_{k<l\le m} \pi_{\mathbf{v}_l} \right] \quad (4)$$

where

$$\mathbf{u}_j = [j_2, j_3, \cdots, j_h, j_{k+1}, j_{h+1}, \cdots, j_k, j_{k+2}, \cdots, j_m, j]$$

and

$$\mathbf{v}_l = [j_2, j_3, \cdots, j_h, j_{k+1}, j_{h+1}, \cdots, j_k, j_{k+2}, \cdots, j_l, j_1, j_{l+1}, \cdots, j_m]$$

have again been constructed to result in state $\mathbf{s}$ upon the referencing of page j_1.

Lemma 2: For the $H_{\text{RAND-LRU}}$ policy with parameter k, the equilibrium probability of state $\mathbf{s} = [j_1, j_2, j_3, \cdots, j_m]$ is given by the same expression as for the $H_{\text{FIFO-LRU}}$ policy with the same parameter.

Proof: As in the proof of Lemma 1, we substitute the proposed solution for $\pi_{\mathbf{s}}$, into (4) and reduce it to an identity. □

Theorem 2: The steady-state fault rate for the $H_{\text{RAND-LRU}}$ policy with parameter k is equal to that of the $H_{\text{FIFO-LRU}}$ policy, i.e.,

$$F(H_{\text{RAND-LRU}}(k)) = \mathrm{F}(H_{FIFO\text{-}LRU}(k)).$$

Proof: The proof trivially follows from Lemma 2 by conditioning on the states of the Markov chain. □

Corollary 3: For programs whose behavior is perfectly represented by the IRM, the FIFO and RAND policies result in identical steady-state fault rates. That is,

$$F(\text{FIFO}) = F(\text{RAND}) \qquad \text{under the IRM.}$$

Proof: The proof trivially follows from Theorem 2 upon observing that, for $k = m$, the $H_{\text{RAND-LRU}}$ policy degenerates into the pure RAND policy and the $H_{\text{FIFO-LRU}}$ policy degenerates into the pure FIFO policy. □

Note that the above result has been obtained through a different method by Gelenbe [16].

B. Variable Partition Hybrids

In this section we consider hybrid policies that use WS management for **B**, thus resulting in variable size memory partitions. Note, however, that the partition size can never become less than k pages (the size of **T**) where k is a static parameter of the policy. Recall that the pure WS algorithm with parameter τ retains a page in memory if and only if it has been referenced at least once during the previous τ time units [11]. In our case since we have no information about references to a page during the period of its membership in **T**, we must somehow estimate the last time the page was referenced when it leaves **T**.

1) *The FIFO-WS Hybrid Policy:* In this algorithm, with each page, we associate a time t_i that is an estimate of the time of last reference to that page. We can now describe the operation of the $H_{\text{FIFO-WS}}$ policy with parameters k and τ.

W1: If $r_t \in \mathbf{T}$, no action is taken (as before, no action *can* be taken).

W2: If $r_t \notin \mathbf{T}$, then $\mathbf{T} \leftarrow \mathbf{T} + r_t - i$ where page i is the FIFO page in **T**. We provide our estimate of the time of last reference to page i as the current time: $t_i \leftarrow t$. We update **B** by removing all pages with last reference time estimates earlier than $t - \tau$; $\mathbf{B} \leftarrow \mathbf{B} + i - \mathbf{J}$, where $\mathbf{J} = \{j \in \mathbf{B}; t_j \le t - \tau\}$.

Note that in W2 above, we know page i could not have been referenced at time t since at that instant it was in **T**. Alternative estimates for the time of last reference to a page during its membership in **T** are the time of entry into **T** and the arithmetic mean of the times of entry into and exit from **T**. In practice, however, these variations have a negligible effect on performance [3].

For the variable partition hybrid policies, we are interested in obtaining expressions not only for the steady-state fault rate $F(A)$, but also for the mean memory occupancy $M(A)$, which is the expected number of pages that are in memory at steady state.

Let $\mathbf{s} = [j_1, j_2, j_3, \cdots, j_k]$ denote the states of **T** and let $\mathbf{P}_i$ be those states such that $i \notin \mathbf{s}$. After an analysis of the FIFO top in isolation, we can prove the following:

Lemma 3: The steady-state probability that page i is in a memory that is managed by the $H_{\text{FIFO-WS}}$ policy with parameters k and τ is given by

$$\Pr(i \in \{\text{memory}\}) = \pi_i + (1 - \pi_i) \cdot \sum_{j=1}^{\tau} \left[\Gamma_i (1 - \Gamma_i)^{j-1} \cdot \sum_{l=j}^{\tau} \binom{l-1}{l-j} p_i^j (1 - p_i)^{l-j} \right] \quad (5)$$

where

$$\pi_i = \Pr(i \in \mathbf{T}) = 1 - \sum_{\mathbf{s} \in \mathbf{P}_i} \pi_{\mathbf{s}},$$

$$\Gamma_i = \lim_{t \to \infty} \Pr(r_t = i | r_t \notin \mathbf{T}, i \notin \mathbf{T}) = \frac{(1 - \pi_i)\beta_i}{\sum_{\mathbf{s} \in \mathbf{P}_i} D_1(\mathbf{s}) \pi_{\mathbf{s}}},$$

$$p_i = \lim_{t \to \infty} \Pr(r_t \notin \mathbf{T} | i \notin \mathbf{T}) = \frac{\sum_{s \in P_i} \left[(1 - \sum_{j \in \mathbf{s}} \beta_j) \prod_{j \in \mathbf{s}} \beta_j \right]}{G(k)(1 - \pi_i)}.$$

The proof can be found in [3].

Theorem 3: The steady-state fault rate for the $H_{\text{FIFO-WS}}$ policy with parameters k and τ is given by

$$F(\text{H}_{\text{FIFO-WS}}) = 1 - \sum_{i=1}^{n} \beta_i \left[\pi_i + (1 - \pi_i) \cdot \sum_{j=1}^{\tau} \left[\Gamma_i (1 - \Gamma_i)^{j-1} \sum_{l=j}^{\tau} \binom{l-1}{l-j} p_i^j (1 - p_i)^{l-j} \right] \right]$$

where π_i, Γ_i, and p_i are as defined in Lemma 3.

Proof: In steady state, the probability that page i causes a fault is simply the steady-state probability that it is not in memory and is referenced. Thus, conditioning on the page,

$$F(H_{\text{FIFO-WS}}) = \sum_{i=1} (1 - \Pr(i \in \{\text{memory}\})) \cdot \beta_i.$$

The result then follows trivially upon the substitution of (5) into the above expression. □

Theorem 4: The mean memory occupancy due to the $H_{\text{FIFO-WS}}$ policy with parameters k and τ is given by

$$M(H_{\text{FIFO-WS}}) = \sum_{i=1}^{n} \left[\pi_i + (1 - \pi_i) \cdot \sum_{j=1}^{\tau} \left[\Gamma_i (1 - \Gamma_i)^{j-1} \cdot \sum_{l=j}^{\tau} \binom{l-1}{l-j} p_i^j (1 - p_i)^{l-j} \right] \right]$$

where π_i, Γ_i, and p_i are as defined as Lemma 3.

Proof: Conditioning on the page, we have

$$M(H_{\text{FIFO-WS}}) = \sum_{i=1}^{n} \Pr(i \in \{\text{memory}\}).$$

The result follows immediately from the substitution of (5) into the above expression. □

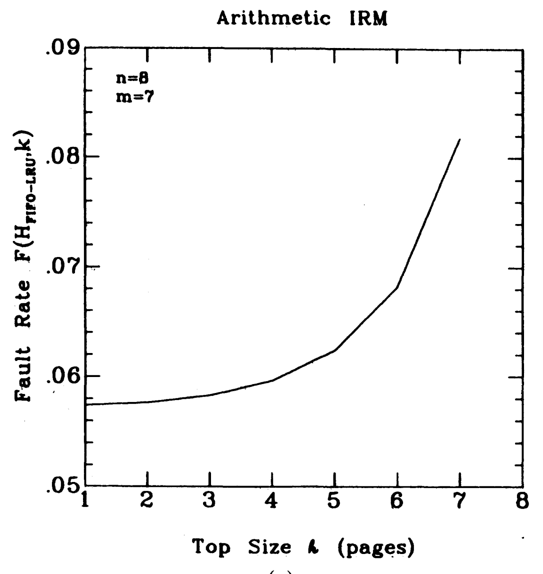
2) *The RAND-WS Hybrid Policy:* Here, we consider a variation of the $H_{\text{FIFO-WS}}$ policy that operates exactly as described in the previous section except that, at the times of events, the page to leave the top is selected at random uniformly over the pages currently in **T**.

Theorem 5: The steady-state fault rates and the mean memory occupancies for the $H_{\text{RAND-WS}}$ policy are identical to those of the $H_{\text{FIFO-WS}}$ policy with the same parameters.

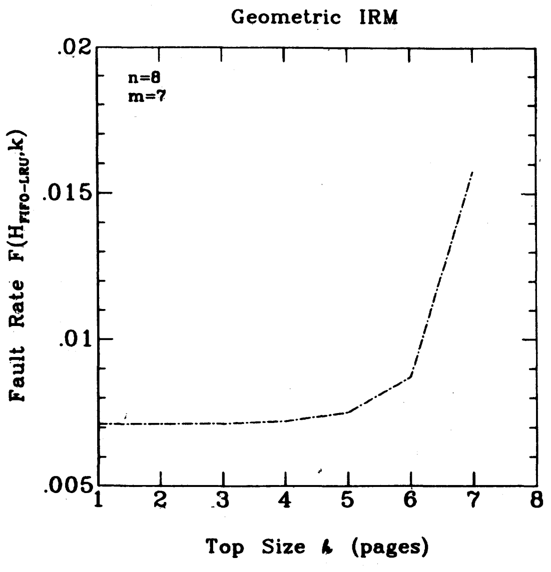
Proof: By Lemma 2 we know that the equilibrium probabilities for the Markov chain states are the same for both the RAND and the FIFO policies. Also, by Corollary 3, we know that the two policies result in the same steady-state fault rate. Based on these observations, the proof follows trivially. □

IV. Numerical Results

To study the behavior of the $F(H_{\text{FIFO-LRU}}(k))$ function between the two end points corresponding to pure LRU and pure FIFO fault rates, we computed numerical values of the expression given in (3) for various choices of the IRM parameters. To minimize the number of parameters involved, the two instances of the IRM we consider were generated through the equations $\beta_i = ci$ and $\beta_i = c^i$, which are called the *arithmetic* and the *geometric* models, respectively. In both cases, the constant c is chosen such that $\sum_{i=1}^{n} \beta_i = 1$. In Fig. 1,



(a)



(b)

Fig. 1. $H_{\text{FIFO-LRU}}$ and $H_{\text{RAND-LRU}}$ fault rates for two sample programs as a function of the policy parameter k.

$F(H_{\text{FIFO-LRU}}(k))$ is plotted as a function of k for the fixed values $n = 8$ and $m = 7$. Note the strong convexity of the two curves; particularly the one corresponding to the geometric IRM, where fault rates that are practically identical to the pure LRU fault rate are achieved even for top sizes of 5 pages (or equivalently, bottom sizes of 2 pages). Outside of the uniform IRM case (i.e., $\beta_i = 1/n$ for all i), the strict convexity of $F(H_{\text{FIFO-LRU}}(k))$ as a function k for all m and IRM parameter values remains a conjecture.

The above results have to be interpreted with caution for two reasons.

1) They are based on a model of program behavior that is known to lack many of the properties of real programs.

2) Due to the factorial growth of the complexity of the expressions involved, numerical results can only be presented for unrealistically small values of the program size n and of the memory size m.

To eliminate these concerns, trace-driven simulations of the hybrid policies were carried out using three program samples called WATFIV, APL, and FFT consisting of 96, 114, and 82 pages, respectively. The page size for the WATFIV program is 1024 bytes, whereas the APL and FFT program page size is 512 bytes. The reader should consult [22] for further details about the programs. The resulting values of the fixed partition hybrid fault rates as a function of the policy parameter are displayed in Fig. 2. These curves indicate that the conclusions based on analytic results are also valid for the three programs available to us. All three programs display a strongly convex curve with a well-defined knee at approximately $k = 40$ pages. Note that this represents a bottom size of 10 pages or, equivalently, 20 percent of the total memory size. This observation is not due to an anomaly of the three sample programs and the particular parameter values but is a more general property of the hyrid policies. If the pure LRU fault rate of a program were to remain relatively constant for memory sizes greater than 10 pages, there would be little incentive to add 40 more pages to the memory no matter how inexpensive their management was. However, from the data presented in [22], we observe that the pure LRU fault rates for these programs continue to decrease and do not flatten out for memory sizes greater than 10 pages.

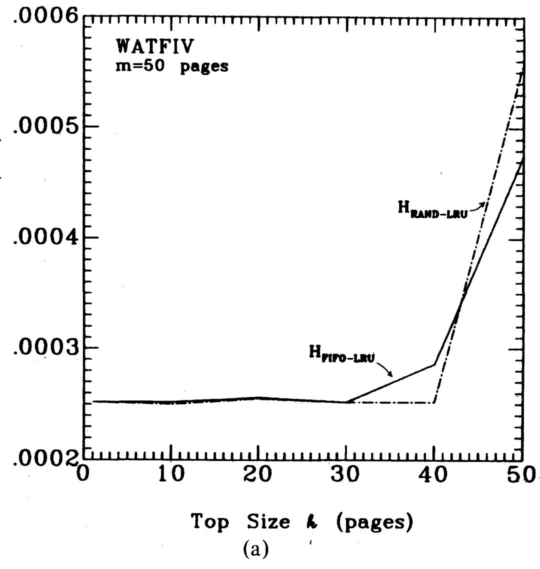
We also note that the $H_{\text{FIFO-LRU}}$ and $H_{\text{RAND-LRU}}$ policies result in different fault rates for these programs. This confirms the fact that these programs do not satisfy the IRM assumptions. No general conclusion about the relative performances of the $H_{\text{FIFO-LRU}}$ and $H_{\text{RAND-LRU}}$ policies however, can be derived from these results, as each is uniformly superior for one of the programs, while both perform about the same for the remaining program.

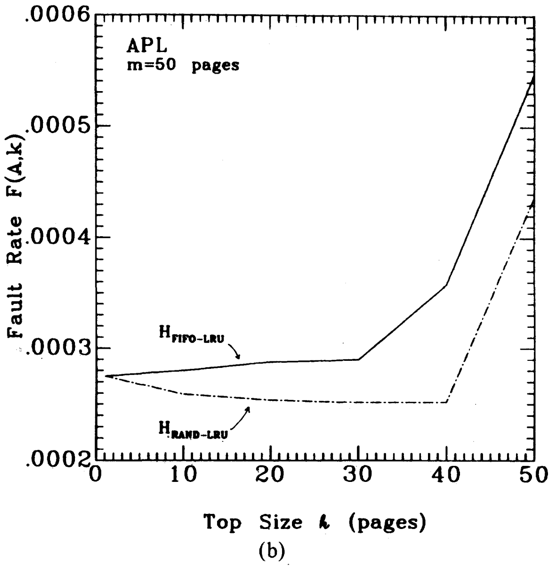
V. Cost Considerations

Up to this point, the only performance measure we have been concerned with has been the steady-state fault rate as a function of the mean memory occupancy. The operation of these hybrid policies requires setting the policy parameter k to some value. If the cost of referencing a page that is in **B** were negligible, then setting $k = 1$ would almost always produce optimum performance with respect to the page fault rate (there are few points in the graphs of Fig. 2 where the fault rate actually drops as k is increased beyond 1). However, since a finite cost is incurred each time a page in **B** is referenced, the selection of the policy parameter should be guided by the desire to keep the fault rate close to the value given by pure LRU algorithm while minimizing the size of **B** (i.e., minimizing the rate of reclaims). Intuitively, the policy should be operated with the parameter set to a value close to the knee that occurs in all three fault rate graphs. Formally, we define a performance measure $C(\cdot)$, that is the weighted sum of the fault rate and the reclaim rate for a given value of the policy parameter. For a page replacement algorithm A and a ratio of page fault service time to page reclaim service time given by α, let

$$C(A, m, k, \alpha) = f(A, m - k) + \alpha \cdot F(A, m, k)$$

where m is the memory size, k is the policy parameter (or the size of **T**) and $f(\cdot)$ is the reclaim rate. Note that while appro-





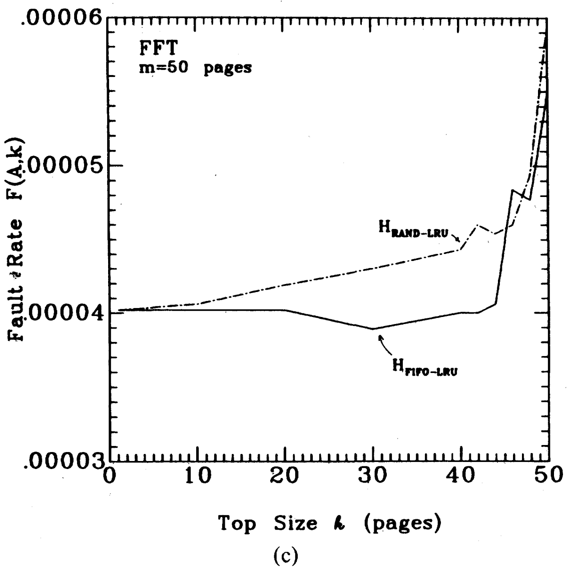


Fig. 2. $H_{\text{FIFO-LRU}}$ and $H_{\text{RAND-LRU}}$ fault rates as a function of the policy parameter k for (a) WATFIV, (b) APL, and (c) FFT.

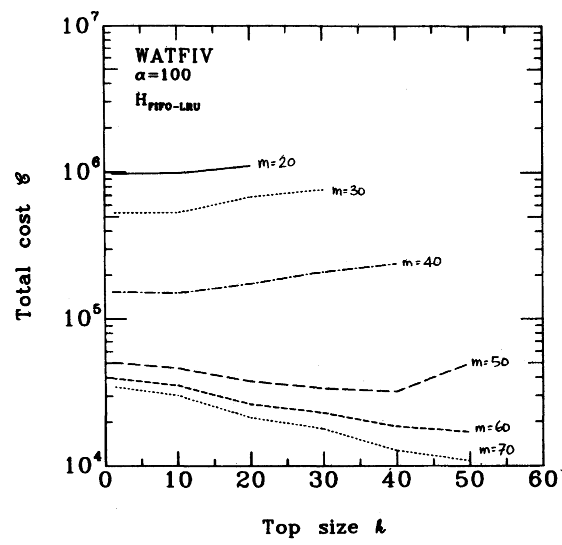
priate as a *responsiveness* measure from the point of view of an individual program, the cost function $C(\cdot)$ is inappropriate for multiprogramming system *throughput* considerations since the major part of the delay due to a page fault results from the paging I/O operation, which can be overlapped with other CPU activity. On the other hand, a reclaim operation is performed entirely by the CPU and cannot be overlapped.

Due to the complexity of the expressions for the steady-state fault and reclaim rates, an analytic minimization of the cost function defined with respect to k cannot be carried out. As before, we resort to trace-driven simulation to study the behavior of this cost function as the policy parameter changes. Fig. 3 displays this cost function for the three programs under the $H_{\text{FIFO-LRU}}$ policy. For the sake of brevity, we restrict the presentation of the results to the case $\alpha = 100$ under the $H_{\text{FIFO-LRU}}$ policy. Our conclusions, however, are applicable to the $H_{\text{RAND-LRU}}$ policy as well as to the cases when $\alpha = 10$ and $\alpha = 1000$ under both policies. The particular choice of the value of α for the results reported here is further supported by measurements from an actual system [4]. As can be seen in Fig. 3, the optimum setting for the policy parameter k varies significantly depending on the program as well as on the total amount of memory allocated to it. In [24], Turner and Levy report on the behavior of this cost function for different values of α. The observed insensitivity of the optimum value of the policy parameter to variations of α is unfortunately not very valuable since, for a given implementation, α tends to be rather static anyway. In a real system, the characteristics of the different programs and the amount of memory allocated to each program are far more dynamic. Our results demonstrate that it is precisely these variations that require the parameter to be adjusted for the hybrid policies to perform optimally. Conditions which make m a variable rather than a fixed system parameter are discussed in the next section.

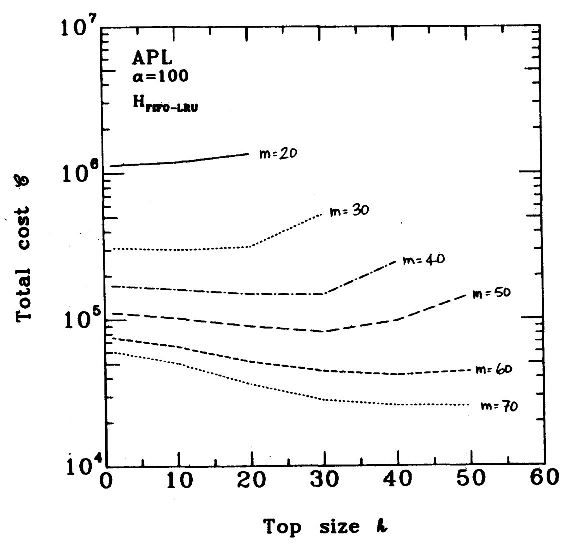
Fig. 4 compares the performance of the two variable partition hybrids to that of the fixed partition hybrid. For the three programs studied, the $H_{\text{FIFO-WS}}$ and $H_{\text{RAND-WS}}$ policies have similar performances, with neither exhibiting uniform superiority. This is not surprising after our observation of the FIFO and RAND policies in conjunction with the LRU bottom. Both of these variable partition policies, however, outperform the fixed partition hybrids (by more than an order of magnitude in some cases) for equal values of the parameter and of the mean memory size in all three programs. Furthermore, these variable partition hybrids tend to have performances that are more uniform with respect to changes in the memory size than their fixed partition counterparts. Note that Fig. 4(c) contains mean memory sizes that are less than the parameter value for the case $k = 50$. This results from the fact that our simulation studies assume an empty initial memory that may take a long time to fill.

VI. Extensions

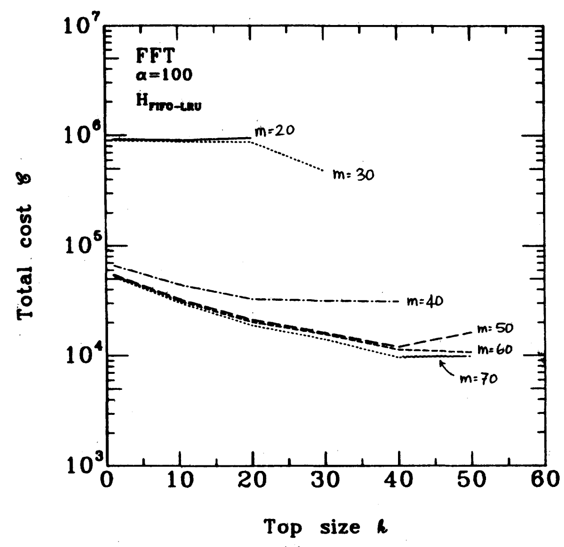
In a multiprogramming environment, the fixed partition hybrids can be extended in a natural way to operate as FIFO-Global LRU and RAND-Global LRU hybrids, thus resulting in variable partitions for the individual programs. This can be accomplished simply by maintaining a single fixed



(a)



(b)



(c)

Fig. 3. The weighted sum cost observed under the $H_{\text{FIFO-LRU}}$ policy for varying amounts of memory as a function of the policy parameter and programs (a) WATFIV, (b) APL, and (c) FFT.

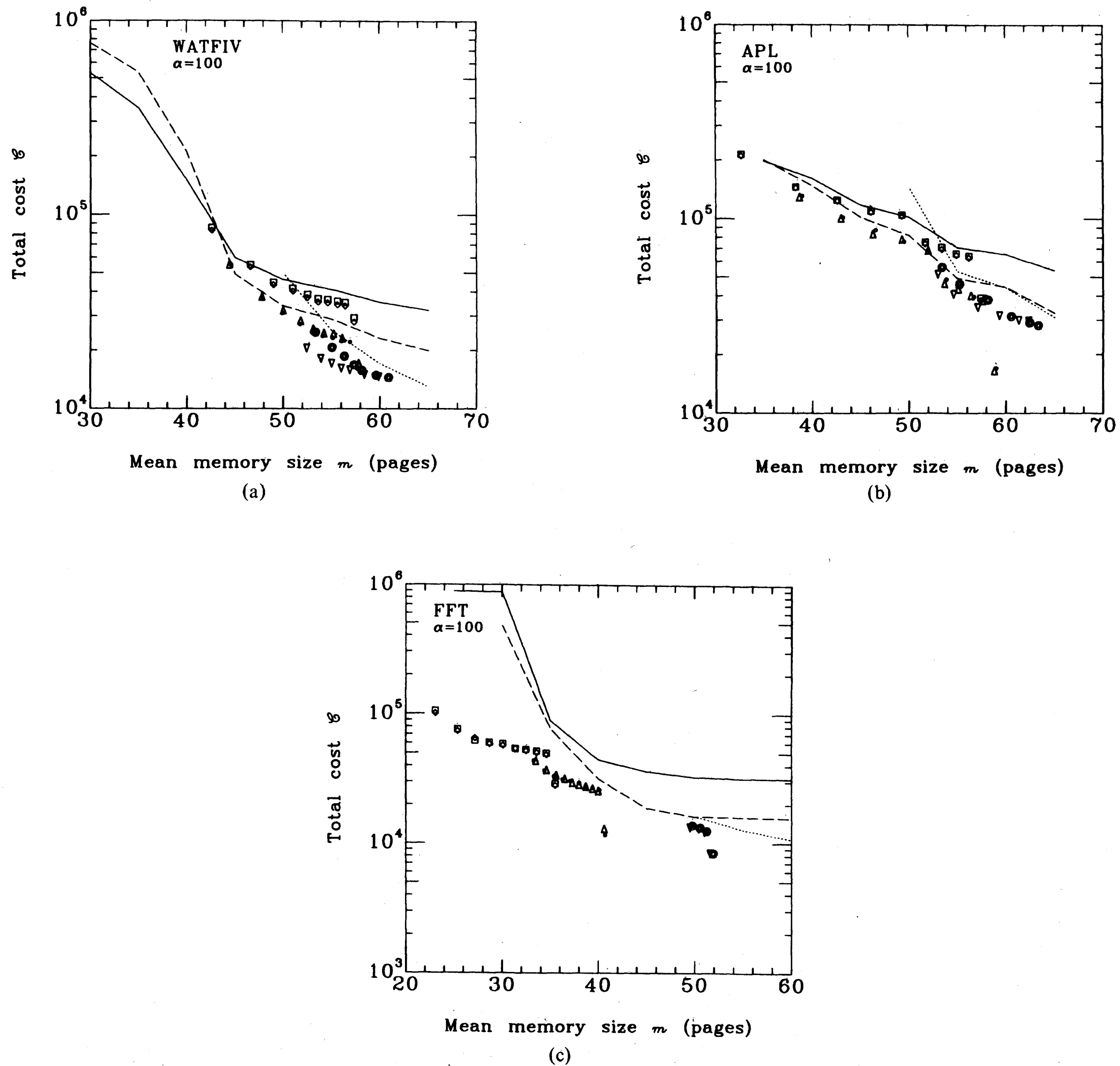


Fig. 4. $H_{\text{FIFO-LRU}}$ (lines), $H_{\text{FIFO-WS}}$, and $H_{\text{RAND-WS}}$ policy weighted sum costs as a function of the memory size m for (a) WATFIV, (b) APL, and (c) FFT. The window size τ was varied between 11730 and 117300 references to obtain the variable partition results. Key: □, △, and ▽ represent $H_{\text{FIFO-WS}}$ with k = 10, 30, 50, respectively, whereas ◇, ○, and @ represent $H_{\text{RAND-WS}}$ with k = 10, 30, 50, respectively.

size bottom containing the reclaimable pages of all the programs that are currently being multiprogrammed (the tops for each of the programs, however, are still maintained separately). In such an environment, the total number of page frames allocated to a program at any given time will be the size of its top plus a random variable that represents the number of pages in **B** belonging to the given program. Under this extension, the study of the performance of individual programs is severely complicated due to their interactions with the other concurrently running programs.

Although we have not studied it analytically, we note that this extension of the $H_{\text{FIFO-LRU}}$ policy adequately models the memory management policy employed in the VMS operating system for the VAX-11/780 computer system [10], [24], [18]. Unfortunately, for optimality to be preserved under this extension, not only must the hybrid policy parameter be set on a per-program basis, but it should also vary dynamically as the amount of memory allocated to the program changes. We note that heuristics such as those that try to keep the top size to the bottom size ratio a constant or that try to maintain a constant reclaim rate can be employed to vary the policy parameter dynamically. The current version of VMS, in fact, implements a scheme similar to the second option to adjust the size of **T** as each program executes [24]. In [4], Babaoğlu and Joy present a considerably different approach to the page replacement problem in an architecture lacking reference bits that was implemented as part of the Berkeley UNIX[1] operating system.

VII. Conclusions

We have introduced a class of hybrid algorithms that are suitable for page replacement decisions in hierarchical stores that lack hardware support other than the address translation

[1] UNIX is a trademark of Bell Laboratories.

mechanism. Expressions for the steady-state fault rates generated by these policies have been derived based on the Independent Reference Model of program behavior. With the appropriate setting of the policy parameter, numerical and empirical results suggest that these algorithms are capable of achieving fault rates close to those of the pure LRU and WS policies while incurring costs comparable to those of the FIFO and RAND policies.

We have defined a cost function that includes explicitly, the cost of reclaim operations; this function is more suitable than the simple fault rate for the evaluation of hybrid algorithms. Based on this cost function, the parameter of the hybrid policies we analyzed was seen to be sensitive to variations in the program characteristics as well as to the amount of memory allocated to it. We have suggested reasonable heuristics that can be incorporated into these policies to allow the dynamic adjustment of their parameters.

References

[1] A. V. Aho, P. J. Denning, and J. D. Ullman, "Principles of optimal page replacement," *J. Ass. Comput. Mach.*, vol. 18, pp. 80–93, Jan. 1971.

[2] O. I. Aven, L. B. Boguslavsky, and Y. A. Kogan, "Some results on distribution-free analysis of paging algorithms," *IEEE Trans. Comput.*, vol. C-25, pp. 737–745, July 1976.

[3] Ö. Babaoğlu, "Virtual storage management in the absence of reference bits," Ph.D. dissertation, Univ. of California, Berkeley, Memo ERL M81/92, Nov. 1981.

[4] Ö. Babaoğlu and W. N. Joy, "Converting a swap-based system to do paging in an architecture lacking page-referenced bits," in *Proc. 8th Symp. Operating Syst. Principles, SIGOPS*, Dec. 1981, pp. 78–86.

[5] Ö. Babaoğlu, "Hierarchical replacement decisions in hierarchical stores," in *Proc. ACM SIGMETRICS Conf. Measurement and Modeling Comput. Syst.*, Aug. 1982, pp. 11–19; also in *Performance Eval. Rev.*, Winter 1982–1983.

[6] F. Baskett and A. Rafii, "The A_0 inversion model of program paging behavior," Dep. Comput. Sci., Stanford Univ., Rep. STAN-CS-76-579, Oct. 1976.

[7] L. A. Belady, "A study of replacement algorithms for a virtual storage computer," *IBM Syst. J.*, vol. 5, pp. 78–101, 1966.

[8] E. G. Coffman and P. J. Denning, *Operating Systems Theory*. Englewood Cliffs, NJ: Prentice-Hall, 1973.

[9] F. J. Corbato, "A paging experiment with the multics system," MIT, Cambridge, MA, Project MAC Memo. MAC-M-384, July 1968. Also in *In Honor of P. M. Morse ed.* Cambridge, MA: Ingard MIT Press, 1969, pp. 217–228.

[10] *VAX-11/780 Software Handbook*. Bedford, MA: Digital, 1978.

[11] P. J. Denning, "The working set model of program behavior," *Commun. Ass. Comput. Mach.*, vol. 11, pp, 323–333, May 1968.

[12] ——, "Virtual memory," *Comput. Surveys*, vol. 2, no. 3, pp. 153–189, Sept. 1970.

[13] M. Easton and P. A. Franaszek, "Use bit scanning in replacement decisions," *IEEE Trans. Comput.*, vol. C-28, pp. 133–141, Feb. 1979.

[14] M. H. Fogel, "The VMOS paging algorithm—A practical implementation of a working set model," *Oper. Syst. Rev.*, vol. 8, pp. 8–17, Jan. 1974.

[15] J. Fotheringham, "Dynamic storage allocation in the Atlas computer including an automatic use of backing store," *Commun. Ass. Comput. Mach.*, vol. 4, pp. 435–436, Oct. 1961.

[16] E. Gelenbe, "A unified approach to the evaluation of a class of replacement algorithms," *IEEE Trans. Comput.*, vol. C-22, pp. 611–618, June 1973.

[17] T. Kilburn, D.B.G. Edwards, M. J. Lanigan, and F. H. Sumner, "One level storage systems," *IRE Trans. Electron. Comput.*, vol. EC-11, pp. 223–235, Apr. 1962.

[18] E. D. Lazowska, "The benchmarking, tuning and analytical modeling of VAX/VMS," in *Proc. Conf. Simulation, Measurement and Modeling Comput. Syst.*, Boulder, CO, Aug. 1979, pp. 57–64.

[19] 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.

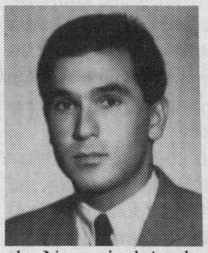
[20] B. G. Prieve, "A page partition replacement algorithm," Ph.D. dissertation, Univ. California, Berkeley, 1974.

[21] S. M. Ross, *Applied Probability Models with Optimization Applications*. San Francisco: Holden-Day, 1970.

[22] A. J. Smith, "Two simple methods for the efficient analysis of memory address trace data," *IEEE Trans. Software Eng.*, vol. SE-3, pp. 94–101, Jan. 1977.

[23] ——, "Bibliography on paging and related topics," *Oper. Syst. Rev.*, vol. 12, no. 4, pp. 39–56, Oct. 1978.

[24] R. Turner and H. Levy, "Segmented FIFO page replacement," in *Proc. ACM SIGMETRICS Conf. Measurement and Modeling Comput. Syst.*, Sept. 1981, pp. 48–51; also in *Performance Eval. Rev.*, Fall 1981.



Özalp Babaoğlu (S'78–M'81) was born in Ankara, Turkey, in 1955. He received the B.Sc. degree in electrical engineering from George Washington University, Washington, DC, in 1976 and the M.Sc. and Ph.D. degrees in computer science from the University of California, Berkeley, in 1977 and 1981, respectively.

He spent the summer of 1978 at IBM San Jose Research Laboratory as an Academic Associate and the first half of 1980 as a Foreign Scholar at the Numerical Analysis Institute of the Italian National Research Council in Pavia, Italy. Before joining the faculty at Cornell University, Ithaca, NY, in 1981 as an Assistant Professor, he was a Staff Scientist with the Computer Science and Applied Math Group, Lawrence Berkeley Laboratories, Berkeley, CA. He was the principal designer and coimplementor of the virtual memory extensions to the UNIX operating system, known as 3.0bsd, for the VAX computer. He was the corecipient of the 1982 Sakrison Memorial Award for his contributions to the Berkeley UNIX operating system. His current research interests include performance evaluation, modeling, and distributed and fault-tolerant computing systems.

Dr. Babaoğlu is a member of the Association for Computing Machinery and the IEEE Computer Society.

Domenico Ferrari (M'67–SM'83) received the Dr.Ing. degree in electronic engineering from the Polytechnic Institute of Milan, Italy, in 1963.

In 1964, he joined the Computer Laboratory, Polytechnic Institute of Milan, and started doing research in switching theory, high-speed arithmetic units, logical design, and later in computer-aided electronic circuit design. He became an Assitant Professor in 1967, and was awarded the Libera Docenza in computer science in 1969. In 1970, he joined the faculty of the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, where he is now Professor of Computer Science. Since then, he has been academically and professionally involved in computer systems performance evaluation activities, with particular emphasis on instrumentation design, virtual-memory system improvement, and workload characterization. His more recent interests include program behavior modeling, computer systems dynamics, and distributed system performance measurement and evaluation. From 1970 to 1973, he participated in the development of PRIME, an interactive multiprocessor system designed by the Computer Systems Research Project of the University of California with ARPA support. Since 1974, he has directed the PROGRESS Group, which does research in performance evaluation with the support of the National Science Foundation.

Dr. Ferrari has organized conferences and advanced courses, lectured extensively in the U.S. and in Europe, and consulted for EDP centers, university installations, computer manufacturers, and software houses in the areas of performance improvement and capacity planning. He is the author of *Computer Systems Performance Evaluation*, Prentice-Hall, 1978, a coauthor of *Measurement and Tuning of Computer Systems*, Prentice-Hall, 1983, the editor of *Performance of Computer Installations—Evaluation and Management* (North-Holland, 1978) and a coeditor of *Experimental Computer Performance Evaluation* North-Holland, 1981, as well as of *Theory and Practice of Software Technology*, North-Holland, 1983. He is on the International Board of Editors of *Performance Evaluation*, and is a member of the ACM and of the IEEE Computer Society.