HP-UX Performance Cookbook

By Stephen Ciullo, HP Senior Technical Consultant
and
Doug Grumann, HP Performance Technology Center R&D Lead Engineer
revision 27JUL02

You are about to enjoy reading a how-to cookbook for general analysis of HP-UX system performance. This cookbook has been "out" for a couple years now, with various minor edits and improvements over time (though Stephen's slang never seems to improve). We hope to keep the cookbook a living document as long as we are. One needs to have goals in life. Speaking of goals: optimizing performance is a good goal, and a very complex subject, which is why we put this together! We don't delve deeply into any single facet of tuning, such as optimizing for particular disk types or apps, but you will see good general information that will make your life easier as you continue to pursue *optimal performance* on the systems you manage. You know you've achieved *optimal performance* when you say to yourself: "Hey, nobody is bugging me about how slow their application is running!"

Our target audience is the system administrator who is somewhat familiar with the HP performance tools. We'll use Glance and OpenView Performance Agent metrics for our thresholds (OVPA was called MeasureWare). Some of these metrics are also available in other tools. Our focus will be on PA HP-UX 11x.

Let's get some general rules of thumb straight right at the beginning, so they won't gum up the works later:

- Don't fix things that ain't broke. If your users are happy with performance, then why muck with it? You got better things to do. Take some time to build up knowledge of what "normal" looks like on your systems. Later, if something goes wrong, you'll be able to look at historical data and use that knowledge to drill down quickly to the problem.

- You have to be willing to do the work to know what you're doing. Did you read that sentence twice? If you really have no idea why you're changing something, or what it means, then do the research first before you shoot yourself in the foot.

- When you do make changes, *try* to change one thing at a time. If you reconfigure 12 kernel variables all at once, chances are things will get worse anyway, but even if it helps you'll never know which change made the difference. If you tweak only one thing, you'll be able to evaluate the impact and build on that knowledge.

- None of the information in this paper comes with a guarantee. If this stuff were simple, we would have to find something else to keep us employed (like Web page design). If any recommendation we provide doesn't work for you, please let us know — but don't sue us!

- The age-old answer to every performance question is: "It Depends." Every system is different, and approaches that work great on one system may not work on another. You know your systems better than we do, so keep that in mind.

If you want to get your money's worth out of reading this document (remember how much you paid for it?), then scour every paragraph from here to the end. If you're feeling lazy (like us), then skip down to the Resource Bottlenecks section unless you are setting up a new machine. For each bottleneck area down there, we'll have a short list of ingredients. If your system doesn't have those ingredients, then skip that subsection. If your situation doesn't match *any* of our bottleneck recipes, then you can tell your boss that you're done, you're bored, and you really think they should throw you more work to keep you busy!

System Setup

If you are setting up a system for the first time, you have some choices available to you that people trying to tune 24x7 production servers don't have. We're sure you have intensely researched system requirements, analyzed various hardware options, and of course you've had the *most bestest* advice from HP as to how to configure the system. Or *not*. It's hard to tell whether you've bought the right combination of hardware and software, but don't worry, because you'll know shortly after it goes into production.

CPU Setup

If you're not CPU-bottlenecked on a given system, then adding more processors will do no good. If you have a CPU-intensive workload (and this is common), then more CPUs are *usually* better. Some applications scale well (nearly linearly) as the number of CPUs increases: this is more likely to happen for workloads spending most of their CPU time in User mode as opposed to System mode, though there are no guarantees. Some applications definitely don't scale well with more processors (for example, an application that has only one single-threaded process!). For some workloads, adding more processors introduces more lock contention, which reduces scaling benefits. In any case, faster (newer) processors will significantly improve throughput on CPU-intensive workloads, no matter how many processors you have in the system. Processor speed is, of course, a factor, but newer processors like the PA8500 to PA8700 have significant TLB, cache, and pipelining improvements that generally make them *even more lots better*!

Memory Setup

Hey, memory is cheap so buy lots (a hardware vendor's point of view). Application providers will usually supply some guidelines for you to use for how much memory you'll need, though in practice it can be tough to predict memory utilization. You do *not* want to get into a memory bottleneck situation, so you want enough memory to hold the resident memory sets for all the applications you'll be running, plus the memory needed for the kernel, plus dynamic system memory such as the filesystem buffer cache.

If you're going to be hosting a database, or something else that benefits from a large in-memory cache, then its even more essential to have enough memory. Oracle installations, for example, can benefit from "huge" SGA configurations (gigabyte range) for buffer pools and shared table caches.

Resident memory and virtual memory can be tricky. Operating systems pretend to the applications that there is more memory on your system than there really is. This is called Virtual Memory, and is essentially the amount of memory allocated by programs for all their data, including shared memory, heap space, program text, shared libraries, and memory-mapped files. The total amount of virtual memory allocated to all processes on your system roughly translates to the amount of swap space that is reserved (with the exception of program text). Virtual memory actually has little to do with how much actual physical memory is allocated because not all data mapped into virtual memory will be active ("Resident") in physical memory. Confused yet? Hey, memory is cheap so buy lots.

Disk Setup

You may have planned for enough disk space to meet your needs, but also think about how you're going to distribute your data. In general, more smaller disks are better than fewer bigger disks, as this gives you more flexibility to move things around to relieve I/O bottlenecks. You should try to split your most heavily used logical volumes across several different disks and I/O channels if possible.

When determining directory paths for applications, try to keep the number of levels from the filesystem root to a minimum. Extremely deep directory trees may impact performance by requiring more lookups to access files. Conversely, file access can be slowed when you have too many files (multiple thousands) in a given directory.

Swap Devices

You're going to want to configure enough swap to cover the largest virtual memory demand your system is likely to hit (at least as much as the size of physical memory). Do

not enable pseudo swap unless you must (if you don't have enough spare disk space for swap). If you have plenty of swap configured, enabling pseudo swap does *nothing* for your system…it was invented so that those systems that had *less* swap configured than physical memory would be able to *use* all of their memory. The idea is to *configure* lots of swap so that you don't run into limits reserving virtual memory in applications, without, in the end, actually *using* it (i.e., have it there but avoid paging to it). You avoid paging out to swap by having enough physical memory so that you don't get into a memory bottleneck.

For the disk partitions that you dedicate to swap, the best scenario is to divide the space evenly among drives with equivalent performance (preferably on different cards/controllers). For example, if you need 16GB of swap and you can dedicate four 4GB drives of the same type hanging off four separate cards, then you're set. If you only have differing drives of different sizes available for swap, take at least two that are of the same type and size and make them the highest priority (lowest number). This enables page interleaving, meaning that paging requests will "round robin" to them. You don't want to page out to swap, but if you do start paging then you want it to go fast.

You can configure other lower-priority swap devices to make up the difference. The ones you had set at the highest priority are the ones that will be paged to first, and in most cases the lower-priority swap areas will have their space "reserved" but not "used," so performance won't be an issue with them. It's OK for the lower-priority areas to be slower and not interleaved. We'll talk more about swap in the Disk and Memory Bottlenecks sections below.

Logical Volumes

Generally, your application/middleware vendor will have the best recommendations for optimizing the disk layouts for their software. Database vendors may recommend bypassing the filesystem (using raw logical volumes) for best performance. With newer disk arrays like the XP, performance on "cooked" volumes is equivalent. In any case, it's a good idea to assign different applications to unique volume groups (physical disks) to reduce the chance of them impacting each other.

There's a lot of LVM functionality builtin to support High Availability. Options such as LVM Mirroring (writing multiple times) and the LVM Mirror Write Cache are "anti-performance" in most cases. Sometimes for read-intensive workloads, mirroring can improve performance because reads can be satisfied from the fastest disk in the mirror, but in most cases you should think of LVM as a space management tool — it's not built for performance. Stephen tells customers "There comes a time when you have to decide whether you want High Availability or Performance: ya can't have both, but you can make your HA environment perform better."

LVM Parallel scheduling policy is better than Serial/Sequential. LVM striping can help with disk I/O-intensive workloads. You want to set up striping across disks that are

similar in size and speed. If you are going to use LVM striping, then make the stripe size the same as the underlying filesystem block size. In our experience (over *many* years) the block size should not be less than 64K. In fact, it should be quite a bit larger than 64KB when you are using LVM striping on a volume mounted over a hardware-striped disk array. Many large installations are experimenting with LVM striping on large disk arrays such as XP and EMC. A general rule of thumb: use hardware (array) striping first, then software (LVM) striping when necessary for performance or capacity reasons. *Be careful* using LVM striping on disk arrays: you should understand the combined effect of software over array striping in light of your expected workload. For example, LVM striping many ways across an array, using a sub-megabyte block size will probably defeat the sequential prefetch algorithms of the array.

Optimizing disk I/O is a science unto itself. Use of in-depth array-specific tools, Dynamic Multi-Pathing, and Storage Area Management mechanisms are beyond the scope of this discussion.

Filesystems - VxFS

If you are using filesystems, VxFS (JFS) is preferable to HFS. If you use HFS, set the block size to 64K and the fragment size to 8K. The JFS block size is generally not important.

For best performance, it is wise to have the Online (advanced) JFS product. Using it, you can better manipulate specific mount options and adjust for performance (see man pages for `fsadm_vxfs` and `mount_vxfs`). Some of the options below are available only with Online JFS.

In general, for VxFS filesystems use these mount options:

```
delaylog, nodatainlog
```

For VxFS filesystems with primarily *random* access, use:

```
mincache=direct, convosync=direct
```

"What???" The short version: When access is primarily random, any read-ahead I/O performed by the buffer cache routines is "wasted": logical read requests will invoke routines that will look through buffer cache and *not* get hits. Then, performance degradation results because a physical read to disk will be performed for nearly every logical read request. When mincache=direct is used, it causes the routines to bypass buffer cache: I/O goes *directly* from disk to the process's own buffer space, eliminating the "middle" steps of searching the buffer cache and moving data from the disk to the buffer cache, and from there into the process memory. If mincache=direct is used when read patterns are very sequential, you will get *hammered* in the performance arena, because very sequential reading will take *big* advantage of read ahead in the buffer cache, making logical I/O wait less often for physical reads. You want much more logical than physical reading for performance (when patterns are sequential).

Here is an example of the exception to the rule: We have seen special cases such as a large, *32-bit* Oracle application in which the amount of shared memory limited the size of the SGA, thus limiting the amount of memory allocated to the buffer pool space; *and* (more important) Oracle was found to be reading *sequentially* 68 percent of the time! When the mincache=direct option was *removed*, (and the buffer cache *enlarged*) the number of physical I/Os was greatly reduced which increased performance *substantially*. Remember: this was a specific, unique, *pathological* case; often experimentation and/or research is required to know if your system/application will behave this way.

On /tmp and other "scratch" filesystems where data integrity *in the unlikely event of a system failure* is not critical, use the following mount options:

`tmplog` *OR* `nolog`, `mincache=tmpcache`, `convosync=delay`

There is almost always a JFS "mega-patch" available. Keep current on JFS patch levels for best performance.

If your application will be I/O-intensive and HFS filesystem-based, then we recommend you turn the kernel configuration option `fs_async` on. This decreases recoverability somewhat, as more data could be lost if the system crashes, but in most cases the risk will be worth it. You should have a decent backup/recovery strategy in place regardless, and UPS to avoid downtime due to power outages.

Network Setup

Every networking situation is unique, and although networking can be the most important performance factor in today's distributed application environments, there is little available at the system level to tune networking, at least via sam. There is the `ndd` tool. This `ndd(1M)` program (formerly named `nettune`) is an acronym for: Novices Don't Dare. You can dig up more information about ndd and net tuning in general from the "briefs" directory in the HP Networking tools contrib archive mentioned in the References section at the end of this paper.

Some general tips:

- Make sure your servers are running on at least as fast a network as their clients. We've seen servers on a 10Mbit LAN trying to handle many clients running on 100Mbit!
- Make sure your network card is running full duplex. Some auto-negotiation protocols with the network hardware may inadvertently set your card into half-duplex mode. Stephen recommends you never ever turn autonegotiation on! It seems to mess up all the time! In any case make absolutely sure the duplex settings match at both ends of the cable. If your system's Network Interface Card is "hardwired" to 100FD, then your corresponding switch port *must* be at 100FD or you will be in a world of hurt!

- Record and periodically examine the network topology and performance, as things always tend to degrade over time. Invest in Network Node Manager or other network monitoring tools.
- Use PCI NICs wherever possible. If that is not possible, use HSC NICs. If that is not possible, then and only then use EISA or HP-PB NICs, but really consider upgrading to a system with PCI slots as soon as you can. HP-PB and EISA do not do justice to 100Mbit networking. HSC does not do justice to Gigabit Networking.

NFS setup

Here's some general advice when setting up an NFS environment:
- Read Dave Olker's presentation on NFS performance (see References section at end).
- Use NFS V3. Remember the clients need to be talking V3 as well as the server. Consider going to NFS/TCP.
- If using NFS V2, export filesystems with the async option whenever possible.
- Use of the automounter can cause unproductive flushes of data from the buffer cache. If possible, use AutoFS instead of Automount. Avoid CacheFS for now.
- Bump up the number of nfsd daemons on the server to twice the number of physical disks you are exporting, but don't go above 200 nfsds.
- Keep the filesystem buffer cache big on the server (see discussion below) but small on clients.
- On clients, run 16 biods unless you want to experiment around.
- If you want to experiment around, read the NFS performance presentation first!

For both clients and servers, make sure you keep current on the latest NFS, networking, and performance-oriented kernel patches!

Kernel Tunables

Stephen has an old story about some sam templates (no longer shipped!) that had a bad timeout tunable value in them. The moral of the story is never to blindly accept anybody's recommendations about kernel tunables (sometimes even HP's recommendations — hey wait who do we work for again?!? Oh yeah... the *new* HP!). Stephen tends to get passionate (not in a good way) about people who come up with simple-minded "one size fits all" guidelines for setting up the configurable kernel parameters. If you manage thousands of systems with similar loads, then by all means come up with settings that work for you, and propagate them. But if you can take the time to tune a kernel specific to the load you expect on any given system, then Stephen says: "Do that". Stephen has also been working closely with various groups at HP… many kernel parameter defaults will be more better, and there are much more better (of the afore-mentioned) templates coming your way in 11.2x! What follows is a brief rundown of our general recommendations for the tunables that are most important to performance. For background as to the definitions of these parameters, their ranges, and

additional information, look at the sam utility's online help. Some of the tunables below are controlled (by default) via formulas calculating off other tunables: this is OK as long as you don't skyrocket something like `maxusers` to a high value, without thinking about other parameters that may be influenced. Many of the default tunable settings are OK, and as HP-UX 11i progresses into the 11.2x's, more of the tunable become dynamic, which helps, but in any case what follows are the ones we normally think are worth normally thinking about:

`bufpages`
Use this to set the number of pages in a fixed-size file system buffer cache. If you set `bufpages`, then make sure `nbuf` is zero. If `bufpages` or `nbuf` are non-zero, `dbc_min_pct` and `dbc_max_pct` are *ignored*. In order to get a 400-megabyte fixed buffer cache, which is our recommendation on any 11.0 system with a gigabyte or more of memory, set `bufpages` to 102400. For HP-UX 11i version 1 (otherwise, and henceforth, known as 11.11), or for big 11.0 file servers such as NFS, ftp, or Web servers; you can increase the buffer cache size so long as you don't cause memory pressure. With a gigabyte or more of memory, start with `bufpages` at 204800 on 11.11. See our Buffer Cache discussion under the Disk Bottlenecks section for more information. Also, take note of the "exception" in the Filesystem –VxFS section above.

`create_fastlinks`
This will speed up path lookups when you create links on HFS filesystems. It doesn't do anything for VxFS (VxFS does this already). Set to 1 if your key filesystems are on HFS.

`dbc_max_pct`
This determines the percentage of main memory to which the dynamic filesystem buffer cache is allowed to grow (when `nbuf` and `bufpages` are zero). The default is 50 percent of memory, but this is major overkill in most cases. With a huge bufcache, you're more likely to get into a situation where free memory is low and you'll need to pageout or shrink the buffer cache in order to meet memory demands for active processes. You do not want to get into that situation. If you really want to use a dynamic buffer cache, start with `dbc_max_pct` at 25. If you have over 2 gigabytes of physical memory, especially on 11.0, start with it even smaller. We have a subsection below delving more into Buffer Cache issues.

`default_disk_ir`
This setting tells all disk devices on the system to enable immediate reporting (no wait on disk I/O completions). This is equivalent to doing a `scsictl –m ir` on every disk device. It has NO effect on complex storage devices that have their own cache mechanisms (like XP), but most systems have some "regular old disks" in them. The default is 0, but set this to 1 as a rule. There is no downside that we know of to having this set to 1 (no impact on data integrity).

`eqmemsize`
One of the more esoteric tunables, used for I/O memory pool sizing. Normally don't muck with this unless directed to by HP: on large systems with lots of LUNs this may

need to be bumped up to 8192 or higher. It doesn't have anything to do with performance but can cause system problems if too low.

`fs_async`
This will allow asynchronous writes of file metadata to disk for HFS filesystems. This speeds up performance at a nominal potential risk to data integrity (if the system crashes then some filesystems may be more likely to need repair). This has no effect on VxFS filesystems where metadata is recoverable from the intent log. Don't turn `fs_async` on when Oracle (or any other third party) tells you that they won't support you if it's on. Otherwise, the only reason not to have it on is if you don't do backups, or you don't trust us!

`max_thread_proc`
Maximum number of threads allowed in each process, set to 200, unless told otherwise by your more knowledgeable software vendor. If you are running Java workloads or you're configuring an NFS/TCP server (running `nfsktcpd`), bump this up to 1000.

`maxdsiz` and **`maxdsiz_64`**
Data size limit for 32bit and 64bit applications, respectively. The default 32bit limit (just 64MB) is frequently hit by processes, and so it should be bumped up. Set them both to the highest data set size of any program you'll be running, or just bump them up to their maximum values. The only risk with huge limits is that programs with memory leaks are more likely to degrade performance rather than aborting. To be safe you can bump up `maxssiz` and `maxtsiz` as well, but these limits are less commonly hit.

`maxfiles` and **`maxfiles_lim`**
Soft and hard limits on per-process file opens — quite often set too low. Bump `maxfiles` and `maxfiles_lim` to 200 if they're not already 200, and higher on Web and file servers (2000). Bump them up if you encounter "Too many open files" program failures. For 11.11, `maxfiles_lim` was one of the new "dynamic tunables," which means you can bump it with the `kmtune(1M)` command... see our discussion of dynamic kernel parameters at the end of this section.

`maxswapchunks`
Maximum number of swap "chunks" (a rocket scientist technical term). With the default `swchunk` tunable of 2048, setting `maxswapchunks` to 16384 allows for a maximum possible swap configuration total of 32 gigabytes. There's no reason to set it lower, even if you're not going to configure that much swap. Don't modify `swchunk` unless you need to in order to get past the limit. By setting this too small, we can defeat the whole idea of being able to "add device swap on the fly." The kernel will multiply `maxswapchunks` times `swchunk` times `dev_bsize` – whatever that becomes, you won't be able to add any more swap than that.

`maxuprc`

Limit on processes for non-root users. Set this to 200 if its not already more than that. Fifty processes per user default limit is often not enough. This is another dynamic tunable in 11.11.

`maxusers`
This parameter has nothing to do with the maximum number of users. Don't try to figure it out. A good value for most systems to start with is 128. This is used to size many other tunables, by default, so don't go overboard. Normally 512 would be the max `maxusers` we'd recommend. By default, this is used in a bunch of those sam configuration parameter formula thingys to size different other tunables: so watch it.

`nfile`
The maximum number of concurrent open files (that is, the number of concurrent `open()`s) on the system. The default formula usually works this out to around this value, but 3000 is a pretty good starting point. If you have a lot of filesystem activity, you can bump this up higher without issues (a large Informix shop had this set to 80000!). Bump `nfile` up if you see high File Table utilization (>80 percent) in Glance (System Tables Report) or get "File table overflow" program errors. Use a similar approach for `nflocks` (max file locks). If you are configuring a big filesystem server then you're more likely to want to bump up these limits.

`ninode`
This only affects the inode cache size for HFS filesystems. The VxFS cache is configurable starting in 11.2x and also 11.11 patch bundles via `vx_ninode` but its default value is fine. Note, though, that the kernel configurable `ncsize` that controls the Dynamic Name Lookup Cache (DNLC) is based (by default) on `ninode`. Set `ninode` to 4000 if only /stand is on HFS; set it to 15000 or higher to be safe if you have many filesystems on HFS. Even higher values are useful for dedicated file servers.

`nkthread`
The maximum number of kernel threads allowed on the system. The default sam formula sets this based on `nproc`, which is fine for most workloads. If you know that you have a multi-threaded workload (uses pthreads), then you may want to adjust the formula to bump this even higher: just make sure it always works out to be greater than `nproc`. Keep in mind the potential for cascading formulas: `maxusers` may be used in the formula for `nproc` that may be used in the formula for `nkthread` that may be used in the calculation for `ncallout`, etc. This is not necessarily a bad thing: If you're smart about setting nproc to something reasonable, then these other guys will just follow suit.

`nproc`
This is heavily dependent on your expected workload, but for most systems, 1024 is enough for the maximum number of processes. If you know better, set it higher. Don't blindly overconfigure this by setting it to 30000 when you'll have only 400 processes in your workload, as `nproc` influences various formulas in sam, and also has secondary effects, like increasing the size of the midaemon's shared memory segment (used by Glance to keep track of process data). Process table utilization is tracked in Glance’s

System Tables Report: check the utilization periodically and plan to bump up `nproc` when you see its gets over 80 percent utilization during normal processing.

`npty`
EDA/MDA applications may need this bumped up (number of remote sessions). Like `nfile`, you can leave this alone unless you see the number of Pseudo Terminals nearing the limit in Glance.

`swapmem_on`
Pseudo swap is used to increase the amount of reservable virtual memory. This is useful when you can't configure as much device swap as you need. For example, say you have more physical memory installed than you have disks available to use as swap: in this case, if pseudo swap is not turned on, you'll never be able to allocate all the physical memory you have. Legend had it that if you had plenty of disk swap reservable (way more than physmem), then also enabling pseudo swap could slow performance. Doug spent a good few days trying to confirm this with benchmarks on some test systems and could not find *any* effect of pseudo swap on performance, *unless* your system is trying to reserve more swap than you have device swap available to cover. So: pseudo swap can slow down performance only when it "kicks in". When your total reserved swap space increases beyond the amount *available* for device swap, if you do not have pseudo swap enabled, programs will fail ("out of memory"). If your total swap reservation exceeds available device swap and you *do* have pseudo swap enabled, then programs will *not* fail, *but* the kernel will start locking their pages into physical memory. If this happens, the number for "Used" memory swap shown in glance will go up quickly. We realize this is a real head-spinner. Rule of thumb: if you have enough device swap available to cover the amount you will reserve, then you don't need to worry about how this parameter is set. If you need to set it because you're short on device swap, then beware that it could affect performance. Bottom line is to try and configure enough swap disk to cover your expected workload.

`timeslice`
Leave this set at 10. If this is set to 1, which used to happen because of that old sam configuration template with a bug in it, excessive context switching overhead will usually result. The system would spend, oh, 10 times what it normally does simply handling timeslice interrupts. It can possibly also cause lock contention issues if set too low. We've never seen a production system benefit from having timeslice set less than 10. Forget the "It Depends" on this one: leave it set at 10!

dynamic kernel parameters

Historically, modifying kernel parameters caused a rebuild of the kernel and a reboot of the system. Support for dynamic kernel parameters was introduced in HP-UX 11.11. That is, parameters that you can modify *without* causing a system reboot using the `kmtune(1M)` command. There is talk about parameters "dynamically tuning themselves", but… this appears to be in the future (being neither in past nor present). The parameters that became

dynamic in 11.11 are typically responsible for **5%** of the system reboots historically. These parameters are:

- `core_addshmem_read`
- `core_addshmem_write`
- `maxfiles_lim`
- `maxtsiz`
- `maxtsiz_64bit`
- `maxuprc`
- `msgmax`
- `msgmnb`
- `semmsl`
- `shmmax`
- `shmseg`

The parameters that have become dynamic in HP-UX 11.22, combined with the above, account for approximately **40%** of system reboots. These are:

- `callouts_extra`
- `executable_stack`
- `ksi_alloc_max`
- `maxdsiz`
- `maxdsiz_64bit`
- `maxssiz`
- `maxssiz_64bit`
- `max_acct_file_size`
- `max_thread_proc`
- `nkthread`
- `nproc`
- `scsi_max_qdepth`
- `secure_sid_scripts`
- `shmmni`

What's Yer Problem?

OK, so let's talk about real life now, which begins after you've been thrust into a situation on a critical server where some (or all) the applications are running slow and nobody has any idea what's wrong but you're supposed to fix it. Now…

If you're good, *really good*, then you've been collecting some historical information on the system you manage and you have a decent understanding of how the system looks when it's behaving normally. Some people just leave glance running occasionally to see what resources the system is usually consuming (cpu, memory, disk, network, and kernel tables). For 24x7 logging and alarming, the OpenView Performance Agent (OVPA) is ideal. In addition to local export, you can view the OVPA metrics remotely with the OV Performance Manager or OV Reporter. It's important to understand the baseline, because then when things go awry you can see right off what resource is out of whack (*awry* and *out of whack* being technical terms). If you have been bad, *very bad*, or unlucky, then you have no idea what's normal and you'll need to start from scratch: chase the most likely

bottlenecks that show up in the tools and hope you're on the right track. Start from the global level (system-wide view) and then drill down to get more detail on specific resources that are busy.

It's very helpful to understand the structure of the applications that are running and how they use resources. For example, if you know your system is a dedicated database server and that all the critical databases are on raw logical volumes, then you will not waste your time by trying to tune filesystem options and buffercache efficiency: they would not be relevant when all the disk I/O is in raw mode. If you've taken the time to bucket all the important processes into applications via OVPA's parm file, then you can compare relative application resource usage and (hopefully) jump right to the set of processes involved in the problem. There are typically many active processes on busy servers, so you want to understand enough about the performance problem to know which processes are the ones you need to focus on.

If an application or process is actually failing to run or it is aborting after some amount of time, then you may not have a performance problem; instead the failure probably has something to do with a limit being exceeded. Common problems include underconfigured kernel parameters or swap space. You can usually look these errors up in the HP-UX documentation and it will point you to which kernel tunable to bump up. Glance's System Tables report can be helpful. Also, make sure you've kept the system updated with the most recent patch bundles relevant to performance and the subsystems your workload uses (like networking!). If nothing is actually failing, but things are just running slowly, then the real fun begins!

Resource Bottlenecks

The bottom line on system resources is that you would actually like to see them fully utilized. After all, you paid for them! High utilization is not the same as a bottleneck. A bottleneck is a symptom of a resource that is fully utilized *and* has a queue of processes or threads waiting for it. The processes stuck waiting will run slower than they would if there were no queue on the bottlenecked resource.

Generic Bottleneck Recipe Ingredients:

- A resource is in use, and
- Processes or threads are spending time waiting on that resource.

Starting with the next section, we'll start drilling down into specific bottleneck types. Of course, we'll not be able to categorize every potential bottleneck, but will try to cover the most common ones. At the beginning of each type of bottleneck, we'll start with the few primary indicators we look at to categorize problems ourselves, then drill down into subcategories as needed. You can quickly scan the "ingredients" lists to see which one matches what you have. After all, all great cooks start with the right ingredients!

If you'd like to understand more about what makes a bottleneck, consider the example of a disk backup. A process involved in the backup application will be reading from disk and writing to a backup device (another disk, a tape device, or over the network). This process cannot back up data infinitely fast. It will be limited by some resource. That resource could be the disk that it's backing up (indicated by the source disk being nearly 100 percent busy). That resource could be the output device for the backup. The backup could also be limited by the CPU (perhaps in a compression algorithm, indicated by that process using 100 percent CPU). You could make the backup go faster if you added some speed to the specific resource it is constrained by, but if the backup completes in the timeframe you need it to and it doesn't impact any other processing, then there is no problem and making it run faster is not the best use of your time.

Now, if your backup is not finishing before your server starts to get busy, you may find that applications running concurrently with it are dog-slow. This would be because your applications are contending for the same resource that the backup has in use. Now you have a true performance bottleneck! One of the most common performance problem scenarios is a backup running too long and interfering with daily processing. Often the easiest way to "solve" that problem is to tune which specific files and disks are being backed up, to make sure you balance the need for data integrity with performance.

If you are starting your performance analysis knowing what application and processes are running slower than they should, then look at those specific processes and see what they're waiting on most of the time. This is not always as easy as it sounds, because unix is not typically very good at telling what things are waiting for. Glance and OpenView Performance Agent (OVPA was called MeasureWare) have the concept of Blocked States (also known as Wait Reasons). You can select a process in Glance, and then get into the Wait States screen for it to see what percentage of time it is waiting for different resources. Unfortunately, these don't always point you directly to the source of the problem. Some of them, such as Priority, are easier: if a process is blocked on Priority that means that it was stuck waiting for CPU time as a higher-priority process ran. Some other wait reasons, such as Streams (Streams subsystem I/O) are trickier. If a process is spending most of its time blocked on Streams, then it may be waiting because a network is bottlenecked, but (more likely) it is idle reading from a Stream waiting until something writes to it. User login shells sit in Stream wait when waiting for terminal input.

Metrics

We're focusing on performance, not performance metrics. We'll need to discuss some of the various metrics as we drill down, but we don't want to get into the gory details of the exact metric definitions or how they are derived. If you have Glance on a system, run gpm and click on the Help -> User's Guide menu selection, then in the help window click on the Performance Metrics section to see all the definitions. Alternatively, in gpm use the Configure -> Choose Metrics selection from one of the Report windows to see the list of all available metrics in that area, and use the "?" button to conjure up the metric definitions. If you have OVPA on your system, a place to go for the definitions is

`/opt/perf/paperdocs/mwa/C/methp.txt`. In general, “all” the performance metrics are in gpm and available to the Glance product’s adviser. A subset of the performance metrics are shown in character-mode glance and logged by OVPA. There are other documents focusing on tools and metrics. We'll include some pointers in the References section below.

We'll use the word "process" to mean either a process or a thread. A few apps are multi-threaded, and each thread in HP-UX 11 is a schedulable, runnable entity. Therefore, a single process with 10 threads can fully load 10 processors (each thread using 100 percent cpu, the parent process using "1000 percent" cpu – note process metrics do not take the number of CPUs into account). This is similar to 10 separate single-threaded processes each using 100 percent CPU. For the sake of simplicity, we'll say "processes" instead of "processes or threads" in the following discussions.

CPU Bottlenecks

CPU Bottleneck Recipe Ingredients:

- Consistent high global CPU utilization (`GBL_CPU_TOTAL_UTIL > 90%`), and
- Significant "Run Queue" or Load Average (`GBL_PRI_QUEUE or GBL_RUN_QUEUE > 3`).
- Processes blocked on Priority (`PROC_STOP_REASON = PRI`).

It's easy to tell if you have a CPU bottleneck. The overall CPU utilization (averaged over all processors) will be near 100 percent and some processes are always waiting to run. It is not always easy to find out why the CPU bottleneck is happening. Here’s where it is important to have that baseline knowledge of what the system looks like when it’s running normally, so you’ll have an easier time spotting the processes and applications that are contributing to a problem. Stephen likes to call these the offending process(es). The priority queue metric, derived from process-blocked states and available only in OVPA and Glance, shows the average number of processes waiting for any CPU (that, is, blocked on PRI). It doesn't matter how many processors there are on the system. Stephen likes to use this more than the Run Queue. The Run Queue is an average of how many processes were "runnable" on each processor. This works out to be similar to or the same as the Load Average metric, displayed by the `top` or `uptime` commands. Different perftools use either the running average or the instantaneous value. Prior to HP-UX 11.11, the run queue includes processes waiting for disk I/O (“short waited” processes), which could be confusing.

To diagnose CPU bottlenecks, look first to see whether most of the total CPU time is spent in System (kernel) mode or User (outside kernel) mode. Jump to the subsection below that most closely matches your situation.

System CPU Bottlenecks

System CPU Bottleneck Recipe Ingredients:

- CPU bottleneck symptoms from above, and
- Most of the time spent in the kernel (`GBL_CPU_SYS_MODE_UTIL > 50%`).

If you are spending most of your CPU time in System mode, then you'll want to break that down further and see what activity is causing processes to spend so much time in the kernel. First, check to see if most of the overhead is due to context switching. This is the kernel running different processes all the time. If you're doing a lot of context switching, then you'll want to figure out why, because this is not productive work. This is a whole topic in itself, so jump down to the next section on Context Switching Bottlenecks. Assuming it isn't that, see if `GBL_CPU_INTERRUPT_UTIL` is > 30 percent. If so, you likely have some kind of I/O bottleneck instead of a CPU bottleneck (that is, your CPU bottleneck is being caused by an I/O bottleneck), or just maybe you have a flaky I/O card. Switch gears and address the I/O issue first (Disk or Networking bottleneck). Memory bottlenecks can also come disguised as System CPU bottlenecks: if memory is fully utilized and you see paging, look at the memory issue first.

Assuming at this point that most of your kernel time is spent in system calls (`GBL_CPU_SYSCALL_UTIL >30%`), then it's time to try to see which specific system calls are going on. It's best if you can use Glance on the system at the time the problem is active. If you can do this, count your lucky stars and skip to the next paragraph. If you are stuck with looking at historical data or using other tools, it won't include specific system call breakdowns, so you'll need to try to work from other metrics. Try looking at process data during the bad time and see which processes are the worst (highest `PROC_CPU_SYSCALL_UTIL`) and look at their other metrics or known behavior to see if you can determine the reason why that process would be doing excessive syscalls.

If you can catch the problem live, you can use Glance to drill down further. We like to use gpm for this because of its more flexible sorting and metric selection. Go into Reports->System Info->System Calls, and in this window configure the sort field to be the syscall rate. The most-often called syscall will be listed first. You can also sort by CPU time to see which syscalls are taking the most CPU time, as some syscalls are significantly more expensive than others are. In gpm's Process List report, you can choose the `PROC_CPU_SYS_MODE_UTIL` metric to sort on and the processes spending the most time in the kernel will be listed first. Select a process from the list and pull down the Process System Calls report and (after a few update intervals) you'll see the syscalls that process is using. Keep in mind that not all system calls map directly to libc interfaces, so you may need to be a little kernel-savvy to translate syscall info back into program source code. Once you find out which processes are involved in the bottleneck, and what they are doing, the tricky part is determining why. We leave this as an exercise for the user!

Common programming mistakes such as repetitive `gettimeofday()` or `select()` calls (we've seen thousands per second in some poorly designed programs) may be at the root of a System CPU bottleneck. Another common cause is excessive `stat`-type filesystem

syscalls (the find command is good at generating these, as well as shells with excessive search PATH variables). Once we traced the root cause of a bottleneck back to a program that was opening and closing /dev/null in a loop!

On busy and large multiprocessor systems, system CPU bottlenecks can be the result of contention over internal kernel resources such as data structures that can only be accessed on behalf of one CPU at a time. You may have heard of "spinlocks," which is what happens when processors must sit and spin waiting for a lock to be released on things like virtual memory or I/O control structures. This shows up in the tools as System CPU time, and it's hard to distinguish from other issues. Typically, this is OK because there's not much from the sysadmin perspective that you can do about it anyway. Spinlocks are an efficient way to keep processors from tromping over critical kernel structures, but some workloads (like those doing a lot of file manipulations) tend to have more contention. If programs never make system calls, then they won't be slowed down by the kernel. Unfortunately, this is not always possible!

Here's a plug for a contrib system trace utility put together by a very good friend of ours at HP. Its called `tusc`, and it's very useful for tracing activity and system calls made by specific processes: very useful for application developers. It's currently available via the HP Networking Contrib Archive (see References section at the end of this paper) under the `tools` directory. We would be remiss if we did not say that some applications have been written that perform an enormous amount of system calls and there is not much that we can do about it, especially if the application is a "third-party" application. We have also seen developers "choose" the wrong calls for performance. It's a complex topic that Stephen is prepared to go into at length over a beer.

Context Switching Bottlenecks

Context Switching System CPU Bottleneck Recipe Ingredients:

- System CPU bottleneck symptoms from above, and
- Lots of CPU time spent Switching (`GBL_CPU_CSWITCH_UTIL > 30%`).

A context switch can occur for one of two reasons: either the currently executing process puts itself to sleep (by making a library or system call that waits), or the currently executing process is forced off the CPU because the OS has determined that it needs to schedule a different (higher priority) process. When a system spends a lot of time context switching (which is essentially overhead), useful processing can be bogged down. One common cause of extreme context switching is workloads that have a very high fork rate. In other words, processes are being created (and presumably completed) very often. Frequent logins are a great source of high fork rates, as shell login profiles often run many short-lived processes. Keeping user shell rc files clean can avoid a lot of this overhead. Since faster systems can handle faster fork rates, it's hard to set a rule of thumb, but you can monitor `GBL_STARTED_PROC` over time and watch for values in the thousands and spikes. Trying to track down who's forking too much is easy with gpm; just use Choose

Metrics to get PROC_FORK into the Process List report, and sort on it. Another good sort column for this type of problem is PROC_CPU_CSWITCH_UTIL.

If you don't have a high process creation rate, then high context switch rates are probably an issue with the application. Semaphore contention is a common cause of context switches, as processes repeatedly block on semaphore waits. There's typically very little you can do to change the behavior of the application itself, but there may be some external controls that you can change to make it more efficient. Often by lengthening the amount of time each process can hold a CPU, you can decrease scheduler thrashing. Make sure the kernel timeslice parameter is at least at the default of 10 (10 10-millisecond clock ticks is .1 second), and consider doubling it.

User CPU Bottlenecks

User CPU Bottleneck Recipe Ingredients:

- CPU bottleneck symptoms from above, and
- Most of the time spent in user code (GBL_CPU_USER_MODE_UTIL > 50%).

If your system is spending most of its time executing outside the kernel, then that's typically a good thing. You just may want to make sure you are executing the "right" user code. Look at the processes using most of the cpu (sort the Glance process list by PROC_CPU_TOTAL_UTIL) and see if the processes getting most of the time are the ones you'd want to get most of the time. In Glance, you can select a process and drill down to see more detailed information. If a process is spending all of its time in user mode, making no system calls (thus no I/O), it might be stuck in a loop. If shell processes (sh, ksh, csh...) are hogging the CPU, check the user to make sure they aren't stuck (sometimes network disconnects can lead to stale shells stuck in loops).

If the wrong applications are getting all the CPU time at the expense of the applications you want, this will be shown as important processes being blocked on Priority a lot. There are several tools that you can use to adjust process priorities. The HP PRM product (Process Resource Manager) is worth checking into to provide CPU control per application. Its companion product HP WorkLoad Manager provides for automation of PRM controls. A more short-term remedy may be judicious use of the renice command, which you can also invoke via Glance on a selected process. Increasing the nice value will decrease its processing priority relative to other timeshare processes.

The easiest way to solve a CPU bottleneck may simply be to buy more processing power. In general, *more better faster* CPUs will make things run *more better faster*. Another approach is application optimization, and various programming tools can be useful if you have source code access to your applications. The HP Developer and Solution Partner portal mentioned in the References section below can be a good place to search for tools.

Disk Bottlenecks

Disk Bottleneck Recipe Ingredients:

- Consistent high utilization on at least one disk device (`GBL_DISK_UTIL_PEAK or highest BYDSK_UTIL > 50%`).
- Significant queuing lengths (`GBL_DISK_SUBSYSTEM_QUEUE > 3 or any BYDSK_REQUEST_QUEUE > 1`).
- Processes or threads blocked on I/O wait reasons (`PROC_STOP_REASON = CACHE, DISK, IO`).

Disk bottlenecks are easy to solve: Just recode all your programs to keep all their data locked in memory all the time! Hey, memory is cheap! Sadly, this isn't always (say ever) possible, so the next bestest alternative is to focus your disk tuning efforts on the I/O hotspots. The perfect scenario for disk I/O is to spread the applications' I/O activity out over as many different I/O cards, LUNs, and physical spindles as possible to maximize overall throughput and avoid bottlenecks on any particular I/O path. Sadly, this isn't always possible either, because of the constraints of the application, downtime for reconfigurations, etc.

To find the hotspots, use a performance tool that shows utilization on the different disk devices. Both `sar` and `iostat` have by-disk information, as of course do Glance and OVPA. We usually start by looking at historical data and focus on the disks that are most heavily utilized at the specific times when there is a perceived problem with performance. Using PerfView, you can draw a Class Compare graph of all disks using the `BYDSK_UTIL` metric to see utilization trends, and use the `BYDSK_REQUEST_QUEUE` to look for queuing. If you're not looking at the data from times when a problem occurs, you may be tuning the wrong things! If a disk is busy over 50 percent of the time, and there's a queue on the disk, then there's an opportunity to tune. Note that OVPA's metric `GBL_DISK_UTIL_PEAK` is not an average, nor does it track just one disk over time. This metric is showing you the utilization of the busiest disk of all the disks for a given interval, and of course a different disk could be the busiest disk every interval. The other useful global metric for disk bottlenecks is the `GBL_DISK_SUBSYSTEM_QUEUE`, which shows you the average number of processes blocked on wait reasons related to Disk I/O, similar to how `GBL_PRI_QUEUE` works for CPU.

If your busiest disk is a swap device, then you have a memory bottleneck masquerading as a disk bottleneck and you should address the memory issues first if possible. Also, see the discussion above under System (Disk) Setup for optimizing swap device configurations for performance.

Glance can be particularly useful if you can run it while a disk bottleneck is in progress, because there are separate reports from the perspective of By-Filesystem, By-Logical Volume, and By-Disk. You can also see the logical (read/write syscall) I/O versus physical I/O breakdown as well as physical I/O split by type (Filesystem, Raw, Virtual Memory (paging), and System (inode activity)). In Glance, you can sort the process list on `PROC_DISK_PHYS_IO_RATE`, then select the processes doing most of the I/O and bring

up their list of open file descriptors, which may help pinpoint the specific files that are involved. The problem with all the system perftools is that the internals of the disk hardware are opaque to them. You can have disk arrays that show up as a single "disk" in the perftool, and specialized tools may be needed to analyze the internals of the array. The disk array vendor is where you'd go for these tools.

Some general tips for improving disk I/O throughput include:

- Spread your disk I/O out as much as possible. It is better to keep 10 disks 10 percent busy than one disk 100 percent busy. Try to spread busy filesystems (and/or logical volumes) out across multiple physical disks.
- Avoid excessive logging. Different applications may have configuration controls that you can manipulate. For VxFS, managing the intent log is important. The `vxtunefs` command may be useful. For suggested VxFS mount options, see the System Setup section above.
- If you're careful, you can try adjusting the scsi disk driver's maximum queue depth for particular disks of importance using `scsictl`. If you have guidelines on this specific to the disk you are working with, try them. Generally increasing the maximum queue depth will increase parellelism at the possible expense of overloading the hardware: if you get QUEUE FULL errors then performance is suffering and you should set the max queue depth down.

In most cases, a very few processes will be responsible for most of the I/O overhead on a system. Watch for I/O “abuse”: applications that create huge numbers of files or ones that do large numbers of opens/closes of scratch files. You can tell if this is a problem if you see a lot of “System”-type I/O on a busy disk (`BYDSK_SYSTEM_IO_RATE`). Back in the Good Old Days, you could look for a low hit rate on the Dynamic Name Lookup Cache (`GBL_MEM_DNLC_HIT`, at the end of Glance’s Disk Report), but the DNLC instrumentation broke in the 11.11 kernel and hasn't been fixed. To track things down, you can look for processes doing lots of I/O and spending significant amounts of time in System CPU. If you catch them live, drill down into Glance’s Process System Calls report to see what calls they’re making. Unfortunately, unless you own the source code to the application (or the owner owes you a big favor), there is little you can do to correct inefficient I/O programming.

Buffer Cache Bottlenecks

Bufcache Bottleneck Recipe Ingredients:

- Moderate utilization on at least one disk device (`GBL_DISK_UTIL_PEAK or highest BYDSK_UTIL > 25`), and
- Consistently low Bufcache read hit percentage (`GBL_MEM_CACHE_HIT_PCT < 90%`).
- Processes or threads blocked on Cache (`PROC_STOP_REASON = CACHE`).

If you're seeing these symptoms, then you may want to bump up the filesystem buffer cache size, especially if you're on 11.11 or managing an NFS, ftp, Web, or other file server where you'd want to buffer a lot of file pages in memory — so long as you don't start paging out because of memory pressure! In practice, we more often find that buffer cache is overconfigured rather than underconfigured.

In 11.11, changes were made to the buffer cache algorithms that addressed some of the worst inefficiencies for large configurations that could plague 11.0, but there are still cases where having an enormous bufcache does more harm than good. For example, a large buffer cache may cause a synchronous activity (such as a read) to queue up behind a lot of asynchronous activity (like writes). A smaller bufcache can improve consistency of response time by smoothing out write activity. In addition, the 11.11 improvements have not solved all algorithmic scaling or super-page pool fragmentation issues. Some filesystem I/O-intensive workloads can benefit from a larger bufcache, but in all cases you want to avoid pageouts! Also, if you manage a database server with primary I/O paths going to raw devices, then the filesystem buffer cache just gets in the way.

You can use the dynamic buffer cache (`dbc_min_pct` and `dbc_max_pct`) instead of configuring a fixed size bufcache (via `nbuf` or `bufpages`), but generally its simpler just to set `bufpages` to a value that works for you. For 11.0 systems that aren't dedicated file servers, if you have a “gig” of memory or more, we recommend you set `bufpages` to 102400 (this times 4K per page equals 400MB). For 11.11 systems, and file servers, with over a gig of memory, set bufpages to 204800 (800MB). Folks with 8GB of bufcache configured today might consider our rules of thumb to be a "9.5 on their sphincter scale", but huge bufcaches lead to additional overhead just managing them, as mentioned above, and are likely to do more harm than good, especially if they contribute to memory pressure.

If you want to be more anal about it, try watching your buffer cache read hit rate over time (`GBL_MEM_CACHE_HIT_PCT`), making sure you watch it when the system is busy. In Glance, this metric appears towards the end of the Disk Report screen. The cache hit rate metrics aren’t very accurate in any tool, because the underlying instrumentation is “screwed up” (another technical term), but they are better than nothing. The hit rate behavior is very dependent on your workload: it should go without saying that if the throughput and load on a system is very low, then the hit rate doesn't matter (if performance is OK then you should find something better to stare at than performance metrics). Also, keep in mind that the hit rate doesn't measure anything that isn't going through the buffer cache. If you are using raw disk device access or mounting vxfs filesystems with mincache=direct (see the Setup section above under Filesystems – VxFS), then I/O through those paths will be neither a bufcache hit nor a miss because it isn't using the buffer cache! Even for filesystem access going through the buffer cache, if the processing is very random, then the size of buffer cache will have *no* bearing on the read hit rate (the random access would likely span a larger data set than any cache could hold in its entirety). In response to the question, “But how do I increase my read hit rate?” Stephen always replies, “Read sequentially. ☺“

Having said all that, bufcache read hit rates consistently over 90 percent probably indicates the buffer cache is big enough and maybe too big: if you usually see the hit rate over 90 percent, *and* you typically run with memory utilization over 90%, *and* your bufcache size (`TBL_BUFFER_CACHE_USED`, found in Glance in the System Tables Report) is bigger than 400MB, *then* reconfigure the bufcache size to be the larger of either half its current size or 400MB. After the reconfiguration, go back and watch the hit rate some more. Lather, Rinse, Repeat. On the other hand, if you often see a read cache hit rate under 90 percent, *and* you always run with memory utilization under 90 percent, *then* think about bumping the bufcache size up.

As mentioned above, there *are* corner cases for justifying large buffer caches. What we want to explain is the *simple* story on buffer cache dynamics: Think about this: you have a buffer cache that is sized 2GB. When you observe the read cache hit rate, you see that it is 97 percent. If you trust that particular metric, (that we *told* you was flakey), this means that 97 percent of the time, processes are "finding in cache" exactly what they are looking for. You then resize buffer cache to 400MB, and when you check the read cache hit rate, it is 97 percent. You can see that at 400MB, processes are finding the exact same stuff they were finding when the cache was much larger. Use that extra memory for something that may matter.

Memory Bottlenecks

Memory Bottleneck Recipe Ingredients:

- High physical memory utilization (`GBL_MEM_UTIL > 95%`), and
- Significant pageout rate (`GBL_MEM_PAGEOUT_RATE > 1`), or
- Any deactivations (`GBL_MEM_SWAPOUT_RATE > 0`), or
- Vhand process consistently active (`vhand's PROC_CPU_TOTAL_UTIL > 5%`).
- Processes or threads blocked on virtual memory (`GBL_MEM_QUEUE > 0 or PROC_STOP_REASON = VM`).

It is a good thing to remember not to forget about your memory.

When a program touches a virtual address on a page that is not in physical memory, the result will be a "page in." When the HP-UX needs to make room in physical memory, or when a memory-mapped file is posted, the result will be a "page out." What used to be called swaps, where whole working sets were transferred from memory to a swap area, has now been replaced by deactivations, where pages belonging to a selected (unfortunate) process are all marked to be paged out. The offending process is taken off the run queue and put on a deactivation queue, so it gets no CPU time and cannot reference any of its pages: thus they are often quickly paged out. This does not mean they are necessarily paged out, though! We could go into a lot of detail on this subject, but we'll spare you.

Here's what you need to know: Ignore pageins. They just happen. When memory utilization is high, then watch out for pageouts, as they are often (but not always!) a memory bottleneck indicator. Don't worry about pageouts that happen when memory utilization is not high (this can be due to memory-mapped file writes). If memory utilization is high and you see any deactivations, then you really have a problem. If memory utilization is less than 90 percent, then don't worry be happy.

OK, so let's say we got you worried. Maybe you're seeing high memory utilization and a few pageouts. Maybe it gets worse over time until the system is rebooted (this is classic: "we reboot once a week just because"). One common cause of a memory bottleneck is an overly large filesystem buffer cache. If your buffer cache size is over 400MB, then think about shrinking it (see previous section about bufcache). Another common cause of memory bottlenecks is a memory "leak" in an application. Memory leaks happen when processes allocate (virtual) memory and forget to release it.

If you have done a good job organizing your OVPA parm file applications, then comparing their virtual memory trends (`APP_MEM_VIRT`) over time can be very helpful to see if any applications have memory leaks. Using PerfView, you can draw a Class Compare graph of all Applications using the `APP_MEM_VIRT` metric to see this graphically. If you don't have applications organized well, you can use Glance and sort on `PROC_MEM_VIRT` to see the processes using most memory. In Glance, select a process with a large virtual set size and drill into the Process Memory Regions report to see great information about each region the process has allocated. Memory leaks are usually characterized by the DATA region growing slowly over time (globally you'll also see `GBL_SWAP_SPACE_UTIL` on the increase). Restarting the app or rebooting are workarounds, of course, but correcting the offending program is a better solution.

If you don't have any memory leaks, your buffer cache is reasonably sized, and you still have memory pressure, then the only solution may be to buy more memory. Most database servers allocate huge shared memory segments, and you'll want to make sure you have enough physical memory to keep them from paging. Be careful about programs getting "out of memory" errors, though, because those are usually related to not having enough swap space reservable or hitting a configuration limit. Relevant kernel parameters are `dbc_min_pct, dbc_max_pct, bufpages, nbuf, maxswapchunks, swapmem_on, maxtsiz, maxssiz,` and especially `maxdsiz` (see System Setup Kernel Tunables section above).

You can also get into some fancy areas for getting around some issues with memory. Some 32bit applications using lots of shared memory benefit from configuring memory windows (usually needed for running multiple instances of applications like 32bit Informix and SAP). Large page sizes is a technique that can be useful for some apps that have very large working sets and good data locality, to avoid TLB thrashing. These topics are a little too deep for this dissertation and are of limited applicability. Only use them if your application supplier recommends it.

Swap

It's very important to realize that there are two separate issues with regards to swap configuration. You always need to have at least as much “reservable” swap as your applications will ever request. This is essentially the system’s limit on virtual memory (for stack, heap, data, and all kinds of shared memory). The amount of swap actually in use is a completely separate issue: the system typically reserves much more swap than is ever in use. Swap only gets used when pageouts occur; it is reserved whenever virtual memory (other than for program text) is allocated.

As mentioned above in the Disk Setup section, you should have at least two fixed device swap partitions allocated on your system for fast paging when you do have paging activity. Make sure they are the same size, on different physical disks, and at the same swap priority, which should be a number less than that of any other swap areas (lower numbers are higher priority). If possible, place the disks on different cards/controllers: Stephen calls this “making sure that the card is not the bottleneck.” Monitor using Glance's Swap Space report or swapinfo to make sure the system keeps most or all of the “used” swap on these devices (or in memory). Once you do that, you can take care of having enough “reservable” swap by several methods (watch `GBL_SWAP_SPACE_UTIL`). Since unused reserved swap never actually has any I/Os done to it, you can bump up the limit of virtual memory by enabling lower-priority swap areas on slow "spare" disks. We used to recommend that you not turn on pseudo swap when you have plenty of disk swap space configured, but its effects on performance turns out to be negligible. You *need* to turn pseudo swap on if you have less disk swap space configured than you have physical memory installed! We recommend against enabling filesystem swap areas, but you can do this as long as you’re sure they don’t get used (set their swap priority to a higher number than all other areas).

Networking Bottlenecks

Networking Bottleneck Recipe Ingredients:

- High (dependent on configuration) network packet or byte rates `(GBL_NET_PACKET_RATE or specific BYNETIF_IN_BYTE_RATE or BYNETIF_OUT_BYTE_RATE > 2*average).`
- Any Output Queuing `(GBL_NET_OUTQUEUE > 0).`
- Higher than normal number of processes or threads blocked networking `(PROC_STOP_REASON = NFS, LAN, RPC, Socket (if not idle), or GBL_NETWORK_SUBSYSTEM_QUEUE > average).`
- One CPU with a high System mode or Interrupt CPU utilization while other CPUs are mostly idle `(BYCPU_CPU_INTERRUPT_UTIL > 30).`
- From lanadmin, frequent incrementing of "Outbound Discards" or "Excessive Collisions".

Networking bottlenecks can be very tricky to analyze. The system-level performance tools do not provide enough information to drill down very much. Glance and OVPA

have metrics for packet, collision, and error rates by interface. Current revisions of the perftools include additional networking metrics such as per-interface byte rates. Collisions in general aren't a good performance indicator. They "just happen" on active networks, but sometimes they can indicate a duplex mismatch or a network out of spec. Excessive collisions are one type of collision that *does* indicate a network bottleneck.

At the global level, look for times when packet rates are higher than normal, and see if those times also have any output queue length. If so, see if there is a repeated pattern and focus on the workload during those times. You may also be able to see network bottlenecks by watching for higher than normal values for networking wait states in processes (which is used to derive OVPA's network subsystem queue metric). The netstat and lanadmin commands give you more detailed information, but they can be tricky to understand. The `ndd(1M)` command can display and change networking-specific parameters. You can dig up more information about ndd and net tuning in general from the `briefs` directory in the HP Networking tools contrib archive (see References). Tools like OpenView Network Node Manager are specifically designed to monitor the network from a non-system-centric point of view.

One thing that has caused problems for several customers is a mismatch regarding duplex auto negotiation between your system's LAN card (NIC) and its switch. Both sides should be either half-duplex or full-duplex. You should check both to make sure they match. You can use `lanscan` to get the card interface number and then: `lanadmin –x CrdIn#` to show its setting (on newer interfaces). Stephen says it’s better just to turn auto negotiation off and manage the configuration "by hand" (though you'll need to go back and recheck if the cables get switched around). High collision rates (which are misleading as they are actually errors) have been seen on systems with mismatches in either duplex or speed settings, and improve (along with performance) when the configuration is corrected.

On some systems, you may see a high Interrupt CPU percentage on a single CPU. This can be caused by interrupts from the NIC saturating the CPU assigned to that NIC. If you see this symptom, you might consider adding a second NIC and possibly trunking it with Auto Port Aggregation to preserve the single IP address.

Here are some things to try if you suspect a network bottleneck: Run the command `netstat –s` twice, spaced 30 seconds apart. Look at the change in tcp sent data packet retransmissions, and any udp socket overflows or ip fragments dropped after timeout (reassembly timeouts). You can grab a copy of the utility `beforeafter` from the HP Networking contrib archive (see References section) under the `tools` directory: it will help parse the output.

For `lanadmin`, you can watch for inbound and outbound discard and error counts and excessive collisions. Be careful, because on fast networks like gigabit ethernet, the 32-bit counters shown by these tools roll over frequently.

If you use NFS a lot, see the configuration tips above in the System Setup section. The `nfsstat` command and Glance's NFS Reports can be helpful in monitoring traffic, especially on the server. If the NFS By System report on the server shows one client causing lots of activity, run Glance on that client and see which processes may be causing it. We've seen users on clients doing repeated (and unnecessary) `find` commands across NFS mounts, which can drag a server down. On the client side, use nfsstat to watch for retransmits and timeouts that can indicate a network or server problem. See our NFS performance reference (References section below).

Other Bottlenecks

Other Bottleneck Recipe Ingredients:

- No obvious major resource bottleneck.
- Processes or threads active, but spending significant time blocked on other resources (`PROC_CPU_TOTAL_UTIL > 0 and PROC_STOP_REASON = IPC, MSG, SEM, PIPE, GRAPH`).

If you dropped down through the cookbook to this last entry (meaning we didn't peg the "easy" bottlenecks), now you really have an interesting situation. Performance is a mess but there's no obvious bottleneck. Your best recourse at this point is to try to focus on the problem from the symptom side. Chances are, performance isn't always bad. At what specific times is it bad? Make a record, then go back and look at your historical performance data or compare glance screens from times when performance tanks versus times when it zips (more technical terms). Do any of the global metrics look significantly different? Pay particular attention to process blocked states (what are active processes blocking on besides Priority?). Semaphore and other Interprocess Communication subsystems often have internal bottlenecks. In OVPA, look for higher than normal values for `GBL_IPC_SUBSYSTEM_QUEUE`.

Once you find out when the problems occur, work on which processes are the focus of the problem. Are all applications equally affected? If the problem is restricted to one application, what are its processes most often waiting on? Does the problem occur only when some other application is active (there could be an interaction issue)? You can drill down in Glance into the process wait states and system calls to see what its doing. In OVPA, be wary of the `PROC_*_WAIT_PCT` metrics as they actually reflect the percentage of time over the life of the process, not during the interval they are logged. You may need some application-specific help at this point to do anything useful. One trial and error method is to move some applications (or users) off the system to see if you can reduce the contention even if you haven't nailed it. Alternatively, you can call Stephen and ask for a consulting engagement!

If you've done your work and tuned the system as best you can, you might wonder, "At what point can I just blame bad performance on the application itself?" Feel free to do this at any time, especially if it makes you feel good.

Conclusion

Conclusion? *Conclusion?* There is no conclusion! The performance saga never ends. But seriously folks, we believe that one of the biggest problems in the world of performance and tuning is misdiagnosis. Be as sure as you can be about what you think your "problem" is; for example, don't let a memory (paging and deactivation) issue fool you into thinking you have a CPU or disk bottleneck. Remember to change ***one*** thing at a time or you'll never know what it was that made performance "better." You might even have attained ***more*** *better* performance if you had not modified one of those 12 things.

The most important thing to keep in mind is: Performance tuning is a Science... ***yeah, right!?!*** We think **NOT!** Performance tuning is more like a mixture of art, witchcraft, voodoo, a little smoke (and mirrors), and a dash of luck. May yours be the good kind.

We would like to acknowledge and thank all the folks inside and outside HP who have contributed to this paper's content "behind the scenes." We don't just make this stuff up, you know: we rely on much *smarter* people to make it up! In particular, we'd like to thank Rick Jones, Dan Dickerman, Dave Olker, Chris Bertin, Martin Skagen and other "perf gurus" for their help and for sharing their wisdom with us. For this edition, Chris White deserves recognition for inciting Doug to spend two days (and two nights, and part of a weekend) characterizing the precise effect of swapmem_on.

References

HP Developer & Solution Partner portal:
http://h21007.www2.hp.com/dspp/tech/tech_TechTypeListingPage_IDX/1,1704,10313,00.html

NFS Performance presentation:
http://h21007.www2.hp.com/dspp/ddl/ddl_Download_File_TRX/1,1249,288,00.pdf

Doug's Making Your GlancePlus Pak Perform paper:
http://h21007.www2.hp.com/dspp/ddl/ddl_Download_File_TRX/1,1249,286,00.pdf

HP Documentation Archives:
http://docs.hp.com
http://ovweb.external.hp.com/lpe/doc_serv/

Tunable kernel parameters reference:
http://www.docs.hp.com/cgi-bin/onlinedocs.py?mpn=TKP-90202&service=hpux&path=../TKP-90202/00/00/1&title=Tunable%20Kernel%20Parameters

Veritas filesystem admin guide:
http://docs.hp.com/hpux/onlinedocs/B3929-90011/B3929-90011.html

HP OpenView performance products:
http://www.openview.hp.com/solutions/categories/performancemgmt/index.asp

HP Networking tools contrib archive:
ftp://ftp.cup.hp.com/dist/networking/

IT Resource Center OpenView Performance Management:
http://forums.itrc.hp.com/cm/CategoryHome/1,11863,166,00.html