Analysis of page-reference strings of an interactive system

by M. G. Kienzle
J. A. Garay
W. H. Tetzlaff

The performance of real-storage-management algorithms in interactive systems suggests that locality of reference extends to a significant degree across users' transactions. This paper investigates this locality of reference by analyzing page-reference strings gathered from live systems. The data confirm the supposition: They suggest that reference patterns are dominated by system data references that are implied by the user's commands. Program references appear to play only a minor role. The user command sequence is an important factor in the reference behavior of an interactive session.

Introduction

- *Scope*

The concept of *locality of reference* has usually been applied to individual programs and their data. The experience with interactive systems, such as IBM's CMS system, suggests that the concept of locality of reference extends to larger contexts. Series of users' interactions with the system can show locality of reference. The user may issue many different commands, invoking different programs and referencing a variety of system and user data. Even so, we can observe locality in the reference behavior shown by users' sessions. As we will see, this locality stems to a large degree from two sources:

- References to operating system data related to a user's session comprise a large component of the total reference pattern of the session.
- The page context of a user's work often remains stable over sequences of transactions.

Most of the existing work in this area concentrates on the behavior of individual programs, excluding the contribution to the reference pattern by the operating system [1–5]. This study expands the scope of the analysis in two directions: It includes the contribution to the reference strings by the operating system; it extends beyond single user commands to sequences of user transactions. In fact, our analysis ignores locality of reference within individual transactions, concentrating on locality of reference across transactions.

The remainder of this section discusses some concepts that are necessary as background for the paper. The next section describes the tools used in the data-collection and data-reduction process. The third section presents and interprets the reference patterns that we analyzed. The following section describes the statistical data analysis, and the final section sums up the most important points of the paper.

Background

About two decades ago, the principle of locality of reference was formulated [1]. This principle holds that most programs do not reference their memory space uniformly. Rather, within a short time window, they reference only a small part of their memory space, called the working set. As program execution proceeds, the contents of the working set will change, and possibly its size will do so as well. But the working set, including the program code as well as the data, is always only a small part of the total memory space of a program.

Since that time, locality of reference has been exploited by virtual-memory systems. These systems attempt to keep in main memory only the working sets of programs. At the same time, they create the appearance to programs that they have their entire memory space available. This leads to a very economical use of main memory while simplifying the task of programming by giving programmers large virtual memories.

The performance of virtual-memory operating systems depends crucially on their ability to recognize the working set of a program. Much work has been done in the areas of measuring the memory demands of programs and modeling their reference behavior [1–5]. The results of this work have been fed back into the design of real-storage-management algorithms.

One model that was found useful to describe program-reference behavior is the phase-transition model [1, 3]. This model assumes that a program favors a subset of its segments (pages) during extended intervals called phases. During these phases, the program's working set changes very little. After some time, a radical transition occurs in which many of the pages in the working set are exchanged. The phase-transition model consists of a macromodel and a micromodel. The macromodel is a semi-Markov model whose states are locality sets and whose "holding times" are phases. This model is used to characterize program behavior at the phase level. The micromodel concentrates on what happens within a phase; e.g., it generates reference strings over the pages of the locality set. Phases and transitions are of equal importance to the reference behavior of a program: Long phases dominate overall reference behavior, and transitions, being unpredictable, account for a substantial part of page faults.

These analyses and models are adequate when users' programs dominate the aspects of system behavior that are relevant to managing the pageable main memory. However, on time-sharing systems supporting large numbers of users, this assumption is no longer true. Most user transactions are short [6], so that the programs called hardly have an opportunity to capitalize on a working set. And many operating-system data structures that had not been part of pageable memory in early time-sharing systems had to be made pageable in order to accommodate large numbers of users. Therefore, operating-system data references substantially influence the total reference pattern caused by a user's interaction.

Despite these changes in the execution environment, virtual-memory-based time-sharing systems can display very good performance. Measurements indicate that a large portion of the pages referenced in one transaction are referenced again in the following transaction. This leads to two suppositions: 1) There is a significant locality of reference across users' transactions. 2) For good performance, the operating-system data structures must show good locality of reference.

The measurements and the analysis presented in this paper test these two suppositions. The impact of the findings on the real-storage-management algorithms is discussed in [7].

Measurement data

Characteristics

The data on which this study is based were measured on an IBM VM/370 HPO system with users running CMS in their virtual machines [8, 9]. CMS is a single-user operating system that in itself does not implement virtual memory. CMS runs in virtual machines supported by CP, the underlying hypervisor supporting virtual memory. In a sense, each virtual machine can be considered a single-user workstation running the CMS operating system, with the CP hypervisor supporting many such workstations on one large computer. In supporting virtual memory, CP manages the hardware reference bits of the storage pages used by the virtual machines [10, 11]. As part of that support, CP can record the numbers of pages that have been referenced during certain intervals. These page-reference data are the source for the reference strings used in this study. VM uses pages of 4K bytes. Consequently, the address resolution of the data is to blocks of 4K bytes.

The data generally include all references by a virtual machine, with some exceptions noted later. The CMS functions included in the references are program management, command interpretation, terminal management, and the file system. The user commands invoke system utilities, such as editors and compilers, user-written command procedures, and user programs. The data from each virtual machine are recorded separately, so that it is possible to construct reference strings for each virtual machine.

The CP-provided functions whose references are not inclued in the data comprise mainly dispatching of virtual machines, paging, and parts of the I/O supervisor. In any operating system, these functions are so heavily used that either they are not pageable or their pages are naturally resident in main memory. Their references would contribute little if anything to paging.

The CP scheduler divides transactions into scheduling intervals. The first interval spans about 80 milliseconds of virtual CPU time. The subsequent intervals are longer. At the end of each interval, the real-storage manager resets the reference bits in order to adjust the working set. We call the pages referenced in one scheduling interval the *reference set*[1] of that interval. The VM Monitor records the last reference set of a user transaction. Short transactions—and the majority of transactions fall into this class—span only one scheduling interval (or queue stay, in VM/370 terminology). For these transactions, all references are recorded. If a transaction extends over several intervals, reference data are lost. Thus, the time resolution of the reference data may be viewed in terms of scheduling intervals, with only the last reference set of a transaction being recorded.

We assume that the references within transactions show at least as much locality as across transactions. This would mean that the locality of complete reference strings would be higher than shown in the data here. Also, we expect that reference strings of long transactions are dominated by the user programs rather than by the operating system. In this case, the characteristics as described in prior studies probably apply.

The emphasis of this study is on locality across transactions and on short transactions that are prevalent in interactive computing [6]. Therefore, the loss of data is not as great a problem as it may look at first. However, for a more general use of the reference data, a complete trace would clearly be desirable. **Table 1** shows the magnitude of the data loss. The first data column gives the total number of scheduling intervals; the next data column gives the number of intervals for which reference data were recorded; and the last column shows the percentage of intervals for which data were recorded. These figures show that the large majority of the transactions recorded are small (as is characteristic of interactive systems) and that the extent of the missing data is small enough not to invalidate the analysis.

For an extension of this study, more complete data would be helpful. By writing the reference-data records at every scheduling interval, we could obtain complete reference-string traces. This would be especially helpful in the analysis of systems whose major workload involves large transactions. Production virtual machines that use MVS, DOS/VSE, or VS/1 as their virtual operating systems generally fall into this category.

• *Sources*

All data presented in the following sections were recorded at three different installations, from interactive user sessions or from server virtual machines running CMS. One installation is an educational institution, one supports a manufacturing plant, and one installation is in a service industry. All three installations were running VM/HPO 3.4 at the time of recording, and from each installation we analyzed the data of five virtual machines running CMS. From the total of 15 virtual machines, we selected six virtual machines that had the longest and the most complete reference strings. We also made sure to include in the study a broad sample of the different effects we observed.

Table 1 Missing data in reference strings.

Virtual machine ID	*Characterization*	*Scheduling intervals*		
		Total	*With ref. data*	*Ref. data (%)*
VM 1	Initial example	296	229	77.3
VM 2	Interactive application•	425	402	94.5
VM 4	Service machine	553	448	81.0
VM 7	Typical example	298	220	73.8
VM 8	Example of phases	357	316	88.5
VM 9	Service machine	400	222	55.5

• *Measurement and analysis tools used*

The data used in this analysis are available through the VM/Monitor [8] for VM/HPO Releases 3.4 to 4.2. Earlier and later releases of VM/HPO have real-storage-management algorithms that would make it very expensive to record the reference data. Therefore, on those other releases the VM/Monitor does not provide page-reference data.

For the reduction of the VM/Monitor data, we used a data-reduction package developed at the IBM Thomas J. Watson Research Center, and known as the Generalized Reduction of Information (GRIN) program [12]. GRIN is a program generator that has built-in knowledge of VM/Monitor data and that offers a very-high-level language for the specification of data-reduction requests. This makes it well suited for exploratory data analysis.

After we extracted the relevant data from the VM/Monitor files using the GRIN package, we used the GRAFSTAT system [13] to perform the statistical analysis for the graphic presentation of the data. GRAFSTAT allows presentation of graphs on high-resolution displays as well as on high-resolution printers.

• *CMS storage layout*

The storage layout of CMS is important for the interpretation of the reference strings. We can identify four major areas, as shown in **Figure 1.**

The *low-CMS area* occupies pages 0 to 31 of the address space. This area contains a *system-data area*, a *user-data area*, and the *transient area*. The system-data area holds CMS system control blocks that are relatively static during

[1] CP uses the reference set as an approximation to the working set. But because the intervals are quite long for working-set windows, and have different lengths, the reference set is not a working set in the strict sense of the definition [1].

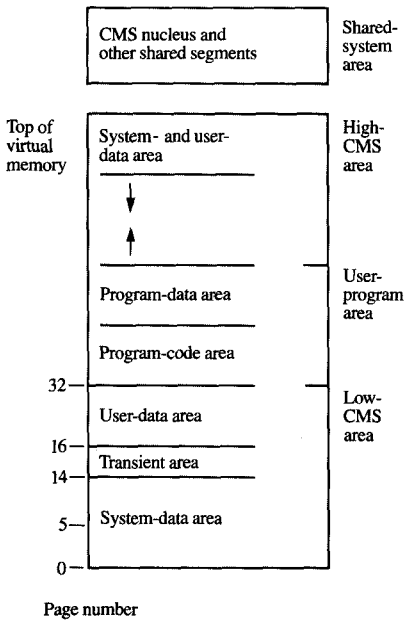

Figure 1

CMS storage layout.

the course of a user session. The user-data area can hold any data that a user allocates which fit into the given amount of space. (The terms *user data* and *user programs*, when used subsequently, refer to all data and programs that are not part of CMS, but are part of application programs, and are not in the shared area.) The transient area is intended for special commands whose programs fit into 8K bytes of memory. The low-CMS area is generally broken down into many small areas that are referenced independently of one another.

The *user-program area* starts at page 32. User-program space is usually allocated upward from this address. First comes the *program-code area*, where the CMS loader loads the programs. Immediately above it is the *program-data area*, where programs may allocate private data space.

The *high-CMS area* occupies the rest of a CMS virtual machine. There, CMS allocates system-data areas and user-data areas. The allocation direction is downward from the top of the virtual machine. System data, such as CMS file-system control blocks, often remain allocated for an entire user session. User data, such as files being edited, are generally transient. Most of the time, the system data are allocated first and reside at the top, and the user data are allocated below them. However, this is not always the case—system data and user data can be intermixed, and there can be unallocated pieces of storage between them.

The *shared-system area* resides in an address range above the virtual machine's address range. It contains shared segments that are very important for the reference behavior and the performance of CMS. Many of the frequently used programs, such as the command interpreter (REXX), the system editor (XEDIT), the text formatter (SCRIPT/VS), and the APL interpreter, reside here. The shared segments also contain file directories for some read-only shared files. These segments are read-only segments, and they are shared among all the virtual machines in the system that connect to them. The references to pages in these segments cannot be traced to individual virtual machines, and are excluded from the analysis here. Since only very frequently used programs are loaded into shared segments, it can be assumed that they are in storage most of the time and that they do not significantly contribute to the paging traffic.

Data presentation

• *Page-reference pattern*

Figure 2 shows the page-reference pattern of a virtual machine running CMS. The *X*-axis dimension represents virtual page numbers. Each point along the *X*-axis represents a page of storage of the virtual machine, sorted in the order of the virtual addresses of the pages. The user whose reference pattern the graph shows has a virtual machine with 256 pages, or one megabyte, of virtual storage.

Each point on the *Y*-axis represents the last or only scheduling interval of a transaction. The transactions are arranged in time sequence, with the time increasing from the bottom to the top of the graph. Thus, the *Y*-axis is not proportional to real time, but it is approximately proportional to the virtual CPU time of the user's virtual machine. The user in the graph originated 267 transactions during the measurement interval.

Each dot in the graph represents a page reference. A vertical line shows all the scheduling periods during which a particular page has been referenced. A horizontal line depicts the pages that were referenced during one scheduling period.

• *Interpretation of the reference pattern*

VM 1 is the only virtual machine whose actual CMS storage map we could obtain, and whose activity during the measurement period we know to some degree. This gives us some help in relating the reference patterns to the users' activities. In the measurement interval, the user of VM1 spent most of his time editing, using the XEDIT editor supplied with CMS. He also sent some files to another system and he built some libraries.

The references in the low-CMS area show a very regular pattern. Some pages are referenced in practically every transaction. Other pages form groups that are often referenced together. Some pages are referenced rarely or not at all. The pattern in this area is quite distinct, and virtually

all CMS reference strings we analyzed show a similar pattern. While the patterns are similar, they rarely match completely. In particular, the pages that are referenced in virtually every transaction are not the same for all users. They depend on the configuration of the virtual machine, on some details of the CMS initialization, and on the initial user-command sequence. The number of pages referenced in every transaction can be as low as five or as high as the entire area between page 0 and page 31.

VM 1 references the user program area very sparsely. The two programs of major size are probably the calls to build a library, and the six programs of only four pages are probably the programs to send files to another system.

The user-data area in the high-CMS area, from approximately page 80 to page 190, shows a sparse reference pattern. The lower part of this area, up to about page 165, contains the files being edited. The upper part contains XEDIT macros. The long contiguous reference strings in the file area are probably due to loading or saving of files being edited, or to commands to make a global change in the file.

The system-data area in the high-CMS area, from about page 190 to the top of the virtual machine, is divided into distinct subareas that are referenced together. Most of these subareas represent CMS-file system directories. These directories are being searched linearly for the resolution of file names. Very often, all directories are searched to resolve one file name.[2]

The references are very sparse, and are scattered over the entire address space. The utilization of virtual storage is quite low. For this kind of reference pattern, virtual storage is very appropriate since it requires only referenced pages to be resident in real storage. This pattern probably developed because CMS always ran on virtual memory systems. Systems where all user storage must be backed by real storage usually cannot afford this generous approach to memory usage.

The reference sets vary frequently and by large numbers of pages. This points to the need for an algorithm that can react quickly to changes in the reference set.

Many blocks of pages are often referenced together and are then dropped together from the reference set. In some cases, pages that are frequently referenced together are not adjacent in the address space. In fact, many of the reference gaps are as consistent as the pages referenced. A good storage-management algorithm can exploit these facts.

While at first glance the reference pattern appears to show some phases, at closer inspection these phases cover only a small extent of the overall reference pattern. The user-data area in particular appears to be referenced quite unpredictably. The low-CMS area and the file directories in the high-CMS area contain some groups of pages that appear to be referenced in phases, but these phases extend only over short ranges of pages. As we will see later, these phases are not significant enough for real-storage management to exploit.

[2] Starting with Release 4 of CMS/SP, the linear search has been replaced by a hashing scheme to resolve file names. This considerably reduces the working-set size. The reference pattern shows how large the impact of this change can be.

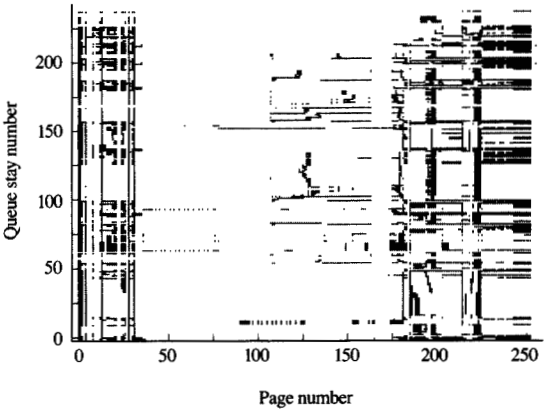

Figure 2

Page-reference pattern of VM 1.

Some special reference patterns

The page-reference pattern of VM 1 shows many important effects and serves well as an introductory example. The following sections show reference patterns from other virtual machines to present a more complete view of the reference patterns we have observed.

User programs and command sequences: An interactive application?

VM 2, as seen in **Figure 3,** shows the greatest user program activity of the virtual machines we examined. The shapes of the reference patterns in the user-program area are highly repetitive. They suggest that the user repeated similar, quite lengthy command sequences several times during the measurement period. A possible explanation for the regularity and repetition in the reference behavior is that this user executed many transactions within one interactive application. The regular use of the file-directory areas and the user-program and data areas supports this hypothesis. Within the repetitive patterns, we can observe phases and phase transitions, even though the working sets vary considerably within the phases. In all areas, many blocks are reused frequently together, a pattern that should lead to good performance of the real-storage-management algorithms. The reference pattern in the low-CMS area is very similar to the pattern shown by VM 1.

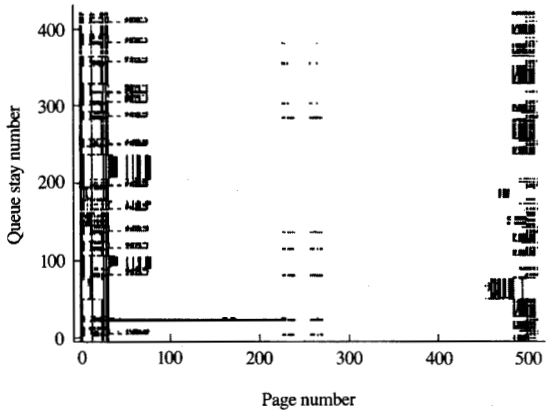

Figure 3

Page-reference pattern of VM 2.

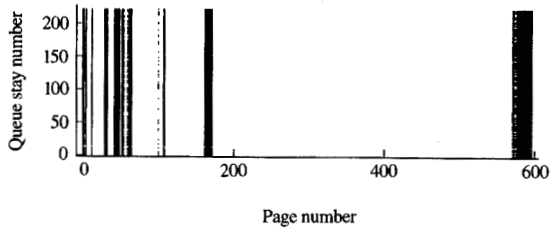

Figure 4

Page-reference pattern of VM 4.

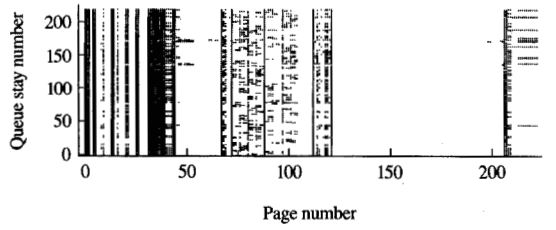

Figure 5

Page-reference pattern of VM 9.

Very repetitive behavior: Server machines?

VM 4 (**Figure 4**) shows the most regular behavior of any of the virtual machines we examined. While we do not know the function of this virtual machine, we conjecture that it performs a very limited function. A special-purpose server virtual machine, for instance, would produce a reference pattern like this. The constant storage usage in the user-program area, in the user-data area, and in the area of the CMS file directories supports this view. Even the low-CMS area shows no variation of the reference pattern. This predictable pattern will put only a minimum load on the real-storage-management algorithms. In terms of the phase/transition model, this user shows only one phase.

VM 9 (**Figure 5**) is a mail server. The pattern shows very consistent use of the low private CMS area, the user-program area, and the area where the file directories are located. It would appear that this virtual machine has only two private CMS minidisks. The user-data area shows some storage blocks whose boundary pages are referenced in every transaction but whose internal data are referenced quite sparsely and in a random pattern. A server machine does not necessarily have a completely predictable reference pattern like that of VM 4. From Table 1, we know that we captured only 55% of the reference sets of VM 9. This means that this machine had many large transactions spanning more than one scheduling interval.

A typical user

The reference pattern of VM 1 serves as an example because it shows most of the important issues quite well. But it is not typical for most of the interactive users observed. While it is difficult to establish a typical user pattern based on 15 users from three installations, VM 7 (**Figure 6**) represents an interactive user typical for our sample. VM 7 has a fairly regular reference pattern with some spurts of user-program executions, but the majority of the references occur in the CMS areas. In particular, there is a concentration of references in the high-system-data area where the CMS file directories are stored, from about page 850 to the top of virtual memory. The user-data area, here from about page 800 to page 850, is referenced heavily and regularly. The mean reference set of this user is considerably larger than the reference sets of the other virtual machines in this study. This is mainly due to the large number of file system directories the user has in memory. The references are quite consistent from one transaction to the next. At a glance, the reference pattern shows obvious phases.

User with phase behavior

VM 8 (**Figure 7**), belonging to an interactive user, shows the most distinct phases in its reference pattern of all patterns we examined. The long series of transactions with no changes in the reference set should lead to very good performance of the real-storage-management algorithms. These phases of

constant reference sets occur particularly when there is no user-program activity. The large reference gap between page 500 and page 1680 indicates an unusual allocation pattern. This area, more than four megabytes of storage, has been allocated before the area around page 400, whose reference pattern indicates that it probably contains file directories. The large area may still be allocated, or it may have been freed. This kind of storage fragmentation is unusual in the CMS environment.

• *Summary of reference patterns*

The CMS-data areas dominate the reference patterns. Most of these areas contain CMS system data and remain allocated through a user's session. They are referenced implicitly by the user's commands, and their reference pattern is consistent, very often for long sequences of transactions. Since a large part of the references are controlled by the operating system, the data structures of the operating system are crucial for the reference and paging behavior of the system.

The user data, referenced explicitly by the user's programs and commands, and the user-program references represent a smaller portion of the pages referenced. Their reference patterns are less predictable, except in server virtual machines or when a user remains in an interactive application for an extended time.

If the virtual machine provides interactive service to a user, the user's behavior ultimately controls the reference behavior. The user's command sequences explicitly reference his data and implicitly reference CMS system data. These data references together far outweigh the impact of the code references, even if the user is running programs in his own virtual machine rather than executing programs from the shared segments. Many data references are also due to I/O operations, rather than to references from the CPU. Because of the small impact of the code references, it is probably not worthwhile organizing short running programs to control code-reference behavior.

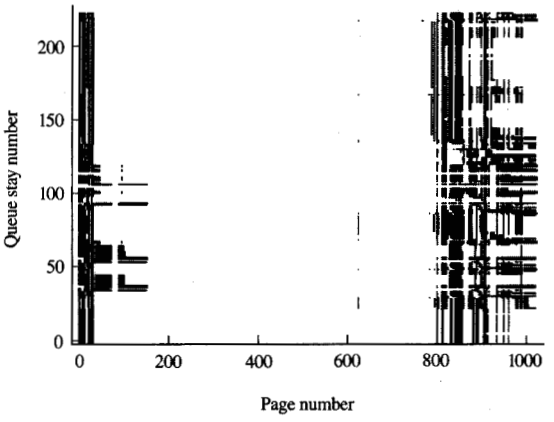

Figure 6

Page-reference pattern of VM 7.

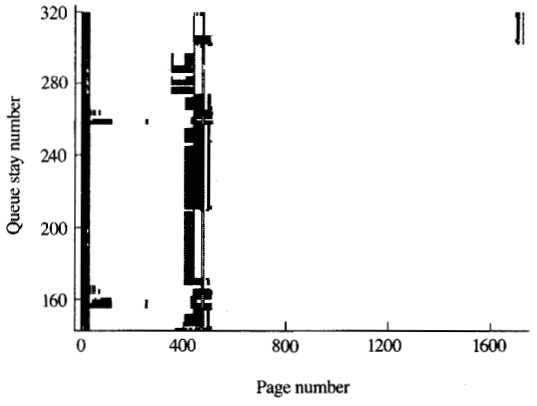

Figure 7

Page-reference pattern of VM 8.

Data analysis

• *Overview*

In this section, we present a statistical analysis of the reference data. **Table 2** gives an overview of some reference-set statistics of the virtual machines analyzed. The column *VM size* gives the virtual-machine size in megabytes. The column *Transactions* shows the total number of transactions the virtual machines handled during the recording interval. The column *RS size mean pages* gives the mean reference-set size in pages. In order to illustrate how the reference-set size varies, the next column, *RS size coeff. var.*, shows the coefficient of variation (i.e., the mean divided by the standard deviation) of the reference-set size. The columns referring to reference-set-contents change show the difference from one transaction to another of the pages that are in the reference set. The first of the two columns, *Mean chg./mean RS*, shows the mean number of pages by which consecutive reference sets change, as a fraction of the reference-set size. The second column, *Coeff. var.*, shows the coefficient of variation of the reference-set changes.

The mean reference-set sizes vary greatly between the virtual machines, but the reference-set size of a given virtual machine does not change very much in the course of a

Table 2 Reference-set statistics.

Virtual machine ID	*Characterization*	*VM size* (MB)	*Transactions*	*RS size*		*RS contents chg.*	
				Mean pages	*Coeff. var.*	*Mean chg. /mean RS*	*Coeff. var.*
VM 1	Initial example	1	296	37.2	0.68	0.67	1.12
VM 2	Interactive application	2	425	29.2	0.65	0.43	1.63
VM 4	Service machine	3	553	54.4	0.08	0.09	1.00
VM 7	Typical example	4	298	95.4	0.47	0.35	1.60
VM 8	Example of phases	7	357	83.7	0.35	0.17	2.18
VM 9	Service machine	1	400	38.3	0.20	0.35	0.71

session. The standard deviation of the reference-set size is smaller than the mean for all the virtual machines examined. The mean change in the reference-set contents appears to be closely related to the change in the mean reference-set size. Most of the time the change is either an addition of pages to the existing reference set or a loss of pages. Less often, there is an addition to one area balanced by a loss in another area. The reference patterns bear out this observation. The coefficient of variation of reference-set changes is generally larger than one. This points to a large variation in the working-set changes.

• *Reference and rereference data*

This section analyzes the reference frequency and the rereference frequency for each of the pages in the virtual machines. The *reference count* of a page is the number of scheduling intervals during the measurement interval in which that page has been referenced. To normalize the reference count, it is expressed as *reference frequency*:

$$\textit{reference frequency} = \frac{\textit{reference count}}{\textit{reference sets in measurement interval}}.$$

A *rereference* is defined as a reference of a page when that page has also been referenced in the immediately preceding scheduling interval. The rereferences are also expressed as frequencies, relative to the total number of references:

$$\textit{rereference frequency} = \frac{\textit{rereference count}}{\textit{reference count}}.$$

The rereference frequency is a measure of locality of reference. There are two types of rereferences. The first type comes from a very dense but random reference pattern. If a page is referenced in most transactions, then its rereference frequency must also be very high. The peaks in the reference graphs in Figures 9 and 10 (shown later) clearly are of this type. The other type of rereference is due to true locality of reference. This type of rereference is possible at either high or low reference frequency. At high reference frequency, the two types of rereference cannot be distinguished, and a distinction is not relevant. If a low reference frequency is accompanied by a high rereference frequency, the cause is true locality of reference. The difference between these two types may be important to real-storage-management algorithms that distinguish between short-term history and longer-term history. When a phase transition occurs, these algorithms should discard the short-term history, the pages referenced due to true locality of reference. But they should maintain the longer-term history, keeping the pages with high reference frequency over an extended time.

Figure 8 gives the mean number of pages referenced and rereferenced for each reference set in the measurement interval. **Figures 9** and **10** show the reference and the rereference frequencies of VM 1 and VM 2, respectively. The frequencies of the other virtual machines display no essential differences from the ones shown here.

The user-data areas in the two virtual machines show relatively high locality. However, due to the low reference frequencies of these areas, this locality has very little impact on real-storage-management performance. The reference frequency varies widely across the address range. This shows a potential that could be exploited by algorithms that take a more detailed reference history into account.

• *Phase/transition model*

This section explores whether phases of working-set use are prominent enough that they could be used by the real-storage manager. If real-storage-management algorithms are to exploit phases of more or less constant reference sets, a large percentage of all reference sets must be part of some phase. The phases of little change in the working set can be important for algorithms that rely on past history for their next decisions. When a phase transition occurs, the past history may not be relevant to future behavior. Algorithms for adjusting working sets and prepaging could take advantage of the phase/transition behavior of reference patterns. For instance, if phases with only small changes in the working sets are prevalent, swapping entire working sets may give better performance than demand paging. When the scheduler observes a phase transition, the prior working set

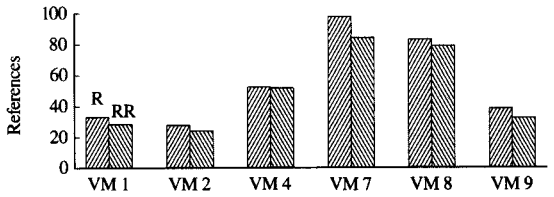

Figure 8

Mean referenced and rereferenced pages per reference set: R = referenced pages, RR = rereferenced pages.

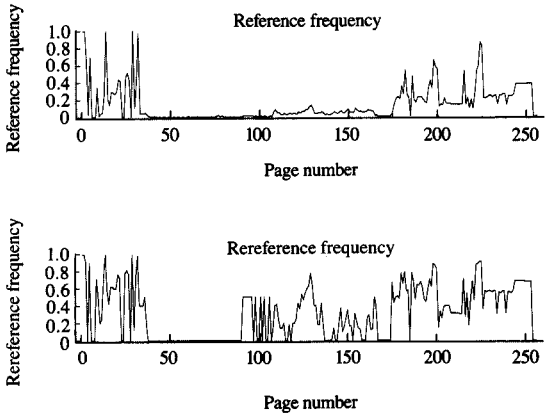

Figure 9

Reference and rereference frequencies of VM 1.

can be discarded and a switch to demand paging made. The phase/transition models reported in the literature, for instance in [3], define the phases on the basis of an analysis of individual programs. This approach is not feasible in our study. The phases must be discerned from a statistical analysis of the reference patterns.

We define the (relative) difference between consecutive reference sets as

$$\frac{|W_i - W_{i-1}|}{S_i} \cdot 100.$$

Since not all the reference sets have the same size, the change percentage can be larger than 100%. Now we can analyze the variation of the reference sets over sequences of transactions. We look first at the extents of the reference-set differences in subsequent transactions, and then at phases of reference sets.

Distribution of reference-set changes between consecutive transactions

Figure 11 shows the cumulative distribution of the differences between subsequent reference sets. The X-axis shows the percentage of reference-set changes. The Y values show the percentage of transactions in which the reference-set changes compared to the previous transactions are greater than or equal to the percentage given by their X values.

Because of the more predictable behavior of the server virtual machines (VM 4 and VM 9), the change distributions drop off sharply. The virtual machines experience practically no changes greater than 75%. The distributions for the other virtual machines drop off sharply only to the 10%-change mark. After that, they decline much more gradually, indicating a significant number of larger changes. These distributions have longer tails, showing considerable numbers of transactions with changes larger than 100%. These large changes may indicate a shift in user activity. VM

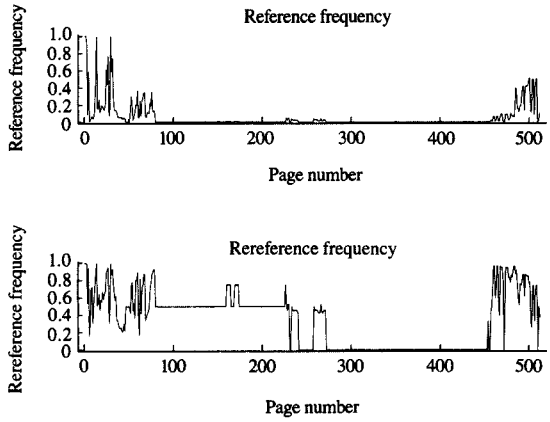

Figure 10

Reference and rereference frequencies of VM 2.

1 shows a nearly triangular form of the distribution. A triangular shape of the distribution would indicate a uniform distribution of the size of the reference-set changes.

Phase distributions

Figure 11 shows that despite the drastic drops in the distributions, there are many adjacent transactions between which the reference sets change only little. The reference patterns, though, show that any phases of completely

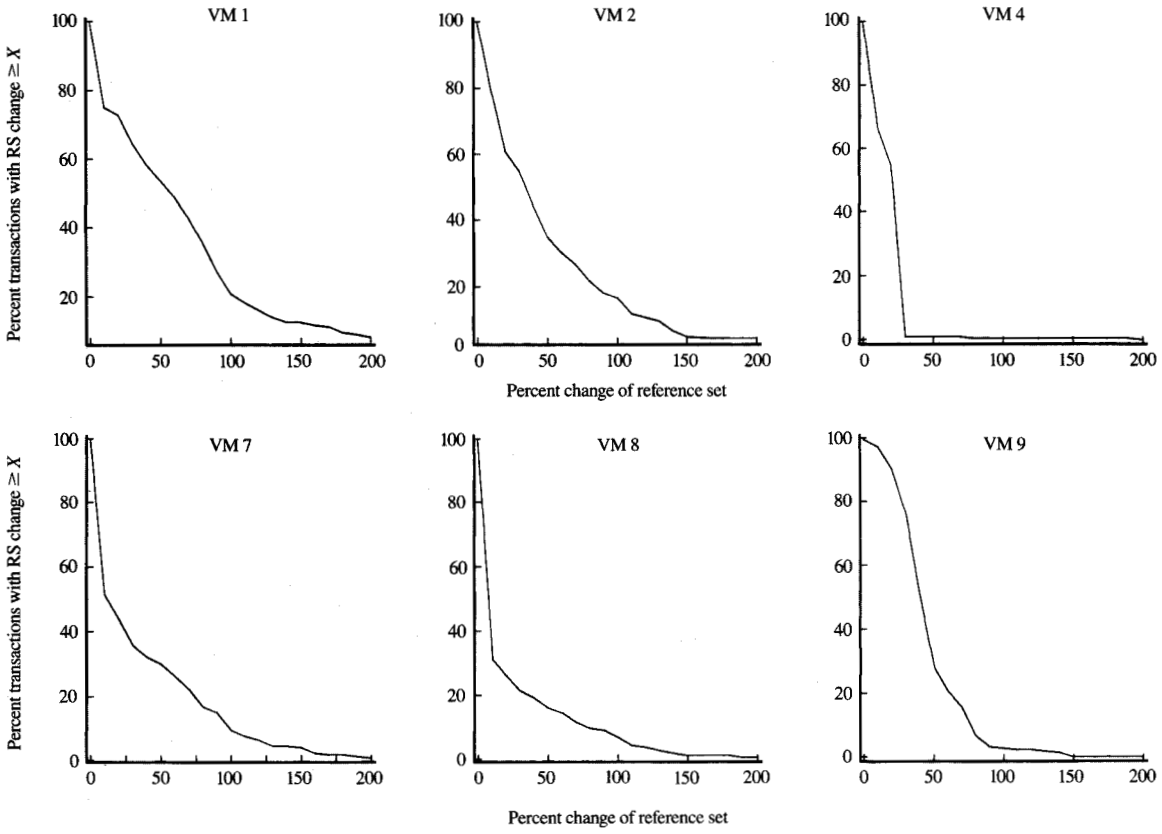

Figure 11

Cumulative distribution of reference-set changes between transactions.

identical reference sets are very short. For phases of reference sets to be significant, the period of little or no change should be considerably longer than just two transactions. Also, defining a phase as a series of reference sets that are completely identical is too restrictive to be useful. Consequently, we define a phase as a series of transactions whose reference sets change less than a given percentage between subsequent transactions,

$$\frac{|W_i - W_{i-1}|}{s_i} \leq CT,$$

where CT is a change-threshold.

Figures 12 and **13** show the extent of phases in the reference patterns of the six virtual machines of the analysis. Figure 12 shows the cumulative distribution of the phase lengths at several change-threshold values. The X-axis shows the phase length in transactions, and the Y-axis gives the percent of phases whose length is greater than or equal to X.

The phases in most virtual machines are very short. Even at change-thresholds as high as 60% of the reference set, most of the phases are shorter than five transactions. VM 4 is an exception. At a change-threshold of 20%, VM 4 spends the entire measurement interval in one phase.

Long phases can stretch over many transactions, so their impact on the system is much larger than the impact of short phases. Figure 13 takes this into account by showing the cumulative distribution of transactions according to the length of the phases to which they belong.

Using a change-threshold of 20%, we see that more than half of the transactions are part of phases that are five transactions or shorter. Even VM 8, whose reference pattern appears to show pronounced phase behavior, shows a substantial number of transactions in longer phases only at a 60% change-threshold. VM 4 again is an exception.

What does this evidence mean for the design of real-storage-management algorithms? An algorithm could, for

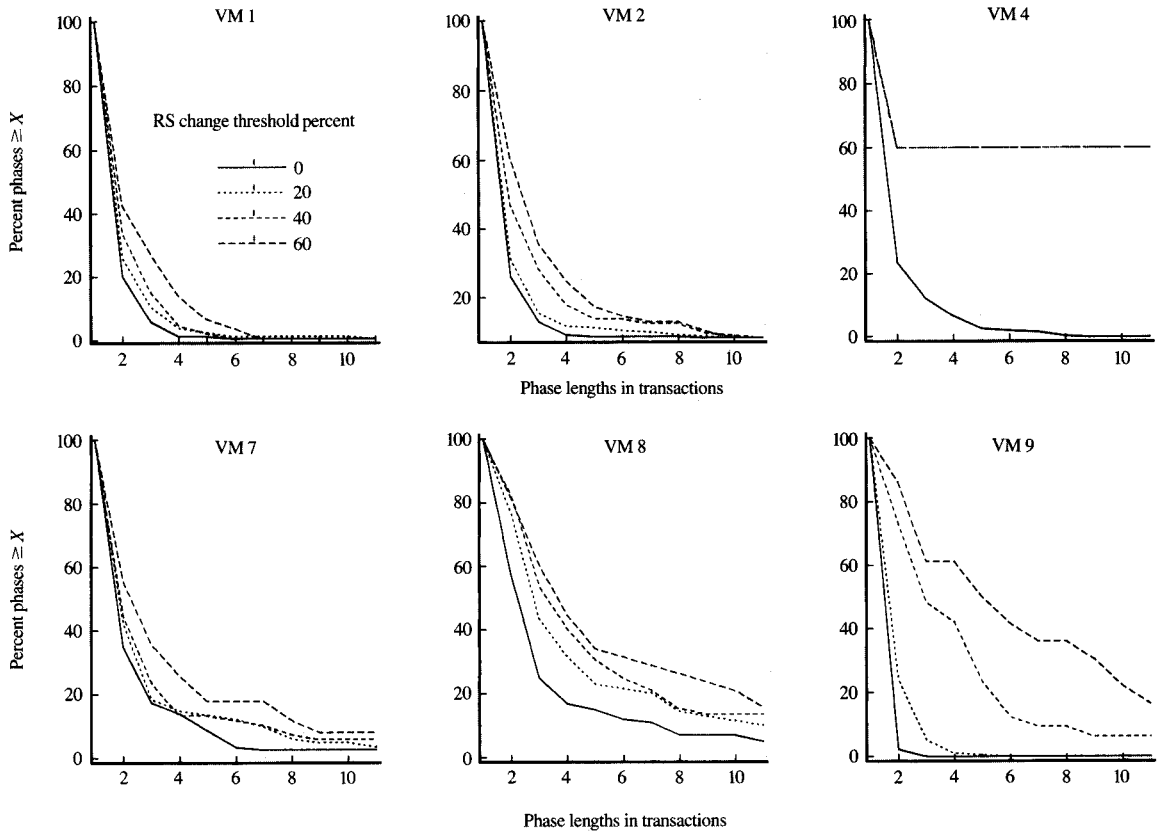

Figure 12

Cumulative distribution of reference-set phase lengths.

instance, exploit phases by swapping entire reference sets while the virtual machine is in a phase of little change, and by using demand paging at other times. For such an approach to be practical, the change-threshold should not be higher than 20%. Moving 20% more pages than needed represents a considerable overhead. Current algorithms often manage to predict reference sets to within 20% [7]. With a 20% change-threshold, most of the virtual machines have some phases that are quite long. However, the majority of the phases are so short that they would be difficult to exploit even if there were an algorithm that could detect the phase transitions instantly. If it took an algorithm several scheduling intervals to detect a phase transition, the resulting excess paging I/O traffic would be quite large. We know of no algorithm that reliably detects phase transitions—other than after the fact and with considerable CPU overhead.

For the exceptional cases where phases can be found, such as in VM 4, most algorithms will perform well even if they do not exploit the phases. Thus, there is no need for algorithms to detect phases in the few instances where they can be exploited.

In addition to the preceding analysis, we tried to determine whether the time-dependent behavior of the virtual machines yields any clues for predicting phase transitions. We produced a scatter plot of the reference-set changes between two subsequent transactions over the real time between the transactions. The plots showed no correlation between the two variables. That is, our sample showed no strong phases of reference behavior that are tied to time periods of user activity.

Conclusions

- *Locality of reference across transactions*

The data show that there is a great amount of locality of reference across transactions in interactive systems. Without

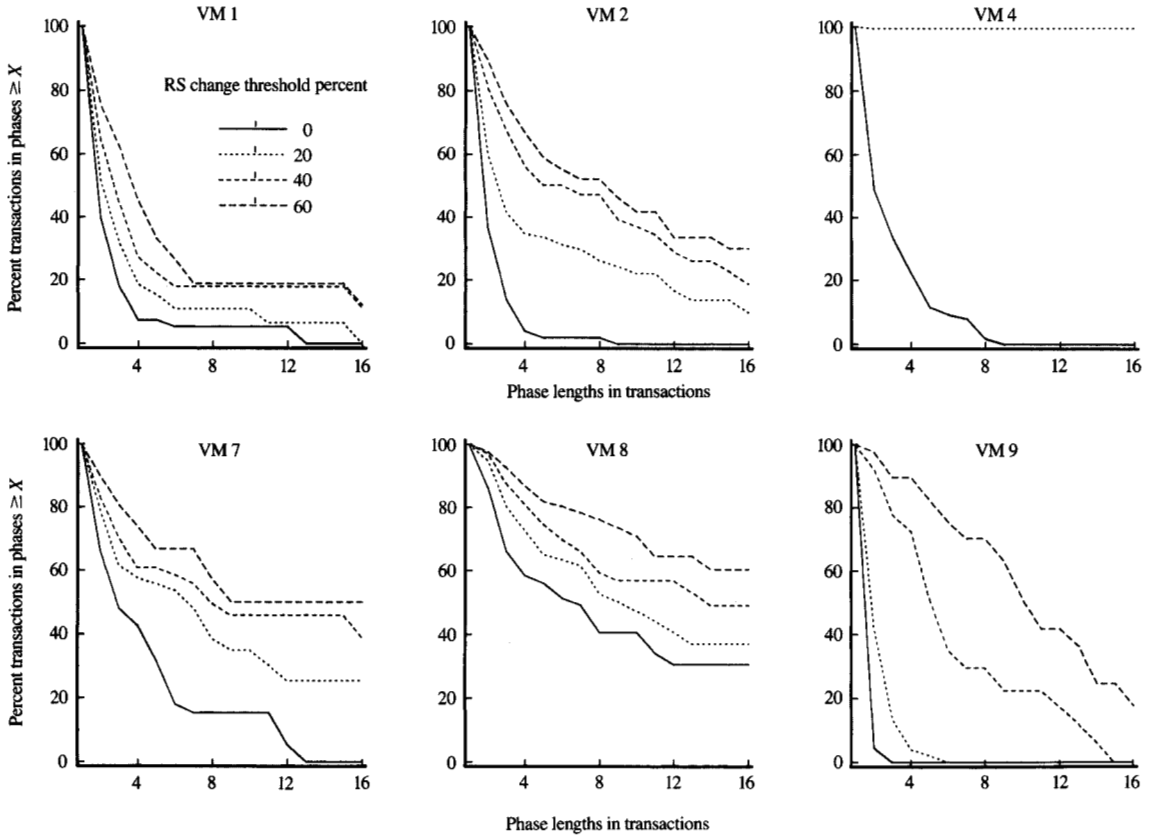

Figure 13

Cumulative distribution of transactions in phases.

this phenomenon, interactive time-sharing systems supporting hundreds of users, such as VM/CMS, would probably not be viable. In a way, the data presented in this study show explicitly what has been either known or assumed for some time.

This finding is relevant beyond time-sharing systems as well. It shows that in a single-user, interactive environment, the locality of reference is significant enough to make virtual-memory operating systems worthwhile. In a single-user workstation, the operating system code cannot be shared; thus, for this case the code references would have to be included in the reference pattern important for paging. But we believe this would not substantially change the overall reference behavior.

• *Dominance of operating-system references*

The reference behavior of highly interactive systems under VM is dominated by the data structures of the virtual operating system, and by the users' behavior, rather than by the behavior of the individual application programs. There are several specific memory areas that each have their own characteristic reference patterns. As the individual transactions get more computation-intensive, the application program and its data references will probably start to dominate the reference pattern. The missing data on long-running transactions in this study mask this effect in the presented data.

• *Dominance of data references*

In the environment we studied, the data references have a far larger impact on the overall reference pattern than the code references. Most of the code references are to shared code segments. Even if code references were counted, data references would probably be predominant, since I/O operations often cause many data references with few code references.

The data structures for shared code must be defined very carefully, since their impact on system performance increases with the degree of sharing.

• *Applications*

VM 4 and VM 9 are both server virtual machines, yet their reference behavior differs radically. Often, server virtual machines constitute a large part of the load on a system, and sometimes they are very ill-behaved. The method of analysis shown in this paper could be used to analyze the reference patterns of such server machines and to improve their reference behavior by redesigning internal data structures. It appears that VM 9 could benefit from such a redesign.

• *Phase-transition model*

The data presented exhibit some phase behavior. But the phases are not pronounced enough for the real-storage-management algorithms to exploit. Even if the phases were more distinct, for effective use the algorithms would have to be able to recognize phase transitions quickly and with little overhead. Considering the data we found and this problem for the algorithms, exploiting the phase behavior appears to be a direction that is not worthwhile pursuing.

Acknowledgments

We would like to express our appreciation to Tom Beretvas, Larry Brenner, Paul Van Leer, and Jerry Spivak. Through their discussions of the algorithms, the analysis, and the results, they contributed greatly to this paper. We would also like to thank David Potter, who helped us untiringly in dealing with the VM Monitor data and with his support of the GRIN system.

References

1. P. J. Denning, "Working Sets Past and Present," *IEEE Trans. Software Eng.* **SE-6,** No. 1, 64–84 (January 1980).
2. M. C. Easton and B. T. Bennett, "Transient-Free Working-Set Statistics," *Commun. ACM* **20,** 93–99 (February 1977).
3. A. P. Batson and W. Madison, "Measurements of Major Locality Phases in Symbolic Reference Strings," ACM SIGMETRICS and IFIPS W.G. 7.3 Conference, March 1976, pp. 75–84.
4. Y. Bard, "Performance Analysis of Virtual Memory Time-Sharing Systems," *IBM Syst. J.* **14,** No. 4, 366–384 (1975).
5. R. B. Hagman and R. S. Fabry, "Program Page Reference Patterns," *Proceedings of the 1982 SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Seattle, 1982; *ACM SIGMETRICS Perform. Eval. Rev.* **11,** No. 4 (1982–1983).
6. H. A. Anderson, G. L. Galati, and M. Reiser, "The Classification of the Interactive Workload for a Virtual Memory Computer System," *Research Report RC-4727*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1974.
7. M. G. Kienzle, J. A. Garay, and W. H. Tetzlaff, "Analysis of Page Reference Strings of an Interactive System," *Research Report RC-11928*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1986.
8. *VM/SP HPO System Programmer's Guide*, Order No. SC19-6224; available through IBM branch offices.
9. *VM/SP CMS User's Guide*, Order No. ST00-1584; available through IBM branch offices.
10. *IBM System/370 Principles of Operation*, Order No. GA-7000; available through IBM branch offices.
11. T. Beretvas and W. Tetzlaff, "Paging Enhancements in VM/SP HPO 3.4," *IBM Washington Systems Center Technical Bulletin*, Order No. GG22-9367, May 1984; available through IBM branch offices.
12. D. Potter and W. Tetzlaff, "Generalized Reduction of Information," *Research Report RC-6676*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1977.
13. G. Burkland, P. Heidelberger, P. Welch, L. Wu, and M. Schatzoff, "An APL System for Interactive Scientific-Engineering Graphics and Data Analysis," *Proceedings of APL 84*, Finland, June 1984.

Received July 23, 1986; accepted for publication September 6, 1987

Martin G. Kienzle *IBM Research Division, T. J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598.* Mr. Kienzle received a Diplom in Informatik from the University of Karlsruhe in 1976, and an M.Sc. in computer science from the University of Toronto in 1977. He joined IBM in 1978 at the Thomas J. Watson Research Center in Yorktown Heights, where he has been working on performance measurements for operating systems, and on operating-system structures. Mr. Kienzle is currently the manager of the Supervisor Kernel Studies group. His main interests are in operating-system structures and primitives for multiprocessors. Mr. Kienzle is a member of the Association for Computing Machinery and the Institute of Electrical and Electronics Engineers.

Juan A. Garay *Computer Science Department, The Pennsylvania State University, University Park, Pennsylvania 16802.* Mr. Garay received the degree of Electrical Engineer from the Universidad Nacional de Rosario in Rosario, Argentina, in 1975, and the Master of Electronic Engineering degree from the Netherlands Universities Foundation in Eindhoven, the Netherlands, in 1981. He also worked for IBM in Argentina as a systems engineer. At present, he is a doctoral candidate in computer science at The Pennsylvania State University. In 1985, he spent six months at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, working on the page-reference-pattern problem.

William H. Tetzlaff *IBM Research Division, T. J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598.* Mr. Tetzlaff joined the Service Bureau Corporation in 1966. In 1969, he joined the Research Division. He has done research in the areas of information retrieval, system performance, capacity planning, file systems, and paging subsystems. Mr. Tetzlaff has published many papers on that research, and has received two IBM Outstanding Contribution Awards for his work, the first for the Statistics Generating Package and the second for a prototype page-swapping subsystem for VM. He is a frequent speaker on system performance and paging at SHARE, GUIDE, and CMG meetings. Mr. Tetzlaff studied engineering sciences at Northwestern University, and he is a graduate of the IBM Systems Research Institute. He is currently Senior Manager of System Structure and File Systems in the Computer Sciences Department.