Guide to AOS/VS Performance

Written by: Phil Horvitz
General Data Systems, Inc.
March 1990

Overview

This document contains a collections of some of the ideas and philosophies that I've collected over the years of tuning AOS/VS systems. The purpose of this guide is not to be the definitive source of information on tuning, but to present some useful guidelines when planning your performance strategy. As you will find in many fields, everyone has a different opinion on tuning although I believe it is important to develop your own tuning philosophies and stick to them as long as they work for you.

Tuning Obsolescence

Many of the tuning philosophies that analysts are using today have become obsolete due to changes in technology and the improved price/performance of hardware. For example, spending countless hours trying to reduce the memory requirements of your system by trimming the Working Set of a process probably made sense when memory cost $5000 per megabyte. Nowadays, with memory at $500 per megabyte or less, it's going to be more cost effective just to go out and buy a memory board rather than spending your money on labor trying to fix the problem.

I guess the thing to recognize is that in the future, some philosophies like well written, efficient programs will always make sense into the future, but many other strategies may end up becoming a big waste of time and money. It is also predictable that as technology changes, performance tuning will still exist, only the emphasis will **shift** from one area to another. For example, as use of networking increases into the future, it will make sense to pay attention to the efficiencies that can be gained in the networking hardware and software. As a result of increased use of networking, you could probably assume that spending a whole lot of time trying to tune the character I/O load on your system is going to be a waste of time.

Proper Assessment: Averages vs. Peak.

Probably the area where I see the most mistakes being made is in the area of performance analysis and assessment. Most of the problem occurs as a result of the design of most of the performance measurement tools and a lack of understanding of peak measurements vs. average measurements.

Consider the following case: An analyst runs a performance monitor on their system for 24 hours because users of the system have been complaining of slow performance sometimes during the day. During the 24 hours, the system had 8 hours of users online , 2 hours of lunch and dinner time, 4 hours of system backups, 2 hours of batch processing and 8 hours of idle time. Now, the statistics from all of this time are averaged together with the 1 hour of slow performance that the users where complaining about. The analyst looks at the numbers and says, "Aha, just as I thought, we have plenty of idle time -- almost 50% -- the users just like to complain!".

Upon closer examination of the example above, we find that the analyst averaged in data from all different unrelated periods with the period in which the users complain the most (probably 10 AM to 11 AM). The resultant average is meaningless, yet I see this mistake made all the time. Smarter analysts run their performance monitor for only the 8 hours during the workday and then usually end up making the same mistake but on a much smaller scale. I was recently called by an analyst and told that the system that he was measuring had plenty of idle time since the 8 hour average was 50%.

What exactly does 50% CPU Idle time mean? It means that during the 8 hours of monitoring, the CPU had 4 hours of work and 4 hours of no work. It doesn't tell you that from 10 to 11 AM when the users complain the most, the system had 0% idle. Of course, graphing the data from the performance monitor would have shown exactly the problem, but that is seldom done and often the graphs are hard to read because they tend to zig-zag all over the place making observation of trends difficult if not impossible.

I think it's important to realize that users only complain when the system is slow, never when it's fast. So, what good does it do to average in all of the system fast time with the slow period you are trying to observe? It does no good at all except to skew your numbers and lead to false assessments.

Monitor with your Eyes and Ears

Making the problem of monitoring using averages more difficult is that fact that even if you monitor just for the 1 or 2 hour peak period during the day, it is difficult to determine how uniform the system response time is during that period. For example, I had one customer that had small bursts of 0% CPU Idle time during the peak periods. The average for the 2 hour period was 40% Idle time. However, the bursts of 0% Idle time would sometimes last as long as 10 to 20 seconds causing the users to complain. What performance monitor would show 10 to 20 second bursts of CPU activity as the source of the problem? And what would you do to solve this problem even if you knew it?

Well, that is exactly why it is important for the analyst to observe the system and observe the response time during the peak periods of the users who scream the loudest. That's kind of hard to do if you try to monitor the system remotely. So, if your analysis doesn't show the problem (or shows a nonexistent problem), get in there and watch the system with your eyes and listen to the complaints of the users -- they're a valuable part of the data that goes into making the correct performance analysis.

Use Peak Averages

To avoid the trap of using averages that tend to skew the performance data, I've come up with a technique that will show me the performance of the system when the system is slow. It is an attempt to answer the question; "When the system is slow, how slow is it?" Too many people make the mistake of asking the question; "When the system is running great, how bad is it?".

In order to accomplish this feat, we need to isolate the "slow" data captured only when the system is slow, average it together, and then compare it to the average for the day. If we see a big discrepancy between the peak average and the daily average, then we have a "Peak Load" problem that needs work as opposed to an "Out of gas" problem (much more serious).

In the early days, I used to collect the data using a cycle time of one minute, dump the data into a spreadsheet and sort the numbers from bad to good. Then I'd average the first 30, first 60, and first 90 to show me the 30, 60, and 90 minute peak averages. In this way, it was a piece of cake to see how bad the system is when it's running slow. I was now able to see what the worst 30, 60 and 90 minutes of the day looked like no matter when those minutes occurred. It was no longer necessary to determine *when* the peak periods of system usage were to analyze the performance of the system. This turns out to be a vastly superior way to measure system performance than either averaging the full days data, or trying to guess when all of the slow periods are.

After getting frustrated with the amount of labor and time lag involved in generating these 30, 60 and 90 minute peak averages, we decided to write our own performance monitor that can provide this kind of information for any system parameter in *real-time*. Much better than waiting till the end of the day to find out what's going on. Contact GDS if you're interested in trying it.

Tuning for Peaks = Overconfiguration

In the case mentioned above where the system was fast most of the time and slow only for small bursts during the day, it is possible that the system is under-configured for the peak periods and over-configured for non-peak periods. Of course, trying to tune the system to reclaim as much CPU and I/O as possible would be the first order of business. But, what happens when you complete all of your tuning and the users still complain, although based on daily statistics, you have plenty of capacity?

Well, if performance tuning doesn't yield the results you need, you really have three choices. The first choice is to let the users grin and bear it. The second choice is to try to spread the load during the day by instituting flex time, 2nd shifts, job scheduling, etc. The third choice is to "over-configure" the system to be able to handle those peak periods. Unfortunately, choice #3 is the selection most often made even though it results in over-capacity for most of the day. In fact, I see choice #3 made even before any serious attempts have been made at system tuning or any of the other options available. That's too bad. There's nothing wrong with over-configuring your system to handle the peak periods during the day -- it's just that it's probably the most expensive way to correct the problem.

Process Priorities

In the problem where we discussed seeing 0% CPU idle time for bursts of 10 to 20 seconds, it is imperative that during that period all users get some CPU time. In order for this to occur, you're going to have to start off with everyone at the same priority. Fancy priority schemes usually don't work when you need them the most -- once the system runs out of idle time.

Sometimes, I've found that users think that they've got everyone at priority 2 swappable but after getting in and taking a look at the PNQ's of the processes using PED, I'll find lots of processes dominating the system with much higher priorities. How does this happen? Well, there are processes that start off at swappable and change themselves to resident, thereby increasing their priority by 255 or so.

The other way processes become higher priority by accident is by getting started from the operators console where they assume the priority of the "master" CLI which comes from DG set at 1 rather than 2 like it used to be. I recently saw a case where a DG networking product was dominating the system and locking out character I/O for small bursts of time. The problem? The process didn't specify it's priority, picked up the default of 1 from the initial root CLI, then after the process started, it changed its process type to resident. Resident Priority=1 is the highest priority possible and exceeds even that of the Peripheral Manager (PMGR). Of course PMGR is responsible for all character I/O under AOS/VS and making another process higher than PMGR or even equal in priority is just plain dumb. Since PMGR was no longer the highest priority process on the system, it was easy to figure out why the customer was experiencing burstiness in character I/O.

Assuming that you've followed my advice and set everyone you can to the Swappable 2, the Heuristic Scheduler under AOS/VS should do the job. In case you forgot, the Heuristic Scheduler changes the Priority (actually the Process Enqueue Priority, or PNQ) to favor highly interactive processes over cpu bound processes. If everything goes as planned and the heuristic scheduler works as advertised, things should work out just the way you want as long as everybody is the same priority.

But what happens when you find that for some reason, the heuristic scheduler just doesn't seem to work out. What do you do? Well, first of all you must remember that process priorities are only important on a system that has no idle time. So, if your system has no idle time, should you fix the problem by messing around with priorities, or should you try to acquire some idle time one way or another?

The first thing you should do before you attempt to mess around with priorities is to tune the system in an attempt to make the system more efficient thereby increasing CPU idle time. If there is sufficient idle time, your priority problems are not an issue since everyone will get serviced in short order. If there is no idle time, and you feel the Heuristic Scheduler is doing a terrible job a deciding who is "highly interactive", you've now got to decide who you want to **steal** CPU time from. Remember, changing priorities is just like robbing Peter to pay Paul. Someone is going to suffer. That's why it's almost always better to let AOS/VS adjust the PNQ in real-time than to try to be "too smart" and come up with a convoluted priority scheme that is essentially static and unaware of the behavior of the process.

Heuristic Bugs

Unfortunately for us, DG's implementation of the heuristic scheduler is not perfect because they forgot to charge users for the time spent performing system calls. Not only did they forgot to charge the correct amount of CPU time, they decided that all system calls are free. So what happens if you've got a program that's performing zillions of system calls? Will the heuristic scheduler notice that the process is eating the system alive?

Nope, it won't have the foggiest notion that the process is misbehaving and thus will let the process gobble up the system without any punishment. Had DG charged the correct amount of CPU time for System Calls, the heuristic scheduler would be pretty much perfect and there'd be no need to mess around with priorities or anything else. In fact, it's possible that in Revision 2.00 of VS II that this problem will be corrected. As for VS Classic, Revision 7.67 does add in a little bit of CPU time for each system call but it's not perfect. For those of you who want to experiment with fixed charges for system calls, I've developed a patch that is supplied in the Appendix.

Rather than correct the problem at the source, a couple of years ago DG added a kludge to the scheduler so that they could selectively punish a process that performed various system calls. The name of this bit mask is TUNPBLK. In revisions of VS Classic prior to 7.67 DG had set the bit to cause PMGR Writes (Screen I/O) to counted as non-blocking events, thus causing a process that performs lots of these system calls to be punished.

Unfortunately, I've always maintained that programs such as Word Perfect by definition are "Highly Interactive" and to punish a user for typing in a word processor is both counter-productive and contrary to DG's published design of the heuristic scheduler. As you already know, users will not tolerate bursty response while typing in a document. In fact, sluggish response while typing is an often heard complaint and can give the impression that the whole system is slow when it's not.

Anyway, once DG charges the correct values for system calls, this kludge will no longer be necessary so it's probably not worth discussing any more. For those of you who like to "fine-tune" your system, I've included the definitions of TUNPBLK in the Appendix.

Idle Time: Good or Bad?

One of the myths I hear often is that Idle time is good. Actually, it depends why you have idle time. I recently tuned a system that had tons of idle time and was incredibly slow. The reason it had so much idle time was because the CPU was waiting for a resource and could do nothing but spin it's wheels while waiting.

And what did the customer think the solution was to the problem? A faster CPU of course. The effect there would have been merely to "Hurry up and wait" (as they say in the military) with no improvement to the problem.

In 90% of the systems that complain of poor performance that I have seen, they all had substantial quantities of CPU idle time. The cause of the idle time was almost always disk bottlenecks with huge queues. After correcting the disk problem, these systems often have much less idle time than they had prior to tuning. After all, why do you want your system to idle when there's work to do?

Of course, it's real easy to get rid of idle time. Just hire a few bad programmers and give them access to a DBMS, and you'll be out of CPU time quicker than you can say "4GL".

Despite all of the things that can lead to misinterpretation of Idle time, some people still want guidelines. Ok, here goes: If you've made sure that you have eliminated all potential disk and other resource bottlenecks that might cause Idle time, I'd say that you ought to have about 25% Idle during the busiest 1 hour of the day. Using the Peak Averages as discussed above will lead to a much more accurate number. Less than that and it's time for serious tuning or consideration of a faster CPU.

Memory: How much is Enough?

The myth regarding memory sizing is that you add enough memory until you no longer have any memory contention. This myth isn't even close to the correct answer.

The correct answer involves two different factors. The first factor is whether or not you are using a DBMS or product that uses INFOS. If you are, then you must have sufficient memory not only for your programs, but at least 4 megabytes of free memory for buffering of Database pages. AOS/VS will automatically put any excess memory to use for holding the most often used pages from your database thereby eliminating a lot of disk I/O and thereby increasing the performance of your system by a bunch. In essence, you must have excess memory on your system for this to take place.

The second factor involves the AOS/VS memory contention algorithm known as PFF. When VS detects memory contention (no more free pages), it runs a subroutine to remove seldom used pages from various processes. When PFF runs, it sometimes has the wonderful result of freeing up mucho megabytes of memory that can be used for holding LRU pages or other programs. The thing to watch for is to make sure that PFF only runs once or twice a day. If it runs more than that, you'd be wise to add more memory.

I've seen systems that have very little free memory for holding LRU pages from INFOS even though they've got lots of processes that are blocked or have lots of memory pages tied up for no reason whatsoever. Sometimes, they'll sit all day with just a few pages available and never run out which would have caused PFF to kick in. In these systems, causing PFF to kick in usually will create tons of free memory. It's too bad that there's no way to guarantee a minimum number of memory pages to be dedicated to buffering Shared Pages on the LRU chain. In systems where you need the extra memory for LRU pages, you can force PFF to run using a program that gobbles up all the free memory on your system. Programs like this are available on several bulletin boards.

So how can you tell if you if you'd benefit from more memory? Well, take a look at the number of pages on your LRU chain (LRUCN.W) and compare it to the total number of pages on the Free Block chain (FBLKC.W) . If LRUCN is big and FBLKC is little (less than 20) then you're using the LRU chain and you'd benefit from some excess memory. In systems that I configure, I usually try to make sure that I have at least 4000 pages on the LRUCN (8 Megabytes). Does it work? My INFOS customers say the system *blazes* (with 6500 DB pages in memory, it ought to blaze).

By the way, since the price of memory has dropped significantly, it no longer pays to spend any time messing around with the tuning parameters related to memory contention. If you've got memory contention, get rid of it immediately. No more discussion.

Finding Pigs

Once you begin tuning the system, one of the first things you want to do is to crank up PED and look for processes that are real porkers. Big Working Set Size (WSS), lots of CPU time, lots of I/O, lots of Page Faults, etc. will qualify a process as a true porker. Once you find the offending process, some judicious tuning may yield a dramatic difference in overall system performance.

Unfortunately, some processes have found a clever way to hide from programs like PED and CPUTIME. Because of the bug in the way AOS/VS charges processes for performing system calls (they're free!), these processes often go undetected and the wrong program gets put on a diet. Happens more frequently than you might think.

In order to trap these little piggies, I'd recommend installing the Patch I've included in the Appendix that causes a fixed charge to be assessed for every system call making these processes CPU time balloon like you wouldn't believe. Using this technique I've nailed one very popular program from a major vendor as the culprit causing poor performance on thousands of systems.

Another way to find programs that are killing the system with lots of system calls is to start up a performance monitor and then selectively block suspicious processes and watch what happens to statistics like # of System Calls performed, Idle Time, etc.

Disk I/O : Performance Enemy #1

Want to know the leading cause of poor performance on AOS/VS systems in the Western Hemisphere? Too many Disk I/O's on too few disk drives. In fact I'd bet that 95% of the systems out there have performance problems directly attributable to disk problems.

It's very easy to see if you've got disk I/O bottlenecks. Just crank up DISCO and look at the column labeled %Busy. If this number is less than 25-30% on all your disks, you're ok. If not, you've got work to do. As discussed above, you've got to perform this measurement correctly for it to be of any significance. For most people that means that <u>you must ZERO the statistics before you start recording</u>. Typing "Z" does it. I'm surprised by the number people who make the mistake of using cumulative system statistics that usually include their nightly backups into their calculation of disk balance and load.

The next step is to monitor the system during the peak period and watch what happens to %Busy. I'm often shocked at the number of people who think that the correct way to tune disks is by Total # of Requests. Absolutely Wrong.

In a previous column in DG Review, I mentioned that the only thing that balancing your disk load using Total Requests would do, is to guarantee that all your drives wore out at the same time and that people who tell you to tune this way probably also rotate their tires every 3000 miles (so they wear out evenly). Don't even look at Total Requests.

Separate System Disk: Myth?

One more performance tuning "rule" bites the dust. This time the concept of making a separate "special" system disk just doesn't make sense anymore and is obsolete. The rational used to be that if the system needed to perform I/O, all users would be affected, whereas if a user needed to use the disk only that user would be affected. Well, since we've already agreed that the system is not going to be performing anymore swapping or paging, what kind of "special" I/O's is the system going to be performing that might affect all the users on the system? I don't know of any -- everything's already in memory.

Therefore, what makes more sense is to balance your disks such that you process the maximum number of requests per second for your *entire* disk subsystem. Arbitrarily limiting one disk to less I/O than other disks will be detrimental to system performance since the total number of I/O's for the entire disk subsystem will be diminished.

It also makes sense to make sure that your system disk does not have any more than 25% Busy, just like the rest of your disks. Sometimes, it is necessary to move files off of the system disk onto other lesser utilized drives in order to accomplish this.

How many Disks?

The sole guideline I use to determine the number of disks is by observing the %Busy during peak periods. (I used to use Average Queue Length, but DG's messed this formula up and I haven't recalibrated my thinking yet). If you can't get all of your disks to stay below 25 to 30% during peak periods, you're going to need more disk drives to reach optimal performance.

Faster Disks

I've never seen bigger performance improvements from systems then after they put on high performance disks to replace or augment the slow disks on a system. Since the majority of systems I see are disk bound, it makes sense that if you purchase disks that are 200% to 300% faster than what you're currently using, you're going to see your elapsed times drop by the same margin.

Adding additional slow disks to a system in order to reduce %busy is not the best way to improve your system. Here's why: When a user needs to use a disk for an extended period of time, the response time for that job is going to be limited by the slow performance of the drive no matter how many drives are added to the system. In this case, the additional I/O capabilities of the extra drives will go unutilized.

To give you some idea on disk performance, I've observed the following average service times on these devices:

DG Argus	28 - 31 milliseconds
DG CSS	24-28 milliseconds
Zetaco SKS	17-20 milliseconds
Zetaco SKS-HP 660	11-14 milliseconds
DG 500 mbyte RAMS	11 -14 milliseconds

As you can see, running a job that was disk bound on a DG Argus drive would be about 2.5 to 3 times slower than running the same job on a DG RAMS or Zetaco SKS-HP 660. There's just no substitute for raw talent. (I think Lefty Driesell once said "You can't teach height").

Tuning Faster Disks

When most people add disks, they usually end up getting disks that are a lot faster then what they were using. As a replacement for existing drives, the performance tuning problem is easy -- just make all of your fast drives balanced by using %Busy the same way you did with your slow drives. Unfortunately, most people retain their old drives making the tuning methodology more complicated.

There are several philosophies to use when you've got drives of significantly different performance levels in the system:

- Keep them all balanced by %Busy.
- Work the Faster Drives Harder than the slower drives.
- Keep the Faster Drives less Busy to assure good response.
- Put "important" users on the faster drives.

I think that there's a certain amount of merit in all of these, however, I am an advocate of working the Faster Drives a little harder than the slower drives. The reason I believe this to be correct is that I want as many people as possible to be affected by the fast drives whenever possible while minimizing the amount of time on the slower drives.

Remember, what we're looking for is the minimum amount of time spent processing all of the disk I/O requests. For those of you who want to whip out your slide-rule and calculate it, you would take the number of requests per unit and multiply it the average response time for the unit, then add all these numbers together to come up with total disk time per hour, day, etc. Dividing this number by the total number of accesses for all disks will yield the average service time for the entire disk subsystem. You can always tune your system this way if you're the scientific type of individual.

Disk Optimizers

Do they work? Well, it depends on how bad things are currently. For example, if you've got a DG Argus drive and are already seeing average service times of less than the numbers indicated in the chart above (less than 28 milliseconds), then I'd seriously doubt that you're going to get the results that make it worth your while. If however you're seeing much worse numbers, it'll probably improve your performance somewhat. If you're looking for one of those big gains of 250% like I described above with Fast Disks, you're looking in the wrong spot. At most you should expect a 15 to 25% improvement if you're like most of the customers I've seen. For some people, that's all they need and they're happy as clams. For others, they should look elsewhere for a big performance boost.

For optimizers to work the best, you've got to be able to predict with some degree of certainty what files get accessed the most and then move these files to the "Hot Spot" on the disk. If you do it right, you can reduce the average seek distance of your drive and thus reduce the average service time for that device. Obviously, a big database is not a good candidate for disk optimization. I also believe that the slower the disk, the better the chances that a disk optimizer will make any measurable difference.

One of the problems that appears to be solved by the latest offering from Eagle Software is that of identifying what files have the most I/O activity automatically and then optimizing based upon these statistics. It certainly has some merit for customers who have a high number of accesses spread of relatively few disk blocks.

The other way that disk optimizers can improve system performance is by placing all directory information an a contiguous area for a single directory. That makes operations like "List Files" go like crazy and also speeds up incremental backups by making the directory scanning go a lot faster. I believe that users probably do notice a big difference after using a disk optimizer because of the above mentioned functions, even though the total performance gain for the entire disk subsystem may be small.

The disk optimizer from DMS Systems offers another feature that may have some merit in the area of eliminating I/O. Their DiskOpt product has the ability to "tune" directory hashframe sizes and file element sizes while performing the optimization. Of course, if you had set your element sizes correctly in the beginning, this wouldn't be necessary. But unfortunately, most applications do a pretty bad job at setting element sizes correctly.

One area that I feel has real potential for disk optimizers is in the AOS/VS II world. Due to the advent of Data Caching under VS II it would be possible for a well designed optimizer to put a whole bunch of related information contiguously on the disk and when VS II's Data Cache reads ahead 16 or 32 blocks, it's going to get whole bunches of the information it's going to need, thus eliminating Disk I/O. We'll have to wait to see if these VS II optimizers do it the right way thought before we jump to any conclusions.

Overcrowded Directories

After all these years of hearing that you're not supposed to use the AOS/VS filing system as a replacement for a DBMS, people do it anyway. In fact, it's not just users that make this mistake, I've seen products on the market that cause you to overload directories with files -- Some of these from people who ought to know better.

I would bet that 90% of the systems that I tune that have performance problems will also have heavily accessed directories that contain thousands of files. Or worse yet, they'll have their searchlist set to scan these overcrowded directories.

The performance penalties of such improper behavior are severe. Just for fun, I found a customer that had 2 directories each with 3000 files. Going in to the first overcrowded directory , I typed F/TYPE=177 knowing the system wouldn't find any files that matched this criteria. Doing this command caused AOS/VS to flush all of it's cache buffers for all other directories. Then, I went into the second overcrowded directory and typed "CREATE JUNK". 9 Seconds later, I received my prompt back. To add further insult to injury, my process was only charged a few milliseconds for this activity even though it brought the system to it's knees.

To sum it up: DON'T OVERCROWD DIRECTORIES.

Element Sizes

Increasing the element size of a file that's going to grow to be fairly large will cause an enormous gain in performance (especially under AOS/VS II). Whenever possible, specify an element size that will cause the file to fit under the following criteria:

- Less than 128 Elements (Single Index Level)
- Wastes only a reasonable amount of space.
- Contiguous for Read Only files.

If you want to see the effect of the proper element size, try the following test:

```
CREATE/EL=32 JUNK
COPY/IMTR=32768/OMTR=32768/A  JUNK  :UTIL:PARU.32.SR
```

then time the following command:

```
COPY  JUNK2      :UTIL:PARU.32.SR
```

See any difference?

Hashframe Sizes for Directories

Not a lot to be gained here unless you're way off. As a crude way to get the hashframe size close, you should divide the number of files in the directory by 20 and use the result of the hashframe size. Utilities are also available to set this number more precisely, although you probably won't notice the difference.

One area that you can probably get a little bit of gain would be to set the hashframe size of any directory that contains just a few files (like less than 10) to 1. Doing that will improve the quality of the entries in the AOS/VS cache and yield a small (theoretical) performance gain.

Of course if you limited your directories to a reasonable number of files, then you'd never have to waste any time messing around with Hashframe sizes looking for minuscule performance gains.

Sizing the AOS/VS Cache

The AOS/VS cache contains directory information for the files on disk that were accessed most recently. Setting the cache large enough to hold a lot of directory information will eliminate I/O at the cost of memory and a tiny bit of CPU. I've looked at the question of how much CPU time is utilized by searching a large cache and found that in most cases it's an insignificant amount (less than 1 millisecond to search the entire cache). I've included my analysis below.

Cache Search Time Calculations

	Cache Size				
	<u>512</u>	<u>768</u>	<u>1024</u>	<u>1536</u>	<u>2048</u>
Processor type					
MV/10000	0.217	0.324	0.432	0.647	0.862
MV/15 Mod 8	0.132	0.198	0.263	0.393	0.524
MV/15 Mod 10	0.132	0.198	0.263	0.393	0.524
MV/15 Mod 20	0.132	0.198	0.263	0.393	0.524

Notes: (1) All timings given in Milliseconds
(2) Worst Case Time required to search entire chain
(3) MV/15 times from Eclipse MV/15000 Principles of Operations Supplement (December 1986)

Therefore, my recommendation is to set the number of buffers in cache quite large in the hope of eliminating some disk I/O.

DG's guidelines for setting the cache size seem to be too conservative for best performance. DG recommends that you look for a cache efficiency of 90 to 95%. I'd recommend that instead of looking at cache efficiency, you look at the total I/O load caused by Cache Misses. This is accomplished by dividing the total cache misses by the sum of (total Read requests + total Write requests). If the resultant number exceeds 15 to 20% you've got problems (a significant portion of your I/O is due to cache misses) and ought to try increasing your cache buffers.

Applications Tuning

Probably the biggest performance gains come from relatively simple changes to applications. In the areas of I/O, the big gains come from modifying the programs to read and write (especially on writes) large chunks as opposed to little tiny reads. Performance gains in the 500% range usually occur when a disk bound application that writes and extends a new file to disk is modified to use larger buffers and less System operations. For most programs, this means you should try to do your I/O in 8K or 16K chunks as opposed to 2K chunks.

It's easy to tell if your applications are efficient. Simply start up DISCO and divide the number of (reads+writes) by total requests. This calculates average blocks per I/O. If the number is less than 2.5 blocks/requests then you can make some big performance improvements by increasing the I/O size of your application.

If your application is performing Agent I/O, it is a piece of cake to tell the Agent that you'd like to use bigger buffers. Just set the ?IMRS word in the ?OPEN packet and you're done. While you're at it, make sure that you've bumped the element size of the files you're reading and writing to as well.

If you're not doing Agent I/O, you'll have to modify the program to use bigger buffers internally. Not too difficult for most people. If you're not sure how to get your application to perform bigger I/O operations, contact a performance consultant to show you.

Under VS II, the same tuning philosophy is also true, but it's hundred times more important. If you're thinking of going to VS II, better take a look at those ratios and do a little fine tuning of your application before you get the shock of your life (poor performance).

IAC Tuning

Mostly a waste of time. The only thing that I've seen be effective at eliminating choppy IAC performance is moving the printers to their own IAC. The theory is that printers don't complain as much as people.

Tuning AOS/VS II (Extracted from Part I, DG Review May 1990)

I keep running into users who want to switch to AOS/VS II but are afraid that their system performance is going to go down the toilet and that they'll be left with no alternative but tossing VS II or living with mediocre performance. Other users who bought their systems with VS II to begin with have no basis for comparison with a VS Classic system and therefore usually don't know that their system could be running a lot better than it is.

Anyway, I've spent a little time watching how VS II behaves as compared to VS Classic and as a result have been able to see why many VS II systems don't perform up to expectations. I started wondering what percentage of VS II systems are running at their full potential (probably none). The good news is, if you've got a VS II system or are thinking about going to VS II you ought to know that I've been able to get huge performance improvements from VS II with relatively minor amounts of tuning.

Fault Tolerant Blues:

There's no shortage of criticism and opinions describing how slow VS II is (although I did write a column saying it's not all that bad if your programs are well written). While I'm sure a lot of the complaints are true, I think it's important to understand where all of the slowness in the new file system comes from so that the proper tuning strategy can be devised.

The first thing I noticed about some VS II systems is that lots of the numbers from the DISCO screen look way out of whack compared to good numbers that I've grown accustomed to from VS Classic. The ratio of Reads to Writes that's supposed to be around 2:1 now looks reversed. The average number of Blocks per Write is hovering around 1.20 giving it the appearance of heavy system I/O rather user I/O. And, the ratio of System Writes to User Writes is way out of balance at 4 System Writes per 1 User Write. As would be expected with this much system I/O being performed, the percent of System CPU time is running close to 50% vs. 30-35% for VS Classic. No big surprise there.

By now I think everyone realizes that DG's implementation of fault tolerance comes with the price of substantially increased levels of system I/O for certain functions. But, which functions really cause your MV to go in the dumper is something that requires a little bit of understanding of the design of the New File System (NewFS -- Please stop calling it NFS!).

Paranoia = No buffering

The NewFS keeps filenames apart from file information. Filenames are kept in the directory with a pointers to the file information which is now stored in the File Information Table (or FIT). Unlike VS Classic, there is only one FIT for the entire system and it is created when you setup your LDU. In addition, VS II directories now only contain filenames, so their size appears to be smaller although this is just an illusion. Unfortunately, the directory size does not indicate the amount of disk space allocated out of the FIT database on behalf of your directory.

Whenever any directory information needs to be updated, VS II does not merely update the information in a buffer in the file system cache and then flush it out to disk when it's good and ready -- Instead, it assumes the worst will happen and with maximum paranoia enabled, proceeds to write every single directory update back to disk every time something gets changed causing lots of extra unnecessary I/O.

Unnecessary you say? Well, that's a debatable point. DG's philosophy with VS II is that there will never be any corruption in the file system due to a crash no matter how many extra I/O's it takes to make that happen. My philosophy is that when a system crash occurs, many of the files being written to are incomplete and are therefore corrupt anyway and probably useless -- therefore, why bother hammering the disk updating the directory and index information while a file is still growing? Other users who are updating their database files at the time of a crash are not going to be helped by VS II's fault tolerance either and are probably going to have to run some kind of Database Fixup. So the idea of "just bring the system right back up under VS II" seems idealistic until all applications running on the machine also become fault tolerant.

A compromise that would allow limited buffering with periodic flushing of heavily used index and directory blocks would probably be acceptable to most users. If this enhancement were made, DG would probably allow users to select their paranoia level during VSGEN or LDU definition. Maybe the question would appear something like:

"Data Loss Acceptability Level: 0 - 3: [1] "

where:

0 = Very Paranoid - Write Every Disk block twice then verify.
1 = No tolerance - Moderate Paranoia - (Current VS II).
2 = Somewhat Tolerant - (Like AOS/VS Classic).
3 = Don't really care - (Use RAM Disk with one flush per hour)

I guess that I'd probably select 2 most of the time and 3 for "those special occasions".

The Big Hit: Extending Files

After analyzing the performance data from some VS II systems I found that the thing that really takes the biggest performance hit for most people is writing to a new file that's growing in size. Programs that generate output files with small element sizes have shown that their performance is much worse under VS II than VS classic.

Here's why; Under AOS/VS Classic, when a new file is being written to and is in the process of growing, the File Information is held in a buffer in memory and not flushed to disk until the file is closed. The index block that contains the pointers to each element of the file are also held in memory (if you use AOS/VS 7.67 or BJ's patch) until the file is closed or the file grows beyond a single level index. So in essence under VS Classic, you get ZERO system writes for every user write to extend the file.

Well, under VS II that kind of buffering isn't allowed so you end up getting at least two writes by VS II for each user write. One write for the File Information Table to update the EOF (and related items) and one write for the index block. Maybe even another one for the bit map depending on the fragmentation of your disk space (I'm sure to find a few more system writes once DG releases a VS II File System Internals Manual).

The point is, it all adds up and pretty soon you're in an environment where the system is doing far more I/O's than you are.

Secret of VS II Performance: No Writes to Disk allowed.

The best way I've figured out avoiding the big performance hit you take in VS II is so simple. Just stop writing to the disk. That means no Creates,no Deletes, no Renames, no Writes, no Anything. Of course it's kind of hard to get any work done under those rules.

So what do you do? Well, the first thing I did was some of the same things that I'd have done if I was performance tuning VS Classic. I analyzed the applications to see where the disk I/O was coming from looking to see if the programs where not careful about things like element sizing, buffering, etc. The stuff we're talking about here should sound familiar to all you VS Classic people. The difference now is that under VS II, it's suddenly a million times more important to pay attention.

Figure it this way: If you reduce the number of times VS II has to extend your file by a factor of 4 by increasing your element size and using bigger buffers, you're going to reduce the number of I/O requests by 75%. The reason is because with every 4 user writes we would have 8 System I/O's associated with it for a grand total of 12. Increasing the size of the write and the element size by a factor of four would allow the same operation to occur in just 3 I/O's.

I'm sure someone genius out there is going to "fine tune" my example here, but it doesn't really matter as long as you understand that you need to reduce the number of times you ask VS II to increase the size of your file.

Dumb Programs and ?IMRS:

If you took any well written program (written by myself of course) and compared it's performance under VS II to VS Classic, you'd probably find that it doesn't take the performance hit that lots of poorly written applications are going to see. The reason is that good programmers always make it a habit to set their element sizes correctly when they create files and always do **gargantuan**, **man sized** reads and writes. (Another reason is that DG hackers as a rule, never take the "Default Value of -1" anywhere in their programs.)

But there's a whole bunch of wimpy software out there that just takes the system default for everything and usually ends up doing itsy-bitsy wimp sized reads and writes. These programs are the ones that are giving VS II a bad image. Get them out of here.

So what if you've got some of these wimpy programs to run under VS II -- What should you do? Well, the first thing is to make sure that all of the programs that write large output files are doing big writes and if they're using ?WRITE rather than ?WRB, you should check to make sure that you've told the Agent to buffer up some data before hitting the disk. The way you do this is to set the word at offset **?IMRS** in the ?OPEN packet to something other than -1 or 2048. Try 8192 for starters. And while you're at it, don't forget to set your element size to be something larger than 4 for a file that's going to grow quite a bit.

Unfortunately, both of these items I've mentioned above require the services of a programmer and changes to the software. What do you do if you can't make any software changes? As an alternative, you might try setting the VS II Secondary Default Element size up to 8 from 4. Keeping the Initial Element Size set to 4 (or smaller) makes sense for most people who have lots of files that are under 2K bytes and don't want to waste space. It's almost always better to set the element size yourself on a case by case basis than to change the system default, but sometimes you don't have that option.

There's also a special patch to the Agent that can be used in special cases where your software is using the default value for ?IMRS using Agent I/O (?READ/?WRITE) and you'd like it to use bigger buffers for I/O but don't have the ability to change the software for some unknown reason. In these cases, you can change the Agent's default buffer size as a last resort. The reason it's a last resort is because it can have adverse effects on some programs that do random I/O, causing them to read/write more than they really need.

To avoid potential problems, I'm not going to publish the Agent Default Buffer Size patch here. Instead, if you think that you'd benefit from the patch give me a call and I'd be glad to explain it to you.

Appendix

PH's Tuning Patches

March 1990, Copyright GDS, Inc.

Disclaimer: The following patches to AOS/VS are unofficial patches and are not supported by Data General or General Data Systems, Inc. Use them at your own discretion.

Trick #1 - Priority tuning on Process Blocking using Discrimination Tuning Word

Explanation: The heuristic scheduler in AOS/VS has the ability to discriminate against a process that performs certain system calls. Currently, AOS/VS is set to discriminate against processes that are performing PMGR writes (console I/O). Discrimination is performed by lowering the process's PNQ (priority) by 1. This is probably not a good idea in a word processing environment.

When a process is blocked, its tuning word is compared to the system tuning word stored in offset TUNPBLK. The bits of TUNPBLK are defined as follows:

Bit 0 - PMGR READ BIT
Bit 1 - PMGR WRITE BIT
Bit 2 - ?SIGNL BIT
Bit 3 - ?IPC BIT
Bit 4 - ?DELAY BIT

The current value (prior to 7.67) of TUNPBLK is 40000K (bit 1 =PMGR WRITES). At 7.67 and beyond, the new value is 74000K. In order to not punish PMGR writes and leave the other bits on, the new value would be 34000K.

If you are in an office automation environment, it is advisable to not let AOS/VS penalize users for interacting with the system. In order to clear TUNPBLK use the following patch to your system file.

```
; Patch #1 - Clear TUNPBLK to not discriminate on PMGR writes.
;  PHH - January 1989 (for Pre 7.67 Systems).
%AOSVS
TUNPBLK            [40000]            [0]
;
; Note: For 7.67, change the value from 74000 to 34000.
; Other values may be acceptable, please read DG Review May 1990.
;
; End of patch
```

Patch #2 - Reset the number of buffers in cache without SYSGENing.

Explanation: A quick way to change the default number of buffers in cache is to simply patch the value in the system image without going through the process of SYSGENing and patching the system.

```
; Patch 2 - Set the number of buffers in cache via the patch utility

%AOSVS
PCACH          [old # buffers]       [new # buffers]

; End of patch
```

Patch #3 - Increasing the maximum number of allowable buffers in cache

Explanation: For many large MV systems that have high user counts, the maximum number of buffers in cache is too small to minimize disk I/O. Although the limit was recently raised to 2048 (from 1024) buffers in revision 7.62, this may still be too small for some systems. The following patch will reset the maximum number to be 4096.

```
; Patch #3 - Reset maximum number of buffers in cache for pre 7.60 systems.
; PHH - January 1989
%AOSVS
BFALL+133      [1024.]          [4096.]

; End of patch

; Patch #3A - Reset maximum number of buffers in cache for post 7.60 systems.
; PHH - January 1989
%AOSVS
BFALL+133      [2048.]          [4096.]

; End of patch
```

Patch #4 - Improving responsiveness of AOS/VS by reducing the Subslice.

Explanation: The current setting for the system subslice is the same for all systems regardless of the systems ability to process many users per second. Faster MV's such as the MV/20000 and MV/40000 can handle many users each second. The current subslice of 32 milliseconds is too long to avoid uneven or "bursty" response on interactive systems with large user counts. Reducing the subslice to 20 milliseconds will greatly reduce this burstiness. This patch should only be applied to systems that fall into the above categories.

```
; Patch #4 - Reduce subslice to 20 milliseconds.

%AOSVS
SUBSL          [-320.]          [-200.]
PSUBSL         [320.]           [200.]
;
; PSUBSL is required for Rev 7.50 and later systems.
;
; End of Patch
```

Patch #5 - Charge User Fixed Price for System Calls.

Explanation: Prior to Revivision 7.67, AOS/VS did not charge the user for CPU time used performing a system call. This leads to both scheduling problems as well as making processes that perform lots of system calls "invisible" to the performance analyst. See DG Review September 1989 for more information.

```
; Patch #5 - Charge Fixed prices for system calls
%AOSVS
;
CALLS+14       [NADDI 10. , 0]        [NLDAI 10. ,0]
^+1            []                     []
CALLS+22       [NADDI 90. ,0]         [NLDAI 100. ,0]
^+1            []                     []
^+1            [XNSTA 0 152 2]        [ LJMP #SPATCH]
^+1            []                     []
^+1            @.                     []

#SPATCH        0                      [XWLDA 1 42 2]
^+1            0                      []
^+1            0                      [WADD 0 1]
^+1            0                      [XWSTA 1 42 2]
^+1            0                      []
^+1            0                      [XNADD 0 152 2]
; Continued on next page
```

```
;
^+1             0                 []
^+1             0                 [ XNSTA 0 152 2]
^+1             0                 [ ]
^+1             0                 [XNLDA 1 PSUBSL]
^+1             0                 []
^+1             0                 {LJMP CALLS+30}
^+1             0                 []
^+1             0                 []
;
SPATCH          #SPATCH\          #SPATCH+14.\
; End of patch
```

Patch #6 - Stop low level index block flushing while extending a file.

Explanation: The following patch developed by Brian Johnson allows the system to allocate elements in a file without forcing the lowest level index block to be rewritten to disk on each allocation. The net effect is to eliminate a large number of I/O's and head movements, especially noticeable during disk to disk file moves or copies. DG's philosophy was to minimize data loss in the advent of a crash. However, the output file that was being written at the time of the crash was probably incomplete anyway so there was no reason to force flush the index blocks while the file copy was in progress.

DG will be supplying this patch as of 7.67.

```
; For AOS/VS systems prior to 7.67
%AOSVS
RUNLC1+3605     [RELF]            [RELM]

; End of patch
```

Patch #7 - Set Agent Default Buffer Size to reduce I/O Requests

Explanation: Many programs perform I/O using ?READ and ?WRITE which go through the Agent without specifying the size of the I/O the Agent should perform. Since the default size of I/O through the Agent is only 2048 bytes, programs that do a lot of reading and writing to big files will suffer. Although the first choice is to modify the program to perform larger reads/writes, this is not always possible. In these cases, it is possible to change the default Agent buffer size, realizing of course that the performance of programs that perform Random I/O will suffer. Please Use Discretion.

```
; Patch to AGENT.PR (For VS Classic or VS II)

%USER
.DEFBUFSZ       [2048.]           [4096.]
;
; Note: This variable  can be set to any multiple of 1024 up to 32768.  Use Caution.
```