Windows NT page replacement policies

Mark B. Friedman
Demand Technology
1020 Eighth Avenue South, Suite 6
Naples, FL USA 34102
markf@demandtech.com

Abstract.

The Microsoft Windows NT operating system supports virtual memory on a variety of hardware platforms, the most common being the Intel IA-32 architecture. This paper provides an overview of Windows NT virtual memory management, with a particular emphasis on the hardware-independent mechansim it uses for page replacement.

The Microsoft Windows NT operating system is a demand paged, virtual memory operating system. NT currently supports 32-bit virtual memory addresses on Intel-compatible and Digital Alpha processors. This paper describes the virtual memory management features of Windows NT and its implications for performance monitoring and capacity planning. Because Windows NT runs primarily on PC architectures which have limited I/O bandwidth, ensuring there is ample physical memory is one of the most important configuration and tuning options.

The first section of this paper provides an overview of the NT Virtual Memory Manager. The second section describes the page replacement policies implemented in NT. The third section describes the memory management performance measurement data that NT provides and also discusses practical strategies for sizing and tuning Windows NT systems.

Virtual addressing.

Virtual memory is a feature common to most advanced processors that uses a hardware mechanism to map from logical (i.e., virtual) memory address that application programs use to real physical memory addresses [1]. The logical memory address range of an application is divided into fixed size chunks called *pages* that are mapped to similarly-sized physical page frames that are resident in real memory. This mapping is dynamic such that logical addresses that are frequently referenced tend to reside in physical memory, while infrequently referenced pages are relegated to secondary disk storage. The resident set of virtual memory pages is called the process's *working set* because those are its currently active pages.

Virtual addressing is designed to be transparent to application programs, which can then be written without regard to specific real memory limitations of this or that computer. Since their introduction in the late 60's, virtual memory architectures quickly came to dominate the computing environment due to their flexibility compared to the fixed memory partitioning schemes that preceded them. One pervasive concern that virtual memory schemes encounter is that the performance of application programs may suffer when there is a shortage of real memory in which to execute. Because real memory is allocated to executing programs on demand, there is also a sense that programs compete for what physical memory is available. The inevitable result of competiton when memory is a scarce resource is that the memory access pattern of one program can unduly influence others.

Virtual memory addressing is decidedly not transparent to operating systems. The OS must perform a number of critical functions to enable the scheme to work. Chief among them is the responsibility for constructing and maintaining address space virtual-to-real memory translation tables that the hardware refers to in the course of executing instructions. The function of these *page tables* is to map logical program virtual addresses to real memory

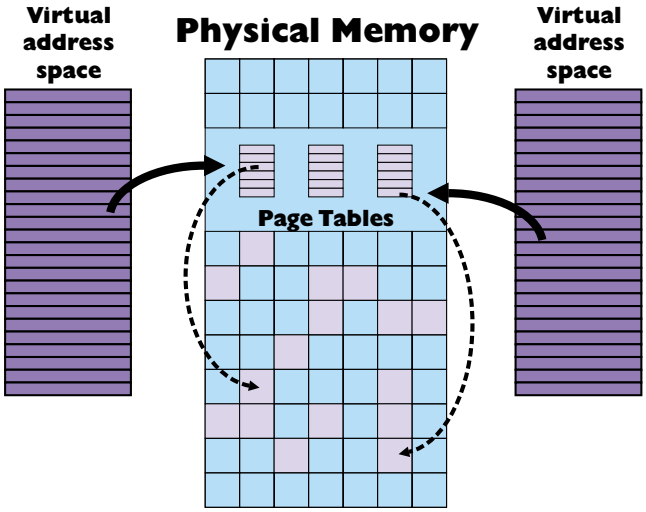

FIGURE 1. VIRTUAL ADDRESSING. VIRTUAL MEMORY IS A FEATURE COMMON TO MOST ADVANCED PROCESSORS THAT SUPPORTS MAPPING PROGAM LOGICAL ADDRESSES INTO PHYSICAL MEMORY. THE LOGICAL MEMORY ADDRESS RANGE OF AN APPLICATION IS DIVIDED INTO FIXED SIZE CHUNKS CALLED PAGES THAT ARE MAPPED TO SIMILARLY-SIZED PHYSICAL PAGE FRAMES THAT ARE RESIDENT IN REAL MEMORY. PAGE TABLES STORED IN MEMORY PERFORM THE MAPPING.

FIGURE 2. A WINDOWS NT OPERATING SYSTEM WARNING MESSAGE ISSUED WHEN OVERALL VIRTUAL MEMORY COMMITTED BYTES REACH A THRESHOLD OF 90% OF THE VIRTUAL ALLOCATION COMMIT LIMIT, CALCULATED AS THE SUM OF RAM AND THE TOTAL PAGING FILE SPACE.

locations, as illustrated in Figure 1. Another key role the operating system plays is to manage the contents of real memory effectively. This requires implementing a *page replacement policy* to ensure that frequently referenced pages remain in real memory and the handling of *page faults* when a program accesses a virtual address which is not currently mapped to real memory.

Windows NT builds page tables on behalf of each process when it is created. The page tables potentially map the entire 32-bit 4 GB virtual process address space depicted in Figure 1. The Win32 **VirtualAlloc** API call provides both for *reserving* contiguous virtual address ranges and *committing* specific virtual addresses. Committing virtual memory addresses causes the NT Virtual Memory Manager (VMM) to construct a page table entry to map the address into physical RAM, or, alternatively, to one or more paging overflow files that reside on disk. Since

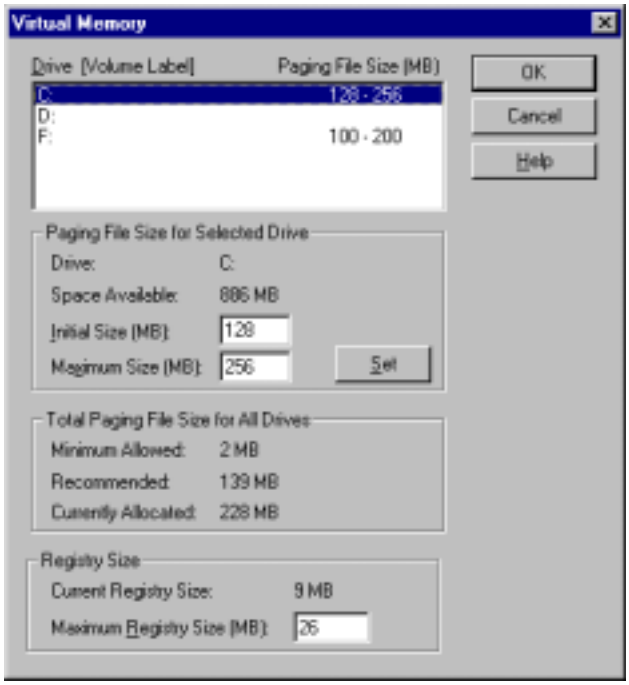

FIGURE 3. THE VIRTUAL MEMORY DIALOG BOX, ACCESSED FROM THE PERFORMANCE TAB OF THE SYSTEM PROPERTIES, PROVIDES FOR THE DEFINITION OF ADDITIONAL PAGING FILES OR CHANGING THE ALLOCATION BOUNDARIES OF EXISTING ONES. UP TO 16 PAGING FILES CAN BE ALLOCATED ON SEPARATE LOGICAL DISKS. PAGING FILES CAN BE ADDED OR CHANGED ON THE FLY.

page table entries themselves are built upon demand, all other unreserved virtual memory addresses are termed *free*.

NT maintains a *Commit Limit* across all process allocated virtual memory. This prevents page tables from being built for virtual memory pages which will not fit in either physical RAM or the paging files. The Commit Limit is the sum of the amount of physical memory and the allotted space on the paging files. NT currently supports a maximum of 16 paging files, each of which must reside on distinct logical disk partitions. Each paging file can be as a large as 4 GB. When the virtual memory workload of Committed Bytes exceeds 90% of the Commit Limit, NT issues a distinctive warning message to the Console, illustrated in Figure 2. Following the instructions in the message directs the Operator to the Performance Tab (see Figure 3) of the System applet in the Control Panel where additional paging file space can be defined.

The Virtual Memory paging file dialog box allows for defining a range of allocated paging file space for each paging file defined. After several seconds of processing above the 90% Commit Limit without an operator intervention occurring, Windows NT will automatically extend any paging files that are defined with flexible extents, subject to space being available on the specified logical disk. Windows NT performance experts [2, 3, 4] are in agreement that such a unilateral action jeopardizes the performance of the system, subjecting subsequent paging file operations to the potential of crossing noncontiguous segments of the paging file. This in turn leads to time-consuming embedded disk seek operations. The experts unanimously recommend that paging file allocations be strictly limited so that the initial allocation cannot be extended. This prescription presumes that initially paging file allocations themselves acquire a set of physically contiguous disk sectors. This is often a safe assumption for the primary paging file built by the Windows NT installation program on the same logical disk partition as the operating system itself when the logical disk is normally in a pristine state. (The initial paging file is built with a minimum allocation equal to the amount of physical memory plus about 15 MB. It is defined by default so that it can extend to approximately 2X the size of RAM.) This may not be a good assumption for paging files which are added subse-

quently, which are likely to be subject to some degree of fragmentation.

While in no way trying to minimize the performance impact of paging operations forced to seek long distances from one noncontiguous disk sector to another associated with a fragmented paging file, I invite Readers to consider the alternative. Not being able to extend the paging files when the system passes the 90% Commit Limit exposes running programs to the possibility of virtual memory allocation failures, a contingency that few programs allow for. Not allowing the paging files to expand under stress redirects that stress onto applications, which will surely fail as the absolute Commit Limit looms. So long as the virtual memory allocation peak itself is transient, allowing the paging files to expand at least ensures that the system stays up. For my money, system availability is always worth preserving, even at some sacrifice of optimal performance levels.

A more proactive approach using continuous performance monitoring would alert the performance analyst when **% Committed Bytes** exceeds a 70% threshold value. If you are operating a system above this level of memory utilization, it is appropriate to explore longer term remedial measures that might involve (1) adding more physical memory, (2) defining additional paging file space, or both. Computers systematically exceeding a 70-80% Committed Bytes threshold may suffer from serious performance degradation due to excessive demand paging activity anyway.

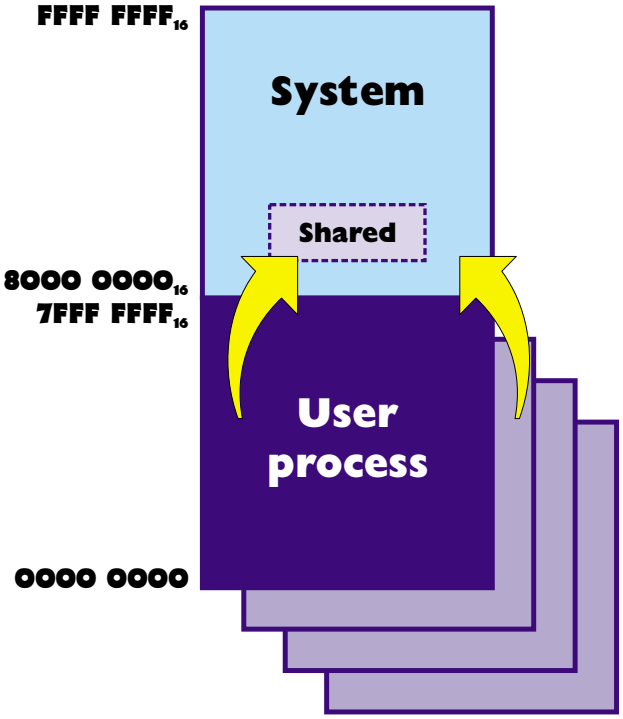

Figure 4. Per Process virtual address spaces in Windows NT. The 4 GB virtual address range granted to each process is divided into 2 equal parts. The lower 2 GB range is associated with private process addresses. The upper 2 GB range is associated with commonly addressable system addresses. No process can access private virtual memory locations associated with different processes. The shared system addresses are protected from inadvertant access by user threads because the storage is accessible only by authorized kernel threads running at Ring 0 priority. Shared memory mapped file Objects can be created in system addresses and accessed by multiple user processes to facilitate moving data between different user processes.

Process virtual address space. As indicated above, each NT Process acquires a virtual address space of 4 GB, divided conventionally into two equal parts as depicted in Figure 4. The lower 2 GB of each process address space is private. This range of addresses refers to pages that can only be accessed by threads running in this process address space *context*. The upper 2 GB of each process address space maps common system addresses. (Note: Windows NT Enterprise Edition permits a different partitioning of user and system addressable storage locations which extends the private User address range to 3 GB and shrinks the system area to 1 GB.) While addresses in the system range are commonly accessible from threads running in each and every process, system addresses are allocated using supervisor (Ring 0 on Intel hardware) authority, which restricts memory access to kernel threads with similar authority. In this fashion, application program threads running in user mode are unable to access virtual memory locations associated with different process address spaces and are also prevented from directly accessing commonly addressable virtual memory locations associated with the operating system. Specific virtual memory management API functions are used to allocate portions of common addressable system areas to share data between two or more distinct processes, as illustrated. The most popular technique for doing this in NT is for one process to map a file into commonly addressable system storage and point one or more additional processes to the range of system virtual addresses associated with the mapped file Object.

Virtual address translation. The hardware defines the precise mapping function that is used to translate a running program's logical "virtual" addresses into real physical memory addresses. Hardware specifications include (1) the mechanism by which the specific virtual translation *context* is established for specific address spaces, (2) the format of the actual translation tables, and (3) the method for notifying the operating system that page faults have occurred. The Intel Architecture which dominates the Windows NT computing platform supports 32-bit virtual addresses. Intel's IA-32 processor architecture mandates the format of the page tables that the Windows NT operating must maintain to enable the computer to perform this virtual-to-real address translation. [5]

Figure 5 shows the specific IA-32 mechanism for translating virtual to real addresses. Translation generally occurs using a 4K page. (So-called large 1 MB pages are

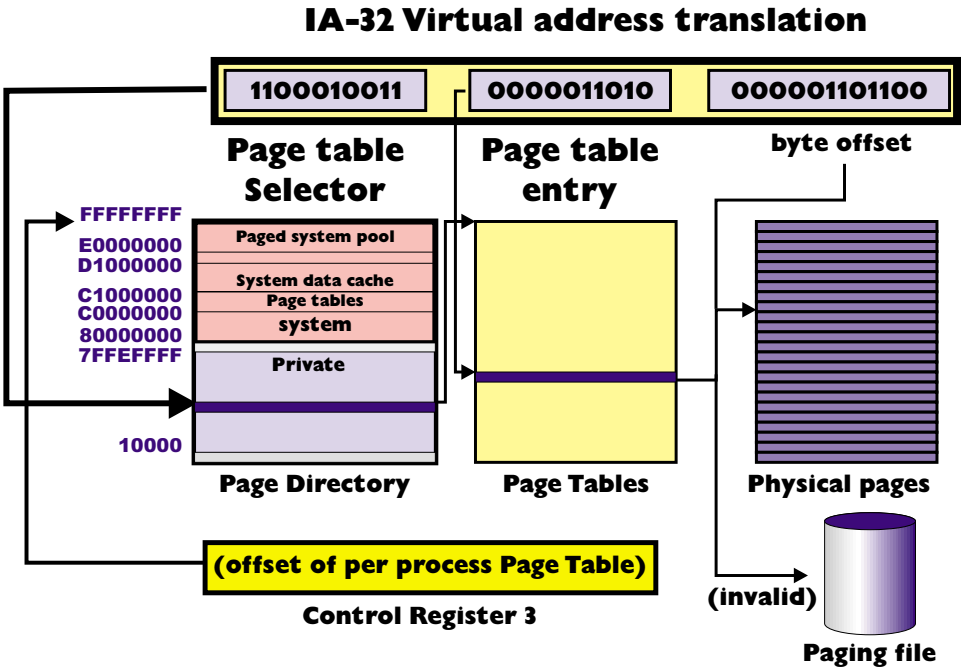

Figure 5. Virtual address translation. Translating virtual addresses to real ones is a hardware function that requires specific operating system support. The OS maintains page tables which the hardware uses to translate virtual addresses to real ones. NT divides the 4 GB addressable range of virtual storage into two sections. The upper 2 GB of addressable storage is reserved for NT itself and is common to every process. The virtual addresses reserved for a few key operating system data structures are inidcated. For example, the file system cache is allocated to a 512 MB range of virtual addresses between x'C1000000' and x'D0FFFFFF'.

The operating system is responsible for loading the address of the origin of the page tables for the currently running process in Control Register 3 on an Intel x86 processor. Virtual address translation takes place in hardware which uses the first ten bits in the virtual address as an index into the Page Table Directory to obtain a pointer to a specific set of Page Tables. The second ten bits of the virtual address are used as an offset into the Page Tables to locate a specific Page Table entry. The Page Table entry contains the corresponding real memory address of the page, if the page is resident in memory. If the page is invalid, the entry specifies a paging file address where it can found.

also supported, which NT uses for mapping the operating system kernel.) A two level, hierarchical indexing scheme is employed using a Page Directory, which then points to the page tables themselves. The Page Directory resides in a single 4K page which is always resident in memory while the process executes. Internal Control Register 3 points to the origin of the Page Directory. Page Tables, themselves also 4K in size, are built on demand as virtual memory locations are committed. These consist of 32-bit Page Table Entries (PTEs) which contain the physical memory address where the page of virtual addresses is currently mapped. Each Page Table can map 1024 4K pages (a 4 MB range), while the Page Directory can point to 1024 Page Tables. The combination supports the full 4 GB addressing scheme.

As the processor hardware encounters virtual addresses, representing either instructions or data areas, it performs a table look-up. The first 10 bits of the virtual address are used as an index into the Page Table Directory to locate the Page Table associated with that 4 MB range. The second 10 bits are used to index into the Page Table to find the PTE. The PTE entry contains the high order twenty bits of the real memory address of the page. The low order twelve bits from the original virtual address, capable of representing offsets 0-4095 into the page, are added to the PTE entry to form the full 32-bit physical memory address.

Since virtual address translation is a repetitive task, significant performance benefits are achieved by buffering these mapping tables in very fast memory on board the processor chip. Like other computer architectures that support virtual memory, Intel processors provide hardware Translation Lookaside Buffers (TLBs) to speed up virtual address translation. A *context switch* occurs when Control Register 3 is reloaded to point to a new set of per process Page Tables. The performance impact of a context switch encompasses flushing the TLB, slowing down instruction execution for a transitional period known as a cache *cold start*.

The PTE also contains a "Valid" page flag which indicates whether or not the virtual memory address has a counterpart in physical memory. In the course of performing virtual memory addressing, whenever the processor encounters a PTE with the invalid bit set it generates an interrupt. It is up to the operating system to determine the root cause of the virtual address translation exception. It is possible the executing program has merely referenced a virtual address which exists, but is not physically present. This is a *page fault*. The operating system is then responsible for fetching the designated page from the paging file where it is stored, copying it into some available physical memory slot, and re-executing the failed instruction. Alternatively, due to a program error, the address specified may not actually be a valid one, i.e., one that was never allocated in the first place. Upon determining that this sort of (very common) programming error has occurred, the operating system performs a diagnostic memory dump and terminates the application program. Windows NT, for example, writes Access Violation diagnostics to the end of

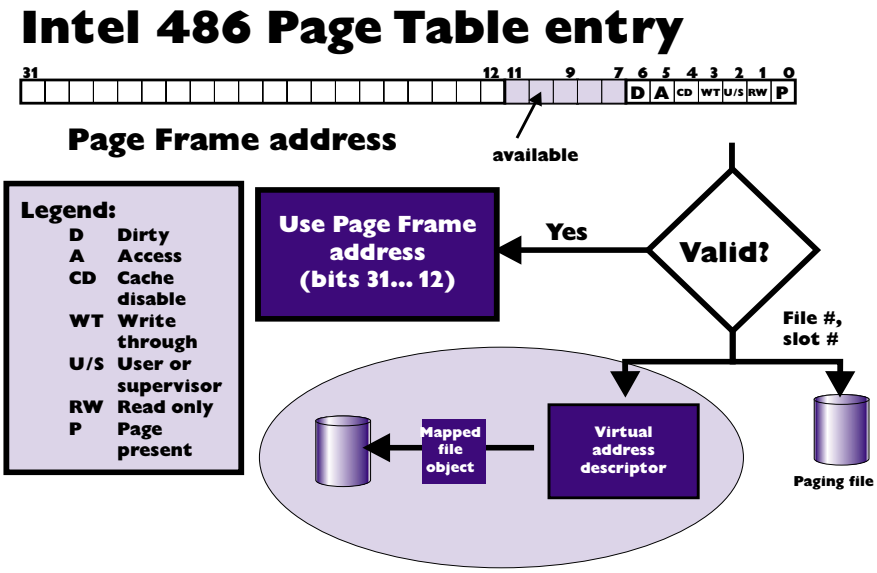

Figure 6. Under Windows NT, an Intel IA-32 Page table entry points to either (1) a real page in memory if the page is valid, (2) a location in the paging file, or (3) a Virtual Address Descriptor (VAD). Bit 0 of the entry is the "Present" bit that indicates whether of not the virtual address is currently resident in physical memory. Bits 1-6 are used by the hardware to maintain additional status information about the real memory page. Bit 2, for example, is set to prevent programs executing in User moide (Ring 3) from accessing authorized operating system locations allocated by kernel threads running in Ring 0. The dirty bit is set whenever the contents of a page are changed. The operating system refers to the Dirty bit during page replacement to determine if the copy of the page on the paging file is current.

the Dr. Watson log file, **drwtsn32.log** stored in the **%system32** root directory.

To assist in trapping common programming mistakes, Windows NT deliberately designates x'0001 0000' as the starting point for all programs, marking the first 64K of virtual memory addresses as invalid. A common programming error is to reference an inadvertently null pointer, leading to an address calculation that points to low storage. By flagging the Page Table entries for the first 64K of each process address space as invalid, NT is able to catch many of these errors before they are destined to do serious damage. The 64K boundary is a legacy of Intel's 64K segmented virtual addressing scheme for the original 80286 processors.

As illustrated in Figure 6, the Intel IA-32 hardware PTE contains a number of status bits. Bit 0 of the entry is the "Present" bit that indicates whether of not the virtual address currently resides in physical memory. Bit 0 determines how to interpret the remainder of the information stored in the PTE. If Bit 0 is set, the interpretation of the other bits is determined as shown. For a valid page, the high order twenty bits of the PTE reference the address of the physical memory location where the page resides. Bits 1-6 are maintained by the hardware to give additional status information about the real memory page. Bit 2, for example, is an authorization bit set to prevent programs executing in User mode (Ring 3) from accessing operating system memory locations allocated by kernel threads running in Ring 0. The "Dirty" bit 6 is set whenever the contents of a page are changed. The operating system refers to the Dirty bit during page replacement to determine if the copy of the page on the paging file is current. Bit 5 is an "Access" bit that the hardware sets whenever the page is referenced. It is designed to play a role in page replacement, and it is used for that purpose by Windows NT under circumstances which will be described shortly. Likewise, Windows NT turns off the Read/Write bit to protect code pages from being overwritten inadvertently by executing programs. NT does not utilize additional Intel hardware status bits designed to influence the behavior of the processor cache.

The Page Table format used in the Digital Alpha, which Windows NT also supports, is conceptually similar, but somewhat different in its implementation. In 32-bit addressing mode, the Alpha uses 8K pages. The Page Directory lookup uses the first 8 bits of a virtual address, while the next 11 bits are an offset into the Page Table entries. These architectural differences were crucial to the design of the NT page replacement policy, which was intended to function on both Intel and Digital Alpha hardware. The Alpha also maintains a similar set of page status bits, with at least one striking difference — there is no "Access" bit. When deciding how to replace pages on the Alpha, the operating system gets no information from the hardware on an executing program's pattern of reference to its virtual addresses. The Alpha does not use separate Write protection and Dirty bits either; it uses one Write bit that essentially serves both functions.

Portability. What is unique about the approach to virtual memory management that NT employs is that it is designed to satisfy processor specific-hardware requirements, while retaining a large degree of hardware independence. Over its relatively brief lifetime, versions of Windows NT have been developed that support quite distinct and different MIPS, PowerPC, Digital Alpha, and Intel IA-32 processor architectures. The NT component responsible for achieving a high degree of portability is known as the Hardware Abstraction Layer, or HAL. The NT OS kernel only interfaces to hardware functions like those that are required to perform virtual memory man-

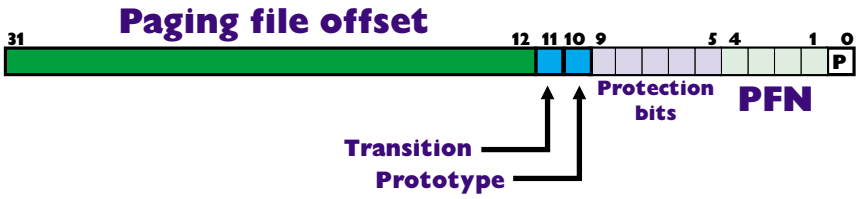

Figure 7. The format of an invalid PTE. When the page is invalid, NT stores a 20-bit paging file offset and a 4-bit paging file number (PFN) to show where the page is stored in the paging subsystem. The operating system uses the Transition bit to determine memory access patterns. Prototype PTEs are used for, among other things, mapping shared memory pages into multiple process address spaces.

agement by calling HAL interfaces, which, in turn, are performed by processor-specific code. Porting Windows NT, which is in the main written in "C" language, requires first a "C" language compiler for the architecture, and then developing a processor-specific HAL module.

One example of NT's platform independence is the way it utilizes the storage locations associated with PTEs when they are *not* subject to the hardware's strict specification. When a PTE is invalid, NT is able to store processor-independent information regarding the page, as illustrated in Figure 7. The format is the same whether NT is running on Intel or Alpha hardware. Information about where the page is located within the paging subsystem is stored in the PTE for "invalid" pages, as shown. The paging file number (PFN) is a 4-bit index that is used to reference up to 16 unique paging files. The 20-bit paging file offset then references a specific paging file offset up to x'10000' or 1 million. A transition bit is maintained by the operating system for determining which pages in a process working set are active. Its role in page replacement is discussed below. The Prototype PTE is a mechanism for mapping shared memory pages into multiple process address spaces. Its use is beyond the scope of the current discussion.

Maintaining a high degree of portability across a wide variety of processor architectures means ignoring opportunities to exploit some unique processor-specific hardware features. This is a temptation that operating systems developed and maintained by hardware manufacturers are usually unable to resist, something that has undermined the goal of portability espoused by the original developers of Unix. What Windows NT may sacrifice in performance by not being optimized around any specific computing hardware, it gains in portability. The goal of portability has influenced the development of the page replacement policies that NT employs to manage virtual memory. The NT page replacement policy was designed to work with processor hardware such as the Digital Alpha which does not maintain a hardware Access bit. Early versions of the operating system ignored the hardware Access bit even where it was available, namely on Intel processors. Later releases of NT, beginning with version 4.0, do make use of the Intel hardware Access bits under some circumstances, the inevitable result of Microsoft's decision to optimize NT for its most widely used hardware platform.

Page Replacement.

A common problem virtual memory operating systems face is what to do when a page fault occurs, a valid page must be retrieved from the paging file, and there is little or no room in physical memory for the referenced page. Following a policy of allocating real memory page slots *on demand* as they are referenced inevitably leads to a situation where physical memory fills up. Then, when a new page is referenced, something has to give. The general solution adopted by operating system designers involves identifying currently resident memory pages which have not been referenced recently and removing them from memory. (Before "dirty" pages can be removed, the operating system must first copy their contents to the paging file.) This popular solution to the page replace problem is designated Least Recently Used or LRU, which captures the overall flavor of the strategy. LRU, strictly speaking, is not optimal. Its performance can be improved upon substantially if an executing program provides "hints" with regard to its expected pattern of memory reference. In the absence of specific hints from the program, however, LRU generally performs well over a wide range of circumstances. In particular, its behavior is very well understood since it is practiced in a variety of contexts not limited to just virtual memory management. [6]

A popular variant of LRU for virtual memory page replacement is known as the "Clock" algorithm. This is practiced in both MVS and Unix, for example. Every clock interval (in MVS, once per second), an OS memory management function resets all hardware access bits. In the process, the OS accumulates information about how long pages resident in physical memory have gone unreferenced. This aging information about resident pages is then used to order them by the time of their most recent reference. Using this ordering, page replacement proceeds by replacing the oldest resident pages first. Least Recently Used pages are subject to replacement when physical memory is depleted. The Clock algorithm has known limitations, not the least of which is its difficulty scaling to very large memory configurations. For instance, the aging algorithm executes in time proportional to the size of it resident page frame database. An optimization introduced in the IBM mainframe MVS operating system, for example, varies the frequency of Clock updates based on the age of the oldest page in the system. [7]

Global and Local LRU. Page replacement policies commonly maintain a pool of available space from which new allocations can be satisfied on demand, then trigger a

cycle of page replacement when the pool drops below some threshold value. In many operating systems, this is called *page stealing*, but Windows prefers a more polite designation, namely *page trimming*. Call it what you will, the specific issue is that one program's pattern of memory access can influence the performance of other executing programs since they all share a common memory pool. This is generally the trade-off between a policy of *global LRU*, which treats all process virtual address spaces on an even footing, and *local LRU*, which applies the page replacement on a process by process basis.

While it has some global aspects, Windows NT decisively practices a localized per process working set management policy. Each process has a designated **working set maximum**, initially set by default by the operating system. The default maximum process working set size is 345 pages on any system with at least 64 MB of RAM installed. Any process running at its maximum value that attempts to reference a new page finds that NT removes an existing page *before* adding the new one to the process working set. Based on the availability of physical memory, the working set maximums of all processes running at their maximums are adjusted upward. This allows processes that require it to add more pages to their designated maximums and grow their working sets over time. The global triggering threshold for this working set maximum adjustment is that it occurs every second that the pool of memory available for new allocations (memory counter Available Bytes) is approximately 4 MB or more. So long as this cushion of available memory exists, process working set maximums are allowed to drift upward. [8]

NT also provides an Application Programming Interface (API) to allow individual processes to specify their real memory requirements to the operating system and take the guesswork out of page replacement. Many applications developed specifically for Windows NT, including Microsoft SQL Server and Internet Information Services utilize a Win32 API call to inform Windows NT of their physical memory requirements. SQL Server, for instance, establishes process working set minimum and maximum values based on a database tuning parameter. Windows NT will only trim pages from a process working set below its minimum value when the system is under stress, defined as when the number of **Available Bytes** falls below 1 MB.

Soft faults. Windows NT's page replacement policy is designed to obtain information about memory reference patterns without recourse to hardware access bits. (Where hardware access bits are available, in Intel hardware, for example, NT does use that information to supplement its basic policy in some circumstances.) Information about memory reference patterns is obtained by periodically trimming all pages above a process's working set minimum, again set by default unless specifically overwritten. The system default for an NT Server with at least 64 MB establishes working set minimums of 50 pages per process.

Trimmed pages are not stolen definitively. Instead, they are marked as being in a state of *transition*. In the PTE, the Valid bit is turned off, but the value of the real memory offset is retained while the page is in the transitional state. The next time a trimmed page in transition is referenced, a page fault is occurs. Responding to the transition fault, the operating system immediately returns the page to the Valid state, adds it to the process working set, and re-executes the instruction.

Figure 8 illustrates the overall page trimming mechanism. The size of the three threaded list structures, the Standby List, the Free List, and the Zero List, is calculated as the number of Available Bytes, the Memory Counter accessible using the NT Performance Monitor. (Perfmon reports these values in bytes for consistency across the Intel and Digital Alpha platforms which use

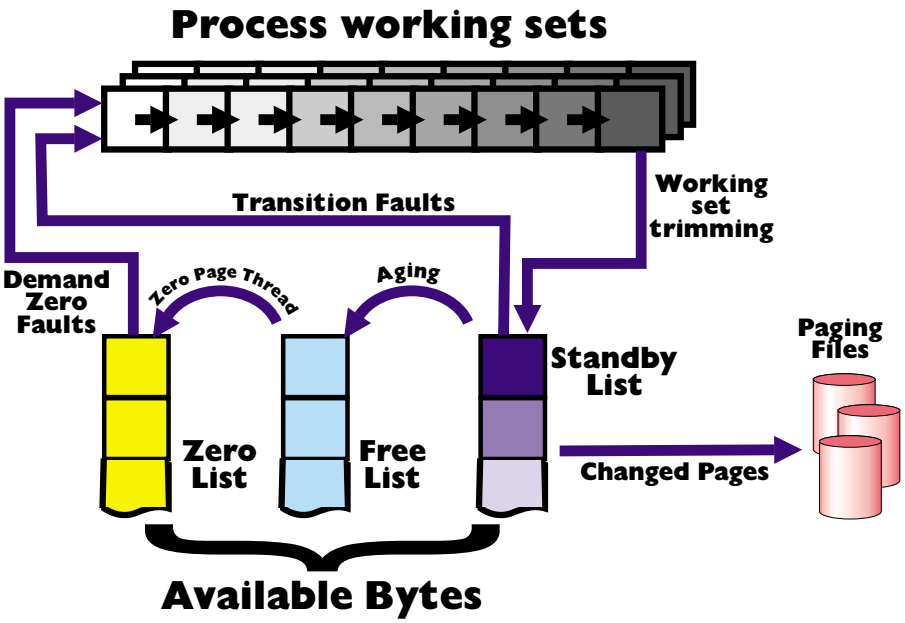

FIGURE 8. PROCESS WORKING SET TRIMMING IN WINDOWS NT. PERIODICALLY, ALL PAGES ABOVE THE PROCESS MINIMUM WORKING SET ARE STOLEN AND PLACED ON THE STANDBY LIST. TRIMMED PAGES AGE AND MOVE TO THE FREE LIST AND, ULTIMATELY, TO THE ZERO LIST. MEANWHILE, TRIMMED PAGES WHICH ARE STILL ACTIVE ARE "SOFT-FAULTED" BACK INTO THE PROCESS WORKING SET WHEN REFERENCED. MODIFIED PAGES MUST BE COPIED TO A PAGING FILE BEFORE MOVING TO THE FREE LIST. A MODIFIED PAGE WRITER THREAD OF THE KERNEL WRITES PAGES TO DISK IN BULK WHENEVER A DESIGNATED THRESHOLD OF CHANGED PAGES ACCUMULATES.

different page sizes. The linked list structures themselves actually reference pages.) New page allocations are generally satisified from the Zero List. A low priority system Zero Page thread has responsibility for taking pages off the free list and zeroing their contents. (This is done for security reasons.) Pages transition from the Standby List to the Free List as required by the demand for new allocations. Page trimming is invoked once per second by default. It is also triggered on demand when the number of available pages drops below 1 MB. Trimmed pages that are modified must be written to disk before transitioning to the Free List. Modified pages are written in bulk to disk by Modified Page Writer threads, again based on threshold values being exceeded. The Modified Page Writer threshold value for a system with 64 MB or more of RAM waits until 300 dirty pages accumulate before physical disk writes are initiated.

Transition faults are also designated as "soft faults" in some Windows NT documentation because they are resolved without performing the time-consuming paging file I/O operations associated with resolving "hard" page faults. Performance analysts accustomed to more conventional LRU page stealing algorithms may need some time to warm to this approach. One byproduct is the very large number of transition faults that can occur. The memory Counter Page Faults/sec includes both *transition faults* and *cache faults*. (Cache faults are read file requests automatically redirected to the NT paging subsystem that *miss* the file cache.) From a configuration and tuning perspective, it makes sense to focus on the memory Counters which break out page faults into the three relevant categories:

- Transition faults/sec: the "soft" page faults described above
- Cache faults/sec: a byproduct of normal file I/O operations
- Page Reads/sec: "hard" page faults which require demand paging file I/O operating to restart the running thread that incurred the request.

Figure 9 provides an example of an NT performance report that takes this approach. This example illustrates a 128 MB NT Workstation reporting 40 MB worth of Available Bytes at the same time it was experiencing over 300 transition faults per second. Meanwhile, the number of "hard" page faults/second never exceeds 25, a much more reasonable number for a single disk system. By the way, hard page faults only impact the execution status of the thread, not the entire process. Other threads in the same process can be scheduled for execution.

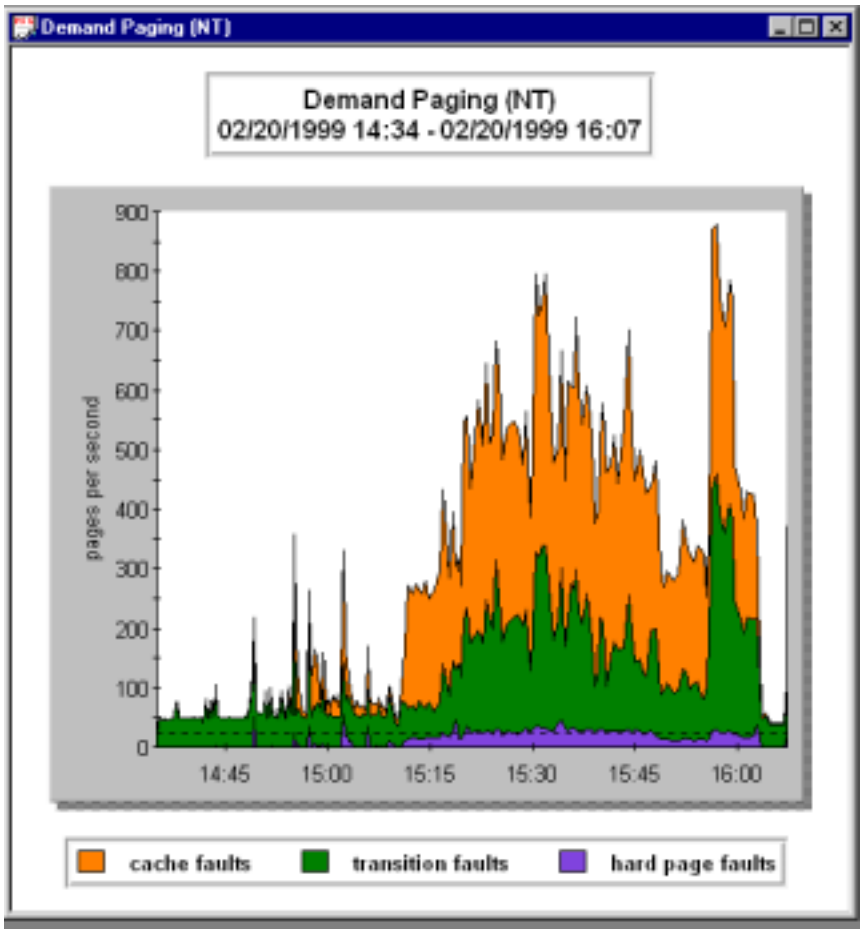

FIGURE 9. DEMAND PAGING ACTIVITY IN NT. THE TOTAL NUMBER OF PAGE FAULTS REPORTED INCLUDES BOTH TRANSITION FAULTS AND CACHE FAULTS. ONLY "HARD" PAGE FAULTS ARE A PERFORMANCE AND TUNING CONCERN.

An alternative reporting approach is to calculate the number of "hard" page faults by subtracting transition faults and cache faults from the total number of page faults. The remainder should be equal to the number of "hard" faults. Experience in making this calculation indicates there is some logical inconsistency in the measurement data because the number of hard fault caluclated in this fashion is sometimes a negative value.

Configuration guidelines. This brings up the usual matter of determining an optimal number of hard page faults for a particular environment. Rather than merely provide the usual answer, which is, "It depends," let's try to see what it depends on. Any usable configuration guideline that attempts to quantify the number of "hard" page faults a particular NT configuration is capable of performing should consider the following:

- the number of physical disks available for paging, since each disk is a single server with some finite I/O throughput capability
- the speed of these disks
- other I/O capacity constraints, such as the speed of the bus connections or the fact that disk I/O competes with other peripherals for access to the shared PCI bus, and
- the fact that any paging to those disks reduces proportionally their availability to satisfy normal file I/O requests

Given that most PC-based Server architectures have limited I/O bandwidth, it seems reasonable to suggest keeping paging I/O traffic to a minmum, perhaps never to exceed 10-20% of overall disk bandwidth.

Using the Intel hardware access bits. Given that 98% or more of all copies of the NT operating system that are sold run on Intel hardware, it was inevitable that some optimizations for the Intel platform would creep into the OS. Without modifying the basic transition fault mechanism, the NT developers have found a way to exploit the hardware memory access bits used to determine the age of a page that Intel IA-32 maintaines. In its original form, the page trimming algorithm treats every process working set as a FIFO (First In, First Out) queue, stealing from the front and transition faulting back into the working set at the rear of the queue. Windows NT Server on uniprocessors was modified to trim unaccessed pages from the working set first, then reset the Access bit of each page remaining in the process working set at the conclusion of the page trimming cycle. This has the effect of adding an element of the popular Least Recently Used page stealing algorithm to the NT page trimming.

Both Windows NT Workstation and instances of Server running on multiprocessors run the original page trimming routine that does not bother with the hardware Access bits. On multiprocessors, Microsoft developers reported a performance problem when the trmming routine resets the hardware Access bits of all executing processes. Processes currently executing on different processors from the one where page trimming is running have their TLBs flushed when the Access bits are reset. The performance hit for threads executing on other processors, especially associated with applications like SQL Server that perform their own working set management, when TLBs are flushed was significant enough for Microsoft developers to revert back to the original FIFO scheme.

Measurement and Prediction.

The preceding discussion highlights several key metrics for tracking the impact of virtual memory addressing under Windows NT. In this section, I will look at the memory and paging Counters that are available and suggest suitable

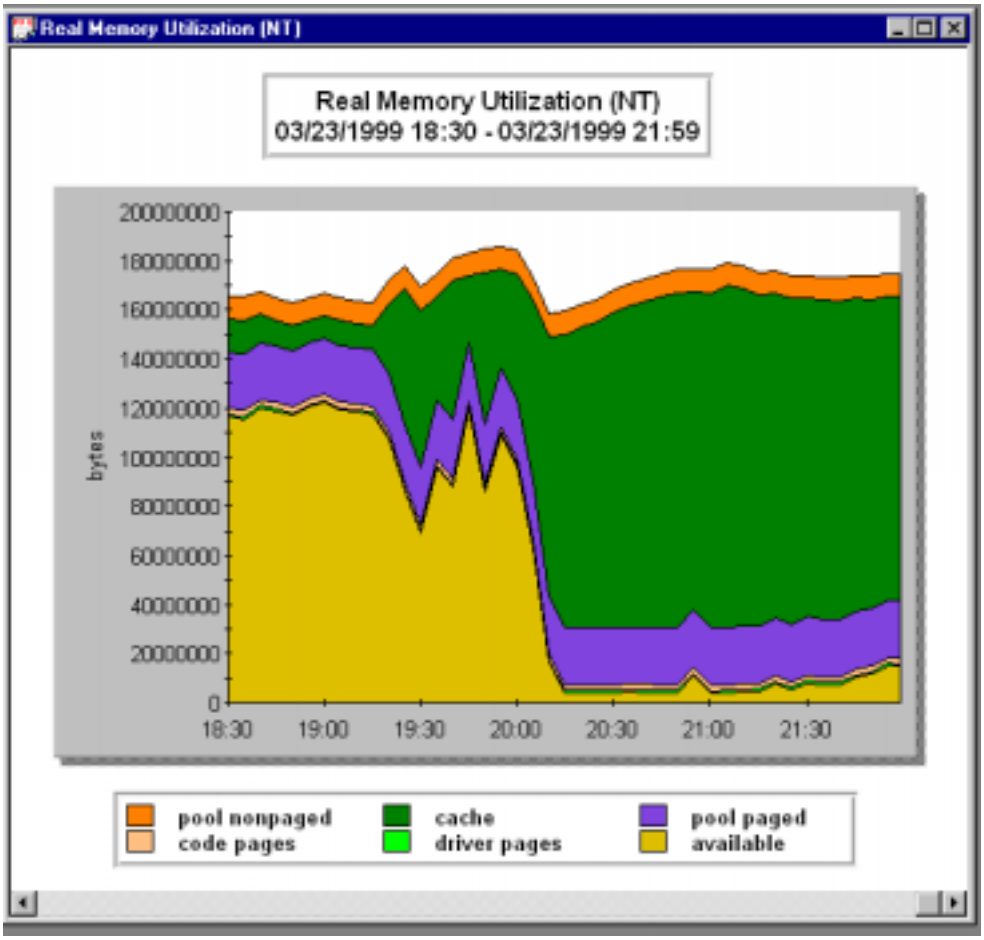

Figure 10. Real Memory Utilization in NT. Six Memory Object Counters report the operating system's use of physical memory: Available Bytes, Pool non-paged Bytes, Pool Paged Resident Bytes, System Cache Resident Bytes, System Code Resident Bytes, and System Driver Resident Bytes. The sum of these six Counters subtracted from the total amount of RAM installed shows the portion of physical memory used by process address spaces.

guidelines for reporting on NT virtual memory performance. Finally, I discuss details for calculating a memory contention index that can be used predictively to anticipate performance problems.

Memory Object. In the main, the performance metrics associated with system memory allocation and paging are Performance Monitor Counters grouped under the Memory Object. For convenience, I divide these Counters into three areas: virtual memory allocation, physical memory allocation, and paging activity. A brief description of the key metrics follows [9]:

Virtual memory allocations. The key metrics are the total number of **Committed Bytes** and the **% Committed Bytes in Use**. The latter is the ratio between Committed Bytes and the Commit Limit. Useful configuration guidelines for medium to large scale systems is that **% Committed Bytes in Use** not exceed 70%. When **% Committed Bytes in Use** exceeds 90%, NT automatically extends those paging files with flexible extents. This can introduce embedded seek operations into paging file I/O requests.

When multiple paging files are configured, it is sometimes useful to access the Paging File Object, which is instanced by paging file, and report the **% Usage** Counter. The advantage of multiple paging files is the ability to utilize multiple physical disks for paging operations.

Physical memory allocations. Physical memory allocated to operating system functions is recorded in six Counters, as illustrated in Figure 10: **Available Bytes**, **Pool non-paged Bytes**, **Pool Paged Resident Bytes**, **System Cache Resident Bytes**, **System Code Resident Bytes**, and **System Driver Resident Bytes**. The sum of these six Counters subtracted from the total amount of RAM installed shows the portion of physical memory used by process address spaces. **Available Bytes** represents the sum of three lists, the Zero List, the Free List, and Standby List. When **Available Bytes** exceeds approximately 4 MB, process working set minimum and maximum threshold used to control page trimming are allowed to be adjusted upwards. A system showing less than 1 MB **Available Bytes** is under stress.

The Process **Working set** Counter (in bytes) reports the size of each per process working set. Resident pages from shared memory Objects, which are quite common due to the resuse of common program runtime DLLs, are counted in *each* process address space they are referenced in. The **_Total** instance of the Process Object reflects this double, triple, quadruple, etc., counting of shared memory Objects.

Paging activity. **Page Faults/sec** is the sum of **Cache Faults/sec**, **Transition Faults/sec** (so-called "soft" faults), and "hard" page faults. Very little can be done about the number of transition faults that occur because the transition fault mechanism is essential to NT's page replacement policy. It is the way page aging information is obtained. **Demand Zero Faults/sec** represent application requests for new pages, including new pages acquired during program loading and initialization. **Pages Read/sec** counts hard page faults directly. **Pages Written/sec** counts the number of Modify Page Writer thread-initiated write operations. Since NT typically performs bulk paging operations, the number of **Pages input/sec** is normally greater than the number of **Pages Read/sec**. Likewise for **Pages output/sec** and **Page Written/sec**. Any paging I/O performed by the operating system normally reduces the capability of the system to perform file I/O operations on behalf of application programs. A suggested configuration guideline is that not more than 10-20% of the available disk I/O bandwidth be absorbed by system paging operations.

References.

[1] Peter J. Denning, Working sets past and present. *IEEE Trans. of Soft. Eng.*, 6, 1 (January 1980) 64-84.

[2] Sean Dailey, *Optimizing Windows NT*. Foster City, CA, WA: IDG Books, 1998.

[3] Curt Aubley, *Tuning & Sizing NT Server*, Upper Saddle River, NJ: Prentice Hall, 1998.

[4] Mark T. Edmead and Paul Hinsberg, *Windows NT Performance: Monitoring, Benchmarking, and Tuning*. Indianapolis, IN: New Riders, 1998.

[5] Intel *Pentium Processor Family Developers Manual, Volume 3: Architecture and Programming Manual*, 1995.

[6] Alan Jay Smith, Cache Memory. *ACM Computing Surveys*, 14, 3 (September 1982) 473-530.

[7] Mark B. Friedman, MVS Memory Management. CMG '91 *Proceedings*, (December 1991) 747-759.

[8] David A. Solomon, *Inside Microsoft Windows NT, 2nd Edition*. Redmond, WA: Microsoft Press, 1998.

[9] *Microsoft Windows NT 4.0 Workstation Resource Kit*. Redmond, WA: Microsoft Press, 1996.

[10] Mark B. Friedman, Practical techniques for Modeling Memory in MVS. CMG '90 *Proceedings*, (December 1990).

Memory contention index. Besides monitoring measures of virtual memory, physical memory, and paging activity, it is useful to develop predictive measures. One useful calculation is to derive a memory contention index which may be of use in predicting hard paging activity. A useful working hypothesis is that demand paging activity, in general, is caused by virtual memory address spaces contending for limited physical memory resources. A simple index computed from the ratio of virtual memory **Committed Bytes** to available RAM can have predictive value. See [10] for a fuller exposition. Because of the limited I/O bandwidth that is usually available, this author monitors this memory contention index and normally intervenes (where possible) to add RAM before the index reaches a value of 2:1.

Another useful memory contention index can be calculated as the ratio of **Pool Paged Bytes**:**Pool Paged Resident Bytes**. This index corresponds to the virtual memory demand associated with the system's pageable pool and the actual number of resident pages within that pool. As this ratio increases, there is more contention for physical memory, as illustrated in Figure 11. While much more investigation is warranted, the correlation between this memory contention index and the hard page fault rate shown in Figure 11 suggests this may be a promising line of inquiry.

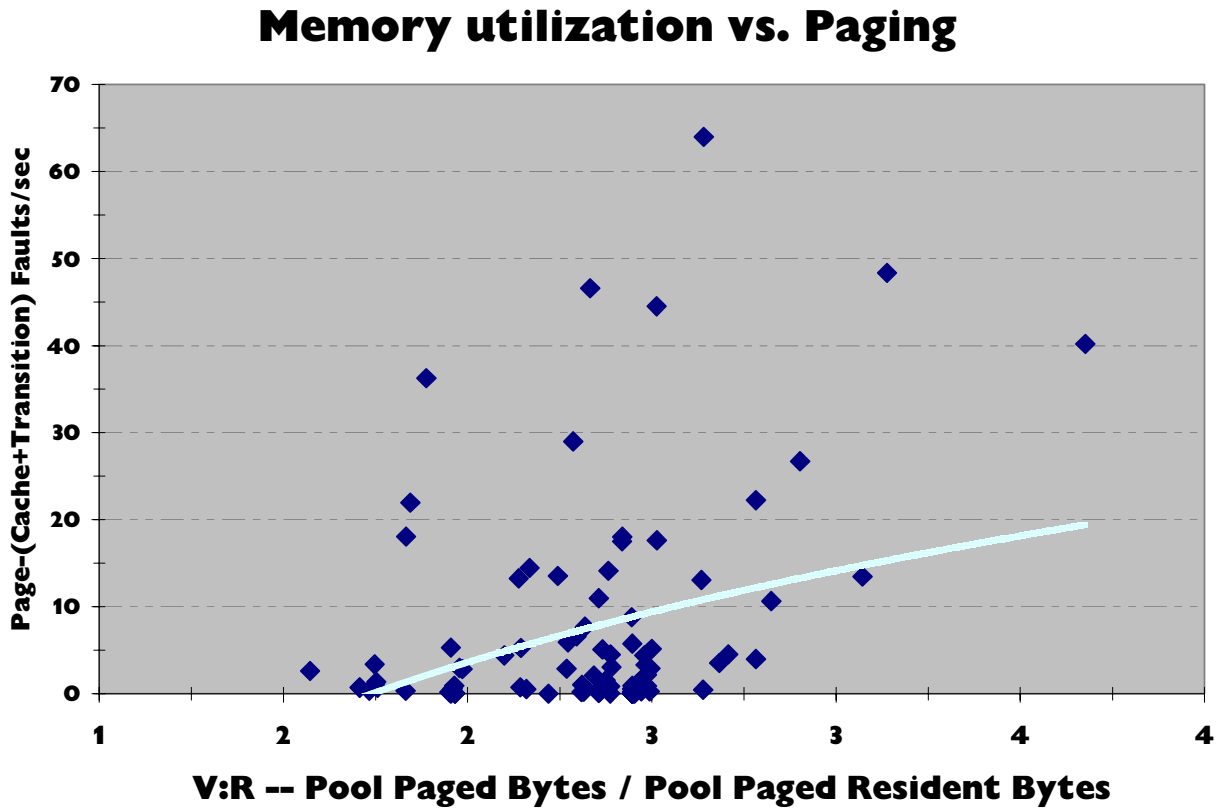

FIGURE 11. A MEMORY CONTENTION INDEX BASED ON THE RATIO OF **POOL PAGED BYTES** TO **POOL PAGED RESIDENT BYTES** *IS CALCULATED AND COMPARED TO THE RATE OF HARD PAGE FAULTS*. THIS INDEX CORRESPONDS TO THE VIRTUAL MEMORY DEMAND ASSOCIATED WITH THE SYSTEM'S PAGEABLE POOL AND THE ACTUAL NUMBER OF RESIDENT PAGES WITHIN THAT POOL. AS THIS RATIO INCREASES, THERE IS MORE CONTENTION FOR PHYSICAL MEMORY. A CORRELATION BETWEEN THIS MEMORY CONTENTION INDEX AND THE HARD PAGE FAULT RATE IS EVIDENT IN THIS EXAMPLE. WHILE THIS IS A PROMISING LINE OF INQUIRY, MORE RESEARCH IS NEEDED BEFORE THIS RESULT CAN BE GENERALIZED.