Making your HP GlancePlus Pak perform

by Doug Grumann
Hewlett-Packard Performance Technology Center

The HP performance products GlancePlus and the MeasureWare Agent have proven over the years to be very useful to system administrators trying to monitor, manage, and troubleshoot performance issues. Many people have GlancePlus, the MeasureWare Agent, or the GlancePlus Pak on their systems but aren't aware of all the different capabilities of the tools, what all the performance metrics mean, or what user interface works best in a given situation. This article is not meant as an introduction to users who are not at all familiar with the tools. Rather, I'll assume some familiarity with the tools and look at their configuration, use, and best practices, with an emphasis on how you can customize them.

Besides the GlancePlus and MeasureWare Agent, the Vantage Point suite of resource and performance management products also includes the Service Reporter and PerfView which provide graphical reporting capabilities for MeasureWare performance data, as well as various Smart Plug Ins for additional monitoring capability of common applications. Look under the OpenView web page at http://www.openview.hp.com for information about the entire suite of performance management software. The focus of this paper will be on GlancePlus and the MeasureWare Agent for HP-UX, although much of the discussion can also be applied to the use of these products on their multivendor versions.

The GlancePlus Pak product is simply a combination of the two products GlancePlus and MeasureWare Agent, which are also available individually. In the Pak, we've also included integration with Vantage Point Operations for event monitoring and configuration. Besides integration of MeasureWare alarms along with system events in the Operations event browser, there are also actions provided to conveniently edit config files.

MeasureWare and GlancePlus on HP-UX are delivered via Software Distributor bundles named by product number. The bundles point to SD-products that in turn point to the filesets that contain the files. Once you have the products installed on your system, you'll see the product files under the directory /opt/perf, with the executables in /opt/perf/bin. Since /opt/perf/bin is added to the system PATH after installation, I'll just refer to the executables henceforth without the full path. The release notes are under /opt/perf/ReleaseNotes. The configuration files that we'll be playing with, as well as other files created during and after installation, go under the /var/opt/perf directory. New default config files (like alarm definitions) are installed into the /opt/perf/newconfig directory, and at the end of installation they are conditionally copied into /var/opt/perf if there are not already config files there (from a previous installation). This prevents your customizations to the /var/opt/perf config files from being overridden when you update to

a new release. If you want the new default configuration files, remove them before swinstalling the new version, or simply copy them over from newconfig afterwards. In recent releases, we've put some supplementary example files under /opt/perf/examples. I'll refer to a few of these below. In addition to the hardcopy manuals and man-pages available, we include electronic versions of documentation in directories under /opt/perf/paperdocs.

Glance in brief – "don't blink"

Here's a quick overview of GlancePlus: There are two user interfaces - *glance* and *gpm*. Both interfaces provide access to the same performance measurement data. The gpm interface is a X-window Motif interface that is easier to navigate but takes a bit longer to start up. The glance character-mode interface is useful when you aren't using an X-display or you want to run Glance remotely over a slow network connection. I prefer to use glance when I just want a quick look at a system, and I use gpm when I want to do more in-depth analysis. Because of the greater flexibility of the Motif interface, gpm gives you access to some detailed metrics not available in character-mode glance.

When you start gpm, it will bring up the Main window along with any other windows you had open the last time you ran it. In the Main window, you'll see information about the update intervals as well as summary graphs of some of the most important global performance data (CPU, memory, disk, and network activity). From there, you can use menu selections to bring up detail Report windows to show more information about various aspects of system performance. We'll go deeper into them later. Another button on the main window shows the status of any current performance alarms, which are configured using the glance "Adviser" functionality. The adviser is a useful way to access specific metrics and alarm on them using a script-like language (more on that below!). You can adjust gpm's fonts, update interval, icons and other properties from the Main window menu. From this and any window, you also can get into gpm's online help facility. There's a lot of useful information contained in the online User's Guide. Finally, there's the "?" button which you'll see on every gpm window. That gets you directly to the on-item help for the windows or metrics in the display. There are over a thousand different performance metrics being collected by Glance, all of which may be useful to you at some point. The Choose Metrics capability of every list window lets you customize which metrics are in the report, and you can define alternate sort columns and filters to highlight the most interesting data. To avoid getting lost in the complexity of all the different gpm reports and metrics, concentrate on the areas of performance most important to you, or to which the metrics displayed in the gpm Main window guides you. Iconify gpm and you'll see the CPU, memory, disk, and network activity continue to update live inside the gpm icon!

The character-mode glance display also shows the main areas of performance in the "bar graphs" at the top of the display. In the bottom of glance's initial screen is the Process List, which gpm puts into a separate window. You can use terminal function keys to

navigate through glance's various report screens, or you can use shortcut letters to go between screens. If you type "?" inside glance, it will go to the Commands Menu screen which lists all the screens available. If a list such as the Process List is longer than the terminal window dimensions allow, you can use the Spacebar to scroll forward through the entire list and the "-" key to move back. The "h" key will get you into the character-mode online help facility (which by necessity has less content than gpm's online help). Although glance has access to all the same performance metrics as gpm, there is not enough space in the screens to display them all, but you can still access them all with the adviser functionality.

Whichever user interface you choose, the way you use Glance is often the same: You take a quick look at the global performance indicators to see what general areas might be affecting performance, and then you drill down into that area. At some point, you'll often end up looking at the Process List to focus in on specific activities that are at the heart of the issue you're trying to resolve. From a specific process perspective, you can drill further if needed into its open files, memory regions, wait states, and threads (for HP-UX 11 releases). In general, Glance is very good at answering "what's going on right now."

MeasureWare in brief – "Moi? No: MWA"

The MeasureWare Agent product (MWA) is structurally more complex than Glance, as those who use it soon realize. MeasureWare is looking at the same performance data as Glance (actually, a subset), but it is logging the information for future reference, using the metrics to trigger alarms, and passing alarm information as well as performance data up into DCE/RPC daemons that make it available to analysis products such as ITO/VPO, Service Reporter and PerfView. The mechanisms are somewhat complex, but looking at the big picture, it's essentially a set of daemon processes (which get started and stopped using the appropriately named "mwa" script) that run in the background to collect, log, and process the performance metrics.

At the heart of MWA is scopeux (also known as "Scope"), which is the daemon that obtains the metrics for use by the rest of MWA. Scope uses the same measurement interfaces as glance and gpm do. Scope is collecting data continuously and logging process data every minute, and other (global, application, disk, network, and transaction) data every five minutes. Scope writes the performance metrics in logfiles that are read by the other MWA daemons. The perfstat script provides a good way to check on these daemons to make sure they are all up. For example, if you misconfigure an alarm definition in the alarmdef file, the alarmgen daemon will exit with an error and the perfstat script will highlight that alarmgen is no longer running. All the daemons write status messages to files under /var/opt/perf, so to diagnose alarmgen problems you'd look in the status.alarmgen file. If you are not relying on other analysis products to view MWA data, you'll want to become familiar with the extract program that you can run to extract and export data from the scope logfiles. Using report files that specify to the extract program which metrics to export, you can customize the output to suit your needs.

Whereas Glance only runs when you invoke it, MeasureWare is designed to run continuously in the background. It starts up automatically at system boot time (unless you turn it off via the /etc/rc.config.d/mwa file). If you do nothing with the data, MWA will continue to collect and log the data forever, but don't worry about running out of disk space because Scope will automatically delete old data from its logs once they reach a maximum size. MWA complements Glance in that it's good at answering questions like "what went on earlier today and has it been happening all this month?".

CPU Scenario: "Some hogs don't live on farms"

Let's start by looking at a simple common performance problem. Say your users are complaining about a server system running slower than it should. You can rlogin and run glance to see if it's obvious what the problem is. In this scenario, let's say glance's main screen shows the CPU Util bar filled up with activity. You would also see the current CPU percentage column at the right pegged at 100%. If you have the default process filters set up (the "o" screen in glance), the Process List will be sorted with the highest CPU user at the top. In this case, let's say you see the process "imahog" listed first, showing CPU utilization of 99%. Note that whenever two metrics in a character-mode glance screen are separated by a slash, it's showing the current value followed by the cumulative average value for that metric since glance was started. Right from the Process List, you can see the username that is running the imahog process. If you then go into the Process Resource screen, you'll see lots more information about the process, including what kind of CPU time it is consuming. You get to this screen by typing "s" and then typing in the imahog process ID (pid), or by just typing ">". Additional process-specific metrics are available from there in the different drilldown screens. Some of this information can be very interesting to developers looking at how their programs are spending their time and what resources they are consuming, but in this simple example you're just trying to improve the system response time. At this point, you could renice the imahog process to change its scheduling priority, allowing other higher priority jobs to get more CPU time, or you could just kill it (assuming you logged in as superuser). Your best bet might be to check with the user who started the imahog process first!

If you had decided to use gpm instead of glance, you would have done the same rlogin and made sure the DISPLAY variable was set correctly. Once you started gpm, you'd see the global activity in the CPU window, and calling up the Process List would have shown the imahog process chewing up all the CPU. Manipulating gpm's Process List with its filtering and sorting capabilities can be fun, as there are many opportunities for customization. You could set it up so that only processes using more than 10% CPU or doing more than 10 disk I/Os per second are listed. You could highlight processes that are using more than 50% CPU. You could sort by username to see what the different users are running. Another useful option in any of gpm's row/column textual reports like the Process List is that you can invoke the Choose Metrics screen to customize what fields are shown in the list. If your system is often subject to disk bottlenecks, you may

want to pick metrics like PROC_DISK_LOGL_IO_RATE and PROC_IO_BYTE_RATE to add to the Process List. If you are bedazzled by the sheer number of metrics and want to access the definitions, this is easy in gpm using the "?" button. Click on "?" and then click on a metric (for example, in the Choose Metrics window), and the metric definition will pop up. You'll notice some of the naming conventions (system-wide "global" metrics start with GBL_, process-specific metrics start with PROC_, etc). You can re-arrange the columns in gpm reports for easier viewing, too. Your changes will be automatically stored in binary-format user-specific and system-specific configuration files in your home directory ("`ls ~/.g??*`" will show them). This means that next time you run gpm on the same system you'll see all the customizations you had made in your last session.

What if the imahog process usually does useful work and you want to do a little research into when it started using excessive CPU (perhaps looping)? This is where MWA can come in handy. If you have PerfView available, it would be easy to connect to the system and look at the history graphs, perhaps drilling down into the process data. Not depending on a separate analysis tool, you could instead create a quick report template for MWA's extract to dump out the data. Try this: Copy the reptall file from /var/opt/perf into /tmp. Then edit the /tmp/reptall file, scrolling down to where you see the GLOBAL metrics listed. Remove the asterisk from in front of the DATE, TIME, and GBL_CPU_TOTAL_UTIL metrics. Then go down to where you see the PROCESS metrics listed. Remove the asterisks from in front of the DATE, TIME, PROC_PROC_ID, PROC_PROC_NAME, and PROC_CPU_TOTAL_UTIL metrics as well. Save the changed file and run the following command:

```
extract -xp -v -gp -r /tmp/reptall -b today
```

When finished, extract will have created two files in your current directory, xfrdGLOBAL.asc and xfrdPROCESS.asc, containing the metrics you requested. Both files will show data starting from midnight last night. The Global output might show you that the system was moderately busy starting at 07:00 but that CPU percentage was near 100% starting at 13:00. Looking at the Process report output, you'll see all the processes that MWA considered "interesting" (more on that later). You can look for the imahog process records and perhaps see that at about 13:00 it started chewing up excessive amounts of CPU. Looking at what other processes were interesting around that same time period might help determine what caused imahog to spin… or not! Performance analysis, no matter how useful the tools, will always remain somewhat of an art.

This simple example touches on some of the capabilities of the tools. You can keep sets of customized MWA extract export report files around to help you with specific tasks. For example, you might want to have a report file which concentrates on displaying CPU data as above, and another that reports on disk data. By looking periodically at Glance and MWA data on your important systems during times when they are running smoothly, you'll get a good idea of what to expect. On busy systems, GBL_CPU_TOTAL_UTIL might not be very useful because it's always near 100%. In that case, it might be good to watch GBL_PRI_QUEUE to monitor how many processes are typically waiting on CPU time. If the average number of processes blocked on Priority gets abnormally high, users will experience poorer response time.

Perhaps you solved this particular problem, but you want to be more proactive about runaway processes on this system for next time. One option might be to keep a glance or gpm session active on your server so that you can keep an eye on the global metrics. Some administrators keep a bunch of gpm windows iconified on their workstation (sometimes I call this "Poor Man's PerfView").. the icons will blink red when an adviser bottleneck alarm goes off. MWA serves this function even better as bottleneck alarms will be sent to PerfView or ITO/VPO if you have them configured (refer to the MWA User's Guide). If you want to set up a special configuration inside MWA to send yourself mail if processes start looping again in the future, you could do the following: Copy the /var/opt/perf/alarmdef file into /tmp. Edit /tmp/alarmdef and append the following syntax to it:

```
hogpid = hogpid
PROCESS LOOP
{
  # Send mail to sysadmin when processes are hogging the CPU now
  # and have accumulated over 1000 seconds cpu time total.
  # Avoid processes with pids < 100 assuming they're system processes
  # and they know what they're doing.
  if (PROC_CPU_TOTAL_UTIL > 95) and
     (PROC_CPU_TOTAL_TIME_CUM > 1000) and
     (PROC_PROC_ID > 100) and
     (PROC_PROC_ID != hogpid) then
  {
    exec "echo 'runaway process detected by mwa' | mail root@adminbox"
    hogpid = PROC_PROC_ID
  }
}
```

Change the root mail address from "adminbox" to something that makes sense for you and save the file. Run the MWA utility program to verify its OK:

```
utility -xc /tmp/alarmdef
```

If no errors are found, copy it back into /var/opt/perf and issue the command:

```
mwa restart alarms
```

This will tell MWA's alarmgen daemon to reread alarmdef and start processing the syntax that you just added. From then on, you should get an email notification of processes that may be stuck in loops. Here's another version of this same check, which has more bells and whistles in the exec action:

```
hogpid = hogpid
PROCESS LOOP
{
  # Send mail to sysadmin when processes are hogging the CPU now
  # and have accumulated over 1000 seconds cpu time total.
  # Avoid processes with pids < 100 assuming they're system processes
  # and they know what they're doing.
  if (PROC_CPU_TOTAL_UTIL > 95) and
     (PROC_CPU_TOTAL_TIME_CUM > 1000) and
     (PROC_PROC_ID > 100) and
     (PROC_PROC_ID != hogpid) then
  {
    exec "echo \"Possible runaway process detected by mwa",
         "\\nname = ", PROC_PROC_NAME,
         "\\npid = ", PROC_PROC_ID,
```

```
          "\\ncpu util = ", PROC_CPU_TOTAL_UTIL,
          "\\nusername = ", PROC_USER_NAME,
          "\\ndetected on ", DATE, " at ", TIME,
          "\" | mailx -s \"runaway process alert from `hostname` \" ",
          "root@adminbox"
    hogpid = PROC_PROC_ID
  }
}
```

There may be several things here that aren't "intuitively obvious" if you haven't played with alarmdef syntax before. Don't worry, I'll have more information below, and pointers to where to find the documentation for it.

An email message from this syntax might look like this:

```
Possible runaway process detected by mwa
name = imahog
pid =  7684
cpu util =  96.28
username = dgrumann
detected on 06/02/00 at 17:28
```

There's a lot of flexibility in the alarmdef syntax, and it can be pretty tricky (especially complex EXEC statements like the one above), so take it easy, look at the examples, and always check your edited alarmdef syntax with utility –xc before you move it into /var/opt/perf/alarmdef. The alarm syntax is explained in the MWA User's Guide, and there are several examples of alarms commented out in the default alarmdef file itself.

In this example, you can use similar syntax inside Glance as well! If you didn't have MWA available on that system, you could change the line:

```
"\\ndetected on ", DATE, " at ", TIME,
```

to be:

```
"\\ndetected at ", GBL_STATTIME,
```

then you could save just this alarm into a file and read it into gpm's adviser, or leaving it in a separate file on its own (say, /tmp/procmail) you could start up character-mode glance in the background like so:

```
glance -adviser_only -syntax /tmp/procmail -j 60 &
```

Essentially, you're telling glance to run in the background and check once a minute for possible runaway processes. I call this trick "Poor Man's MeasureWare." The reason why you needed to change that one line when converting the syntax from MWA alarmdef to Glance adviser is that the tools store time/date differently. Scope uses the metric names `DATE` and `TIME` while Glance uses `GBL_STATTIME`. There are only a few cases like this where the metric names in the tools aren't the same. The gpm online help has a whole section devoted to using the adviser. That section of the online help is also shipped as a printable document under /opt/perf/paperdocs/gp/C.

Symptoms – "Don't be alarmed"

Wading through all the data is important for some situations, but it can be tedious. The Glance adviser syntax and the MWA alarmdef syntax are there to make things simpler, to add a bit of "intelligence" to the tools. Both tools come pre-configured with a set of syntax that you can modify based on the system workload.

For character-mode glance, the default adviser syntax is in the /var/opt/perf/adviser.syntax file. The first time you install Glance, this file is created, but it won't be overlaid if you re-install so your previous changes are preserved. If you get a new version of Glance, you can compare the latest default adviser.syntax file in /opt/perf/newconfig to the one in /var/opt/perf to see if you want to pick up any changes. With gpm, its adviser syntax is encoded in the startup file in your home directory that also holds your window customizations. You can reset it to the default via menu selections in the gpm Adviser windows. Both interfaces have the same default alarms. Look at the files, and you'll see they're divided up into sections. The "symptoms" are definitions that we've come up with to represent probabilities for bottlenecks. For example, the CPU_Bottleneck symptom assigns percentage probabilities based on metric values for some CPU metrics. It looks like this:

```
symptom CPU_Bottleneck type=CPU
rule GBL_CPU_TOTAL_UTIL        >   75  prob 25
rule GBL_CPU_TOTAL_UTIL        >   85  prob 25
rule GBL_CPU_TOTAL_UTIL        >   90  prob 25
rule GBL_PRI_QUEUE             >    3  prob 25
```

This means that if the average CPU utilization on the system exceeds 75%, the probability of a CPU bottleneck is 25%. If CPU utilization exceeds 85%, then the probability goes up to 50% (the "probability" fields at the right are added up for all the rules which evaluate true). If CPU utilization exceeds 90%, the probability of a bottleneck goes up to 75%, and then things depend on the Priority Queue metric which is the average number of processes waiting for CPU time.

Consider this: If your CPU is 100% busy, meaning that processes are using all available CPU resources, this is a good thing! You want the money your company invested in the processing power of the machine to be used. There is the potential of a problem only if the CPU resource is saturated *and* processes are kept waiting for the CPU. If the CPU is busy, and more than 3 processes are, on average, waiting for the CPU, then Glance has decided there is a 100% chance of a CPU bottleneck occurring.

These rules are based on experience from many systems, but don't necessarily apply in every case. On very active servers, the CPU may always be busy and the Pri Queue may often run high even when things are going smoothly. You don't want your gpm icon flashing red all the time, so on some systems you may want to tweak the rules so that alarms are more likely to go off when response time or throughput is suffering. Remember that what you want to improve is overall application performance. These performance metrics are indirect measures of that. Direct application response time metrics are harder to come by (though we'll get to that later).

The Glance symptom statements for the other bottleneck areas are similar to the one for CPU. We combine various metrics that are good indicators into a bottleneck probability, which is then reflected in alarms and alerts. In Glance, we also pre-define some alarms based on system table utilizations. Something like a system running out of available swap space is not so much a performance problem as a configuration issue, but we have it in there. Periodically looking at the System Table reports in Glance is a good practice to see if you are nearing any limits and need to schedule time for reconfigurations.

MeasureWare similarly has bottleneck symptoms and alarms defined in its default alarmdef file. Like adviser.syntax, the MWA alarmdef lives in /var/opt/perf and is not overwritten when you re-install so that your changes are preserved across updates. Compare your alarmdef to the default in /opt/perf/newconfig to see whether there are any changes you want to pick up. In nearly every release we try to improve the symptoms and add more examples. Recently we've added some fancier examples under the /opt/perf/examples/adviser and /opt/perf/examples/mwaconfig directories. Check them out!

MWA has a facility built into the utility program that makes tuning your alarmdef file easier. If you run the command:

```
utility -xa -D
```

it will process the historical scope logfile data and show you how your current alarmdef file would behave. Alarms are shown and EXEC statements are printed but not actually executed. You'll probably see different alarms start, repeat, and (hopefully) end over time. You can look at the historical data with extract or one of the analysis tools to see why alarms went off at different times. If alarms are never going off and your users never complain about response time, then sit back and read your horoscope instead of this article! If alarms are going off too frequently, or they go off at odd times when you know the system is behaving, or (worse) they aren't going off during periods when you know the system is in trouble, then it's time to review the thresholds and alarms and customize them better so that next time they'll alert you when performance is an issue.

Metrics – "No answers without data"

So what are the "best" metrics to look at to solve performance problems? The answer is: "It Depends!". This is the first rule to learn about the art of system performance tuning. Every system's configuration and workload is different, and every problem might require you to look at different metrics to characterize what is going on. Nobody can give you a short list of golden metrics that will characterize all performance issues. With experience on your systems, you will find that some metrics are more useful than others.

When looking at a system's performance, it's best to begin at the global level. Glance and MWA's bottleneck alarms and their global metrics give a summary of how the system as a whole is behaving. The global metrics shown in Glance's main window and the ones used in the bottleneck symptom definitions are important indicators. From the global level, you drill down into different metric classes. There are several different

views of disk I/O, from the device, filesystem, and logical volume levels. There's another metric class to give the by-process view. There's a class of metrics to track information about each network interface. The sum of input packet rates (BYNETIF_IN_PACKET_RATE) for all network interfaces on the system equals the global metric GBL_NET_PACKET_RATE, just as the sum of physical I/O rates (BYDSK_PHYS_IO_RATE) for all disks equals the global metric GBL_DISK_PHYS_IO_RATE. In turn, some of these class metrics are broken down further. For example, the Disk I/O rates are a sum of the Read and Write rates, which are also broken down by type (I/O to a raw device, a filesystem, virtual memory -paging, or system -inode accesses).

Many metrics are subsets or recalculations of others. The global metric GBL_CPU_TOTAL_UTIL is the average utilization of all the CPU resource on the system over the collection interval, while GBL_CPU_TOTAL_TIME is the amount of CPU busy time over the last collection interval. If the update interval was 5 minutes (as is the case for MWA), then if GBL_CPU_TOTAL_UTIL was 10 at a particular time, GBL_CPU_TOTAL_TIME would be 30. There are 300 seconds in 5 minutes, and the CPU was busy 10% of that time, or 30 CPU-seconds. The multiple metrics are there so you can use whichever you feel most comfortable with. For example, if you are benchmarking a program you may care more about how many CPU-seconds it used than about how busy the CPU was while it was running.

In the default alarmdef file, you'll see a few references to APP_ metrics. These are application metrics. This is a special set of metrics that summarize data from groups of processes to make it easier for you to understand the performance data. For example, if there are 3 processes running in the "backup" application and each of them is doing 100 I/Os per second, then APP_DISK_PHYS_IO_RATE for the backup application would be 300. Applications are defined in the /var/opt/perf/parm file, which is a file shared by both Glance and MWA. This is another file that isn't overwritten during installation in order to preserve your changes. The latest default is /opt/perf/newconfig/parm. We ship some application definition defaults that are just examples. You may want to change them considering the applications running on every system are different! The MeasureWare Installation and User's Guide go into some detail about all the contents of the parm file, but the important thing to understand about application data is that you don't *need* to define applications. MWA and Glance work perfectly fine if there are no application definitions in the parm file at all! Every process that doesn't match an application definition gets its data bucketed in the "Other" application. Application definitions *only* affect the APP_ metrics.. there is no affect on the GBL_ or PROC_ metrics. If you want to look at the overall performance characteristics of an application conveniently, then you can set up your parm file to match your environment and use the Application metrics reported in both Glance and MWA. Examples of some application definitions you could use are in the /opt/perf/examples/mwaconfig/parm_apps file.

Another important thing to remember about application and process data is that the application data will reflect activity for all the processes that are grouped into the application, whether or not any of those specific processes were reported. Glance has

filters that allow you to restrict the number of processes shown in the Process List, but even the processes not shown in the Process List still contribute to the application and global summary data. In other words, filters you set up in one window will not affect data in the other windows.

MWA also has "interesting" process filtering, which controls what processes are logged by Scope. The process thresholds for MWA are set in the parm file. The default threshold line from parm looks like this:

```
threshold cpu = 5.0, disk = 5.0, nonew, nokilled
```

This means that only processes using more than 5% CPU or generating over 5 physical disk I/Os per second average during the 1 minute collection interval for process data will be "interesting," and thus logged in the MWA logfiles. If there were 100 processes in an application, each of which only used 1% of the CPU, then none of them would be logged individually in the process data class with the default thresholds. However, the application and global metrics would still reflect the activity. Its wise to tune the filters for process data being logged by MWA by adjusting the parm thresholds to suit your needs. If you want to see a lot of historical process-level detail, you can log processes using more than, say, 1% CPU over an interval. You could also switch on "new" and "killed" process logging by removing the "nonew" and the "nokilled" flags from the threshold line.. this will cause MWA to log processes that started and/or exited during an interval and ran for more than 1 second, regardless of how much CPU or disk activity they generated. Be cautious, though. A threshold line such as this:

```
threshold cpu = 1, disk = 0
```

would tell Scope to log all processes which used more than 1% CPU, processes that generated any disk activity, *and* all processes alive for more than a second that started and stopped during an interval. This setting would increase the scopeux daemon's overhead, especially on busy systems, and it might cause the process logfile (/var/opt/perf/datafiles/logproc) to fill up and roll over more quickly than you would want. However, such a setting may sometimes be useful for debugging problems. By controlling the amount of data being logged as well as the maximum size of the logfiles (also defined in the parm file), you can keep the right amount of historical data to meet your needs. If you rarely look at the MWA process data, then set your process logging thresholds high or turn off logging of process data altogether. The other data classes (and Glance) will not be affected.

Memory Scenario – "There's a hole in my bucket"

Let's go through another common problem situation. Let's say that you're managing a server with a mix of applications. You've recently installed a new version of a client/server application that your development group gave you, and although online users reported good response times at first, they started complaining about the application being slow after a couple of days, and eventually the server process "ileek" aborted and had to be restarted. Now it's a few days later and users are complaining again that it's slower.

Since you know history is repeating itself, you might want to take a look at the historical data from MWA. What was going on with the system when the application response degraded last time? Let's say that there was a noticeable sustained increase in disk activity (GBL_DISK_PHYS_IO_RATE and GBL_DISK_UTIL_PEAK) until the time the application was restarted. You could look at the type of disk I/Os going on, and which disk and logical volume was busy. Sometimes by knowing what's on the disk (datasets, home directories, swap) you'll have a clue as to what was happening. You can also look at the MWA application and/or process data to find anything out of the ordinary. Comparing the data for the specific client/server application over time, you might see that the Virtual Memory (VM) I/O rate (APP_DISK_VM_IO_RATE) and the Virtual Set Size summation (APP_MEM_VIRT) was increasing. In the process data, you might see that the process "ileek" had more memory faults (PROC_MAJOR_FAULT), VM I/Os, and a steadily increasing VSS (PROC_MEM_VIRT).

These are classic symptoms of a memory leak in the program. Your new version of ileek is probably repeatedly allocating memory and not releasing it. You could look at the latest logged MWA data to see if that same trend is occurring now. In general, this type of problem will often first show up as a decrease in the amount of system free memory (GBL_MEM_UTIL will climb to 100%) along with a sustained increase in the amount of swap space reserved (GBL_SWAP_SPACE_UTIL). As memory pressure increases you'll start seeing more VM I/O activity (GBL_MEM_PAGEOUT_RATE is an excellent indicator) and you'll start seeing swap space or memory bottleneck alarms from the Glance and MWA. It's usually pretty easy to spot the application and process that is causing the problem by browsing the data over long periods (if a process leaks 1 kilobyte a minute, its VSS will increase over 1 megabyte a day).

Glance can come in useful at this point to get more detail. Just as in the MWA data, you'll see the same set of metrics having larger than "normal" values for your system. The Glance adviser may tell you that there's a memory bottleneck or a disk bottleneck or both (as HP-UX is doing a lot of VM I/O to the swap devices). In the gpm Process List, you can set your sort fields to the Virtual Memory field, and the biggest memory users will pop to the top. Select the ileek process and bring up the Process Memory Regions report window. You can browse all the memory regions that the process has allocated, looking for the one that has a very large VSS. Often this will be Type=DATA, meaning it's the process's own heap space. At this point, you call in the developer who gave you the new version of the application and bawl them out.

So we solved this memory hog issue this time. Let's draw up some MWA alarmdef syntax to watch for this in the future. Using the copy-and-modify method of development, let's start with a variation of the syntax that solved our CPU hog problem:

```
memhogpid = memhogpid
# set VSS threshold to a value that makes sense for the workload.
# example set to 50000 kilobytes (~50 megabytes) Virtual Set Size:
VSSthreshold = 50000
PROCESS LOOP
{
```

```
  # Send mail to sysadmin for processes, which are hogging memory
  # and have been running for at least an hour (3600 seconds).
  if (PROC_MEM_VIRT > VSSthreshold) and
     (PROC_RUN_TIME > 3600) and
     (PROC_PROC_ID != memhogpid) then
  {
    exec "echo \"Possible memory hog process detected by mwa",
         "\\nname = ", PROC_PROC_NAME,
         "\\npid = ", PROC_PROC_ID,
         "\\nVSS = ", PROC_MEM_VIRT, "KB",
         "\\nusername = ", PROC_USER_NAME,
         "\\ndetected on ", DATE, " at ", TIME,
         "\" | mailx -s \"memory hog process alert from `hostname` \" ",
         "root@adminbox"
    memhogpid = PROC_PROC_ID
  }
}
```

Like last time, you could add this to the MWA alarmdef file by first adding it to your existing alarmdef, and then running "utility –xc" on it to make sure there are no syntax errors. Once you move it into /var/opt/perf you can use "mwa restart alarms" to get alarmgen to pick it up. Unfortunately, this syntax may not behave quite as you would expect, for a couple of different reasons. First of all, it is possible that your VSS threshold may be low enough that more than one process qualifies and you may start getting a lot of email. This is because we're setting the "memhogpid" to the pid of a process that passes our test in the loop, but if there is more than one process that qualifies, then the condition that checks the memhogpid inside the IF statement will always fail, and we'll generate two or more email messages every time. The second potential problem here involves the fact that alarmgen will only see process data that is logged by scopeux. In the case of CPU, a looping process will be "interesting" to Scope and thus get logged in its Process logfile and available to alarm on. For memory hogs that don't have much CPU or disk activity, there's a possibility they'll never exceed your parm file process thresholds and thus never be logged. When you use a loop construct in the Glance adviser, it will always loop through all instances of a class regardless of filter settings, but in MWA the process loop construct will only loop through data that you configured Scope to regard as interesting... in other words, it may not check all instances.

So how can we improve this syntax so that we catch the memory hogs consistently and avoid getting too much email? If possible, I'd recommend using Application data instead of Process data for MWA alarms. This presumes that you've made the time to edit the parm file applications such that you are capturing what you want. For our example, let's presume you've done this for the DBServer application. The parm file entry for it might look something like this:

```
application = DBServer
user = dbadmin
or
file = ileek,idbmaint,idbdaemon
```

Remember that when you change the parm file, you need to restart Glance or MWA ("mwa restart") to pick up the changes. In order to set a good value for the VSSthreshold

in our new alarm, you can use Glance and look at the Application List's Virt Mem (APP_MEM_VIRT) field for DBServer to see what a "normal" value is. The application metric is the sum of all the processes in that application, so for example if ileek and idbdaemon share access to a large shared memory segment, its VSS contribution would be reflected twice in the application's VSS metric. Let's say we settle on 80 megabytes (about 80000kb) as a normal value. Let's send email if we see it go over 100 megabytes. We can replace our alarmdef memory hog syntax with this:

```
# Watch for DBServer application using over 100MB memory VSS sum:
VSSthreshold = 100000
application loop {
  if (APP_NAME == "DBServer") and
     (APP_MEM_VIRT > VSSthreshold) then
  {
    exec "echo 'DBServer app memory VSS alert' | mail root@adminbox"
  }
}
```

Although this MWA syntax example works good, when it starts going off you'll get email messages continuously until you do something to bring the VSS for the DBServer application back down. This might be inconvenient, and if the condition starts going off at night then you could get a pretty full mailbox by morning. This is where the MWA alarmdef ALARM syntax statement comes in useful. You'll notice in the default alarmdef file that the bottleneck alerts are used inside specific alarm statements (as opposed to loops or "if" statements). Alarms do a couple of nice things for you. They allow you to define a duration over which a condition is true before it starts, and a repeat interval (if desired) to remind you the condition is still true. An alarm can have a different action specified for when it starts, repeats, and ends. Finally, alarms let you reference specific parm file applications without needing to use an application loop (in fact, you can't use the loop construct inside an alarm statement). Here's a version of our memory hog syntax using an alarm:

```
# Watch for DBServer application using over 100MB memory VSS sum:
VSSthreshold = 100000
alarm DBServer:APP_MEM_VIRT > VSSthreshold for 5 minutes
  start {
    yellow alert "DBServer application memory threshold exceeded"
    exec "echo 'DBServer app memory alert' | mail root@adminsys"
  }
  repeat every 60 minutes {
    yellow alert "DBServer application still hogging memory"
    exec "echo 'DBServer app alert continuing' | mail root@adminsys"
  }
  end
    reset alert "DBServer application memory demand now below threshold"
```

It's best to try to set up most of your alerts inside alarms like this so you can control the frequency of notifications. In recent releases of MWA we put some examples of cpuhog and memory hog alarms in /opt/perf/examples/mwaconfig/alarmdef_procloop that show how you can create alarms on data from process loops.

If you want to tune up your application definitions in the parm file, try using the gpm Application List. Double clicking on any app in the list will bring up the list of processes that are being bucketed in that application. You can change the parm file to get processes bucketed better, unfortunately you'll need to stop and restart gpm to see those changes reflected in its Application List. When I do this, I usually delete applications (like some of the ones in the default parm) that don't make sense on the particular system I'm working on, and then add ones which logically group the processes according to my knowledge of the workload. One goal might be to make sure that fewer processes that are using a lot of CPU are bucketed under the "Other" application. Once the applications make sense for you in gpm, you can do a "mwa restart" so that the MWA Scope data will reflect your changes. Looking back at the historical MWA process data later, you can look at the `PROC_APP_ID` to see whether you need to make more changes (Application Ids are assigned according to the order of your application definitions in the parm file).

Multiprocessor considerations – "Give it your 200%"

It can be tricky to create CPU alarms using application data. Since several processes are usually active in an application, it can be tough to tell the difference between a normal situation where different processes are active, and a situation where one of the processes is looping. In addition, when looking at CPU metrics on multiprocessor systems, you need to be aware that the application and global CPU metrics are "normalized." That means that these classes use the number of processors to help calculate relative CPU times and percentages. For example, imagine a 3-way MP system that has two processes in the same application, both looping. They could each use nearly 100% of a CPU. Over a 10-second interval, each process uses nearly 10 seconds of CPU time, so the application and the system as a whole have used nearly 20 seconds of CPU time in 10 seconds of elapsed time. When calculating utilization from these numbers, we leave the process data alone (i.e.: each process would show that it used nearly 100% CPU over the interval) but we divide application and global CPU activity by the number of processors. In this case, both the application and the global total CPU Utilization would be about 66%, not 100+100=200%. When you are looking at the overall system workload, or comparing one system to another, you usually consider CPU utilization as a number relative to the total processing power of the system. So, 66% global CPU utilization means that 2/3^{rds} of the entire processing power of the system was used over the interval.

This explains why you can see, for example, processes listed in the Glance Process List or MWA Process data for which the CPU percentages in an interval, if you sum them up, exceed 100%. Things get even more complex in HP-UX 11, where processes can have multiple threads, each of which can consume CPU independently of the others. You could imagine a process on a 4-way MP system that has three threads all of which are looping. In this case, the process as a whole is using 30 seconds of CPU time over a 10-second interval. Both MWA and Glance would report the process using 300% CPU. The application and global metric classes however, would normalize this CPU utilization and report the overall CPU resource as 75% busy.

Importing Glance Metrics into MWA – "Log whatever"

One important feature of the MWA product that hasn't been discussed is its Data Source Integration (DSI) capability. It can be a complex topic, and the DSI manual distributed with the MWA product goes into a lot of detail with examples of how to import data from other tools to log and alarm on. Smart Plug In products use this capability to, for example, log database application information into MWA. You can use this facility to set up data feeds from your own applications or monitoring tools.

Glance as a diagnostic tool is designed to provide an incredible number of different performance metrics. As mentioned before, the Scope collection daemon takes a subset of all these metrics and logs them for MWA. Sometimes there are specific metrics available in Glance that customers want to see MWA log. As time goes on, we add these to the list of metrics that Scope logs. For example, in HP-UX 11 versions of MWA we now log the most important System Table utilization metrics. Additionally, because the adviser can access any of the metrics shown in Glance, there is a way for you to bring any other data from Glance into MWA via DSI.

Let's look at an example from a multiprocessor system. Glance has a CPU By Processor report that lists all the CPUs on your system with statistics relevant to each. Normally HP-UX does a very good job of balancing the CPU load across the different processors, but sometimes you may find these BYCPU_ metrics useful. If you have an application that uses the mpctl system call to bind a process to a processor, it may be important for you to monitor individual processor activity. The BYCPU_ metric class is not logged by Scope, so if you want to get these metrics logged, you can set up a DSI data feed from Glance.

The first step in this process would be to write some adviser syntax to print out the values of the metrics you'll want to log. Let's pick the processor CPU utilization and the 1-minute running Load Average. Note: like the per-process CPU Util, this metric is not normalized on MP. The only thing tricky about writing this syntax is that you want all the fields going into the dsilog process to be printed on one line. We use the "\c" trick to keep each processor's output from coming out on a separate line inside the CPU loop below. You could put the following lines in a file named /tmp/bycpusyntax:

```
# adviser syntax file for dsi data feed
#
CPU LOOP
{
  exec "echo \"", bycpu_cpu_total_util, bycpu_run_queue_1_min, "\\c\""
}
print " "
```

And then run a command like:

```
glance -adviser_only -syntax /tmp/bycpusyntax
```

to see how every interval, both metrics will be printed for each processor on your system.

Then, to set up a DSI data feed, you need to define a DSI specification file. DSI needs to know how many metrics to expect on each line of input to the feed, so your specification file will need to have enough metrics to match the number of processors on the system. The example below works for a system with two processors. You'd put the following lines in a file named /tmp/bycpuspec:

```
# The following MWA DSI spec file is for logging BYCPU_ Glance metrics
CLASS BYCPU = 2000;

METRICS

CPU0_UTIL = 100
LABEL "CPU ID 0 Utilization"
PRECISION 1;

CPU0_LOADAVG_1_MIN = 101
LABEL "CPU ID 0 LoadAverage"
PRECISION 1;

CPU1_UTIL = 200
LABEL "CPU ID 1 Utilization"
PRECISION 1;

CPU1_LOADAVG_1_MIN = 201
LABEL "CPU ID 1 LoadAverage"
PRECISION 1;
```

The specification file syntax is explained in detail in the DSI manual. Here we define a new class of metrics for MWA, describe how to label them, and set a precision at which to store them (one decimal place). You can then "compile" this specification to initialize the "gpdata" data feed:

```
sdlcomp /tmp/bycpuspec /tmp/gpdata
```

Now you can start writing data to the data feed by backgrounding glance piping its adviser output into dsilog:

```
glance -adviser_only -j 60 -syntax /tmp/bycpusyntax \
      | dsilog /tmp/gpdata bycpu &
```

You can use the perfstat script to see that there's a glance and dsilog process running now. After a minute, you can check the data going into the data feed with the command:

```
sdlexpt /tmp/gpdata bycpu -H
```

This data is now being logged and you can make it available to the analysis tools. This is just an example, but it shows some of the flexibility available to you.

Application Response Measurement – "ARM your APPs"

CPU utilization, Virtual Set Sizes, I/O rates, and all the other system performance data help you understand what's going on with your system, but its not easy to understand how all this relates to the application's performance. Your users care about response time and throughput of their applications. Without instrumentation inside the application itself, monitoring response time often relies on user perception (help desk calls!). You can buy a stopwatch and stand over users as they enter data to see how quickly the application is responding, *or* you can get developers to instrument their code. The Glance and MWA

products support the ARM industry standard API for application instrumentation. This is the "transaction" data (TT_ metrics) you see in the tools. The Tracking Your Transaction manual (available online under /opt/perf/paperdocs/arm/C) discusses this facility of the performance tools in depth, and touches on the subject of defining Service Level Objectives for your applications. If you have an internal software development group, turn them on to this technology. If you buy your applications, encourage your vendors to investigate and instrument with ARM.

Once you have the data, you can focus less on trying to monitor system performance metrics and instead establish Service Level Objectives (SLOs), which the tools can use to monitor and alarm. By registering a transaction and then calling the ARM API when a transaction starts and stops, your application will make available to Glance and MWA useful data on response times, throughput, and even resource consumption per transaction. You can instrument client/server applications to pass correlation information so that from the server's perspective you can diagnose response time problems on a client-by-client basis. For example, you might notice that SLO violations are occurring only on a set of clients on a particular subnet and you'd be able to pinpoint a networking problem quickly.

If you are lucky enough to have your major applications instrumented with ARM, then you'll want to spend some time tuning the /var/opt/perf/ttd.conf file so that the response time bins and SLO thresholds make sense for your system. Going back to our DBServer application example, it might be instrumented to provide ARM data on a few different transactions such as "dbupdate" and "dbdelete". Perhaps a "dbupdate" transaction usually takes between .2 and 1 second to be completed, and "dbdelete" a little longer. You can define some response time "bins" to measure the distribution of completed transactions and add a SLO threshold like this in ttd.conf:

```
[DBServer]
tran=dbupdate range=.1,.2,1,5,10 slo=1
tran=dbdelete range=1,2,5,10 slo=2
```

The application can register these transactions and Glance and MeasureWare will show the transaction data. The most important transaction metrics are the number of completed transactions (TT_COUNT) and the number of completed transactions which exceeded your SLO threshold (TT_SLO_COUNT). You can also monitor the distribution of completed transactions in the response time bins to get a relative feel for what percentage of transactions are completing quickly or slowly and comparing this to the system-level metrics. For example, you may find that response times are excellent for your DBServer application except when certain other applications are running (like your program that searches for signs of extraterrestrial intelligence). It will be easier to balance the workload with this information. By keeping watch on how many transactions exceed your service level objective, you can be more proactive about monitoring application response and responding to problems. We ship a sample alarm that is based on SLO violations in the MWA default alarmdef, and there's some example adviser syntax shipped with Glance in the /opt/perf/examples/adviser/arm file.

There's a lot of information packed into this article, but there is also a lot more to the GlancePlus Pak that I haven't gone into. Rather than try to understand all the capabilities and extensions to the products right away, start with the simpler task of monitoring your workload to gain an understanding of what's normal for your system. Browse gpm's online help, the MWA manuals, and work from the examples we've provided. Have fun! Soon you'll count yourself among the elite performance "artistes." Together, Glance and the MeasureWare provide a well-rounded solution for system performance analysis. You can find more information about the other OpenView solutions for application and system management on the web at http://www.openview.hp.com. We look forward to customer feedback so we can make these tools even more useful for you in future releases.