# Designing Performance Tools for the Every Day System Manager

Gerry Wade
Hewlett Packard, Performance Technology Center
8010 Foothills Blvd, Roseville, CA

For years computer science has held a cloak of mystery about itself. At first it surrounded the entire field. Anyone who knew anything at all about computers could speak the jargon and baffle every day folks with seemingly meaningless acronyms. Gradually this shroud has been drawn back as more and more people were required to interact with these magic boxes. Now there remain only a few "computer wizards" who profess to understand the inner mysteries of the "System Internals". It is now time to open the cloak a bit wider and cast light on yet one more area normally understood only by the wizards.

The time has come to start designing computer system performance tools to be used by normal people instead of just by computer wizards. This article will discuss some of the requirements, techniques, and goals of such an endeavor.

## Who is going to use this tool?

As you might guess from the title of this paper, I feel that there are at least two categories of performance tool users. There are the "wizards" who have spent their lives learning the obscure jargon and off-the-wall acronyms common in discussions about the inner workings of computer operating systems. They have immersed themselves into the bits and bytes to such an extent that many of them have a hard time communicating with anyone except each other. There are also a lot of people who don't know much about computers and don't really want to know any more. Finally, there is a group of people we will characterize as "Every Day System Managers" or *Ed Syms* for short. (Oh dear, another acronym !!!)

The "Every Day" in Ed Sym indicates that these folks probably don't spend a lot of time pondering the trade offs in one memory management scheme versus another. They don't have the time or the need to design operating systems; that is why they bought a computer system that came with its own operating system. The "System Manager" part indicates that they are slightly more involved in the management of the computer system than the average data entry clerk. They probably understand that batch jobs use the computer resources in a different manner than interactive sessions. They no doubt have a passing knowledge of MPE's execution priority queues (but they might be a little confused about all the details on how they work).

Ed Sym is a member of a group of front line personnel whose task it is to keep their systems running at peak performance while they deal with the day to day tasks involved in managing those systems. (Some of them actually have to deal with USERS !!!)

So why are we talking about designing performance tools for this group of Every Day System Managers ? Surely they already have their hands full working with the tools they already have . Exactly! The current crop of performance tools are a real hand full. They can provide enough data to completely overwhelm most Ed Syms. It is not so much that these tools provide useless data, or even that they do not provide enough data. The problem is that they provide data in a way that takes a performance wizard to understand.

How this came about is pretty easy to understand. Initially the performance tools were written by the wizards for their own uses in understanding and tuning the operating system. In order to expose the inner workings of the various parts of the operating system many often obscure data items were collected. Many times these items were really just local variables used by some routine in the operating system and didn't have much meaning outside the context of that routine.

During the debugging phase an attitude of "better too much data rather than too little" prevailed so these early tools collected a vast array of data items. Now when the computer systems were placed in the real world, System Managers began to ask for some tool to help them answer questions about their performance. The easiest way to provide these tools was to take what already existed, add a jazzy front end to format the data and *VOILA* performance tool. Unfortunately no one took the time to see just what their intended users would need or how they would like to see the data presented. (Even if they did, most Ed Syms did not understand performance well enough to ask for the tool they really needed).

When the initial reaction to the tools came in indicating that they were too difficult to interpret, it was all too easy for the wizards to pass it off with comments like "that's why you need us to explain it to you. It is really a complex topic after all". Now before I alienate all the wizards in the audience, I'd better move on to the next topic. In summary, the existing tools suffer from an attitude that it's OK for performance to be hard to understand and a single tool can handle everybody's needs.

Now it is time to design performance tools specifically for the Every Day System Managers to use. These tools will probably not excite traditional performance wizards but that is not their intent. The tools will be written to address the needs of the Ed Syms of the world, which brings us to our next topic...

## What needs should be addressed?

There are probably as many ways to group performance tools as there are people who would want to do such a thing. For the sake of this discussion we will break performance tasks into four requirement levels. A given performance tool might address one or more than one of these areas. Personal experience suggests that any tool which attempts to address ALL of these areas is probably doomed to do none of them well.

**Quick Scan** or "What's going on NOW!". This is probably the most common tool. It examines what is happening on the system in "real time" (you see what is happening as it happens). The most common use of these tools is when the system performance becomes poor, to attempt to place the blame somewhere. "The system is really sluggish. Who is using up all the CPU ?" This is a *reactive* use of a performance tool.

**State of the System** This is the next generation of performance tools. It attempts to classify a system as to what resource is causing slow downs or perhaps to plan for hardware upgrades. Questions like "Should I buy more memory or another disc drive?" or "Would I run out of CPU if I added another 120 users?" are common. Performance tools in this area usually deal with the system as a whole and concentrate on certain key "*bottleneck*" resources.

**Application Tuning** In this area of performance, attention is usually focused on a particular user application or situation as regards making it go faster. "How can I reduce the response time in my order entry application?" and "What can I do so I finish the nightly batch programs before noon the next day?" are typical questions. For a performance tool to be useful answering these types of questions, it should be able to focus on particular parts of the system instead of the system as a whole.

**Trouble Shooting** What do you do if you have an application that runs well all day, but for some unknown reason every day from 3:35-3:40 everybody using it stops processing. How about if you have two identical systems running the same application, only one is getting half second response times while the other is taking 10 seconds for each transaction. All traditional performance tools indicate that there are no bottlenecks during these times. Obviously you have a need for some serious help. This is the case where you call you local Performance Wizard and have them bring their magic bones, tom toms, and gibberish producing performance tools to your rescue.

## Terms and Jargon

Jargon can be used to simplify discussions about complex topics. How much simpler it is to discuss the performance of a *CPU* instead of referring to a *Central Processing Unit* every time. Although simplifying communications might have been the first intent, today jargon is often used for just the opposite effect. If enough abbreviations and special purpose words are used in a discussion then the uninitiated can be totally excluded from understanding it. If you wish to be an elitist then it is easy to fall into the trap of using your own specialized language to prevent you from having to communicate with "just one of the crowd". It is also easier to disguise your own shortcomings if no one can challenge what you say since they don't know what it means.

The computer industry has been very good at inventing new words and developing a language that isolates its wizards from the common man. Computer performance tools are most often created by those most high wizards that understand the operating system internals. Many of these wizards are so involved in their own world that they have completely forgotten common English. It is small wonder that traditional performance tools measure obscure quantities with even more obscure names. (How many people do you know who REALLY UNDERSTAND what a "*Memory Manager Clock Cycle*" is? Would you assume it is good if this number goes UP and bad if it goes DOWN, or the other way around?).

The first step in designing performance tools, is to decide what things these tools should measure, and what names these "things" should have. I suggest that someone who has forgotten English is not the proper person to make design decisions of this type. Please don't misunderstand. I am not suggesting that all performance tools use terms found in a sixth grade reader, nor am I suggesting that all computer acronyms be eliminated. I think we might agree that most people who are interested in computer performance would understand what a "CPU" was (or at least could understand it if the acronym was spelled out one time). I am suggesting that instead of breaking up the activities of the CPU as:

MAM, PROCESS, ICS, DISP, CACHE, PAUSE, and IDLE

we use terms like:

SWAPPING, BUSY, OTHER, DISC CACHING, WAITING ON DISC, IDLE.

Note that renaming terms may involve regrouping them as well. In my example, ICS and DISP were merged to form OTHER since not all users will care to differentiate between *Handling Interrupts while on the Interrupt Control Stack* and *Dispatching the next process*. (I can hear the whines from the techno-nerd wizards right now. "But you MUST differentiate ICS and DISP or else you can't tell about gizormplatzing v09". Well, most every day system managers never even HEARD about gizormplatzing V09 and really don't need to know about it to take care of day to day business! If they DO need to deal with it, they will probably ask a wizard to help them and they can bring out their performance tool to measure things the way they want to see them).

## What tools are already available?

The HP3000 is not totally void of performance tools today so it makes sense to see what is available, and more importantly how the existing tools are being used. Here is what we discovered:

**Bundled (free) tools:** MPE comes with a few features which can be used to glean performance data. Most users will use commands like *SHOWJOB* for "quick scan" or "reactive" performance monitors. (Often system slow downs can be explained simply by knowing how many people are logged on to the system. Sometimes monitoring times of poor system performance with SHOWJOB can detect a correlation between system performance and certain user logons). More sophisticated system managers can even make some discoveries using the cryptic *SHOWQ* command.

We found many users performing "state of the system" analysis based on the *REPORT* command and on the wealth of information contained in the *MPE LOG FILES*. The REPORT command has only two time scales represented (NOW, and SINCE LAST RESET) and often needs to be augmented by recording its output periodically and then calculating differences. We found many tools available to assist in analyzing the REPORT command and MPE LOG FILES. Some of these tools were public domain or part of the TELESUP remote support package used by HP. Many others were available from the Users Group Contributed Library Tape or from independent software vendors.

**HP Services:** Hewlett Packard offers a variety of services which can address different performance needs. *HPTREND* is provided with some system support contracts. This service supplies plots of system resource usage over time periods of one to twelve months. The use of these plots is usually for a "State of the System" function and for long range planning purposes. *HPSNAPSHOT* is a consulting service which will provide detailed analysis of a short period of time (one to two hours) on a system. This service is normally used in the "Trouble Shooting" area although it has been used for "State of the System" or even "Application Tuning" functions. *HPCAPLAN* is another consulting service. It attempts to answer "what if" questions about a system's performance in the future. It uses an analytic model which is calibrated to your system using data from an HPSNAPSHOT data collection. Once the model has been calibrated it can be used to forecast the results of changing loads or potential hardware modifications.

**HP Products:** Hewlett Packard also offers two performance monitoring tools as purchasable products. *OPT* is probably the most well known. The "On Line Performance Tool" provides a variety of features. We found it being used primarily in two areas: For "Quick Scan" analysis, it's GLOBAL and PROCESS screens were most often used to determine what was happening on the system during times of concern. For "State of the System" analysis, OPT was run in batch mode generating log files or periodic summary reports.

*APS* the "Application Program Sampler" is being used almost exclusively for its most obvious function, Tuning Applications. While some site use APS as an effective "Quick Scan" facility, it has its strongest uses in identifying local bottlenecks in a particular application program.

**Non HP Products:** We found quite a few performance tools in use from sources other than HP. Many independent software vendors are providing a variety of tools for measuring and reporting the performance of the MPE/VE based HP3000 systems. I would be doing a disservice to attempt to list them all here as I know I would leave some out and misrepresent others. I suggest you visit the vendor's areas in any user's group meeting and scan the many periodicals and newspapers for informations on these vendors.

A few tools are worthy of special mention. These tools are often found on a system, but few people understand where they came from. I speak about the many progeny of the *OVERLORD* program (both direct descendants and relatives by association). Include in this list such names as "SOO" (Son Of Overlord), "MOO" (Mother Of Overlord), "GOO" (Grandson Of Overlord) and even "SCOOTR" (Second Cousin Of Overlord, Twice Removed). These tools sprang up from a need for "Quick Scan" information and someone's willingness to spend some time with a *MPE Tables Manual* (The Talmud of the MPE Operating System) trying to satisfy that need. These brave souls faced programming in "Super Privileged Mode" and crashing many systems in order to discover the many hidden secrets of the operating system and lay them bare for all to see. That they succeeded is evidenced by the survival of their tools. The nice thing about most of these tools is that they provide a lot of needed information and they are usually free for the asking (and the knowing of someone with connections that you can ask). The bad thing is that the original authors can rarely take the time away from their regular job to support and distribute updates to these tools. And updates are required since, unlike the real Talmud, the MPE Tables Manual is being continually changed (or at least the tables themselves are being changed). Many times these changes will cause older versions of these tools to fail or even crash your system. (No, I'm not unduly picking on the authors of these tools. I was one of those hidden folks involved in the "Overlord Legacy" myself).

## How is performance data obtained?

In order to understand the relationships between different performance tools, we must dip into a slightly technical area. What does it make sense to measure on a system and how do you go about measuring it? The intent of this paper is not to make you all performance wizards, capable of instrumenting an operating system. Rather it is to expose some of the different techniques commonly in use to measure performance of a system and the trade offs between them. Knowing this will arm you with the proper questions to ask when comparing the performance numbers from one tool with that from another. You should see that it is not only possible, but likely that two performance tools can measure the same quantity and get different results if they are using different measuring techniques. The temptation is to say they one of the tools must be wrong. Some might even argue that BOTH the tools may be in error, but few would admit that both tools may indeed be correct even though that is often the case. The thing that is missing in this discussion is a strict definition about WHAT is being measured and that will often involve HOW it is being measured so let's begin.

The most commonly used techniques for capturing performance data are **SAMPLING**, **TRACING**, and **COUNTING**. Lets take measuring CPU utilization to illustrate the three techniques. **SAMPLING** interrupts the system periodically and determines its current status. It accumulates the status according to what is being measured then resumes the system from the point of interruption. To sample CPU Utilization, the sampler would simply count the number of times the CPU was BUSY when it was interrupted, and the number of times it was IDLE. The CPU Utilization can then be calculated as the number of samples in the BUSY counter divided by the total number of samples (BUSY plus IDLE).

**TRACING** will insert code into the operating system to log the occurrence of key events as the system runs. To measure CPU Utilization, it would probably log the time that the CPU left and entered the IDLE state. (Actually it is more common to log the time the CPU was given to a particular process and the time that process had the CPU taken away from it since these are two well defined points in the dispatcher routines). Each log record will have a time stamp to show precisely when that event occurred. By subtracting the process start time from the process stop time you get the process busy time. Adding all process busy times found in a particular interval in the log file and then dividing by the elapsed time of that interval, you get CPU Utilization.

**COUNTING** involves placing measurement code into the operating system much like tracing does. The difference is that the code increments a pre defined accumulator or accumulates elapsed time into that accumulator each time an event occurs rather than logging the event itself. To measure CPU Utilization, the dispatcher code that gives the CPU to a process will save the time in a special place. When the process has the CPU taken away the measurement code will subtract this starting time from the current time to get the process busy time. This process busy time is then added to a global accumulator. CPU Utilization is obtained by sampling the accumulator, waiting a set interval then sampling it again. The difference between the two values in this accumulator is the total time the CPU was busy during the interval. Divide this value by the elapsed time and you have CPU Utilization.

How do the numbers from these different techniques compare ? As you might guess, TRACING and COUNTING can achieve very comparable numbers providing they are

measuring at the same points within the operating system. Either can make a mistake if all places within the operating system where a particular event can occur are not instrumented. Counting requires a pre defined array of accumulators and can consume significant amounts of main memory. Tracing can rapidly fill a log file if the events being trace occur with any frequency. (It is not unusual for process launches to occur at over 1000 times per second under MPE/VE. If you are tracing CPU as described then you will be generating 2000 log records per second). The overhead for tracing high frequency events often prohibits using this technique in other than a laboratory environment.

Sampling is often the easiest technique to implement since it does not require instrumenting many places within the operating system. It is often thought of as the least biased approach to measurement since it should interrupt the system and sample it no matter what else is happening (actually this isn't true but it is a common misconception). Sampling has the advantage of counting since it doesn't log individual events, merely counters. It has an advantage over counting since the counters do not have to exist as a part of the operating system and different measurements can define different things to count.

The problem with sampling is that a large number of samples must be taken to insure statistical validity. This assumes that the environment being measured will remain stable over the sampling interval. Sampling once each second might produce a good picture of the average CPU utilization over the period of one day, but the number would be very untrustworthy to measure the CPU utilization over one minute. This is especially true on most HP3000 systems where interactive terminal applications use CPU in short "bursts". I mentioned before that sampling was considered "unbiased" since it interrupts at random times and samples whatever is happening at those times. Actually in many cases sampling is implemented by asking the system time base generator hardware to interrupt the system at a given rate. This causes two phenomenon. First the operating system can elect to disallow interrupts during critical processing phases. This guarantees that no sample will occur during these times. Second, the system time base generator is often used by processes to pause, wait for a given time. If sampling is occurring based on this same clock, then it increases the probably that whenever a sample occurs, the "pausing" processes are in the same state (just waking up for example).

If the performance measures on your tool are obtained by sampling, you should understand the relationship between sampling frequency and duration of measurement in order to obtain a valid measurement. There are many books on the subject of statistical validity so I won't explore the issue further at this time. Also be aware that sampling works best where the system activity is uniform during the sampling interval.

When would you use one technique over another ? You would use sampling when instrumentation does not currently exist within the code you want to measure and when the values being measured may not be defined until the measurement is taken. (APS/3000 uses a sampling technique to measure CPU activity on a user application).

You would use Tracing when ever you needed to maintain a strict time sequence of events to analyze a situation. This is especially useful if the analysis must isolate a single event from all the others. (For instance, if you wanted to see a particular terminal transaction as opposed to seeing the average response time and transaction rate for a process). Tracing is especially useful in laboratory environment to trouble shoot systems where a causal

relationship must be established. (Parts of HPSNAPSHOT utilize tracing of IO events to analyze disc and terminal transfers).

Counting is useful when you can pre define the events to be measured and then instrument the operating system to measure them. It is useful for long term data collections since its counters can be sampled and logged quite slowly with no loss in accuracy. It will not miss events as sampling does, nor does it require massive log files as tracing does. Counting will mask most sequencing information and so is not as useful as tracing in trouble shooting operating systems. (OPT/3000 uses the HP3000 Measurement Interface which is predominately a counting technique).

## How much data is enough?

There is an old folk story about a man who captured a leprechaun. According to legend, such a capture requires the leprechaun to reveal the location of his pot of gold. Knowing the legend, the man forced his captive to lead him to the tree in the forest under which the gold was buried. After the tree was located, the man discovered that he needed a shovel to uncover it. To mark the tree, the man removed his bright yellow scarf and tied it around the tree. Now the man knew that left alone, the leprechaun would remove the scarf so the tree couldn't be relocated. He also knew that despite their mischievous nature, leprechauns were honest to a fault. Before leaving to fetch a shovel, the man elicited a promise from his captive that he would not remove the scarf or move it to another tree. Thus assured, the man hurried home to get a shovel. When the man returned, sure enough, his scarf was still tied around the tree just like he left it. However there was also an identical scarf tied around every other tree in the forest. It is said that the man died penniless after spending the rest of his life digging under all the wrong trees.

The moral to this story is that it isn't enough to present the correct answer. You must also make sure that answer is not obscured by a lot of other details. In other words, it is as important to know what NOT to present, as it is to know WHAT to present.

I suggest the following approach to guide you in discovering what is useful information, and what is distracting data. (Note that I use the term *data* to mean some measured quantity and *information* to mean data to which a MEANING is attached). Instead of starting with a list of what data is available, start with what information is required. In order to know what information is needed, you will need to decide who will be using the information and for what purpose. You should not begin to develop a tool until you have the answers to these questions firmly in your mind.

## How do you develop the tools?

The uses of a performance tool can vary. You can guess what Ed Sym would want to do with a tool, or if you can find Ed, you can ask them. In either case you will probably get an answer based on what is currently available. (Most answers are similar to "I need something just like XYZ only that does blah blah blah"). You have to be prepared for what happens if you build a tool that does exactly what is asked for. What happens if you go back to your lab and build the perfect XYZ that does blah blah blah better than anyone else? You hand it proudly to Ed Sym and they carry you around on their shoulders for solving all their problems. Right ?? (I guess we're all entitled to be naive once in our life). Actually, what happens is that Ed Sym takes your pride and joy and uses it gleefully for a day or so and then says something like "This is not bad, but if it only did dumdedumdedum it would be a whole lot better". (Talk about your rude shocks!!!).

Usually its at this point that you and Ed get into a shouting match with a lot of phrases like "But it DOES exactly what you asked for" and "why didn't you think of that BEFORE?". Actually this symptom isn't restricted to performance tools. You can probably see examples of it in many different areas. There are two forces at work here. First "things change" and the longer it takes to do something, the more it can change. Second "anything introduced to an environment will change that environment". The very fact that you handed Ed Sym a new performance tool will change what Ed needs in such a tool. For example, there is no way for Ed to know how they would really like a particular feature to work until they get a chance to use it and see what it really IS.

If this sounds like a "can't win" situation then take heart. I think you can solve it through a simple process of *successive refinement* and *prototyping*. All this jumble of words means is that you should PLAN on making the wrong tool the first time. The idea is to make the best tool you can, knowing it isn't the final solution then quickly get it into the end user's hands. Now let the user work with it a bit and then accept their ideas on how to improve the tool. Go back and implement those changes and let the user have the new release then repeat the process until you have the right tool.

Obviously the amount of time it takes to create a tool and to rework it must be minimized in order to achieve the final form in your life time. (If it takes you a year to implement changes then you will probably never catch up to the other changes in the environment. If you can turn a version around in a month, or better still in a week, then you have a good chance of creating the proper tool before it is obsolete). As you might guess, traditional vendor-customer relationships will probably not allow this type of working relationship. You will probably have to sacrifice on the "prettiness" of the documentation if it must be rewritten every week. The first releases of your tool should be only minimally implemented with major pieces delayed until the underlying framework is in place. Most customers would not purchase a product that was incomplete and subject to such rapid changes.

You will probably have to develop a special relationship with a few customers during the product development process. For their help in developing the product, the customer gets to influence its course and feature set. They also get to use the tool earlier than they would if they waited for it to be finished. On the other hand, the customer must be willing to quickly install and test new versions of the tool and provide accurate feed back on their changing uses and requirements of the tool. I strongly suggest that you use more than one

prototyping test site if you want to develop a generally applicable tool. This avoids anyone pressing features or procedures which will not be acceptable to the general user population.

A real problem in this type of development is knowing when to quit. Your prototyping test sites will blossom with an entire lab responsive to their every whim (more or less) and soon you may find an ever increasing list of enhancements for your tool. You will have to decide on just what constitutes a tool that is viable in the commercial market. No doubt many of the enhancements on your list would be beneficial, but at some point the tool will meet enough of the marketplace's needs to warrant making it a product. You will also have to decide what enhancements are not appropriate to the tool you are building. In order to avoid a product that looks like a camel ("a horse designed by a committee") you will have to weigh each enhancement against a design goal for your tool. It is not that the enhancements are not a good idea, but maybe they are better implemented in a separate tool.

As you can see, it isn't necessary to carefully define every detail about a performance tool's final appearance before starting to starting to write it. It IS important to have a crystal clear understanding of WHO will use the tool, and for WHAT purpose. Once these items are decided, you can form the design goals against which all enhancement requests and delayed design decisions will be weighed.

## An Example: (Setting the Goals)

Let us examine a real life example to illustrate the process we have described. We have already decided WHO will be using the performance tool (Ed Sym, the Every Day System Manager). We know HOW we will build the tool (Successive refinement through prototyping). Now all we need to decide is WHAT the tool will be used for. We take as our starting point the large installed base of HP3000 systems running the MPE/VE operating system. This system was chosen first because a large number of Ed Sym's out there are asking for a more appropriate performance tool, and because the operating system is fairly stable and well instrumented for performance data. Given this wealth of performance tools on the HP3000, why even bother creating yet another one ? Why indeed ? It seems that we were constantly being hammered to provide better performance tools which were supported by HP and filled out the performance offering. To sum it all up, people were asking for a tool they could purchase for a reasonable price that would provide more performance *information* on their systems.

As I mentioned before, the specific requests were based upon what was currently available. "We need a tool like OPT but which will log more data over long time periods and doesn't require a training class to use". "We like HPTREND but we want to produce the plots ourselves and we need more information in them". "We need to be able to 'tune' a performance monitor so it looks at our applications the way we do". " We would like HPSNAPSHOT to run all the time so we don't miss any important times, but we can't afford the disc space it requires".

Our analysis of all these inputs was that what was needed is a low overhead, continuous data collection facility coupled with a friendly and flexible user presentation module. The data collected might be used for "Quick Scan" work but since OPT was satisfying that need, it wasn't critical. "Trouble Shooting" was being successfully handled by the tools in the HPSNAPSHOT package so the new tool would not have to collect the level of detail necessary to support that function. "Application Tuning" was being handled well by APS but perhaps it needed some help in identifying which applications to analyze and when. The primary area this new tool would address is the "State of the System" functions over time periods between HPSNAPSHOT (1-2 hours) and HPTREND (1-12 months).

Our initial goals statement was to design a performance data collection and presentation tool which could be used to monitor the state of the system over periods of one hour to several months. It should be a low overhead tool which does not require large amounts of disc space or other system resources since its collection would be continuous and system wide. Much flexibility as to what was collected and how it was presented is required. The data collected should be in sufficient detail to allow Every Day System Managers to identify and analyze ninety percent of their performance problems but not so detailed as to require significant training to understand.

## Example continued: (Designing to meet the goals)

Now that we had our goals statement, we could begin our work. As part of our design strategy, we kept in mind that even though we were building a specific tool, much of what we did might later be extended. (For example in choosing a data structure, we purposefully choose one that could be expanded should we have a need. This decision didn't cost much in the original design, but would have been painful if we had to retrofit it later. Also, even though the original goal is not to create a "quick scan" facility, we found that it was fairly easy to design a system which would allow one to be added if desired).

Based on our goals strategy, we selected a few prototyping test sites and began to develop a working relationship with them. We spent some time getting them to tell us what they needed in a tool and then translating it back to them to make sure we understood. We were careful at this stage to keep the discussions fairly general since we weren't ready to fix the final tool features at this time.

Armed with the inputs from our test sites, we created the first pass tool and sent them a copy. As expected, at first they all said that everything was great and it was exactly what they wanted even if a few pieces were missing and it was a bit rough around the edges. After a week or so of using the tool, enhancement requests began to roll in. We filtered and combined all the requests, decided which ones were appropriate to the goals for the tool, then decided on the best way to meet the needs. (NOTE: It is at this stage that you have to be careful to ferret out the real need which is at the root of a request. If you do exactly what is asked in all cases then you will probably not design a good tool. Don't be afraid to go back to your test sites and ask them WHY they want a particular thing and then explore alternate ways to achieve it. You should also not be afraid to refuse some enhancement requests which are not appropriate to the tool's goals. Remember, the test sites are providing inputs, but YOU are responsible for the overall design of the tool).

## Example continued: (Lessons Learned)

In our example, we made many design decisions after repeated cycles between our test sites and the lab. Here are a few of the most important ones:

- A continuous data collection facility is required in order to provide long term information and allow analysis of an event after that event has passed.

- The data collection facility must be of sufficiently low overhead as to not impact the performance or usability of the system it is monitoring. While some users would tolerate as much as 10% CPU overhead, most would allow from 2% to 5% maximum.

- Data which is logged should be "pre reduced" from the raw data which is available in order to reduce the amount of disc space required. It was discovered that the processing requirements to pre reduce the data did not add significantly to the overhead of the collection facility due to the reduction in the resources needed to log that data.

- The interval at which data is sampled and logged will vary based on the nature of that data. It is often possible to choose data which requires lower logging rates and provides sufficient information for the task. (You don't really have to log every disc IO which occurred for example, if you just keep track of how many were done and by whom).

- The time retention requirements may vary depending upon the type of data being logged. It was found that *GLOBAL* data (concerning the system as a whole) was often required as much as 3-12 months after it was taken (although at the longer retention times a lower number of data points per unit time was usually satisfactory). *PROCESS* data (concerning the condition of a particular user's program at one instant in time), was rarely required for more than 1-4 weeks. At longer retention times, then volume of process specific data can become overwhelming unless it is reduced by summarizing.

- Summarizing process level information into user defined groupings at data collection time allows detailed data to be logged for those groupings which would be excessive if logged by individual processes. As an example, keeping track of disc reads and writes by logical device would be prohibitive on a process by process basis.

  If you keep read & writes (2 values) for each disc drive (maximum of 32 disc drives) for each process (current MPE/VE maximum of 628 processes can be collected) then you get 2 times 32 times 628 or 40192 values. Logging this number of samples once each minute will consume 115 Megabytes of disc each day, far too much for a continuous collector.

  When we asked our test sites what they did with this process level data, they responded that they added processes together into groups which represented certain applications. By pre summarizing this data we can reduce the volume of data significantly.

  Now you keep read & write (2 values) for each disc drive (32) for each user defined process grouping (arbitrarily set to a maximum of 20 although most users indicated that 6-10 would suffice) then you get 2 times 32 times 20 or 1280 values. Now by logging this information every five minutes instead of every minute (averages of processes don't

need to be logged as often as individual processes) then you collect only three quarters of a megabyte of data each day.

- Log data only when it will be used. (Avoid logging an event if it didn't occur or if it will not be utilized during analysis). Taking our previous example, if we only log a grouping of processes whenever there was some activity in that group then we can further reduce disc (and CPU) overhead without losing any information. If a user has defined 20 groupings of programs then we typically see only 3-5 groupings active during any one five minute interval. By only logging those groupings which had non zero activities, we can reduce the disc requirements to less than 100 Kilobytes of data per day.

- Applying this principal to individual process data, we found we could go even further. (With up to 628 processes on the system, each potentially logable every minute, we could easily swamp the data collector if we logged every process every minute). We found by observing the test sites, that in the "Quick Scan" mode they were ignoring most of the processes and concentrating on just those with certain characteristics. The "*interesting*" processes were those using significant amounts of one of the system resources. If we allow the user to set thresholds, then we can only log a process if it used more than the threshold amount of a given system resource. For example we would log a process if it consumed more than 10% of the CPU during a one minute sample, or if it caused more than 10 disc IOs per second during the interval. We also discovered that we should log processes which are entering large number of terminal transactions or are experiencing poor system response times. Now, to take care of the processes which never cross any threshold, we should log every process at least two times (once when it is first created so we know when it began processing, and again when it terminates so we know the total amount of resources it consumed and the amount of work it completed). With these additions, the concept of logging a process only when it became "interesting" instead of every minute, we can preserve having a lot of information about those processes without carrying a lot of unneeded data.

## Example Continued: (Presenting the data)

This pretty well defined a data collection facility which was separate from the data display facility. The data collector would be running continuously on the system (in order to not miss any important events and to be able to provide a continuum of available data). What about displaying the data ?

The first approach to displaying the data is to simply format it and print it. We found this approach to be quite limited unless you could select a specific small time interval for display. This was helpful in the trouble shooting functions, but not really what was needed for the tool we wanted to build. We did discover that many of our users were now performing the "quick scan" function not in real time as they had in the past, but on recent historical data. In fact many of them commented that it was great to be able to examine what was causing performance problems this morning, or even five minutes ago, even if you weren't logged on and running a performance tool at the time. It seems that a common scenario with tools like OPT was that the system manager would get a call that the system was running slow. They would then log on to the system and run OPT to see why. They often discovered that there was no real problem and the system was running fine at the time. When they called the user back their reaction was "Yes, it is fine now, but *you should have seen it a few minutes ago*!". Now with the continuous collector, they WERE there a few minutes ago. They could just take a look at the log file and confirm or deny the actual performance and any causes for it.

We had a good tool if the System Manager wanted to know what was happening at a particular time. What we didn't have yet was a way to tell the system manager which times were worth looking at. We rapidly discovered that the best way to examine a large volume of data of similar content was to graph it. A simple plot showing how the overall use of CPU varies throughout the day could go a long way toward identifying times of interest. If we subdivided the CPU into general classes as to what kinds of things it was being used for, then the information was much more useful. This is where we discovered what kinds of terms the Every Day System Manager would like to see. From a system manager's point of view, the CPU was used by Batch Jobs, Interactive Sessions, and System Overhead. Some system managers would like to see System Overhead have Disc Caching broken out separately since they could control this facility by enabling and disabling it.

Things really got interesting when we were able to graph one resource at the same time as another. For example, if we plotted the CPU usage, Disc IO Utilization, and Memory Manager activity all at the same time then we might make some observations about the three classic system bottlenecks (CPU, DISC, and MEMORY) and whether our system suffered from any of them throughout the day. Now if you add another plot showing terminal transaction rates and response times, we can visually correlate times of heavy terminal use and poor response times against potential bottlenecks in these areas.

As we explored ways to present the performance data we soon found that almost every user will want to present the data in slightly different ways. Some will be concerned about spreading disc transfers across all their disc drives evenly to forestall a potential disc transfer bottleneck. They will want to show the disc activity broken down by individual device throughout the day. Others will be more interested in the response time for certain key applications and would rather graph application response times alongside overall CPU,

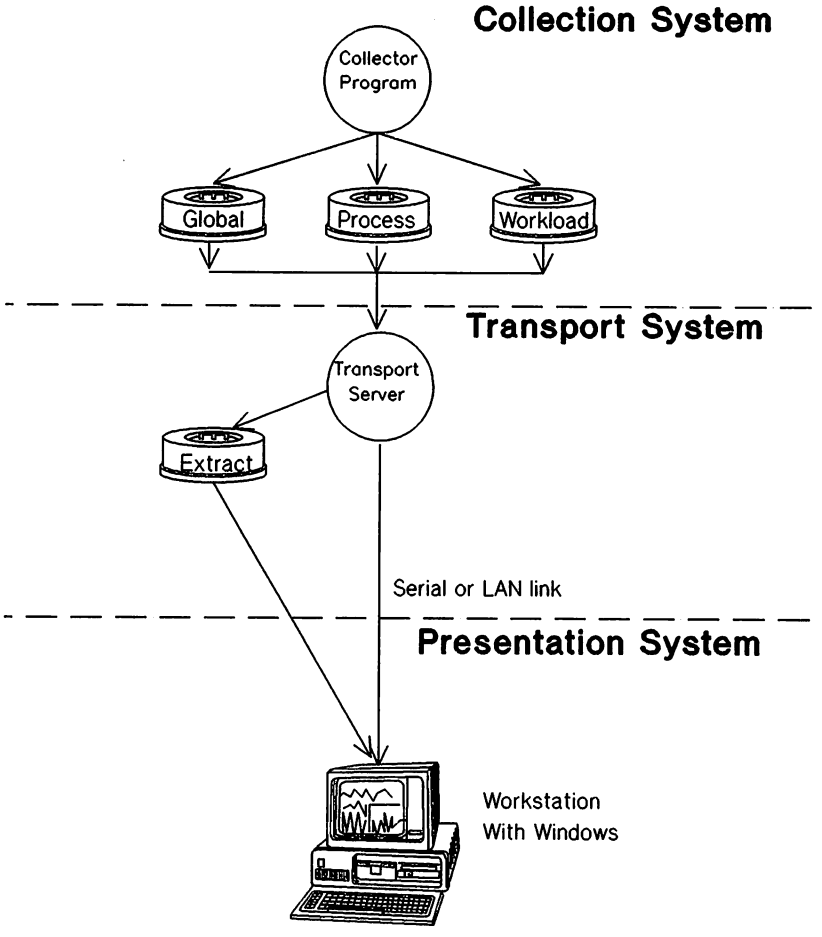
DISC, and MEMORY usage. The possibilities are staggering considering the amount of data that is available.

In order to meet this need, we tried a "let the user's roll their own" approach. That is, let's just make all the data available and let the user define their own graph. A few of our test sites really went for this, but most said that they didn't have time to spend developing graphs and furthermore, they didn't feel good that they would graph the right things given all this freedom. Back to the drawing board. (I DID mention that the development process allowed for making a lot of false starts didn't I??)

We now have settled on using a workstation to display the data and to take advantage of the features of an industry standard windowing package to allow combining data in different ways. In order to simplify life, we have defined a set of graphs which appear to be the most generally useful from all our test site's inputs. To take care of the special cases, we allow the raw data to be subsetted from one of the standard graphs and then exported to other graphics or analysis packages (such as Graphics Gallery or a Spread Sheet). This allows the more sophisticated system manager to display or manipulate the data as they see fit, without making the original tool overly complex.

## Example continued: (Final Design)

Please note that I use the word *Final* with caution since with prototype development you are allowed to make adjustments as necessary up until the product is shipped.



**Data Collection:**

The data collection facility will be a single process running in a batch job on the HP3000. It will gather most of its information from the MPE Measurement Interface Facility whenever possible. the Measurement Interface (MI) provides COUNTER type data for most GLOBAL and PROCESS items. The collection program will pre reduce the data before logging it. It will also form the grouping of process type data into user defined groups (called CLASSES) and log enhanced details for each class.

Some of the desired information is not available from the GLOBAL and PROCESS areas of the MI. Specifically we found that you could not obtain accurate values for Terminal Response Times, Disc Utilization Factors, or MultiProgramming Levels. The collector obtains some of this information from the MI IO Tracing facility while avoiding the overhead of tracing by analyzing the IO trace before it is logged then not logging it. Some items such as the disc utilization factor and the class MultiProgramming Level (Number of processes in a class that are waiting for the CPU) by SAMPLING at a rate of once per second. If later versions of the MI include these items as counter data items then the collector will be modified to use them in place of the tracing or sampling techniques.

The collector should be running at all times on the system being measured. It will collect and summarize its data and periodically log it to disc. Care was taken to insure that if the collector was stopped and restarted, or if the system crashed, then no significant amount of data would be lost from the log file. When restarted, the collector would continue where it left off in the original log files.

Up to three log files are used to hold three different types of data. This division of data was primarily to allow a different retention time commensurate with the usability of that data. One log file holds GLOBAL data records which report system performance on the system as a whole. Records are added to this file at the rate of one every five minutes. The second log file holds PROCESS data. A record can be added to this file once each minute for any process which is found to be "interesting". (The precise definition of "interesting" defined below). The third log file holds WORKLOAD data. One record can be added to this file every five minutes for any user defined grouping of processes (CLASS). In order to be logged, a CLASS must have had at least one process consume some CPU during the five minute sample.

The user may specify whether to log GLOBAL, PROCESS, and WORKLOAD data independently in any combination. They may also set the amount of disc space they are willing to consume to hold data of each type. The log files are "circular", that is when they are filled, old data is removed to make room for new data. In this way the user need only specify the maximum size of each file and the collector will automatically remove aged data to make room for new records. You will always have the latest data available for analysis. Since each log file can be built at a different size, different amounts of data can be maintained for each data type.

In order to reduce overhead and conserve disc space, a process will not be logged to the PROCESS log file unless it is "interesting". A process will ALWAYS be summarized into its proper CLASS and the CLASS will be logged to the WORKLOAD log file at the proper time. "Uninteresting" processes are not ignored, but their activities are summarized into

classes and logged at longer intervals. A process becomes "interesting" when it meets any one (or more) of the following criteria:

- A process is seen for the first time (it is NEW).

- A process ceases to run (it DIES).

- A process uses more of a given resource than a user defined threshold value. Currently thresholds can be set for the following resources:

  CPU (default is 10% CPU over a one minute interval).

  DISC (default is 10 physical IOs per second for one minute).

  TERMINAL TRANSACTIONS (Default is 100 transactions in one minute).

  TIME TO FIRST RESPONSE (Default is 1.0 seconds from entering the data at the terminal to completing the first write back to that terminal).

  TERMINAL RESPONSE TIME (Default is 5.0 seconds from entering the data at the terminal to completing the last write prior to the next read back to that terminal).

**Data Transport:**

In order to present the collected data, it may be necessary to transport it to the data presentation workstation. There are several options available.

Single system connections may be made using either a Local Area Network (fastest, most expensive) or RS-232 Serial (Least expensive, slower) data comm link. Data may be analyzed directly from the three log files (even while the collector is still collecting data into them) or it may be moved to local workstation storage. The capabilities are not affected in either case, merely the access times and the amount of workstation storage required. You may also decide to use a combination of workstation (LOCAL) and HP3000 (REMOTE) data accesses. For example, if you are utilizing a low speed serial connection and you want to analyze a large amount of data, then you may download that data to the local disc storage overnight. You may still use remote access directly to the log files if you want more current data (such as data on what happened only a few minutes ago). Accessing smaller amounts of data will be relatively quick even at lower data comm speeds.

Multiple HP3000 systems may be analyzed from a single display workstation. If all the HP3000 systems are connected to the same LAN as the workstation, then the workstation can directly access the log files on any system as easily as on any other. If the workstation is connected to only one HP3000, then you can use the same data extraction tool to move selected data to the locally connected HP3000. From there the extracted files can be downloaded to workstation storage, or accessed directly from the local HP3000 disc.

As a special feature to aid those sites with multiple HP3000 systems, each log file can be automatically identified with its own user specified identification string. This identity will be displayed when the presentation package accesses each file.

**Data Presentation:**

A user workstation will be used to display the performance data. Industry standards are being followed when designing this display package to allow maximum flexibility and compatibility with future products. Currently the preferred workstation is an HP Vectra Personal Computer (or any fully compatible IBM PC/AT computer) configured to support MS/Windows release 2.0 or later. The PC and MS/Windows provide a rich user environment which is flexible in a standard way. It also eases the conversion to later standards such as HP's New Wave. Other workstations may be supported in later releases as these standards propagate.

Special attention is being paid to insure that the first release product will operate in conjunction with other products offered on the HP Support Workstation. (Products such as the HP LASERROM service will use the HP VECTRA PC and integral CD-ROM Disc Player).

The MS/Windows environment will make it easy for existing PC users to learn to use the data presentation software. Users which are not yet familiar with MS/Windows can follow a simple tutorial supplied with the package to learn to use it in about twenty minutes. Additional tutorial information will also be available utilizing the CD-ROM player to provide Computer Based Training. Training topics will include learning how to use the tool, Basic Performance Topics, and examples of system log files which illustrate different system situations. Using Windows, it is easy for a user to compare two or more standard data graphs by positioning each in its own window.

A Context Sensitive "Help" feature will also be included. This will allow a user to ask for help, then use a pointing device such a mouse to point to anything on the display they want to know more about (Such as the blue line on a particular plot). They will then get a "pop up" window explaining that the blue line represents CPU Busy and can range from zero to 100 percent. They can ask for more help on this or related issues or return to the graphs where they left off.

For those system managers which have a higher than average knowledge of the performance measures or who would like to perform some manipulation or alternate display techniques on the data, a data export facility will be provided. The data can be selected directly from a standard graph by simply using the pointing device to point to the start and end of the interval of interest. All relevant data which occurred in this interval will then be written to an MS DOS file in either a standard printable format, or an industry standard data transfer format such as DIF. Data heading may be included if desired to identify each data item. It is usually an easy matter to then move this data into a wide variety of PC applications such as word processing, spread sheet, statistical analysis, or graphing where additional operations may be performed on them.

**Summation:**

We have found traditional program development techniques to be inappropriate for the development of performance tools for every day use. In particular the design decisions for such tools are best made by a process of early prototype testing. This testing allows the users to adapt to the tools and learn what they can provide. More importantly it allows the tool to adapt to the users once they realize the potentials. This intimate feedback during the early development of a tool can result in a much more useful product and a much shorter overall time to market.

During our investigations, it became clear that performance tools designed by operating system engineers are typically not appropriate for use by end users. The fault with these tools does not always lie in their user interfaces as many believe, but rather with the choices of what data to expose. A common fault is to provide too much data thus hiding what is important. We also discovered that some attention must be paid to the terms used to describe the data. Often a simple change to the name of an item was all that was needed to clarify it. (For Example: Changing 'MAM' to 'SWAPPING OVERHEAD' does not assume that the user knows that the operating system memory management facility is called 'MAM').

A common attitude in performance logging facilities was to delay the decision of which data was important until the data reporting phase. We have found this to be a wasteful practice. By deciding what data will be required and in what formats, we were able to drastically reduce the storage overhead of the logged data. What was not obvious was that we also reduced the data collection CPU requirements. Contrary to popular beliefs, it is often less work to pre-reduce the data than it is to log all the raw data for later reduction. In this case a double savings was discovered, once during collection then again during reporting.

Substantial usability benefits are obtained by using a full time data collector. By designing a data collection facility which is low enough overhead (including the run time resource usage and in the log file storage requirements) users are not pressured to try and "catch" a performance problem while it is still happening. Since the collector is always collecting enough data for analysis, the users can relax and examine a situation after it has passed. System Managers can now afford to take a day off without having to worry about the system performance while they are gone. If any problems arise, they can be examined calmly when the System Manager returns.

The user interface features provided by a workstation such as the Vectra PC make it a lot easier for most users to gain access to their data. The use of a powerful industry standard interface such as MS/Windows (and soon New Wave) allows those users already familiar with this environment to use the performance tool with little or no training. For those users which are not already using MS/Windows or even a PC we have developed a simple tutorial. Our test site users were able to pick up the fundamentals of this environment from a short (20 minute) session following simple examples from the tutorial. Once windows fundamentals are understood, the performance tool becomes very simple to use. Each different context which the user can enter will have familiar windows features which all operate in an identical manner. Thus a user can go into the "Disc Details" context for the first time and notice that they are presented with a horizontal and a vertical "scroll bar". Even though they may not be 100% sure what each scroll bar does, then know enough to

point to the bar and "click" to scroll backward and forward through the data. Additional details can always be found in the powerful "help" facility.

By separating the data collection facility from the data presentation facility several benefits were realized. First as already mentioned, we can take advantage of the powerful user interface provided on the workstation even though data must be captured on the main system. Second, multiple data collection facilities can be supported by a single data presentation facility. This means you don't have to duplicate the data presentation hardware and software for each system you are monitoring. Third, although not quite so obvious from the discussion in this paper, the workstation data presentation facility can be easily made to be independent on the type of system which collected the data. It would be possible to analyze data collected under different operating systems and architectures (say MPE/VE and MPE/XL and even HP-UX) using the same workstation. It might even be possible to examine the resource consumption of an HP-9000 running HP-UX and compare it to the same time interval for an HP-3000 running MPE/XL. To the user, it would simply be two windows on the screen, each one being controlled in the same manner.

Last but not least, we have discovered a wealth of additional uses for the long term performance data collected by the data collection facility. We can now begin to prototype different data analysis and presentation tools which use the same log files. The richness of this concentrated system performance data covering such long time intervals has surprised even our staunchest designers. Watch this space for additional details.....