HP - UX Kernel Tuning and Performance Guide

Getting The Best Performance
From Your Hewlett - Packard Systems
Version 3.1
Revision Date: March 15, 2000

Author: Stephen Ciullo

Index

---

1. Introduction

This document describes how a HP - UX kernel is tuned and configured. The intent is to provide customers, developers, application designers, and HP's technical consultants the information necessary to optimize performance of existing configurations and to make intelligent decisions when running applications on HP - UX. This is not a manual and we do not go into the specific reasons for many of the recommendations contained herein.

Back to the Top

2. Hardware Considerations

HP, and other hardware vendors, offer a broad selection of products with a wide range of CPU performance, memory and disk options. Obviously, the performance of an application will be affected by the hardware on which you choose to run.

There are five key hardware areas that directly affect the performance you will obtain from your application: CPU, Memory, Disk, Graphics, and Network. It is not a wise choice to buy the fastest CPU and configure it with insufficient memory.

There are many things to consider when you are choosing the hardware for your system. The compute needs may vary from the very simple to the incredibly complex. The best way to select the appropriate hardware is to try to answer the following questions:

- How many users need to access the system at any one time?
- Is it a server or a client?
- What are the server needs ?
- What are the application software needs?

There might be several different configurations necessary for your environment. You might need different configurations for different users. The best way to select the appropriate hardware is to perform tests that duplicate your intended use of the system. With test results in hand, you will have the information you need to make well informed hardware decisions.

Back to the Top

3. CPU

CPU performance is the single most important factor when you want to get the most work done in the shortest possible time. If it takes five seconds to perform a particular operation, is it worth it to spend an extra $10,000 to do it in three seconds?

However, if the operation takes five hours and the time can be reduced to one or two hours, it may be worth the additional expense.

Computational tasks are most affected by CPU performance. Be sure to consider investment protection. The CPU that seems adequate today may not meet your needs in the near future. The rapid pace of hardware development makes existing systems obsolete in a very short period of time.

Back to the Top

4. Memory

One of the most commonly asked questions is "How much memory do I need?". Unfortunately, the real answers to this question are "Enough" and "It depends". The amount of memory you need is directly related to the size of the applications with which you are working. While 'X' amount of memory may allow you to run your application, it may not be large enough to allow for optimal performance.

There is a lot of confusion regarding cache memory, configuration of swap space, swap's relationship to physical memory, kernel parameters affecting memory allocation, and performance. It is important to understand memory management in order to understand these relationships.

Back to the Top

A. Physical Memory

Physical memory is composed of hardware known as RAM (Random Access Memory) usually installed in SIMM or DIMM's. For the CPU to execute a process, the relevant pages of a process must exist in the physical memory.

The more physical memory there is, the more processes can be run and/or the larger a process (or processes) can be, without the system having to "page out". When the system has no more room in memory and begins to page out, performance will begin to degrade.

Not all physical memory is available to user processes. The kernel occupies some main memory and it is never paged. The amount of main memory not reserved for the kernel is termed available memory. Available memory is used by the system for executing processes.

Back to the Top

B. Physical Memory and Performance

The amount of memory available to applications is determined by the amount of swap configured plus physical memory. The amount of physical memory available will directly affect when or if paging will occur. Paging imposes a serious performance penalty. There is a critical threshold for physical memory, below which the system spends much of its CPU time paging. When it begins to spend most, if not all of the cpu time paging, it is known as thrashing. Thrashing is evident by the fact that system performance virtually comes to a standstill.

In a best case scenario, paging would never occur. However, not everyone has the luxury of enormous amounts of memory. Understanding how memory size affects performance is important. HP's performance monitoring tools (Glance/ GlancePlus/MeasureWare/PerfView) can help in tracking memory and associated bottlenecks/problems.

Back to the Top

1. Where Is The Memory Going?

   To help you understand minimum memory requirements, it helps to understand how memory is consumed. Minimally, you will have the following resources consuming memory:

■ HP - UX Operating System	10 - 12 MB
■ Windowing System	21 MB (X11) 25 MB (VUE) 32 MB (CDE)

   HP - UX uses paging to move pages in and out of physical memory. The operating system and any pages that have been explicitly (or implicitly) locked are not subject to paging. Only the pages of a process that are required for execution will be paged in. Text (instructions) pages are never "paged out".

   Back to the Top

2. Determining Memory Requirements

   One way to determine whether the amount of physical memory in your system is adequate would be to run benchmarks with increasing levels of physical memory and use one of HP's performance tools to monitor the system. This will tell you if your system is paging. It would be beneficial if you could run your real "production" applications to perform the test.

Back to the Top

C. Memory Management

HP - UX memory management is composed of 3 basic elements: cache, memory and swap space. Swap space can be composed of two types: device swap and file system swap. Device swap can be made up of primary swap space that is defined on the root disk and secondary swap space which is defined on other disk volumes. All of these elements can be configured/optimized through HP - UX kernel parameter tuning.

The data and instructions of any process (a program in execution) must be available to the CPU by residing in physical memory at the time of execution. Physical memory (also called "main memory"), is shared by all processes. To execute a process, the kernel accesses it's parts through a per - process virtual address space that has been mappe into physical memory.

The term "memory management" refers to the rules that govern physical and virtual memory and allow for efficient sharing of the system's resources by user and system processes.

The total size of a user process(es) is permitted to exceed physical memory by using an approach termed demand - paged virtual memory. Demand paged virtual memory enables you to execute a process by bringing into main memory pages of the process as needed (on demand), and writing out to disk the pages of a process that have not been recently accessed.

The HP - UX operating system uses paging to manage virtual memory. Paging involves moving small units (called pages) of a process between main memory and disk space.

One method for increasing the efficiency of memory allocation is the use of the mallopt() command. The mallopt()call must appear prior to any malloc() call in the application. One mallopt() at the beginning of the program should be sufficient, depending on what one is doing. mallopt() is also needed when attempting to obtain "process large data space" (a data segment larger than the deafult maximum of 1.9GB if EXEC_MAGIC). This command is unique to HP - UX and controls the memory allocation algorithm and other optimization options within the malloc library. Use of mallopt() can improve application execution time up to 10X, depending on the data size. It is important that the Maxfast and Numlblks options (i.e. the first two options to mallopt) be defined to reflect the data size links being accessed.

Back to the Top

D. Virtual Address Space

Process virtual address space is mapped via a structure named "vas" (Virtual Address Space). The vas contains information about, and pointers to the various parts of a process, both in memory and on disk. One virtual address space (vas) exists per process and serves several purposes:

- It provides the overall description of each process.
- It contains pointers to other elements in the process structure mapping per - process regions. (pregions)

Each process can have a maximum of 4 Gb virtual address space (this changes in HP - UX 11.0). The four GB virtual address space is divided into four one - GB quadrants. This (and the descriptions following, will change in 11.0). Each quadrant has associated with it:

- The first quadrant always contains the process's text segment (code), and sometimes some of the data (EXEC_MAGIC).
- The second quadrant contains the data segment (static data, stack, and heap, etc.).
- The third quadrant contains shared library code, shared memory mapped files and sometimes shared memory.
- The fourth quadrant contains shared memory segments, shared memory - mapped files, shared library code, and I/O space.

As stated, the vas structure points to per - process regions, or pregions. Pregions represent the specific segments of a process, including text (process instructions), data, u_area and kernel stack, user stack, shared memory, shared libraries, etc.

The maximum size of various memory segments is controlled by the values of certain configurable kernel parameters. It is beyond the scope of this paper to discuss all the process segments. The following, however, is a description of the segments most relevant to this discussion:

Text - The text segment contains a process's instructions and may be shared by multiple processes. The maximum size of the text segment is limited by the configurable parameter maxtsiz.

Data - The data segment contains a process's initialized (data) and uninitialized (.bss) data structures, along with the heap, private "shared" data, "user" stack, etc. A process can dynamically grow its data space (heap). The maximum size for the process data space is governed by the configurable kernel parameter maxdsiz.

Stack - Space used for local variables, subroutine return addresses, kernel routines, etc. The u_area contains information about process characteristics. The kernel stack contains a process's stack while executing in kernel mode. Both the u_area and kernel stack are fixed in size. Space available for the user stack is determined by the configurable parameter maxssiz.

Shared Memory - Address space which is shareable among multiple processes.

Back to the Top

5. Disk

Application data can be quite large. Disk I/O is one of the "Big Three" bottlenecks (CPU and Memory being the other two). Disk I/O can also be the limiting factor in overall performance if a system does an inordinate amount of paging on a frequent basis.

HP's philosophy is to design balanced systems in which no single component becomes a performance bottleneck. HP has made significant enhancements to I/O performance in order to keep pace with the speed of our CPUs. I/O performance depends on several parts of the system working together efficiently. The I/O subsystems have been redesigned so that they now offer the industry's fastest and most functional I/O as standard features.

Back to the Top

A. To Improve Disk I/O Performance:

Distribute the work load across multiple disks. Disk I/O performance can be improved by splitting the work load. In poor configurations, a single drive contains the operating system, swap space, and data file access simultaneously. If these different tasks can be distributed across multiple disks then the job can be shared, providing subsequent performance improvements. For example, a system might be configured with four logical volumes, spread across more than one physical volume. The HP - UX operating system could exist on one volume, the application on a second volume, swap space interleaved across all local disk drives and data files on a fourth volume.

Split swap space across two or more disk volumes. Device swap space can be distributed across disk volumes and interleaved. This will improve performance if your system starts paging. This is discussed in more detail in the section on Swap Space Configuration later in this document.

Enable Asynchronous I/O - By default, HP - UX uses synchronous disk I/O,

when writing file system "meta structures" (super block, directory blocks, inodes, etc.) to disk. This means that any file system activity of this type must complete writing to the disk before the program is allowed to continue; the process does not regain control until completion of the physical I/O. When HP - UX writes to disk asynchronously, I/O is scheduled at some later time and the process regains control immediately, without waiting.

Synchronous writes of the meta structures ensure file system integrity in case of system crash, but this kind of disk writing also impedes system performance. Run - time performance increases significantly (up to roughly ten percent) on I/O intensive applications when all disk writes occur asynchronously; little effect is seen for compute - bound processes. Benchmarks have shown that load times for large files can be improved by as much as 20% using asynchronous I/O. However, if a system using asynchronous disk writes of meta structures crashes, recovery might require system administrator intervention using fsck and, might also cause data loss. You must determine whether the improved performance is worth the slight risk of data loss in the event of a system crash. A UPS device, used in a power failure event will help reduce the risk of lost data.

Asynchronous writing of the file system meta structures is enabled by setting the value of the kernel parameter fs_async to 1 and disabled by setting it to 0, the default. For instructions on how to configure kernel parameters, see the section Kernel Configuration Parameters later in this document.

Back to the Top

B.

File Systems

When using UFS (HFS) file systems, configure them with a block size of 64K and a fragment size of 8K. HFS file systems have historically preferred to perform I/O in 64K block sizes. We have improved performance by using a VxFS (JFS) file system when it is being used as a "scratch" file system...a file system that you do not care about when the application crashes, or when it completes successfully. When doing so, you need to mount this file system with three specific options in order to gain performance. They are:

- nolog
- mincache=tmpcache
- convosync=delay

The on - line (advanced) JFS product is required to use these options. In my experience, the JFS block size is of no consequence when using JFS. JFS likes

to perform I/O in 64K chunks, regardless of the block size. Supported block sizes are 1, 2, 4, and 8K. There is no fragment on a JFS file system.

When striping with LVM, one should make sure that the file system block size and the LVM stripe size are identical. This will aid performance.

When mounting file systems, they should be positioned at mount points that are as close to the "root" of the tree. This will help "shorten" directory search paths. It is very important that file systems that contain "tools" that will be used by the application(s), be mounted as close to the top as possible.

As of the current version of this document, there is a JFS "mega patch" for performance. The patch number is PHKL_14613 for the 700 series and PHKL_14491 for 800 series systems.

NOTE: Be sure to review the patch catalog for these patches to be sure that these are the most current patches.

Check your buffer cache size. Some say 128K for each 1000 IOPS a server expects to deliver.

Check your disk and file system configurations:

- LVM configuration/layout
- Multiple disk striping?
- HFS? ...check your block/fragment sizes
- JFS? ...check your mount options

Reads and writes...server and client block sizes should match. Pay attention to the suggestions for file systems (above).

Back to the Top

C. Logical Volume Manager

The following are simply recommendations...you do not have to do them. Obviously, there are pros and cons with everything. This is not the focus of this document, but there is value in a brief review. Use as many physical disks as possible. Stripe them if you can. If you have followed the file system recommendation of using a 64K block size, use a 64K stripe size as well. I would suggest a 64K stripe size for LVM anyway. Hopefully, you will have identical disks (make, model, size, geometry, etc.). When you have control, place your logical volumes so that the "pieces" a logical volume are located in the same place across the physical devices. For example, having four physical devices, you "stripe" a logical volume so that 25% of appears on each of the

four disks, and, each piece appears at the "top" of the disk.

Back to the Top

D. Secondary Storage

Main memory is where the data and instructions required for program execution are stored. During process execution, data and instructions reside in cpu registers and cache. These are very fast and very expensive pieces of hardware. Program files are kept in secondary storage (disk drives).

Back to the Top

E. Swap

A temporary form of data storage is swap space. It should be noted that HP - UX does not "swap" any more, it pages and, as a "last resort" deactivates processes. The process of deactivation replaces what was formerly known as swapping entire processes out.

While executing a program, data and instructions can be paged (copied) to and from secondary storage, if the system load warrants.

Swap space is initially allocated when the system is configured. HP - UX supports two types of swap space: device swap and file system swap. Device swap is allocated on a disk "outside" of any file system space and can be:

- a entire disk
- an area on a disk
- a logical part of a physical disk

If the entire disk hasn't been designated as swap, the remaining space on the disk can be used for a file system. File system swap space is allocated from a mounted file system and can be added dynamically to a running system. Device swap can also be added dynamically to a running system. Sam or the swapon() command can be used to enable device or file system swap.

NOTE: File - system swap has significantly lower performance than device swap. The I/O for file system swap will contend with user I/O on that particular file system. Using file system swap space should be avoided. Once allocated, you cannot remove either type of swap without rebooting the system. HP - UX uses a swap space reservation method (to insure it has space available), but only allocates the space when it actually needs to write to it.

Back to the Top

1. How much swap do I have?

   SAM, Glance/GlancePlus, top, and swapinfo all show swap information. To see how much swap space is configured on your system, and how much is in use, execute one of the following commands:

   - top
   - Glance / GlancePlus

/usr/perf/bin/glance	HP - UX 9.X systems
/opt/perf/bin/glance	HP - UX 10.X systems

   - sam — requires root
   - /etc/swapinfo -t — HP - UX 9.X systems - requires root
   - /usr/sbin/swapinfo -t — HP - UX 10.X systems - requires root

   Any user can execute top and Glance.

   Back to the Top

2. How Much Swap Do I need?

   The amount of swap, added to the amount of memory available, defines the virtual memory available for processes. The minimum recommendation is twice as much swap space as physical memory. Keep in mind, this is an "old" formula. If you have two, three, four or many more gigabytes of physical memory, this would result in way too much swap space. Granted, there are the pathological cases that would require you to have eight to ten gigabytes of swap with, say, four gigabytes of physical memory. If you try to execute a process that would exceed the amount of available memory plus available swap, you will get a "out of memory" message. If you configure more swap than you will ever need, you are wasting disk space. The correct swap size will vary considerably depending on the size and number of application(s) run on a system.

   The correct swap size can be determined by monitoring swap usage while

working with real data. This could be done either with the swapinfo command or using a tool like HP's GlancePlus. GlancePlus allows you to monitor system resources on a per process basis and will display the high water mark (since you started glance). You would configure a system with more swap space than you expect to need, then run GlancePlus while running an application(s). By monitoring the high water mark, you can determine the maximum swap space used and adjust the swap size accordingly. Obviously, if you experience out of memory errors, swap size is too small.

There are many systems with less swap space than physical memory. This is perfectly fine. The person who specified the system probably recommended all that memory so that they would not have to swap! We have seen systems with with swap space equal to 50% that of physical memory. Just be sure to have swapmem_on EQUAL TO 1!!

NOTE: For best performance, swap space should be distributed evenly across all disks configured with swap, at the same priority .

Back to the Top

3. Configuring Swap Space

   As previously mentioned, device swap is preferred over file system swap to achieve the best performance. The ideal swap configuration is device swap spread across multiple identical disks. Each quantity of swap space being equal in size. Once this is done, assign each space the same priority. This implies interleaving. This means that if your system starts to page, the paging requests will be "round robin'd" among the swap spaces. This is how you would want your system to page, should it run out of memory. If possible, it is even better to place the disks containing the swap space on separate cards/controllers. This eliminates the controller as a "single point of bottleneck".

   SAM is one method for adding and configuring swap space. Swap configuration is under the Disks and File System area of SAM. For more information on configuring swap, please see the on - line Help section within SAM's Swap Configuration. If you wish to use the swapon command, review the man page, swapon(2).

Back to the Top

6. HP - UX Kernel Configuration

This section explains HP - UX configurable kernel parameters that affect system

capacity and/or performance. Most of this section is common for HP - UX 9.X and HP - UX 10.X. Specific differences are noted.

Back to the Top

A. Configuring Kernel Parameters in 9.X

In HP - UX 9.X we recommend manual kernel configuration. All work related to creating a new kernel in 9.X takes place in the /etc/conf directory. Follow these steps:

- cd /etc/conf
- cp dfile dfile.old
- vi dfile
- Modify the dfile to include the kernel parameters and values suggested above.
- config dfile
- make - f config.mk
- mv /hp - ux /hp - ux.old
- mv /etc/conf/hp - ux /hp - ux
- cd / ; shutdown - ry 0

NOTE: For more information on manual kernel configuration, please see the HP - UX System Administration "Systems Administration Tasks" book.

Back to the Top

B. Configuring Kernel Parameters in 10.X

In HP - UX 10.X we recommend modifying the kernel parameters SAM allows, and then manually modifying the hpux_aes_overide parameter. The hpux_aes_override kernel parameter is the only recommended parameter that must be modified manually. We recommend using SAM for the other parameters to take advantage of its built - in kernel parameter rule check function.

NOTE: Apply patch PHCO_11647 if you use this parameter on HP - UX 10.X. Failure to do so can cause some "apparent" corruption in parts of the file system where trasnition links occur.

To configure a kernel manually, you must be super - user. All work related to creating a new kernel in 10.X takes place in the /stand/build directory. Follow these steps:

- cd /stand/build

- /usr/lbin/sysadm/system_prep - s system
- vi system
- Either add or modify the entries to match:
- hpux_aes_override 1
- mk_kernel - s system
- mv /stand/system /stand/system.prev
- mv /stand/build/system /stand/system
- mv /stand/vmunix /stand/vmunix.prev
- mv /stand/build/vmunix_test /stand/vmunix
- cd / ; shutdown - ry 0

NOTE: For more information on manual kernel configuration, please see the HP - UX 10.X System Administration "Systems Administration Tasks" Book. .

To configure the remaining kernel parameters with SAM, follow these steps:

- Login to the system as root
- Place the list of kernel parameter values (above) in the file:
- /usr/sam/lib/kc/tuned/stuff.tune

  (The first line should be "STUFF Applications" in the format shown in the general "Configuring Kernel Parameters" section above.)

- Start SAM by typing the command: sam
- With the mouse, double - click on Kernel Configuration .
- On the next screen, double - click on Configurable Parameters.
- SAM will display a screen with a list of all configurable parameters and their current and pending values. Click on the Actions selection on the menu bar and select Apply Tuned Parameter Set ... on the pull - down menu. Select STUFF Applications from the list and click on the OK button.
- Click on the Actions selection on the menu bar and select Create A New Kernel. A confirmation window will be displayed warning you that a reboot is required. Click on YES to proceed.
- SAM will build the new kernel and then display a form with two options:
  - Move Kernel Into Place and Reboot the System Now
  - Exit Without Moving the Kernel Into Place
  - If you select the first option and then click on OK, the new kernel will be moved into place and the system will be automatically rebooted.
  - If you select the second option move the kernel from the /stand/ build directory into the /stand/vmunix

Back to the Top

C. Configurable Parameters

HP - UX configurable kernel parameters limit the size of the text, data, and stack segments for each individual process. These parameters have pre - defined defaults, but can be reconfigured in the kernel. Some may need to be adjusted when swap space is increased. This is discussed in more detail in the section on configuring the HP - UX kernel.

	Sets number of buffer pages
bufpages	Store symbolic link data in the inode
create_fastlinks	Sets asynchronous write to disk
fs_async	Controls directory creation on automounted disk
hpux_aes_override	drives
maxdsiz	Limits the size of the data segment.
maxfiles	Limits the soft file limit per process
maxfiles_lim	Limits the hard file limit per processes
maxssiz	Limits the size of the stack segment.
maxswapchunks	Limits the maximum number of swap chunks
maxtsiz	Limits the size of the text (code) segment.
maxuprc	Limits the maximum number of user processes
netmemmax	Sets the network dynamic memory limit
nfile	Limits the maximum number of "opens" in the system
ninode	Limits the maximum number of open inodes in
nproc	memory
npty	Limits the maximum number of concurrent processes
	Sets the maximum number of pseudo ttys

Back to the Top

D. Kernel Parameters

1. bufpages
   Bufpages specifies how many 4096 - byte memory pages are allocated for the file system buffer cache. These buffers are used for all file system I/O operations, as well as all other block I/O operations. Bufpages meant much more prior to having dynamic buffer cache (9.X on workstations, 10.X on servers). dbc_min_pct and dbc_max_pct are the appropriate parameters to use now.

   Back to the Top

   In HP - UX 10.X, it is recommended this kernel parameter be set to 0. This will enable dynamic buffer cache.

2. create_fastlinks

When equal to '1', this tells the kernel to store the path name of the "linked - to" file in the inode, rather than in a data block. This reduces disk space usage and eliminates a disk I/O to retrieve the name. By default, this feature is disabled for backward compatibility. We recommend all systems have create_fastlinks enabled by setting this kernel parameter to 1.

Back to the Top

3. dbc_max_pct
   This parameter determines the percentage of main memory to which buffer cache is allowed to grow. When performing a large amount of block I/O, the system will "grow" the buffer cache to this maximum size. If the system begins to feel pressure due to process space memory requirements, the kernel will shrink buffer cache. The problem arises when there is stress due to process space requirements, and, there is block I/O pressure. The system tries to reclaim buffer cache pages to allocate them to running processes. But the system is also trying to allocate as much buffer cache as it can, causing a vicious cycle of allocating and deallocating memory between buffer cache and process memory space, creating a large amount of overhead. There is a good chance that your system is paging at this point, which is causing even more overhead.

   The idea then, is to keep this number reasonably low, allowing you to have cache space but also keep the application space large enough to avoid high levels of conflict between them. The default value is 50%, but we recommend 25% to start. We have seen systems that need buffer cache to have a max of as little as 5%, with a min at 2%. We have also seen systems that require 80 to 90% buffer cache. You need to determine if your system is going to be used for applications performing large amounts of block I/O, or very little I/O but large (or many) processes, or both. If the answer is "both", you will need an enormous amount of physical memory.

   Back to the Top

4. fs_async
   This kernel parameter controls the manner in which writes of file system meta structures are performed. Asynchronous writes to disk can improve file system I/O performance significantly. However, synchronous writes to disk make it easier to restore file system integrity if a system crash occurs while file system meta structures are being updated. Depending on the application, you will need to decide which is more important.You may value file system integrity more than I/O speed. If so, fs_async should be set to 0.

Back to the Top

5. hpux_aes_override
   This value is part of the OSF/AES compliance. It controls directory creation on automounted disk drives. We recommend hpux_aes_override be set to 1. If this value is not set, you may see the following error message:

   mkdir: cannot create /design/ram: Read - only file system.

   This system parameter cannot be set using SAM. The kernel must be manually created. It is best to modify the other parameters with SAM first and then change this parameter second, otherwise SAM will override your 'unsupported' value with the default.

   NOTE: Apply patch PHCO_11647 if you use this parameter on HP - UX 10. X. Failure to do so can cause some "apparent" corruption in parts of the file system where trasnition links occur.

Back to the Top

7. maxdsiz
   Maxdsiz defines the maximum size of the data segment of a process. The default value of 64 MB is too small for most applications. We recommend this value be set to the maximum value of 1.9Gb. If maxdsiz is exceeded, the process will be terminated, usually with a SIGSEGV (segmentation violation) and you will probably see the following message:

   Memory fault(coredump)

   In this case, review the values of maxdsiz, maxssiz and maxtsiz. For more information on these parameters, please see the on - line Help section within SAM's Kernel Configuration. If you need to exceed the specified maximum of 1.9Gb, there are a couple of ways (yet to be supported) to do so. Contact your Hewlett Packard technical consultant for the details. It is important to note that the maxdsiz parameter must be modified in order for these procedures to work. Maxdsiz will need to be set to 2.75Gb or 3.6Gb depending on the method chosen and/or size required. It will also require a manual creation of a new kernel.

Back to the Top

9. maxfiles
   This sets the soft limit for the number of files a process is allowed to have

open. We recommend this value be set to 200.

Back to the Top

10. maxfiles_lim
    This sets the hard limit for number of files a process is allowed to have open. The default for this kernel parameter, and our recommendation, is 2048.

    Back to the Top

11. maxssiz
    Maxssiz defines the maximum size of the stack of a process. The default value is 8Mb. We recommend this value be set to a value of 79 Mb.

    Back to the Top

12. maxswapchunks
    This (in conjunction with some other parameters) sets the maximum amount of swap space configurable on the system. Maxswapchunks should be set to support sufficient swap space to accommodate all swap anticipated. Also remember, swap space, once configured, is made available for paging (at boot) by specifying it in the file /etc/fstab (/etc/checklist on 9.X). The maximum swap space limit calculated in bytes is: (maxswapchunks * swchunk * DEV_BSIZE). We recommend this parameter be set to 4096.

    NOTE: Never modify the kernel parameter, swchunk.

    Back to the Top

13. maxtsiz
    Maxtsiz defines the maximum size of the text segment of a process. We recommend 1024 MB.

    Back to the Top

14. maxuprc
    This determines the number of concurrent processes that a user can run. A user is identified by the user ID number. Maxuprc is used to keep a single user from monopolizing system resources. If maxuprc is too low, the system issues the following error message to the user when attempting to invoke too many processes:

no more processes

We recommend maxuprc be set to 200.

Back to the Top

16. maxusers
This kernel parameter is used in various algorithms and formulae throughout the kernel. It is used to limit system resource allocation and not the actual number of users on the system. It is also used to define some system table sizes. The default values of nproc, ncallout, ninode and nfile are defined in terms of maxusers. We are recommend fixed values for nproc, ninode and nfile. Set maxusers to 124.

Back to the Top

17. netmemmax
This specifies how much memory can be used for holding partial internet - protocol(IP) messages in memory. They are typically held in memory for up to 30 seconds. The default of 0 allows up to 10% of total memory to be used for IP level reassembly of packet fragments. Values for netmemmax are specified as follows:

Value	Description
- 1	No limit, 100% of memory is available for IP packet reassembly.
0	netmemmax limit is 10% of real memory.
> 0	Specifies that X bytes of memory can be be used for IP packet reassembly. The minimum is 200 Kb and the value is rounded up to the next multiple of pages (4096 bytes).

If system network performance is poor, it might be because the system is dropping fragments due to insufficient memory for the fragmentation queue. Setting this parameter to - 1 will improve network performance, but, at the risk of leaving less memory available for processes. We recommend it be set to - 1 for systems acting as data servers only. For all other systems, we recommend a setting of 0.

Back to the Top

18. nfile
    Nfile sizes the system file table. It contains entries in it for each instance of an open of a file. Therefore, it restricts the total number of concurrent "opens" on your system. We suggest that you set this at 2800. This parameter defaults to ((16 * (nproc + 16 + maxusers) / 10) + 32 + 2 * npty). If a process attempts to open one more (than nfile) file, the following message will appear on the console:

    file: table is full

    When this happens, running processes may fail because they cannot open files, and no new processes can be started.

    Back to the Top

20. ninode
    Ninode sizes the in - core inode table, also called the inode cache. For performance, the most recently accessed inodes are kept in memory. Each open file has an inode in the table. An entry is made in the table for each "login directory", each "current directory", each mount point directory, etc. It is recommended that ninode be set to 15,000.

    NOTE: On a multi - processor system running HP - UX 10 - 20, ninode should NOT exceed 4000. This is due to a spinlock contention problem that is fixed in 11.0.

    Back to the Top

21. nproc
    Nproc sizes the process table. It restricts the total number of concurrent processes in the system. When some person/process attempts to start one more (than nproc) process, the system issues these messages:

    at console window : proc: table is full
    at user shell window: no more processes

    Set nproc to 1024.

    Back to the Top

24. npty
    This parameter limits the number of pty data structures that can be opened. These are used by network programs like rlogin, telnet, xterm,

etc. We recommend this parameter be set to 512.

Back to the Top

E. Kernel Parameter Recommendations

The following are the suggested kernel parameter values.

# Parameter	Value	
bufpages	0	# on HP - UX 10.X
create_fastlinks	1	
dbc_max_pct	25	
fs_async	1	
maxdsiz	2063806464	
maxfiles	200	
maxfiles_lim	2048	
maxssiz	(80*1024*1024)	
maxswapchunks	4096	
maxtsiz	(1024*1024*1024)	
maxuprc	200	
maxusers	124	
netmemmax	0	# on desktop systems
	- 1	# on data servers
nfile	2800	
ninode	15000	# 4000 on HP - UX 10.20 multi - processor systems
nproc	1024	
npty	512	

Back to the Top

7. Networks

In today's networked environments, many installations are client/server configurations. Therefore, network configuration is critical to overall performance and throughput. One HP workstation can almost saturate a single ethernet wire with heavy traffic. See the section labeled Networking later in this document for tuning and configuration guidelines.

Back to the Top

A. NFS

Network configuration will also have an impact on performance. Virtually all installations use some form of local area network to facilitate sharing of data files and to simplify system management. Most installations use NFS to mount remote file systems. This imposes a performance penalty, however, because the I/O bandwidth for accessing data on an NFS mounted disk is less than that for a directly connected disk. There are a few system configuration recommendations that can be made to maximize the convenience that NFS and the local area network provide while minimizing the performance penalty.

- NFS configuration. On NFS servers, a good first order approximation is to run two nfsd processes per physical disk (spindle). The default is four total, which is certainly not enough on a server. On 9.x systems, too many nfsd processes can cause context switching bottlenecks, because all the nfsds are awakened any time a request comes in. On 10.x systems, this is not the case and you can safely have extra nfsd processes. Start with 30 or 40 nfsd's. On NFS clients run sixteen biod processes. In general, HP - UX 10.X has much better NFS performance than previous versions of HP - UX.

- Patches - Always install the latest HP - UX NFS patch. HP periodically releases patches that correct problems associated with NFS, many of them performance related. If you are using NFS, you should make sure the latest patch is installed on both the client and server. See the PATCHES section for more details. General HP - UX patch information can be found at http://us - support.external.hp.com for the Americas and Asia. For europe and others, use http://europe - support.external.hp.com/.

- Local vs. Remote. You will need to determine which things you will NFS mount, and which should be local. For performance, it would be great if nothing was accessed over the network. And "if wishes were horses, dreamers would ride", said John Butcher. Consider doing some of these things and using some of the techniques described near the end of this document, under "NFS".

- Subnetting. In general, it is a bad idea to have too many systems on a single wire. Implementation of a switched ethernet configuration with a multi host server or a server backbone configuration can preserve existing wiring while maximizing performance. If you are doing rewiring, seriously consider using fiber for future upgradability.

- Local paging. When applications are located remotely, set the "sticky bit" on the applications binaries, using the chmod +t command. This tells the

system to page the text to the local disk. Otherwise, it is "retrieved" across the network. Of course, this would only apply when there is actual paging occurring. More recently, there is a kernel parameter, page_text_to_local, which when set to 1, will tell the kernel to page all NFS executable text pages to local swap space.

- File locking. Make sure the revisions of statd and lockd throughout the network are compatible; if they are out of sync, it can cause mysterious file locking errors. This particularly affects user mail files and Korn shell history files.

- Design the lan configuration to minimize inter segment traffic. To accomplish this you will have to ensure that heavily used network services (NFS, licensing, etc.) are available on the same local segment as the clients being served. Avoid heavy cross segment automounting.

- Maximize the usage of the automounter. It allows you to centralize administration of the network and also allows greater flexibility in configuring the network. Avoid the use of specific machine names which may change over time in your mount scheme; force mount points that make sense. /net ties you to a particular server, which may change over time.

- You can watch the network performance with Glance, the netstat command, and the nfstat command. There are other tools like NetMetrix or a LAN analyzer to watch lan performance. Additionally, you can use PerfView and MeasureWare/UX to collect data over time and analyze it. You may want to tune the timeo and retrans variables. For HP systems, small numbers 4 for retrans and 7 for timeo are good. The default values for wsize and rsize, 8K, are almost always appropriate. Do NOT use 1024 unless talking to an Apollo system running NFS 2.3 on SR10.3. 8K is appropriate for 10.4 Apollos running NFS 4.1.

- Investigate the use of dedicated servers for computing, file serving, and licensing. A good scenario has a group of dedicated servers connected with a fast "server backbone", which is then connected to an ethernet switch, which is itself connected to the desktop systems.

Make sure that ninode is at least 15000 on HP - UX 10.X. Remember to not go above 4000 on an HP - UX 10.20 system with multi - processors, as previously stated. Some customers have seen performance degradation on Multi Processor systems when ninode is greater 4000. Check it on your system. The details of this problem are much too detailed and complicated for this document.

NFS file systems should be exported with the async option in /etc/exports.

Some items that can be investigated...

nfsd invocations

- nfsstat - s

UDP buffer size

- netstat - an | grep - e Proto - e 2049

How often the UDP buffer overflows / UDP Socket buffer overflows

- netstat - s | grep overflow

NFS timeouts...are they a result of packet loss? Do they correlate to errors reported by the links? Use lanadmin() or netstat - i to check this.

IP fragment reassembly timeouts?

- netstat - p ip

mounting through routers?

- check to see if routers are dropping packets

check for transport bad checksums

- netstat - s

is server dropping requests as duplicates?

- nfsstat

is client getting duplicate replies? (badxid)

- nfsstat on CLIENT

Some customers have mentioned that they have had serious problems because of too many levels of hierarchy within the netgroup file. It seems that this file is re - read many times, and the more hierarchy, the longer it takes to read.

Back to the Top

8. Patches

Since patch numbers change frequently, it is recommended that you always check for the latest information. Here are some general recommendations:

- Always load the latest kernel "megapatch", ARPA transport patch, NFS/ automounter patches, statd/lockd patches, and SCSI patch. Many performance and reliability improvements are delivered by these patches.
- Load the latest C compiler and linker patches.
- Load HP - VUE or CDE, and X/Motif patches at your discretion. These usually contain bug fixes.
- Load the latest X server patches.

A. How to get patches. If you have WWW access go to http://us - support.external. hp.com, and follow the links to the patch list. This is also a good way to browse the latest patch list. You can also get patches by e - mail. If you know what the name of the patch you want is, send a message to support@support.mayfield.hp. com, with the text "send patchname". Don't forget to substitute the name of the patch you want for "patchname". You can get a current list by sending the text "send patchlist". To get a complete guide on using the mail server, send the text "send guide". If you have HP SupportLine access, then patches can be requested from the HP SupportLine at (800)633 - 3600, and are also available for FTP access.

Back to the Top

B. How to tell what patches are loaded. First scan the directory /etc/filesets (9.x) systems, or use the swlist command (10.x). Patches are named PHxx_nnnn, where xx can be KL, NE, CO, or SS. nnnn refers to the patch number, which is always unique no matter what PHxx category is specified. If a patch has been loaded on a 9.x system, a file will exist in /etc/filesets, with the same name as the patch. If a patch has been loaded on a 10.x system, the patch should be listed in the output of swlist.

Back to the Top

C. How to load patches. Patches are shipped as shell archives, named after the patch. Unpack the shell archive, check the README file and follow the directions.

Back to the Top

D. Patch management. Patch management can be a fulltime job for a large site. HP recommends that large sites that don't want to tackle that particular task purchase the PSS support option. This service provides a consultant who,

among other things, provides patch management. It's well worth the money.

Back to the Top

9. Performance and the PATH Variable

A. The PATH Variable

This is one of the most abused areas that causes performance problems. PATH variables that are way too long AND the positioning of the directory that contains the most frequently used tools (by the application), at the end. Try to limit your PATH statement to the paths that are most useful to your primary application. Consider writing startup scripts (wrappers) for the lesser used applications. In these wrappers, you may embed additional PATH statements to meet the needs of the application.

Back to the Top

10. HP - UX 11.0

As of this revision, I feel it is "early in the game" to detail everything for 11.0. I do think it is appropriate to discuss various aspects of 11.0, specifically those that seem to be problematic.

Back to the Top

A. Points of Interest

"FALSE" DEACTIVATIONS. There is an issue regarding process deactivations when there seems to be NO paging. The following is the description and patch and recommended work around AS OF THIS DATE (MARCH 17, 2000).

This applies to BOTH 10.X and 11.X versions of HP - UX.

When there are memory mapped files, the writes show up as pageouts to disk, even though there really is NO MEMORY PRESSURE. Part of the problem is/are the kernel NFS routines. If the pageout rate is anything but zero, there is a change in the logic that assumes (incorrectly) that we are under memory pressure. It causes a process to fire up and start purging buffer cache pages. This causes a performance degradation, especially if the system has a large buffer cache.

Patch PHKL_17869 added a new counter in the kernel, k_pageout_count, used

by thrashing logic to see if we are under memory pressure. This "corrects" the deactivation issues...BUT...the macro (in bcache.h) that NFS relies upon to determine if we are under memory pressure was NOT changed...it still uses the "old" counter for pageout count. It reflects cumulative pageouts including memory mapped I/O files.

Anything that references the old counter is getting a skewed view of whether we're under memory pressure or not. Because NFS thinks that we're under pressure, it tries to "re - use" a bunch of internal kernel structures. The overhead and bookeeping to do this is causing performance problems. It REALLY should use the new counter.

Typically (until the kernel can be modified), the way to get around the problem... reduce the size of buffer cache. This is actually what the response center will recommend to any customer that appears to be suffering from this exact problem.

FALSE SENSE OF PAGING. There is a situation apparent (seemingly) on 11.X systems only, where the system appears to paging when there is absolutely no memory pressure. I have seen systems with as much as 3 or 4GB free and pageouts to disk are occurring. The issue is with applications that are performing operations on memory mapped files. It seems that memory mapped writes are causing this pageout activity. As of this date (March 15, 2000), I have yet to find/get an explanation of why this happens.

This is not pageout in the sense that the system is under memory pressure. I am currently working on an explanation. Once the "mystery" : -) is solved, it should be apparent whether it is the kernel that should be modified...or the tools that are reporting the paging activity.

PLEASE NOTE:

The following paragraph was written in June 1999. It may be obsolete information now, but I wanted to keep it in the paper. You may want to check for a patch, and if it exists, check for installation and/or install it.

As of this writing, there is a problem that causes the system to crash. Currently, there is no patch for it, but, there is a work - around. Until patched, you must make sure that the kernel parameter page_text_to_local is is turned off ('0') AND executables have the sticky bit turned OFF. Refer to the manual pages for chmod(1), if you need to know more about the sticky bit.

Back to the Top

B. lotsfree, desfree and minfree

There are three kernel parameters that have recently become tuneable by mortals. They are the "paging" parameters lotsfree, desfree and minfree. These parameters define thresholds that the kernel uses to determine swapper/vhand (the page daemon) behavior. Here is the short version, in english (sort of :-)), of how these parameters are used...

lotsfree -- vhand begins to "age" pages. There is another parameter that is dynamically modified, gpgslim, which is where vhand begins to "steal" pages. gpgslim starts at one quarter the distance between desfree and lotsfree, and "moves" between them, based on memory pressure.

desfree -- more serious, more furious :-) paging begins here. Much of vhand's behavior is modified at this point... how often it wakes up, how many pages to look at, what is the distance between the age and steal hands, how many pages to steal, etc.

minfree -- at this point, the system is deactivating processes. In the "old" days this would have been where swapping took place. We no longer swap processes.

I have noticed, on several occasions, that these parameters have been set way too high. It was very apparent on several V class machines. The suggested values are:

on a system with up to 2GB of memory:

- lotsfree no larger that 8192 (32MB)
- desfree no larger than 1024 (4MB)
- minfree no larger than 256 (1MB)

on a system with 2GB to 8GB of memory:

- lotsfree no larger than 16384 (64MB)
- desfree no larger than 3072 (12MB)
- minfree no larger than 1280 (5MB)

on a system with a whole group of memory:

- lotsfree 131072
- desfree 32768
- minfree 8192

Back to the Top

C. Text, Data and Shared Objects Maximum Values

The "new" maximums for text, data and shared objects on 64 bit HP - UX, for a 64 bit executable are 4TB for both text and data and 8TB combined for shared objects. The maximum size of any single shared object is 1GB.

Back to the Top

D. EXEC_MAGIC

A EXEC_MAGIC executable is only "legal" if it is a 32 bit application on either 64 or 32 bit HP - UX. The "old" maximums are still in effect... 1.9GB of data space, unless the "current" documented malloc() and mallopt() call combination is used. In that case, you have just under 4GB as a maximum.

Back to the Top

E. The New Parameters for Text, Data and Stack

Don't forget to adjust the new parameters maxtsiz_64bit, maxdsiz_64bit and maxssiz_64bit to meet your application requirements.

Back to the Top

F. Variable Size Pages

The chatr() command

Let's talk a little bit about chatr() first. The command gets its' name from change attribute. It will change a programs' internal attributes. The man page tells you that by default, chatr() prints each file's magic number and file attributes. Let us digress... if you do not "know" magic numbers, or, more important - if you do not know HOW THE HECK to interpret what chatr() is displaying vs. what the man page describes as the magic number...yer DEAD. For example, when chatr()'ing an EXEC_MAGIC, you will see "normal executable" as the first attribute. You obviously know that this means EXEC_MAGIC...right? : -). If you were to od() the executable (- x), you would see that the third and fourth bytes were 0107. Obvious, eh? I think not. Check out the man page for magic(4). You will see that the first group of "#define" statements that you encounter will actually use the words like EXEC_MAGIC and the associated hex number like 0x107 AND the comment field will look something like /* normal executable */. For example:

```
#define EXEC_MAGIC      0x107   /* normal executable */
```

Whew!...had enough? Get it? OK. Let's move on.

The only machines that support variable pages are the PA - 8000 and any "follow - ons" based on the PA 2.0 arcitecture. Partial support of variable size pages has existed since HP - UX 10.20.

The Benefits

First, we need to take a look at the Translation Lookaside Buffer (the tlb). It is a small, high speed piece of hardware. The "current" sizes are:

- PA8000 - 96 entries
- PA8200 - 120 entries
- PA8500 - 160 entries

There are NO hardware walkers. There ia a large penalty (cycle time) to perform tlb miss handling. Applications with large data sets will spend a lot of time handling tlb misses.

Using variable sized pages:

- a larger piece of virtual space can be mapped with a single tlb entry
- large reference sets can be mapped with fewer tlb entries
- fewer entries result in fewer tlb misses.

Many/most applications will NOT realize a performance increase. An application that spends very little time handling tlb misses will not improve much, if at all.

How Do I Use Variable Sized Pages?

Determine the page size that seems to be best for the application. Set the appropriate page size for text and/or data pages. Use the chatr() command to do this.

- +pi used for text pages
- +pd used for data pages

Valid page sizes range from 4K to 256MB. Take care when using the -L option (Large). This option requires that the user executing the application posess MLOCK privelege (see setprivgrp(1m)).

New Kernel Parameters

There are three new kernel parameters, tunable by you. They are:

- vps_pagesize
  - default/minimum page size selected
- vps_ceiling

        - maximum page size selected by the kernel
    - vps_chatr_ceiling
        - maximum size page that can be chatr()'d

Variable Page Size "Inhibitors"

OR...why am I not getting my variable sized pages? I cannot go into the technical details here, but, you can find the more verbose explanation of the reasons in the white paper on variable sized pages. Here is my "list" of reasons.

- physical memory size
- dynamic buffer cache
- start of virtual alignment
- re - running a recently chatr()'d application
- page demotion
- physical size inflation (performance degradation)

G. Memory Windows

Why Use Memory Windows?

In HP - UX 10.X, the limitation on the maximum (total) size of shared "objects" is 1.75GB system wide. Typically, there are many processes sharing a limited amount of memory. This memory is utilized by all processes using any/all of the "shared object" stuff...shared memory, shared libraries and memory mapped files. Often just three or four processes need all or most of the available memory. It should be noted that only 32 bit processes are affected by this limitation. With a 8TB limitation on shared object space in 64 bit land, I assume it will be a couple of months before one of you require more! More important... ya don't need memory windows.

Disadvantages

Applications using a memory window cannot "see" any other memory window. Also, the current memory window is inherited accross fork(). This could cause starnge behaviour. And...should applications in multiple memory windows have a need to communicate, they would need to use the standard system wide shared memory space, .75GB in the 4th quadrant.

When Should Memory Windows Be Used?

On HP - UX 10.X, you could not have (even) two "sets" of processes, each sharing a 1GB segment of shared memory. This is just impossible when the total amount of shared space is 1.75GB (actually, less). On 11.X you can have multiple sets of processes, each sharing 1GB of memory. A typical example of

this would be multiple instances of Oracle, SAP, etc. So. Sites or users that need to run multiple concurrent processes or applications that require the sharing of LARGE amounts of memory, should use memory windows.

Memory Window Usage

Memory windows are created programmatically with two new system calls, getmemwindow(2) and setmemwindow(2). If you are familiar with ipcxxx(2) type system calls, these are similar. Be aware of the new kernel paramater max_mem_window. It defaults to 0, so make sure you modify it.

LARGE Memory Windows

When using memory windows as discussed above, the "global window" (typically the first one) could get a maximum of 1.75GB, and each memory window thereafter could get 1GB. One could obtain 2.75GB for the global and 2GB for sny subsequent windows. This can be accomplished with a SHMEM_MAGIC executable. You make an executable SHMEM_MAGIC using the -M option to ld(), or the -M option to chatr().

The REAL maximums...I always tell people that you will actually get...2.75GB (or 1.75GB) MINUS all shared memory space currently in use, MINUS all shared library space currently in use, MINUS all memory mapped file space currently in use. I have seen as much as 2.6GB.

H. Spinlock Pool Parameters

This was "borrowed" from the web page on Configurable Kernel Parameters.

The following parameters, all related to spinlock pools for multi-processor computers, are used similarly and are documented together here. Each parameter allocates the specified number of spinlocks for the corresponding system resource:

bufcache_hash_locks

Buffer-cache spinlock pool

chanq_hash_locks

Channel queue spinlock pool

ftable_hash_locks

File - table spinlock pool

io_ports_hash_locks

I/O - port spinlock pool

pfdat_hash_locks

Pfdat spinlock pool

region_hash_locks

Process - region spinlock pool

sysv_hash_locks

System V Inter - process - communication spinlock pool

vnode_cd_hash_locks

Vnode clean/dirty spinlock pool

vnode_hash_locks

Vnode spinlock pool

These parameters are for use by advanced users only who have a thorough understanding of how spinlocks are used by multiple processors and how the number of spinlocks needed are related to system size and complexity. Do not change these from their default value unless you understand the consequences of any changes. In general, these values should not be altered without the advice of HP support engineers who are thoroughly familiar with their use.

Setting these parameters to inappropriate values can result in severe performance problems in multi - processor systems.

Following is a list of acceptable values. All of these parameters have the same minimum and maximum values. Only the defaults are different as indicated:

Minimum

64

Maximum

4096

Default

64 (ftable_hash_locks, io_ports_hash_locks)

Default

128 (bufcache_hash_locks, pfdat_hash_locks, region_hash_locks, sysv_hash_locks, vnode_hash_locks, vnode_cd_hash_locks)

Default

256 (chanq_hash_locks)

Specify a value that is an integer exponent of 2. If you specify any other value, SAM or the kernel itself changes the parameter value to the next larger integer exponent of two (for example, specifying 100 results in the spinlock - pool value of 128.

Description

In simple terms, spinlocks are a mechanism used in multiple - processor systems to control the interaction of processors that must be held off while waiting for another processor to finish a task so the results can be passed to the waiting processor. Spinlocks control access to file - system vnodes, I/O ports, buffer cache, and various other resources.

Earlier HP - UX versions allocated a fixed number of spinlocks for all resources, but beginning at HP - UX 11.0, spinlocks can be allocated for each resource type to accommodate very large and complex systems.

In general, if the system is encountering lock contention problems that are associated with one of these hashed pools, first identify the resource spinlock pool that is associated with the contention, then increase the spinlock - pool parameter for that resource. Spinlock pools are always an integer power of two. If you specify a value that is not, the kernel always allocates a value that is the next larger integer exponent of two.

As stated above, these parameters are for use by experienced, knowledgeable system administrators only. They should not be altered unless you are quite certain that what you are doing is the correct thing to do.

Back to the Top