HP-UX Performance Cookbook

By Stephen Ciullo, HP Senior Technical Consultant
and
Doug Grumann, HP System Performance tools lead engineer
revision 10JUN08

Have you ever run across a document that sounded really interesting and useful, but after a short while you found out it was several years old and horribly outdated? Well, if you are reading this revision of the Performance Cookbook in 2015, then go no further. By 2015 this paper will be obsolete because all systems will tune themselves using tachyon beams anyways. If, however, it's more like 2008 or 2009, then you are in luck: you *have* stumbled across an old document, *but* we have managed to update it and keep it (relatively) current! Since the content of this cookbook is based on experience, we may seem a little slow moving to the latest breaking technology, but this revision does include more data on HP-UX 11.31. Not to worry, we still cover the earlier HP-UX revisions that are very common in the field. As with previous releases of the cookbook, note that:

- We're not diving down to nitty gritty detail on any one topic of performance. Entire books have been written on topics such as SAP, Java and Oracle performance. This cookbook is an overview, based on common problems we see our customers hitting across a broad array of environments.

- We continue to take great liberties with the English language. To those of you who know English as a second language, we can only apologize in advance, and give you permission to skip over the parts where Stephen's New Jersey accent gets too thick.

- If you are looking for a professional, inoffensive, reverent, sanitized, Corporate-approved and politically correct document, then read no further. Instead, contact your official HP Support Representative to submit an Enhancement Request. They will send you to a web page. The web page may be down. Opinions expressed herein are the authors', and are not official positions of Hewlett-Packard, its subsidiaries, acquisitions, or distant cousins.

- Our target audience is system administrators who are somewhat familiar with the HP performance tools. We reference metrics often from Glance and what-used-to-be-known-as-OpenView Performance Agent, though some of these metrics are also available in other tools.

This revision's focus is on HP-UX 11.23 and 11.31, both PA-RISC and Itanium (also called IA64, IPF, Integrity, whatever). At this point, you should move your servers off 11.11 if you can. The 11.2x bits have been out for years now, and 11.31 also for a while! They're stable! Well, at least 11.23 is stable…11.31 is getting there. As HP employees,

we're supposed to call 11.23 by its official name "11i version2," and 11.31 by "11i version3" but we REFUSE. If you have read previous generations of this cookbook, and you want to see what's new, then skip ahead to specific mentions of 11.31, UFC, and NUMA/Cell discussions below.

Here are the tried and true general rules of thumb:

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

- You have to be willing to do the work to know what you're doing. In other words, you can't expect to make your systems tick any better if you don't know what makes them tick. So... 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. HP-Education has a good set of classes on HP-UX, and there are several books (such as Chris Cooper's "HP-UX Internals"), as well as numerous papers on HP-UX and performance-related topics.

- When you go to make changes, *try* to change just 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 2.0 stuff). If anything in this cookbook doesn't work for you, then *please let us know* — but don't sue us!

- A performance guru learns to chant the magic words: "IT DEPENDS." While this can be used as a handy excuse for any behavior or result, it is true that every system is different. A configuration that might work great on one system may not work great on another. You know your systems better than we do, so keep that in mind as you proceed.

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 bottleneck ingredients. If your system doesn't have those ingredients (symptoms), then skip that subsection. If your situation doesn't match *any* of our bottleneck recipes, then you can tell your boss that you have nothing to do, and you're officially H.P.U.U.

(Highly Paid and Under-Utilized). This designation may qualify you for certain special programs through your employer!

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 existing 24x7 production servers don't have. In preparing for a new system, we are confident that 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 going to be CPU-bottlenecked on a given system, then buying 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 bottlenecks on 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.

Itanium processors

Integrity servers run programs compiled for Itanium better than programs compiled for PA-RISC (this is not rocket science). It is fine for an application to run under PA emulation as long as it ain't performance-critical. When performance of the app is very important, especially if its working set is large and it is CPU-intensive, then you should try to get an Itanium (native) version. Perhaps surprisingly, we assert that there is *no* difference for performance whether a program uses 64bit address space or 32bit address space on Itanium. Therefore people clamoring for 64bit versions of this or that application are misguided: only programs accessing terabytes of data (like Oracle) take advantage of 64bit addressing. You get the same performance boost compiling for Itanium in native 32bit mode! Therefore the key thing for Itanium performance is to go native, not to go 64bit.

Most multi-core and hyperthreading experience comes from the x86 world, and we are still waiting to see how these chip technologies translate to HP-UX experience over time,

but generally Doug categorizes these features as “ways to pretend you have more CPUs than you really got”. A cynical person might say “thanks for giving me twice as many CPUs running half as fast”. If cost were not a concern, then performance would always be better on eight independent single-core non-hyperthreaded CPUs than on four dual-core CPUs, or four single-core hyperthreaded CPUs, or whatever other combinations that lead to eight logical processing engines. What’s really happening with multi-core systems and hyperthreading are that you are saving hardware costs by making a single chip behave like multiple logical processors. Sometimes this works (when, for example, an application suffers a lot of ‘stalling’ that another app running on a hyperthread or dual core could take advantage of), and sometimes it doesn’t work (when, for example, applications sharing a chip contend for its cache). The problem is that there’s little instrumentation at that low level to tell you what is happening, so you either need to trust benchmarks or experiment yourself. The authors would be interested in hearing your findings. We like to learn too!

OS versions

As of the time of writing this current edition (mid 2008), you will set up with the latest patch bundle of 11.23 (aka 11iv2) or 11.31 (11iv3). We have gotten more exposure and experience with 11.31 systems since the last revision of the cookbook, and we encourage you to try it (with caution, *and* with the latest patches!). The file system buffer cache is replaced by a Unified File Cache (UFC) in 11.31, which is more efficient. Down towards the end of this paper we’ve added a special section dedicated to the UFC. Also, 11.31 made significant performance improvements especially for the type of app that does a lot of I/O (mass storage stack improvements). Generally, 11.31 can do more I/Os per second and take less CPU time to do them than 11.23. The 11.31 kernel now has per-thread locks (which used to be per-process). There are also new kernel architected synchronizations for spinlocks, semaphores and mutexes that should make things generally more zippy. Last year, an official announcement came out from HP that said “HP-UX 11i v3 delivers on average 30% more performance than HP-UX 11i v2 on the same hardware, depending on the application,..”. We have been assured that these results were from real customer applications and not just benchmarks, which is great. What ***we*** can say with confidence is: “your mileage may vary.”

We know some of you are ‘stuck’ on earlier revs because your app has not certified yet on the latest OS. We’re sorry. The 11.23, especially as it has evolved over the past few years, is very solid. Now, 11.31 contains more performance-oriented and scalability enhancements. See what you can do to get your apps rolled forward, to take advantage of the potential better performance from the OS!

Memory Setup

We always say “memory is cheap so buy lots” (yes this is 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 the file system buffer cache (file page cache in 11.31).

If you're going to be hosting a database, or something else that benefits from a large in-memory cache, then it is even more essential to have ample 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 their applications that there is more memory on your system than there really is. This trick is called Virtual Memory, and it essentially includes 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 will be 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. When your program gets an "out of memory" error, it typically means you are out of reservable swap space (Virtual memory), not out of physical (Resident) memory.

With superdomes (and now the "r'fill-in-the-blank' cell-based" systems), you have the added complexity of Cell Local Memory and related stuff. Our general recommendation: do not muck with it yourself unless you have an application specifically tuned to it. Tuning it well is complex. We have learned that Oracle 10gR2 specifically has enhancements that take advantage of CLM. But generally, CLM is not what we would call the 'practical stuff' of system performance (the bread and butter of simple performance management that addresses 95% of issues with 5% of the complexity). CLM and reconfiguring interrupts to specific processors and other topics that we avoid generally fall into what we call 'internals stuff'. We're not saying it's bad to learn about them if it applies to your situation, just don't go overboard. At the end of this paper, we have a new section specific to Cell-based (NUMA) performance, which discusses briefly Oracle and multiple SGAs and PSETS and stuff, BUT…it ain't gonna be in 'kernelese' – it will be more 'Stephenism'! And we do not go into serious detail…just enough to keep you informed and hopefully help you decide if you want to do detailed research on your own to use these things for specific, performance related issues!

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, many 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. Of course, big storage arrays can be virtualized and have their own management systems nearly independent from the server side of things. Managing fancy storage networks is an art unto itself, and something we do not touch on in this cookbook.

An old UNIX tip: when determining directory paths for applications, try to keep the number of levels from the file system 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 want to configure enough swap space to cover the largest virtual memory demand your system is likely to hit (at least as much as the size of physical 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 (in other words, you want to 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 volumes of the same type hanging off four separate I/O cards, then you're perfect. If you only have differing volumes of different sizes available for swap, take at least two that are of the same type and size that map to different physical disks, and make them the highest priority (lowest number…**0**). Note that primary swap is set to priority 1 and cannot be changed, which is why you need to use 0. This enables page interleaving, meaning that paging requests will 'round robin' to them. You don't want to page out to swap at all, 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.

We don't care if you enable pseudo swap (which you must do if you don't have enough spare disk space reservable for swap). If you get into a situation where your workloads' swap reservation exceeds the total amount of disk swap available, this leads to memory-locking pages as pseudo swap becomes more 'used'. If you have plenty of device swap configured, then enabling pseudo swap provides no benefit 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.

Logical Volumes

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

There's a lot of LVM functionality built in 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 file system 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 pre-fetch 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 cookbook.

File systems - VxFS

If you are using file systems (not raw disk access), then use VxFS (JFS) with 8 kilobyte block size. We ***KNOW*** we said we would not talk about things like Oracle, BUT…'corner cases' (exceptions) would be like, oh --- redo and archive file systems. Make 'em 1K block size. Also, these guys should be DIRECT I/O. See Mark Ray's view on this topic in the paper on JFS Tuning and Common Misconfigured HP-UX Resources (updated for 11.31) linked via our References section below.

For best performance, get the HP OnlineJFS. 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 OnlineJFS. AND: some of the options (in more current VxFS versions) can be modified *dynamically* while the file system is mounted…read the man page☺.

In general, for VxFS file systems use these mount options:

```
delaylog, nodatainlog
```

For VxFS file systems with primarily ***random*** access, like your typical Oracle app, 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 (that's bad), because very sequential reading would 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 access patterns are sequential). BUT WAIT: we have seen an improvement in performance with direct I/O (it happened to be a backup) when the process was requesting a large amount of data. The short version: the largest physical I/O that JFS will do is 64K. If a process was consistently reading/requesting 1MB… JFS would break it up into multiple 64K physical reads. In this specific case, using `mincache=direct` caused much fewer physical I/Os… it just went out and got a 1MB chunk of data at a time.

Let's talk about datainlog and nodatainlog a little more. If you take a look at the HP VERITAS File System Administrator's Guide in the Performance and Tuning section under the discussion of `nodatainlog`, you will see a statement that reads "A `nodatainlog` mode file system should be approximately 50 percent slower than a standard mode VxFS file system for synchronous writes. Other operations are not affected". We completely disagree with this statement (by now you should know that we really check these things out…many different ways). When you use `datainlog` it kinda sorta simulates synchronous writes. It allows smallish (8K or less) writes to be written in the intent log. The data and the inode are written asynchronously later. You only use the intent log in case there is a system crash. Using `datainlog` will actually cause more I/O. Large synchronous I/O is not affected. Reads are not affected. Asynchronous I/O is not affected. Only small, synchronous writes are placed in the intent log.

The intent log still has to get flushed to the disk synchronously…there is the opinion that this will be faster than writing the data and the inode asynchronously. This is not *true* synchronous I/O…and does not maintain the data integrity like true synchronous I/O.

Check this scenario out: the flush of the intent log succeeds, so the `write()` returns to the application. Later, when the data is actually written, an I/O error occurs. Since the application is no longer in write, it can't report the error. The syslog will have recorded *vx_dataioerr*, but the application has no clue that the write failed. There is the possibility that a subsequent successful read of the same data would return stale data. We still feel that `nodatainlog` is way much more betta than `datainlog`.

Let's also talk a little `convosync=direct`. Stephen *has* seen a couple of customer systems that have suffered when this option has been used. It does make for more direct I/O (more physical than logical I/O). Performance improvement has been seen when this option has been removed. Afterwards, there appears to be *less* physical I/O taking place. A side effect of this may be a lower read cache hit rate… the `convosync=direct` option acts as if the `VX_DIRECT` caching option is in effect (read `vxfsio(7)`) and buffer cache was not being used. After the option is removed, you *are* using buffer cache more and probably experiencing a more worser (lower) hit rate. Remember: that is a couple of customers…most will not feel negative performance *with* `convosync=direct`.

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' file systems 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`

Nolog acts just like tmplog. Stephen can explain, if you buy him a beer and give him an hour. If you buy him TWO beers you will have to give him TWO hours.

Generally, for file system options the more logging and recoverability you build in, the less performance you have. Generally, consider the cost of data loss versus the cost of additional hardware to support better performance. You should have a decent backup/recovery strategy in place regardless, and UPS to avoid downtime due to power outages.

IMPORTANT NOTE: There is almost always a JFS 'mega-patch' available. Keep current on JFS (VxFS) versions and patch levels for best performance! There are many enhancements, dynamic tunables, etc. READ UP ON 'EM! AND, read Mark Ray's papers!

Since we are talking about file systems, we want to mention one 'odd/special' case – to maybe be aware of! It happened on 11.23 on a system with 400 file systems and a VERY

LARGE buffer cache. The buffer cache was larger than most people could even imagine for just physical memory alone! The problem was that it took more than 7 hours to shutdown the system. We will spare you the internals details of how many times the cache is traversed and why – for each mounted file system. The bottom line: an enhancement was made to allow file systems to be unmounted in parallel. The product is UnmountallEnh, available from the HP Software Depot website. The parallel unmounts occur only upon shutdown and it scales better with more processors. The above-mentioned shutdown then took less than 30 minutes.

OK one more trick to discuss... on unix there are ways to mount tmp and other 'scratch' filesystems in memory-only. On HP-UX this is called the "Memory File System", and there are some references to it on the web under docs.hp.com (search for memfs). It is a mount option and there are various considerations you can read about. Apparently you need a patch on 11.23 to be able to use it. Bottom line: we have not seen this used in the customer base and do not recommend it. If you have ample memory and want to try it, then let us know how it goes.

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. A network performance guru we know says that he typically asks people to get a copy of `netperf` / `ttcp` (for transport layers) or `iozone` (for NFS) and run those benchmark tests to measure the capabilities of their links and if those tests indicate a problem then he starts drilling down with tools like `lanadmin`, network traces, switch statistics, etc. You can dig up more information about different tools and net tuning in general from the HP docs website or 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 and configured properly.
- 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.
- When setting up an NFS environment, use NFS V3 and read Dave Olker's book on *"Optimizing NFS Performance"*, which is out of print but you can find it!
- 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 (obsolete now) that had a bad timeslice tunable value in them. The moral is never to blindly accept anybody's recommendations about kernel tunables (sometimes even HP's recommendations — hey wait who do we work for again??!?). 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 a given system, then Stephen says: "Do that".

Also note that some application vendors have guidelines for configuring tunables. It is best to take their recommendations, especially if they won't support you if you don't! EVEN if you find out that you ain't even usin' SPIT in comparison to what they told ya to configure. They may not support you unless you do what they say!

What follows is a brief rundown of our general recommendations for the tunables that are most important to performance on 11.23 and 11.31. For background as to the definitions of these parameters, their ranges, and additional information, look at the SAM utility's online help. Compared to 11.0 and 11.11, many of the default 11.23 and 11.31 tunable settings are OK. Over time, tunables control a smaller proportion of overall memory, and more tables become dynamic, which also helps. Due to 11.31 and this word's 'smattering' all over all documentation…we might just use it here for both 11.23 and 'behind' (and 11.31). That word would be ***DEPRECATED***! Why can't they just say "we ain't gonna use it anymore"? In any case, what follows are the ones we still worry about:

`bufpages`
This was 'deprecated', along with `nbuf`, in 11.31. In other words, don't worry about it there. On 11.23, you can 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, then the values of `dbc_min_pct` and `dbc_max_pct` are *ignored*. In order to get a 1GB (one gigabyte) fixed buffer cache, which is our recommendation for 11.23 systems with *OVER FOUR* GB of memory, set `bufpages` to 262144. For smaller systems or any system on 11.0 or 11.11, we recommend only a 400MB buffer cache (set `bufpages` to 102400). For big file servers such as NFS, ftp, or web servers; you should increase the buffer cache size so long as you don't cause memory pressure. If you are more comfortable with setting `dbc_min_pct` and `dbc_max_pct` instead of `bufpages`, then set `dbc_max_pct` to a value equivalent to 1GB. We discuss buffer cache tuning in conjunction with the Disk Bottlenecks section below.

`dbc_max_pct`
This is another tunable relevant only to 11.23 (not in 11.31). It determines the percentage of main memory to which the dynamic file system buffer cache is allowed to grow (when `nbuf` and `bufpages` are not set). The default is 50 percent of memory, but this is major overkill in most cases. With a huge buffer cache, 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 want to use a dynamic buffer cache, start with `dbc_max_pct` at a value equivalent

to the recommendation above (for example, on a 11.23 server with 20GB of memory, set `dbc_max_pct` to 5 to ensure a 1GB limit). Set `dbc_min_pct` to the same value or something smaller (it will not affect performance as long as you avoid memory pressure and page outs). We have a subsection below delving more into Buffer Cache issues.

On 11.31, the buffer cache is no longer used for normal file data pages. If you are on 11.31 then don't worry about sizing the buffer cache, instead consider the Unified File Cache settings filecache*, mentioned below.

NOTE: since 11.11, the use of a large buffer cache or UFC (11.31) no longer has performance degradation implications (it has gotten "mo' betta" with each release). If you have ample free memory and you want a large buffer cache – YO, be our guest! Have at it! Stephen has just returned from a customer (11.23) that the more buffer cache he gave 'em…the better the application performed. It happened to be a database that did BOTH: reading a LOT of sequential stuff from a lot of file systems, and then writing (and reading) to raw volumes. One of those special cases, but a good example! They were 'testing' with a bufcache size of 3GB to 5GB.

`default_disk_ir`
This setting tells real 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=1` on every disk device. It has NO effect on complex storage devices that are virtualized and 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. This recommendation may be a '9.5 on your sphincter scale,' but this is an old perception left over from when systems crashed regularly and before data recovery mechanisms were standard. There is no downside that we know of to having this set to 1 (no impact on data integrity!).

`filecache_max` and **`filecache_min`**
Relevant only to 11.31 (and later!), these are the configuration limits of the dynamic Unified File Cache, which (almost entirely) replaces the function of the Buffer Cache. The goal when sizing the is still the same: to avoid memory pressure. You should definitely read through the long-winded man-page: `man 5 filecache_max`, and also take a peek at the UFC section we added towards the end of this paper. Bottom line: the configuration of the UFC defaults to be restricted to between 5% and 50% of physical memory. If you see any sign of a memory bottleneck (discussed below) or you are 'tight' on free memory, you will most likely want to tune `filecache_max` 'down' (to a lower percentage). As was the case with the Buffer Cache in 11.23, having a large UFC, as long as you also have ample free memory, is not a problem.

`max_thread_proc, maxuprc, maxfiles, maxfiles_lim, maxdsiz, maxssiz, maxdsiz_64,` and friends
There are a bunch of tunables that configure the maximum amount of something. These limits used to be more important because 'butthead' applications that went crazy doing dumb things were more common in the past. These days, you're more likely to get annoyed by hitting a limit when you don't want to (because it was set lower than your production workload needed), so we generally tell you to bump them up from the defaults

if you suspect the default may be too low. Or, unless told otherwise by your more knowledgeable software vendor. If you know that nobody is going to run any 'rogue' program, say, that mallocs memory in a loop until it aborts, then bump the `maxdsiz` parameters to their maximum!

The old `maxusers` parameter is gone, thankfully! Doug has overheard Stephen say that tunable formulas generally suck.

`nfile`
The maximum number of file opens 'concurrently at the same time' (that is, not the number of open files but the number of concurrent `open()`s) on the system. The default is normally fine. 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 file system server then you're more likely to want to bump up these limits. We have found that most customers do not realize that multiple locks can be held on a single file…by one process or multiple processes.

`ninode`
This sets the inode cache size for HFS file systems. The VxFS cache is configurable separately (see `vx_ninode` below). Don't worry about it.

`nkthread`
The maximum number of kernel threads allowed on the system. The default is fine for most workloads. If you know that you have a multi-threaded workload, then you may want to bump this higher.

`nproc`
This is heavily dependent on your expected workload, but for most systems, the default is fine. If you know better, set it higher. Don't blindly over configure this by setting it to 30000 when you'll have only 400 processes in your workload, as this 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 that it reaches over 80 percent utilization during normal processing.

`shmmax`
We have seen 64bit Oracle break up it's SGA shared memory allocations (`ipcs -ma`) when this tunable is configured too low. This can hurt performance: if you have the physical memory available, then let the DB allocate as much as it needs in one chunk. Bump the segment limit up to its max (unless you fear 'rogue' applications causing a problem by hogging shared memory, which typically ain't nuthin' to worry about). The default is 1GB… a little too low for big servers.

`swapmem_on`
Pseudo swap is used to increase the amount of reservable virtual memory. This is only 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. There is no 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 do it. FYI: the 'values' used for pseudo swap is 100% of memory in 11.23 and above, *and* it's always turned on in 11.31 (not configurable). 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, 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! As of the writing of this current version of the paper (2008): Stephen STILL finds a system here and there that has timeslice incorrectly set to '1'!

`vx_ninode`

The JFS inode cache is potentially a large chunk of system memory. The limit of the table defaults high if you have over 1GB memory (for example, 8GB physical memory calculates a quarter million maximum VxFS inode entries). But: the table is dynamic by default so it won't use memory without substantial file activity. You can monitor it with the command: `vxfsstat /`. If you notice that the `vxfsd` system process is using excessive CPU, then it might be wasting resources by trying to shrink the cache. If you see this, then consider making the cache a specific size and static. Note that you can't set `vx_ninode` to a value less than `nfile`. For details, refer to lengthy JFS Inode Cache discussion in the "Commonly Misconfigured HP-UX Resources" whitepaper that we point to in our References section at the end of the cookbook. As a general rule, don't muck with it. If you have a file server that is simultaneously accessing a tremendous number of individual files, and you see the error: `vx_iget - inode table overflow` then bump this parameter higher. Most say "YO, it's dynamic…what do I care"? GEE… do you know anyone that might run a find command from root? How fast DO YOU THINK this table will grow to its maximum ☺? If you are on an older OS pre-11.23: set it to 20000.

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 Performance Agent (PA) works good. In addition to local export, you can view the PA metrics remotely with the Performance Manager or other tools that used to be marketed under the term "OpenView". Also, the new HP Capacity Adviser tool can work off the metrics collected by PA. Whatever tools you use, 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 file system options and buffer cache or UFC 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 Glance and the Performance Agent'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, application parameters (like java), or swap space. You can usually look these errors up in the HP-UX or application documentation and it will point you to what limit 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. As they say on cable TV (so it must be true): all great cooks start with the right ingredients! Unless you are Stephen (who is a GREAT cook) and, as usual, has his own unique set of '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 slowest resource in the data flow could be the disk that it's backing up (indicated by the source disk being nearly 100 percent busy). Or, that slowest 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! Making it run faster is not the best use of your time. Remember: a disk (or address) being 100% busy does not necessarily indicate a bottleneck. Coupled with the length of the queue (and *maybe* the average service time)…it might indicate a problem.

Now, if your backup is not finishing before your server starts to get busy as the workday begins in the morning, you may find that applications running 'concurrently at the same time' 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 Performance Agent (PA is also known as MeasureWare) have the concept of Blocked States (which are 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 that it's 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 xglance (same as 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 xglance use the Configure -> Choose Metrics selection from one of the Report windows to see the list of all available metrics in that area, and you can right-click to conjure up the metric definitions. If you have PA on your system, a place to go for the definitions is `/opt/perf/paperdocs/ovpa/C/methp*.txt`. A subset of the performance metrics are shown in character-mode glance and logged by PA. If you need more info on tools and metrics, refer to the web page pointers in the References section below.

We use the word "process" a lot, but in HP-UX it is the actually the thread which is the individually schedulable, runnable entity, and a process can be multi-threaded. 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.

CPU Bottlenecks

CPU Bottleneck Recipe Ingredients:

- Consistent high global CPU utilization (`GBL_CPU_TOTAL_UTIL > 90%`), and
- Significant Run Queue (Load Average) or processes consistently blocked on Priority (`GBL_RUN_QUEUE > 3 or GBL_PRI_QUEUE > 3`).
- Important Processes often showing as blocked on Priority (waiting for CPU) (`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), shows the average number of processes waiting for any CPU (that, is, blocked on PRIority). 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 performance tools use either the running average or the instantaneous value.

We should also mention that you may see other rules of thumb that have been published or presented elsewhere. Feel free to let us know if you find alternatives that work better for you, but our guidelines here have held up well for use by many admins for many years.

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.

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 (and doing no I/O), then it might be stuck in a spin. User-mode processes that are causing I/O may be doing memory-mapped I/O. If shell processes (sh, ksh, or yuck-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 dive deeper into detailed HP-UX application performance, including **Caliper** for Itanium and **Prospect** for PA-RISC.

The HP PRM product (Process Resource Manager) and Global Work Load Manager (gWLM) are worth checking into to provide CPU control per application. Some workloads may benefit by logical separation that you can accomplish via one of HP's Virtual Server Environments (nPars, vPars, or HPVM). If you are engaged in consolidation activities, check out the HP Capacity Adviser product as well. In the race to keep up with changing systems, sometimes the one with the best tools wins!

A 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 it's processing priority relative to other timeshare processes. There are many scheduling 'tricks' that processes can invoke, including POSIX schedulers, although use of these special features are not common. Oracle actually recommends disabling user timeshare priority degrading via `hpux_sched_noage` (sets kernel parameter `SCHED_NOAGE`). It is a long story that Stephen talks about in his 2-day seminars. A simple (right☺) explanation is that many people discuss this using the term 'priority inversion'. When you use `SCHED_NOAGE`, it tells the kernel NOT to adjust/degrade the priority of a process/thread. The most bestest priority that can be set using the rtsched command or system call (with the `SCHED_NOAGE` policy) is 178 – which is the most bestest USER priority in the HP-UX timeshare range.

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.

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 it itself, so jump down to the next section on Context Switching Bottlenecks.

If the system CPU isn't caused by context switching, then see if the metric `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.

Some people have expressed a concern to us over vPars (virtual partitions) and allocating bound versus unbound processors. Apparently I/O interrupts are restricted to bound CPUs. We have not seen this be an issue in the real world… in other words, don't worry about not allocating 'enough' CPUs bound unless you have a shi*pt*load of I/O happening and you see high Interrupt-CPU levels, as above, on your bound processors. Only in that case should you start worrying about 'needing' to make more unbound (floater) CPUs into bound CPUs.

If you aren't burdened by high System CPU caused by Context Switching or Interrupts, then we can assume at this point that most of your kernel time is spent in system calls (`GBL_CPU_SYSCALL_UTIL >30%`). Now 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 system calls.

If you can catch the problem live, you can use Glance to drill down further. We like to use xglance (gpm) for this because of it's 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 system call will then be listed first. You can also sort by CPU time to see which system calls are taking the most CPU time, as some system calls are significantly more expensive than others are. In xglance'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 system calls 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 system call 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(), sched_yield()`, 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 file system 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!

We once saw a case where a system CPU bottleneck was found to be caused by programs communicating with each-other using very small reads and writes. This type of activity

has a side effect of generating a lot of kernel syscall traces which, in turn, causes the midaemon process (which is used by Glance and PA) to use a lot of CPU. So: if you ever see the midaemon process using a lot of CPU on your system, then look for processes *other* than the midaemon using excessive system CPU (as above, sort the glance process list by the `PROC_CPU_SYS_MODE_UTIL` metric). Particularly inefficient applications make very short but incessant system calls.

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 type of situation results in very long-running System Calls. 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 system admin 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. It is called **`tusc`**, and it's very useful for tracing activity and system calls made by specific processes: very useful for application developers. It's 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 touching virtual memory that is not resident, or 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. Also, we have seen high fork/exit rates caused by 'agentless' system monitors that incessantly login from a remote location to run commands. Since faster systems can handle higher fork rates, it's hard to set a rule of thumb, but you can monitor the metric `GBL_STARTED_PROC_RATE` over time and watch for values over 50, or periodic spikes. Trying to track down who's forking too much is easy with xglance; 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 if you can't reduce context switch utilization by changing the workload.

Memory Bottlenecks

Memory Bottleneck Recipe Ingredients:

- High physical memory utilization (`GBL_MEM_UTIL > 95%`), *and*
- Significant pageout rate (`GBL_MEM_PAGEOUT_RATE > 10`), or
- Any 'true' deactivations (`GBL_MEM_SWAPOUT_RATE > 0`), or
- vhand process consistently active (`vhand's PROC_CPU_TOTAL_UTIL > 10%`).
- 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 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, watch out for pageouts, because they are often (but not always, especially in 11.31!) a memory bottleneck indicator. Don't worry about pageouts that happen when memory utilization is not high. If memory utilization is less than 95% and you see pageouts, they are most likely due to memory-mapped file writes. This is much more common in the 11.31 because of the Unified File Cache. The UFC has its own dedicated section at the end of this paper. If memory utilization is high (>95%), *and* you see pageouts along with 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"). A 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 PA 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 Performance Manager, you can draw a 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, but it could also be leaking via memory-mapped files that aren't unmapped (you would see a growing number of MEMMAP/Priv regions). Globally, you'll also see `GBL_SWAP_SPACE_UTIL` on the increase if there is a leak somewhere. Restarting the app or rebooting are workarounds, of course, but correcting the offending program is a better solution.

A common cause of a memory bottleneck is an overly large file system buffer cache on 11.23. On 11.31, we fear similar issues may crop up with an overly large Unified File Cache (UFC). If you have a memory bottleneck, and your 11.23 buffer cache size or 11.31 file cache size is 1GB or over, then think about shrinking it.

If you don't have any memory leaks, your buffer cache or UFC 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 (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 Oracle,

Informix and SAP). Large page size is a technique that can be useful for some apps that have very large working sets and good data locality, to avoid TLB thrashing. Java administers its own virtual memory inside the JVM process as memory-mapped files that are complex and subject to all kinds of java-specific parameters. These topics are a little too deep for this dissertation and are of limited applicability. Only use them if your application supplier recommends it.

Oh yeah, and if this all were not confusing enough: One of Stephen's favorite topics is 'false deactivations'. This is a really interesting situation that HP-UX can get itself into at times, where you may see deactivations when memory if nearly full but NOT full enough to cause pageouts! This appears to be a corner case (rarely seen), but if you notice deactivations on a system with no paging, then you may be hitting this. It is not a 'real' memory bottleneck: The deactivated processes are not paged out and they get reactivated. There is NO VM I/O generated and it is really just a 'preemptive strike' by the O/S *just in case* the system does become 'memory pressurized'! This situation is mostly just an annoyance, because you cannot count solely on deactivations to indicate a memory bottleneck.

Swap sizing

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' volumes. 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 file system 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).

Disk Bottlenecks

Disk Bottleneck Recipe Ingredients:

- Consistent high utilization on at least one disk device (`GBL_DISK_UTIL_PEAK > 50 or highest BYDSK_UTIL > 50%`).
- Significant queuing lengths (`GBL_DISK_SUBSYSTEM_QUEUE > 3 or any BYDSK_REQUEST_QUEUE > 1`).
- High service times on BUSY disks (`BYDSK_SERVICE_TIME > 30` *and* `BYDSK_UTIL > 30`)
- 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 most 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 HBAs, 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 PA. Both Glance and sar have included more detail on I/O for 11.31 via breakdown by HBA. Analysis usually starts 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. Filter your inspection using the `BYDSK_UTIL` metric to see utilization trends, and then 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 PA'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.

A lot of old performance pundits like to use the Average Service Time on disks as a bottleneck indicator. Higher than normal services times *can* indicate a bottleneck. But: be careful that you are only looking at service times for *busy* disks! We say: "Service time metrics are CRAP when the disk is busy less than 10% of the time." Our rule of thumb: if the disk is busy (`BYDSK_UTIL > 30`), *and* service times are bad (`BYDSK_SERVICE_TIME > 30`, measured in milliseconds average per I/O), only then pay attention. Be careful: you will often see average service time (on a graph) look very high for a specific address or addresses. But then drill down and you find that the addresses with the unreasonable

service times are doing little or *no* I/O! The addresses doing massive I/O may have fantastic service times.

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-Disk, By-Filesystem, By-Logical Volume, and in 11.31 also By-HBA. 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 (File system, 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 and offsets, which may help pinpoint the specific files that are involved. The problem with all the system performance tools 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 tool, and specialized tools may be needed to analyze the internals of the array. The specific vendor is where you'd go for these specialized storage management 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 file systems (and/or logical volumes) out across multiple HBAs and physical disks (LUNs) to maximize your throughput.
- 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 parallelism 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 (`scsi_queue_depth`) down.

Some facts to be aware of regarding disks:

- The smaller the I/O, the shorter the service time. The larger the I/O, the longer the typical service time.
- Sequential I/O is faster than random I/O (decreased head movement).
- To maximize throughput, use larger I/O sizes for sequential I/O.
- The maximum buffered I/O size is 64KB.
- Maximum direct I/O size is 256KB (it can be 1MB on 11.23 with a patch for VxFS and a couple of patches for VxVM).

- Crossing various boundaries will result in breaking up an I/O request into smaller I/Os. These boundaries include: file system block, buffer chain, file extent and LVM LTG boundaries.

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`). 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

Buffer Cache Bottleneck Recipe Ingredients:

- Moderate utilization on at least one disk device (`GBL_DISK_UTIL_PEAK or highest BYDSK_UTIL > 25`), and
- Consistently low Buffer Cache 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 in 11.23, then you may want to bump *up* the file system buffer cache size, especially if you have ample free memory and 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! While some file system I/O-intensive workloads can benefit from a larger buffer cache, in all cases you want to avoid pageouts! In practice, we more often find that buffer cache is overconfigured rather than underconfigured.

Also, if you manage a database server with primary I/O paths going to raw devices, then the file system buffer cache just gets in the way. This is also true for the 11.31 UFC, which is discussed in its own special section at the end of this paper.

To adjust the size of the 11.23 buffer cache, refer to the Kernel Tunables section above discussing `bufpages` and `dbc_max_pct`. Since `dbc_max_pct` can be changed without a reboot, it is OK to use that when experimenting with sizing. Just remember that the size of the buffer cache will change later if you subsequently change the amount of physical memory. We used to rail against over-configuration of Buffer Caches, which was a big problem on HP-UX 11.0 and 11.11, but in 11.23 and later there is no performance penalty for having a large cache IF you have the memory.

If you suspect, from the above symptoms, that you may have too large a Buffer Cache, *and* you typically run with memory utilization (`GBL_MEM_UTIL`) over 90%, *and* your

buffer cache size (TBL_BUFFER_CACHE_USED, found in Glance in the System Tables Report) is bigger than 1GB, *then* reconfigure your buffer cache size smaller. Configure it to be the larger of either half its current size or 1GB. After the reconfiguration, go back and watch the hit rate some more. Lather, Rinse, Repeat. Your primary goal is to lower memory utilization so you don't start paging out (see Memory Bottleneck discussion above).

If your applications will take advantage of a very large cache, and you have a lot of free/available memory --- by all means go ahead and configure a large cache! There is a known case (described to Stephen by Mark Ray) of a customer with a buffer cache of 387GB! Now datsa GI-FREAKIN-GANTIC buffer cache, EH?!

Networking Bottlenecks

Networking Bottleneck Recipe Ingredients:

- High network byte rates (dependent on configuration) or utilization (BYNETIF_IN_BYTE_RATE or BYNETIF_OUT_BYTE_RATE or BYNETIF_UTIL > 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 PA have metrics for packet, collision, error rates and utilization by interface (BYNETIF_UTIL). 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 byte rates or utilization (GBL_NET_UTIL_PEAK) is higher than normal, and see if those times also have any output queue length (GBL_NET_OUTQUEUE). Be careful, because we have seen that metric get 'stuck' at some non-zero value when there is no load. That's why you look for a rise in the activity. 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 PA's network subsystem queue metric). The `netstat` and `lanadmin` commands give you more detailed information, but they can be tricky to understand. The `ndd` 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 Network Node Manager are specifically designed to monitor the network from a non-system-centric point of view.

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.

If you use NFS a lot, 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.

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 around the clock. 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 PA, 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 the 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 it's doing. In PA, 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.

11.31's Unified File Cache

In honor of 11.31 'hitting the street', so to speak, we have devoted the following short section to delving a little deeper into a significant change in 11.31 specific to performance: the addition of the Unified File Cache (UFC).

We have already discussed what actually controls the size of the UFC (`filecache_max` and `filecache_min`). Let's just talk a little about this cache *without* going into the internals or exactly how it works, etc. Since the old concept of buffer cache 'goes away' (sorta) with 11.31, we just want you to know a little about this new thing.

The buffer cache still exists…it only caches non-JFS meta data structures. Regarding JFS (VxFS), there are similarities between the file cache and the page cache. Read the other papers mentioned in our References for details on things like the internal parameter `discovered_direct_iosize` (and many other VxFS file system parameters – all of the parameters we are referring to can be found in the man-page for `vxtunefs`), but you should know that the file cache is used for read and write requests smaller than `discovered_direct_iosize`, and it also allows for read ahead and asynchronous writes. While the buffer cache could have buffers of many different sizes, a major difference is that the file cache is based on a 4KB page. Another major difference is that JFS no longer performs 'flush behind'. Now, dirty pages are flushed by vhand or vxfsd, so you will generally see these daemons more active on 11.31.

Something that Stephen has found, that many people he has encountered are unaware of, is something affectionately known as 'read before write'. This is not just 11.31, but…you need to be aware of it. It can happen in both the buffer cache and the file cache and can have performance implications. We will not do the sizes, numbers, etc, which can be found outside of this paper. We will do the short (right ☺) 'Stephenism'. If youz do a small write to either the buffer or file cache and the buffer or page ain't already in the cache --- this condition may just arise. If the write is smaller than an 8K buffer or a 4K page (or there are alignment issues), youz are gonna hafta read the buffer or page, perform the modification and then do the write.

Prior to 11.31, HP-UX used the buffer cache for read, write and sendfile access. It also had a separate Page Cache for memory mapped file [`mmap()`] access. Because they were different, it caused significant problems with data coherency between these two separate entities. You could not guarantee WHAT the data in the file would look like if the file was being 'used' concurrently at the same time by both methods (buffer cache and memory mapped), which prevented some applications from porting to HP-UX. With

11.31, the unified file access allows for more easier portability and automatic synchronization.

OK: remember…no real internals here --- we just want you to sorta understand this new UFC 'animal'☺. The kernel manages the UFC 'mappings' through the Virtual Memory subsystem. It is very similar to the way it manages other kernel objects like shared memory, shared libraries, shared text and uses the 'process structures'. In 11.23, the File Cache just appears to be part of User memory. If you have taken a HP-UX Internals course or read Chris Cooper's HP-UX Internals book or taken one of Stephen's Internals and Performance seminars, then you already know what vas, pregions, regions, virtual frame descriptors (vfd's) and disk block descriptors (dbd's) are. The vfd is used to locate a page in memory and the dbd is used to locate a page on disk. The UFC locates the 4KB pages of the file cache using the btree of vfd's and dbd's. This manner of locating pages in the cache (using the btree) is for fast access. OH YEAH: the UFC is also ccNUMA-aware, and supports large pages.

UFC and bufcache differences

The buffer cache is (duh) buffer based, usually uses an 8KB buffer size, and it uses a Least Recently Used algorithm. The UFC is 4KB page-based and uses a Not Recently Used algorithm. It is managed by vhand (the Virtual Memory Subsystem). File Cache pages are allocated from its own File Cache Memory Resource Group and therefore has its own concept of 'freemem'. In 11.31, vhand is expected to be more active. This is due to the fact that it now has the responsibility of aging and stealing pages in the file cache. Now, when memory pressure begins to happen…vhand might not be able to guarantee the throughput to free enough pages to satisfy the demand. SO: 11.31 introduced *inline paging* – a thread/process/function requesting memory can execute vhand paging code in its own context (if it did not get the memory that it requested and a 'fault' occurred).

One big difference between the bufcache and the UFC shows up in the performance tools. For 11.31 versions, Glance added the metric `GBL_MEM_FILE_PAGE_CACHE` which is supposed to be the size of the UFC, however Doug's testing on 11.31 has shown this *not* to be true! There is, as of the date of this revision, no way to see the exact size of the 11.31 UFC. The Glance metric is actually showing you the amount of UFC in 'active use', but there may be memory allocated to the UFC that is not active. If you want to test this for yourself, you can experiment (on a *non*-production system): try bumping up the minimum filecache on an idle system while watching memory statistics (like Glance's Memory Report)... you will likely see Free memory go down and User (not FileCache) memory jump up! This is because (as of Glance version 4.70), the inactive parts of the UFC are 'hiding' and show up (incorrectly) bucketed as User memory. The FileCache metric can show a value (which is the same as shown by the command `kcusage filecache_max`) that is actually less than your setting for `filecache_min`! Do we have your head spinning yet? Now, we are *sure* that this will be fixed in some future version, but its good to be aware of now. Bottom line: in 11.31 the User (and to a lesser extent, System) memory metrics are suspect. The FileCache size displayed by the tools may not be reflective of the 'real' size of the UFC. Fortunately, you can depend on the Free

memory and total Used memory metrics (`GBL_MEM_FREE` and `GBL_MEM_UTIL`) still being accurate, as well as the other metrics that we rely on as memory bottleneck indicators.

FINALLY, here are a few other UFC options and tunables besides `filecache_max` and `filecache_min` that we're not going to say much about... so if youz wanna know more youz gotta go lookemup:

- `fcache_seqlimit_system/fcache_seqlimit_file` – the sequential access limit on the system and the sequential access limit per file, both are expressed as a percentage of the maximum file cache size. Both default to 100.
- `fcache_fb_policy` – this can enable flush behind. In 11.31 (unlike 11.23) it is disabled by default for performance.
- `fadvise()/fcntl()` – programmer stuff. Options to this library function (3) and this system call (2) can be used to set ccNUMA policies, large page hints and syncer interval options per file.

Cells, Cell Local Memory, Locality Domains, NUMA Latencies and Processor Sets

What follows is a section specific to 'Superdomey' systems (cell-based servers). Definitely optional reading if you have these types of systems and are interested. Considerations specific to cells/ldoms can be pretty intense, so we gotta hurt youz a little here, BUT…methinks it gotta be done and these things certainly have become issues hurting and helping performance... and that is what this paper is all about, EH? It certainly is gonna look like we drifted into INTERNALS, but we really didn't! If this doesn't get nerdy enough for you, we can surely point you places to where you [masochistic people ☺?] can really go hurt yourselves! OK, here goes…

The NUMA Hierarchy

We really gotta talk to youz about 'dis 'NUMA stuff, cause then you'll be 'backgrounded' so we can tell ya what we (eventually!) want to say further on about performance!

The NUMA Physical Hierarchy goes from the potentially multithreaded processor, to the possibly multi-cored socket, to the Front-side bus, to the Cell, up to one or two crossbar hops. From the perspective of the Operating System, the logical abstraction of the Locality Domain (LDOM) maps to a physical cell. Generally, the View of NUMA by the O/S is very simplistic.

The latencies to access data across these hierarchies vary by source and destination. The best latency is for a cache-to-cache copy between cores on the same front-side bus (85ns). Next, is a miss to memory on the same cell as the core requesting the memory (185ns). A cache-to-cache miss is most costly when not on the same font-side bus even when in the same cell (277ns, 466ns). Cache-to-cache misses on large configurations are

more even costly (up to 677ns). A one hop memory miss is 386ns while a two hop is 460ns.

Attempt to reduce data cache miss stall cycles several ways. Use the right system: 1) A small bus-based system will provide better single stream performance. 2) If the workload scales, a cell-based system could be capable of more throughput. In a NUMA system, you should consider placing things closely that share modified data. Finally, it is goodness if applications can make use of Cell Local Memory.

OK, NOW: we can't be teachin' ya the detailed internals, can't be teachin' ya the commands we will reference, can't be teachin' ya Oracle and we can't be teachin' ya exactly how to do all of the things we are about to TRY and educate ya on, K? K! Here goes:

Cell-local memory

We will try to cover this in English (or at least 'Stephen's English as a second language'). Cell Local Memory (CLM) are blocks of memory coming from a given cell appearing at a range of addresses. CLM is not interleaved with memory from other cells. Access times to these addresses will be optimal for all of the processors in the same cell. The amount of CLM is configurable at boot time.

In 11.31, the kernel makes heavy use of CLM. Oracle 10gR2 uses CLM (more Oracle later). Applications that are not NUMA-aware will also benefit from CLM 'cause of the kernel allocation policies. Private data (hopefully, you know what private data is and the various 'parts' like stack, etc.) is allocated in CLM. Shared data (you gotta know this by now!) is allocated in interleaved memory.

In the past, CLM did not really 'help'. Each thread is allocated a home locality – the locality in which it was launched. CLM allocations are made from the home locality and unfortunately, threads seem to quickly migrate from their 'home cells'. The bottom line is that all accesses end up being made to a remote CLM!

There are things to consider when configuring Cell Local Memory and there are (we borrowed 'em☺) some starting point suggestions. What if you have a 6-cell system? 5 out of 6 accesses are remote! It can't get too much worse…can it? If your application can take advantage of CLM, you have a potentially large upside and a potentially small downside.

Those stolen (borrowed?) suggested starting points:

- Any O/S with Oracle 10gR2 87%
- 11.31 with earlier version of Oracle 37.5%
- 11.31 with non-NUMA aware apps 37.5%
- 11.23 with non-NUMA aware apps 25%

Controlling placement

Here's an over of the default launch policies, so you can see how this will affect 'where' processes run on a cell/ldom-based system:

- A new process is created in the least loaded locality
- A new thread is created in the same locality, one per CPU
- It will spill to the next least-loaded locality
- Threads/processes are free to migrate – there is no binding
- The 'home locality' is the chosen locality…Cell Local Memory is allocated here

Threads are usually 'moved' by the HP-UX scheduler due to idle stealing and balancing policies.

In order to get CLM to work, we need to tie each thread to its home locality. This is most easily done using non-default launch policies. One (easy) way to control placement is to use a command like `mpsched -p <policy>`, to use a non-default launch policy. the `mpsched` command controls the processor or locality domain on which a specific process/thread executes.

Consecutive threads should be specified as to which LDOM they will be created in. This ties threads to their home LDOM. Read up on mpsched(1) for policies: **RR, RR_TREE, LL, FILL, FILL_TREE, PACKED, NONE**. We will not explain these in detail…just wanna mention the policies.

You can use mpsched to see localities and CPUs, to bind a process to a specific processor, to execute a command in a specific LDOM and to execute all of a process' threads in the same specific LDOM.

Another way to control placement is by using Processor Sets…AKA **PSETS**. It actually seems that whenever you talk to *almost anyone* and mention 'PSETS' - their eyes glaze over and roll back in their head!

Using PSETS you can:

- dedicate processors to specific workloads
- separate workload pieces to improve cache and TLB behavior
- virtually remove processors to maintain consistent performance when ramping up a new system
- isolate processors with a high interrupt handling load
- you can provide near real-time environments for workloads that are latency sensitive…this can be done with Real Time PSET

There are disadvantages to using PSETS. They can be a large pain to set up until you are familiar with them. PSETS will not persist beyond a reboot and you are taking any flexibility away from the scheduler.

For you to manage processor sets you will need to definitely read up and learn `psrset(1)`. With psrset you can create a new processor set, assign processors to a new

PSET, tie/bind process IDs to a processor set, execute a command in a processor set and assign processes that belong to a specific user to a processor set. You can also display the attributes of a PSET, show the PSET assignments for processes and display the processes that are assigned to a specific PSET.

Finally, we JUST want to mention **RTE PSETS** (Real Time Extensions). This is also done with the psrset command and you are reserving processors for real time tasks. It is a processor set, BUT…there are NO external I/O interrupts taken and NO callout processes and NO kernel daemons. This one we want you to go figure out…not gonna write a novella here!

LAST (so far) and certainly not least: we said we would talk about Oracle and Cell Local Memory and some performance improvement. Here we go…

In Oracle 10g NUMA optimization is enabled by default. Prior to 10g, it was 'available' but not enabled. You had to know about it and enable it! The parameter that is set (in the parameter file) is `_enable_NUMA_optimization=true`**.** We have mentioned in the past that (normally) when one sees multiple Oracle SGAs of equal size attached by the identical number of processes – it is usually due to `shmmax` being smaller than the size that the DBA has requested for the SGA --- and it got 'broken up' into shmmax-sized segments.

These days, you are liable to see several equal sized shared memory segments and they *will not be as large as* shmmax**!** This is due to the above mentioned parameter being set to *true*. Let's just stick with 10g as the example, since it is the default behavior. Still keep in mind: you can set the parameter to true on earlier versions of Oracle. This use of multiple SGAs is expected for performance reasons.

In a paper from Oracle they state that they create one shared memory segment per process group and one segment that 'stripes' across all groups (along with the small bootstrap segment). Performance should be better with multiple segments. They also say that NUMA optimization is an internal optimization on the way the data structures are laid out and how the buffer cache is laid out such that we reduce the total number of remote cache misses an a large system.

SO, here is what Stephen has typically seen: On an 'N' numbered cell-based system there will be 'N' number of equal sized shared memory segments and *one more* (that always appears smaller) which is the segment that is 'striped'. Then you will (as always) see the very small bootstrap segment. Each segment will be attached by the same number of processes (for that specific instance of Oracle). You may see this multiple times on a single system only the sizes (for the 'N+' segments) may differ from group to group. This just means that you have multiple instances of Oracle in the same system…with the same number of cells.

Miscellaneous 11.31 tidbits

OK now that we have your head spinning slightly, we'll toss in a couple of 11.31 details that did not fit well above. We are seriously tired of 'hurting youz here', so: we ain't gonna write a whole shi*pt*load of stuff – just enough to get you acquainted with more new 11.31 pieces.

Mass Storage Stack improvements

There were many improvements, but…we will talk about the **performance** improvements due to the new mass storage stack. they included:

- Automatic load balancing of I/Os on all available lun paths.
- Choice of load balancing algorithms…like cell aware round robin policy: it selects a path from the locality of the CPU where the I/O was initiated.
- Parallel I/O scan reduces scan time significantly (also improves boot time).
- CPU allegiance algorithms reduce cache misses.
- Maximum I/O size increased to 2MB.
- Enhanced performance tracking tools.

The Mercury project

This is another 11.31 'enhancement,' relevant to developers more than sysadmins. Tell your programmers to do a man on '`hg`' in section 3 of the manual. Just some key points here:

- User and kernel cooperative performance hinting and information sharing.
- Performance is faster than the traditional system call interface (supposedly 10x to 30x faster).
- Faster for information like `time` and the current processor that a thread is running on.
- Provides information that is almost impossible to get with standard system calls.
  - Context switch counts
  - Is another cooperating thread running?
- Allows user programs to 'influence' scheduling.
  - Fast way for a thread to tell the kernel (maybe during a critical portion of code) that it does not want to be context switched. This is temporary.

11.31 versus 11.23

Due to the fully parallelized device scanning - boot, scanning and reboot times have been reduced. Ioscan and insf times have been dramatically reduced. There less CPU per MB of data transfer, and locality improvements: the cell local round robin load balancing keeps memory accesses within the cells. LVM has been enhanced to support larger page sizes. The default maximum I/O size has been increased from 1MB to 2MB, and the I/O MAX_queue_depth is more flexible – can set per device, device type, vendor ID, product ID, etc.

Much of the native multi-pathing, load balancing and improved I/O performance is due to improvements in cell locality. 11.31 has taken steps to reduce cache miss and cache line sharing, and keep I/O scheduling in the cell where the CPU that scheduled the I/O is.

Conclusion

There is no conclusion to good performance: the saga never ends. Collect good data, train yourself on what is normal, change ***one*** thing at a time when you can, and don't spend time chasing issues that aren't problems.

What follows are the most common situations that Stephen encounters when he is called in to analyze performance on servers, from most common to least common:

1. No bottleneck at all. Many systems are overconfigured and underutilized. This is what makes virtualization and consolidation popular. If your servers are in this category: congratulations! Now you have some knowledge to verify things are OK on your own, and to know what to look for when they're not OK.
2. Memory bottlenecks. About half the time these can be cured simply by reducing an over configured buffer cache. The other half of the time, the system really does need more memory (or, applications need to use less).
3. Disk bottlenecks. When a disk issue is not a side effect of memory pressure, then resolution usually involves some kind of load rebalancing (like, move your DB onto a striped volume or something). As of the date of this paper, I/O issues are beginning to become more frequent. The GSE Performance Team would probably tell you that most of the issues that land in their lap are I/O related.
4. User CPU bottlenecks. Runaway or inefficient processes of one kind or another are often the cause. You can recode your way out or 'MIP' your way out with faster/more CPUs.
5. System CPU bottlenecks. Pretty rare, and usually caused by bad programming.
6. Buffer cache bottleneck: Underconfigured buffer cache can lead to sucky I/O performance, and is typically configured too low by mistake.
7. Networking or other bottlenecks.

The most important thing to keep in mind is: Performance tuning is a discipline that will soon no longer be needed, as all systems of the future will automagically tune themselves... ***yeah, right!*** We think **NOT!** Performance tuning is around to stay. It is not a science; it is more like a mixture of art, witchcraft, a little smoke (and mirrors), and a dash of luck (possibly drugs). May yours be the good kind.

References

HP Developer & Solution Partner portal:
http://h21007.www2.hp.com/portal/site/dspp

HP Documentation Archives:
http://docs.hp.com
http://support.openview.hp.com/selfsolve/manuals

GSE team's Common Misconfigured HP-UX Resources whitepaper (this has been updated, slightly, for 11.31):
http://docs.hp.com/en/5992-0732/5992-0732.pdf

Mark Ray's JFS Tuning paper:
http://docs.hp.com/en/5576/JFS_Tuning.pdf

HP Software system performance products:
http://managementsoftware.hp.com/solutions/ev_prf/

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

About the Authors

Stephen Ciullo is a Senior Technical Consultant with HP's Engineering Services group in the Technical Services organization. Stephen has a primary focus on Performance and HP-UX Internals consulting with expertise in the areas of core HP-UX operating system, C Language System Calls and [now…some] LVM disk management. Stephen delivers technical seminars around the country on HP-UX Internals and Performance on a regular basis.

Doug Grumann is a Subject Matter Expert for system performance in the HP Software organization. Doug works on the HP system performance tools Glance, Performance Agent and Manager, participating in various aspects of their development, support, and marketing over 17 years in this product domain. He is an acknowledged ambassador for the customer interest, and has provided consulting and training on the HP performance products on many occasions.

Combined, Stephen and Doug have over 50 years of UNIX experience. Both also share background in delivering Technical Education, performance tuning and advanced technical topics. They have been collaborating and inebriating on performance topics for more than 15 years.

Doug and Stephen would like to acknowledge and thank all the folks inside and outside HP who have contributed to this paper's content and revisions. 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 Jan Weaver, Ken Johnson, Mark Ray, Colin Honess, Pat Kilfoyle, Rick Jones, Dave Olker, Chris Bertin, Curt Thiem, Santosh Rao, Chris Cooper and all the other 'perf gurus' we work with in HP, for their help and for sharing their wisdom with us.

Legal gobbledygook