QUICK START GUIDE FOR DRM 3.0 on AIX

Introduction

The DRM3 is an innovative new product developed by WHAM Engineering & Software for measuring the performance and resource utilization of client-server type applications. Today there are a large number businesses utilizing capture/replay technology for generating load in a client-server environment. These tools do a good job of measuring the response time and throughput from the perception of end-user workstation but have no view into the server behavior. Increasingly, it is the server actions that are responsible for the performance measured by the capture/replay client load generators. Thus, when the response time and throughput goals are not achieved, the server behavior must be understood.

The existing products available for measuring application behavior and performance on UNIX systems are not well-suited to the analysis of client-server applications. In particular, the class of products that simply use system call or standard performance data collection interfaces provided by the operating system fail to provide the needed granularity and detail. Products based on the ARM standard require significant instrumentation by the application to be monitored as well as a specialized collector to gather and interpret the data. Each of these classes of products are also quite intrusive to the operating environment of the server applications.

The DRM3 is based on patent-pending technology invented by WHAM Engineering & Software that requires no application instrumentation and impacts the system on which it runs in a negligible fashion. The DRM3 agent consists of a user level daemon process the ***drmd*** which interfaces to a kernel extension through a set of system call interfaces. The kernel component has a timer driven sampler that snapshots the collected metrics 10 times per second. This data is buffered in the kernel and extracted once per second by the ***drmd***. This process is itself a server and can deliver data to multiple concurrent requestors at any rate needed by the individual user. The DRM3 uses wrapper technology to redirect system call interfaces through WHAM's patent-pending measurement software.

Because the DRM is kernel resident it can provide metrics of it's own construction such as the number of threads running for each application and the total number of threads available to run for each application. This kind of measurement allows the developer to determine if the expected level of concurrency has been achieved. Additionally, WHAM has defined the concept of a transaction as it relates to client side and server side behavior. Simply expressed ***a transaction is a request response pattern that takes place on a TCP socket connection***. There is a server side perspective that consists of receiving one or more request messages and responding with one or more response messages. The client perspective is sending one or more request messages and receiving one or more response messages. An application can exhibit either or both behaviors. A web server that simply serves files would be an example of a server in it's transactions with requesting browsers. A web server that implements servlets which make requests to a database would exhibit client behavior from the servlets with respect to the database server.

The DRM3 separates these two behaviors and measures the total time spent by the client process as it sends request messages, receives response messages and waits on the server to respond. These metrics are summarized by a post processing tool that provides an average total response time or end to end time and a throughput rate from the client perspective. On the server side of an application the same measurements are made with respect to server behavior. The throughput of the server is summarized as well as queue length. In addition to these measurements, the average request size and response size are measured and reported. These data are obtained at the socket level in the kernel of the operating system for each application and so no instrumentation is required. This allows the DRM3 to be used to evaluate database servers, web servers and any custom applications developed by your organization against a common set of measures.

The focus of the DRM3 is evaluating scalability and providing sizing information for single applications or groups of applications. The DRM3 also collects all of the available UNIX kernel metrics about a process which generally consist of CPU and memory information. This allows a determination of the notion of a CPU cost per transaction. With all of the information collected by the DRM3 it is possible to make very accurate simulations and models of systems. For the analyst that is simply attempting to understand how the system is performing at different load levels, the DRM3 provides a very clear breakout of many different components of application and system behavior. This detail allows the analyst to identify points of poor performance and move toward a resolution.

Finally, the DRM3 defines a transaction which is common across client-server applications built on TCP/IP. This is distinct from other products that require the user to define a complex set of interactions as a transaction. Such products usually will advertise the mapping as a feature that allows the user to define a transaction. However, what it doesn’t make clear is how the user is expected to know a-priori the mapping between the complex set of interactions and a real transaction. The DRM3 allows the user to map the number of operations performed by a known load onto DRM3 transactions, which can then be used to provide an accurate estimate of the cost of the real operations. This is ideal when used in conjunction with capture/replay technology because the capture component is used to collect a set of user interactions with the target system in order to build up the essence of a real world operation. The replay system can then perform these captured operations at variable rates and with variable numbers of users. The DRM3 is able to collect data from the server during the replay sessions and then the DRM3 data can be used to map user operations to DRM3 transactions in order to form an accurate estimate of the cost of any operation or grouping of operations.

Using the DRM GUI to Collect Data

The DRM GUI is invoked by typing ***drm.gui***. When the program starts it presents two small windows. The first is entitled Distributed Resource Monitor and the second is entitled Collector. The first window is the main window and is shown in Figure 1.

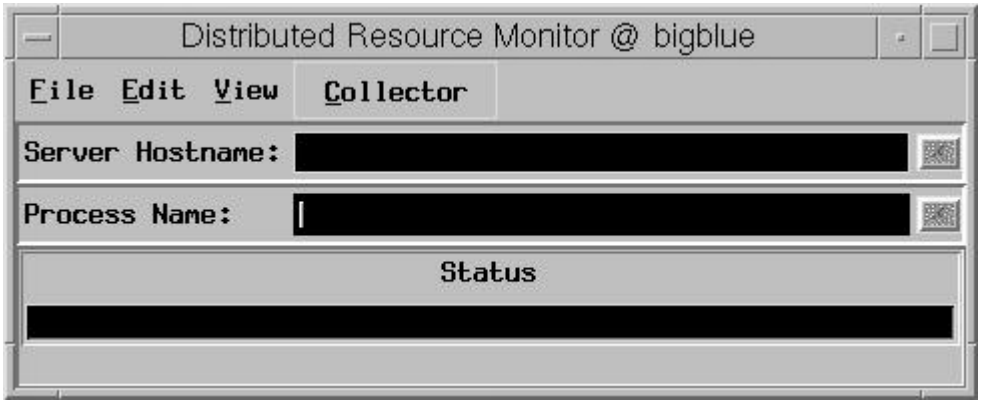

Figure 1

From this window all functions of the DRM can be accessed. The File selection allows the user to create a new data file or open an existing file for analysis or viewing. If creating a new file, the window shown in Figure 2 will be displayed.

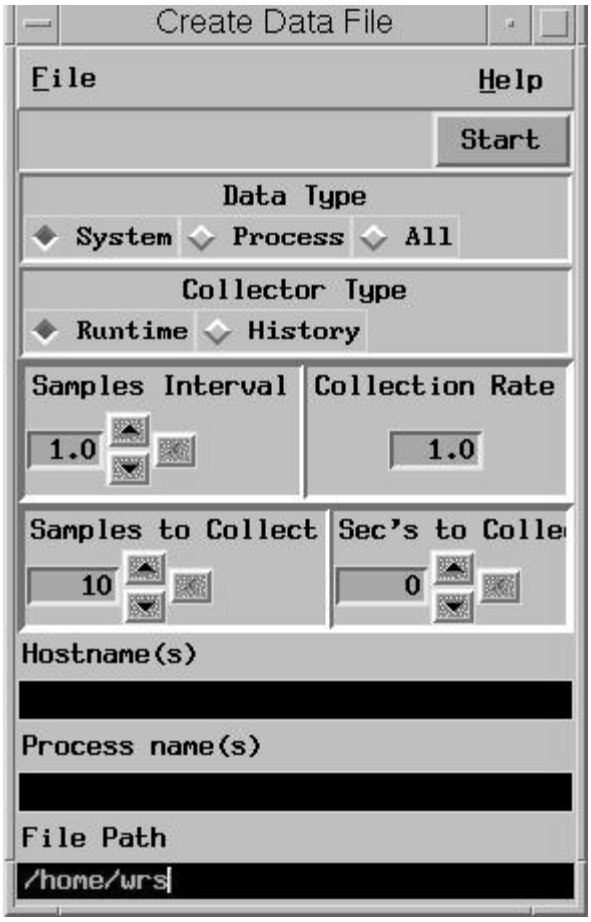
Create Data File
File
Help
Start
Data Type
System
Process
All
Collector Type
Runtime
History
Samples Interval
1.0
Collection Rate
1.0
Samples to Collect
10
Sec's to Colle
0
Hostname(s)
Process name(s)
File Path
/home/wrs

Figure 2

This window allows for specification of the data type to collect, the rate at which data is to be collected, how much data to collect, what and where to collect from and finally, where to store the data. After the fields have been filled in, the collection process is initiated by activating the Start button. The Data type buttons allow for selection of the data type contained in the file to be created. It is best to get both process and system data (All) for most work. The Collector Type should be left at Runtime. This means the data will start from the time the Start button is depressed. History means that only the previous five minutes of data stored in the agent memory is collected.

The Samples Interval or Collection Rate selections allow the sample interval or sample rate to be specified but not both. The up/down buttons can be used to experiment with the allowable values. The fastest sample rate is 10/s which corresponds to a .1s sample interval. The control for the duration of sampling is set by the number of samples or the number of seconds to collect but not both. Again, experimentation with the up/down buttons will show what allowable values are available. For sample rates of greater than 2/s or intervals less than .5s the sampling is only done once per second. In this case the two buttons are identical.

NOTE: On DRM3 the Seconds button should not be used.

The Hostname(s) field is used to specify which host(s) to collect from and if not filled in means the local host. The Process names field specifies the names of the processes for which to collect data. If the processes to be collected are not expected to be present at the time the Start button is activated, make sure to select the init process or the drmd. This allows the agent to establish a data stream to the monitor and when the processes of interest begin, data will be added to the existing stream. If this is not done and the process isn't available when the Start button is activated, there will be a 15s interval before that host is retried.

The File Path field specifies the location of the data file. This must be the name of a file that doesn't yet exist.

Now let's collect some data from a web server that we will do a load test for. The name of the web server is bigblue and the load client will be on blueboy. The web stress client is named web_client and the web server is ns-httpd. The collection window that we have filled in for creating the file is shown in Figure 3.

Figure 3

In this instance we have selected a .1s sample interval collecting all data from hosts blueboy and bigblue about processes web_client, ns-httpd and drmd. The drmd is the DRM3 agent and will be running when the web_client is started on blueboy. The strategy here is to set up and start the collection and then begin the load test. The length of the file has been set to 35 samples which in this case will be approximatley 35 seconds. The web_client program has been set to use 2 threads making 20 requests per second each for a total of 400 requests. The web_client should run 20 seconds and so the selection of 35 seconds should be sufficient to capture all of it's activity.

Using the DRM GUI to Analyze Collected File Data

Now that we have a file of data we can proceed to analyze and view the data that we have collected. To do this we use the main window File selection to Open a file which presents us with the DRM Data File Control window. This window has a Browse selection that is useful for providing a selection of files. In this case since the file was just created we use copy and paste to enter the file name from the Create Window into the DRM Datafile Name selection box of the DRM Data File Control window. After pasting the name the cursor will be focused at the end of the path name. Typing return causes a Hostname selection window to be displayed which allows a selection of either of the hosts which were specified in the Collection Window. The Select Host Name window is shown in Figure 4

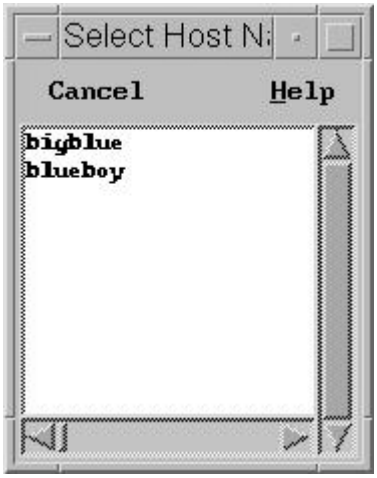

Figure 4

By double clicking on a host name the information about the file and what data is contained relating to that host is now in the memory of the GUI. The File Control window is shown in Figure 5.

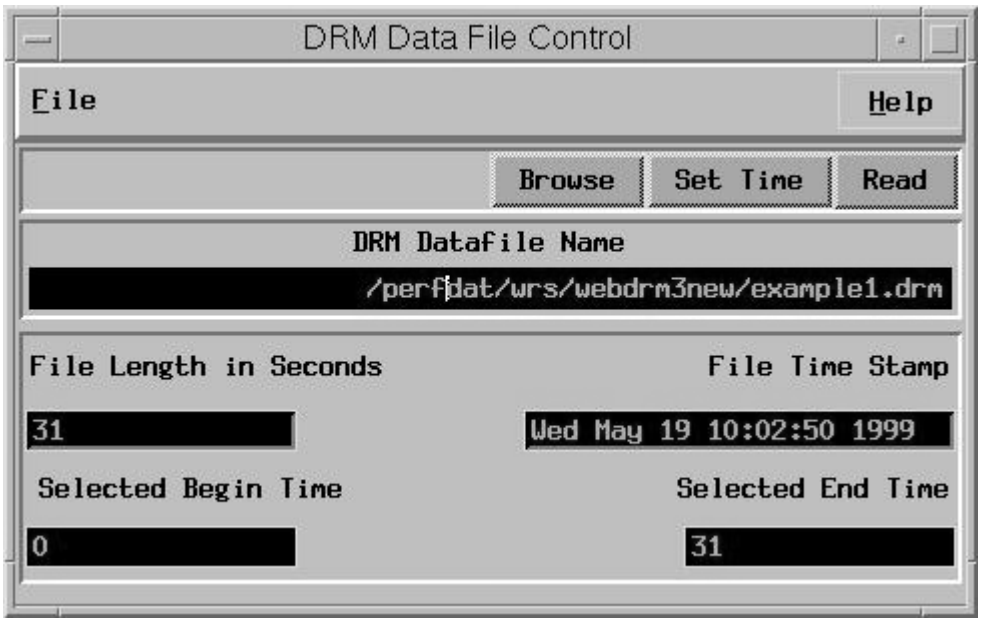

Figure 5

This shows that the file is actually 31 seconds long. The web_client program is instrumented internally and makes measurements of round trip time for each request it makes. The output of the web client program was as follows:

```
ip address of interface is 199.5.191.26
Web Stress Test         Number of Threads: 2
Message Interval: .05 second(s)
Total Number of calls: 800
Total Number of calls: 800

Average = 0.016100400   Variance = 0.043541708  StdDev = 0.208666500
Maximum = 5.903314987           Minimum = 0.000000000
Attempted URLs/s 40.000000, Actual URLs/s 25.713317
```

This shows that the client had an average round-trip time of 16ms with a variance of 4ms and a maximum response time of 5.9 seconds. The client attempted 40 requests per second but was only able to get 25.7. We will use the data in the file to determine where the slowdown was.

To get a first cut idea of what happened in the run it is useful to get a synopsis of the activity occurring on each host by generating a report. This can be done through the View selection on the main window which produces the pop-up shown in Figure 6.

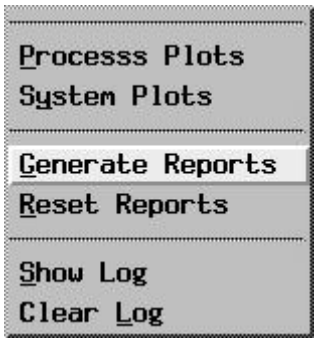

Figure 6

Clicking on Generate Reports produces the Report Setup Window shown in Figure 7. The report generated will list the processes on each host sorted in ascending order of any of the keys shown in the key list. We have selected system CPU because that will allow us to distinguish which ns-httpd process was active. The web_client was specified to run against only a single instance. We don't know which one of the four instances on bigblue was the active one because the client identifies them by port number and the server identifies them by PID.

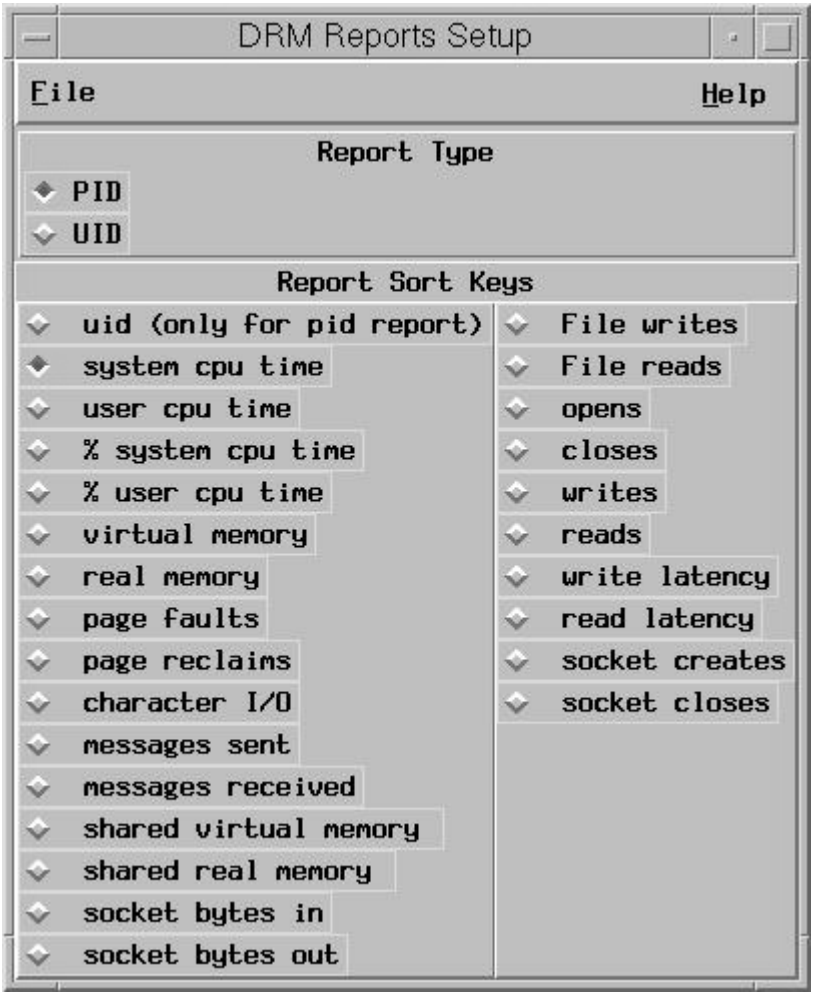

Figure 7

The report data shown in Figure 8 indicates that PID 14548 was the active ns-httpd PID. It shows that from the data in the file we saw 768 transactions (SRVTRANS) and used 1.54s of system CPU time(SYSCPU). The first seven lines of output about PID 14548 are mostly metrics that the agent collects while the last two lines are statistics that we compute from the collected metrics. The Start and End times of the data for that process are shown above the metrics and statistics. For this PID we see that it was active for the entire 31 seconds.

The first line of summarization shows that the average service time per transaction on the server, (AVGSVTIM) was .00052 seconds and the average response messaging time per transaction (AVGSNDTIM) was 1.72 ms. This indicates that the server was taking about 2ms per transaction. Another interesting and often useful server statistic is the server transaction (SXACT/S) rate which is reported as 26.37/s. This corresponds closely with the rate measured and reported by the web_client of 25.7/s. Additional information about the server is the CPU cost per transaction (CPU/SXACT) of 3.29ms. The web_client process is the next thing to look at and so we use the scroll-bar in the right hand of Figure 8 to produce the report info shown in Figure 9.

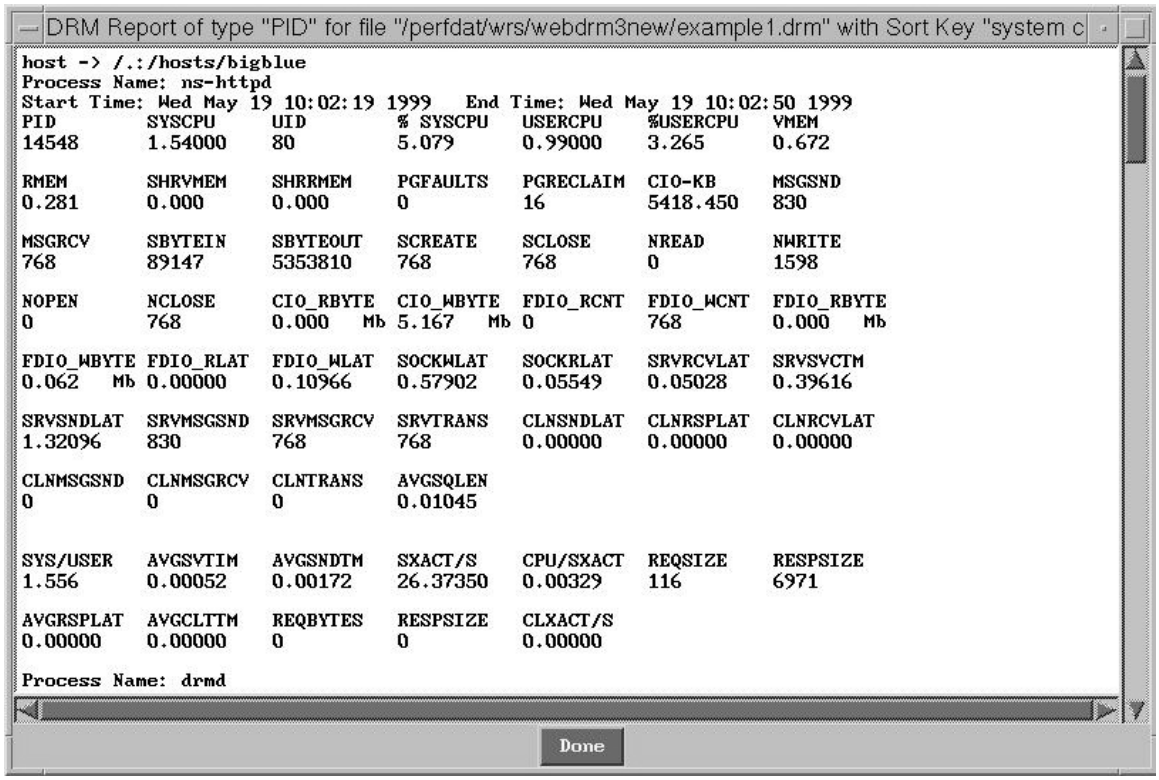

Figure 8

This shows that for the web_client process the file contains information about 768 client transactions (CLNTRANS) which corresponds with the number of server transactions reported in Figure 8. The client transaction rate (CLXACT/S) 26.37/s corresponds with the server rate of 26.37/s and the request size (REQBYTES) of 116 corresponds with the server value of (REQSIZE) 116. It is important to note that the data

has been gathered from two different systems and two different DRM3 agents and that there is excellent correlation in the response time measurements. The final statistic of interest concerning the client is the total round trip time (AVGCLTTM) of 11.5ms. This time and the transaction rate differ from the output of the web_client itself but in consistent fashion. Because the socket layer is below where the timing occurs in the web_client program, there is less time from the sending of the request to the receiving of the response. This means the total round trip time will be longer as measured in the web_client program. Secondly, the average throughput as computed by the program is just the time from when the threads that make the requests are started until the last one has finished divided by the total number of requests. This again adds time to the computation that is not part of the DRM3 measurements. So we see that there is a 5ms difference in average round trip time. What we should do is determine if the client response times are a function of the system on which the client runs or the server it is using or the network. We can illustrate this a little further with an example of additional functionality of the GUI.

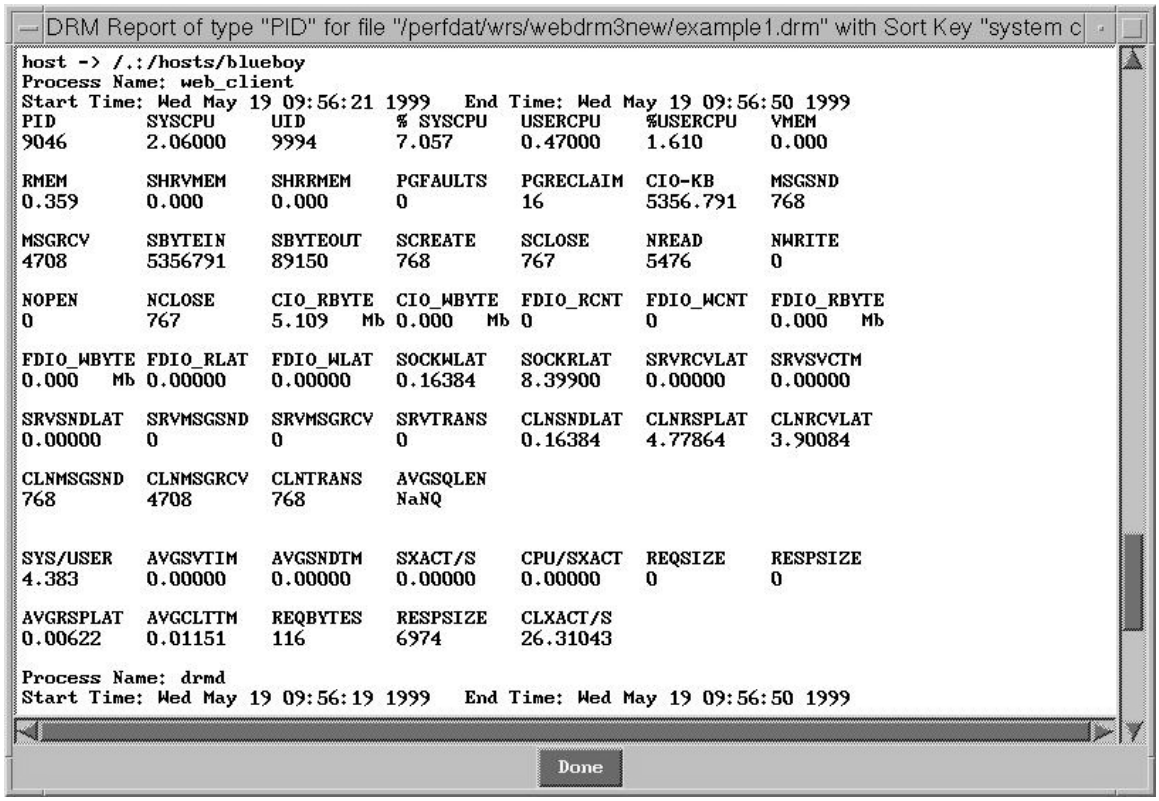

Figure 9

The report data has taken us about as far as it can in this problem so we now want to look at graphs of the different metrics over time. To do this we use the Read button on the File Control window shown in Figure 5 and select Both. We want the data for blueboy so prior to activating this button we should enter blueboy as the hostname in the main window if it isn't already there. Now we get a selection for process names as shown in Figure 10 and we double click on web_client. This causes the GUI to read the data for that process and the system

data in and when it has the data it provides us with the selections for plotting shown in Figure 11 and Figure 12.

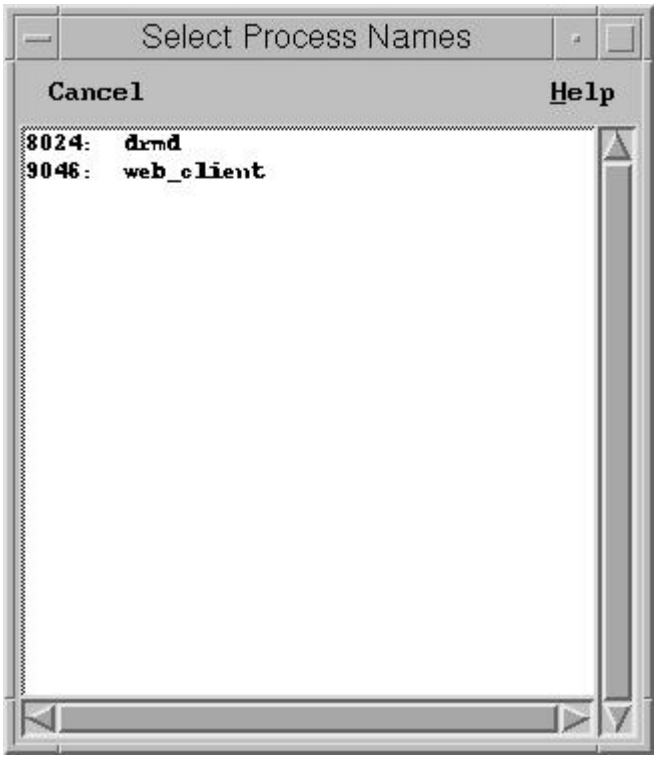

Figure 10

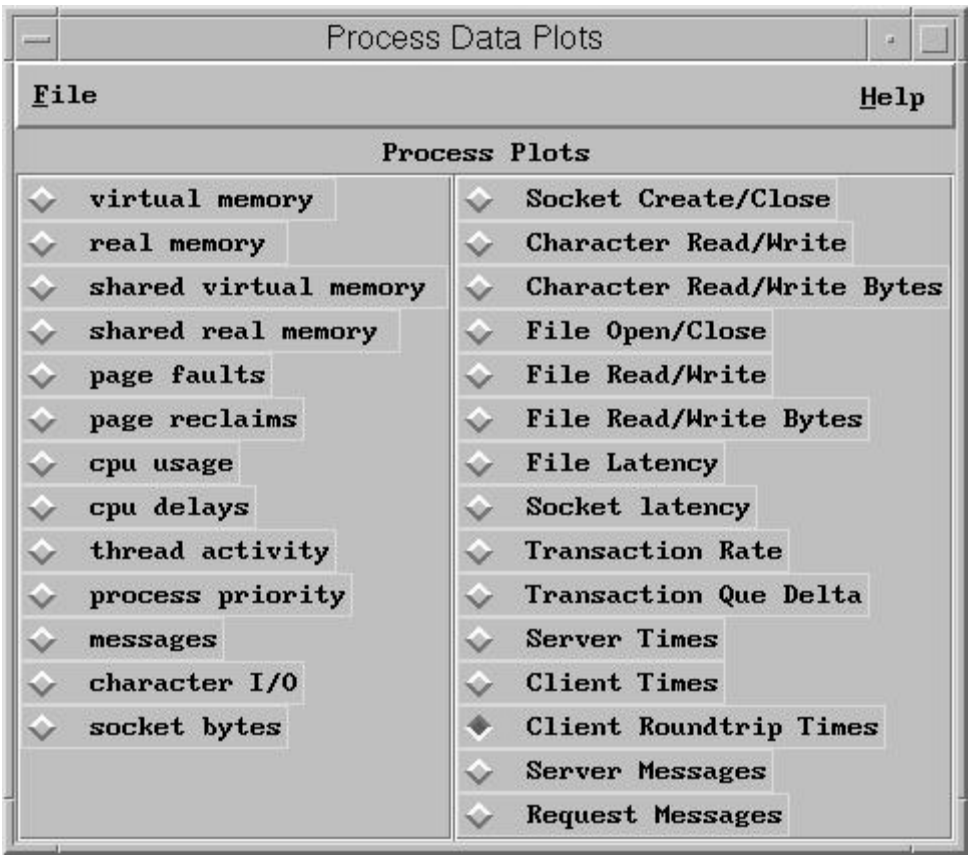

Figure 11

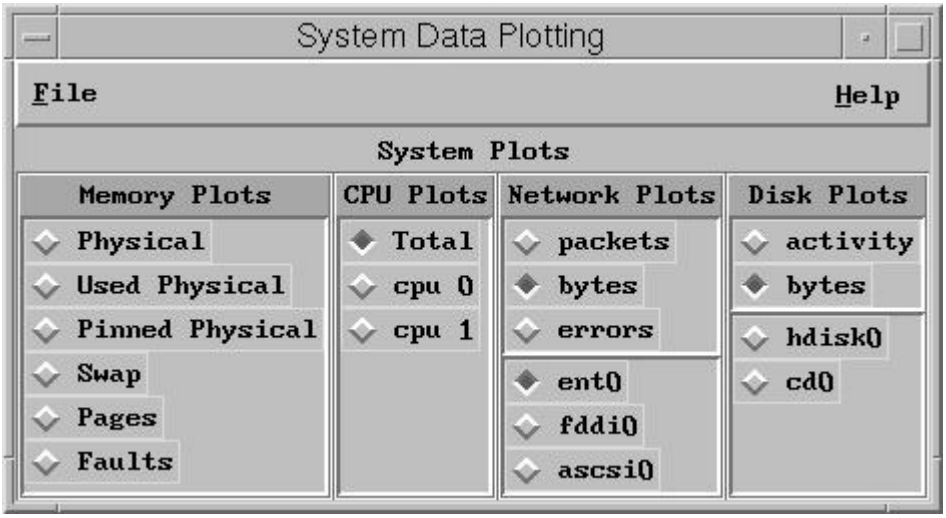

Figure 12

We are interested in looking at the client round trip times as selected in Figure 11. The plotted data is shown in Figure 13.

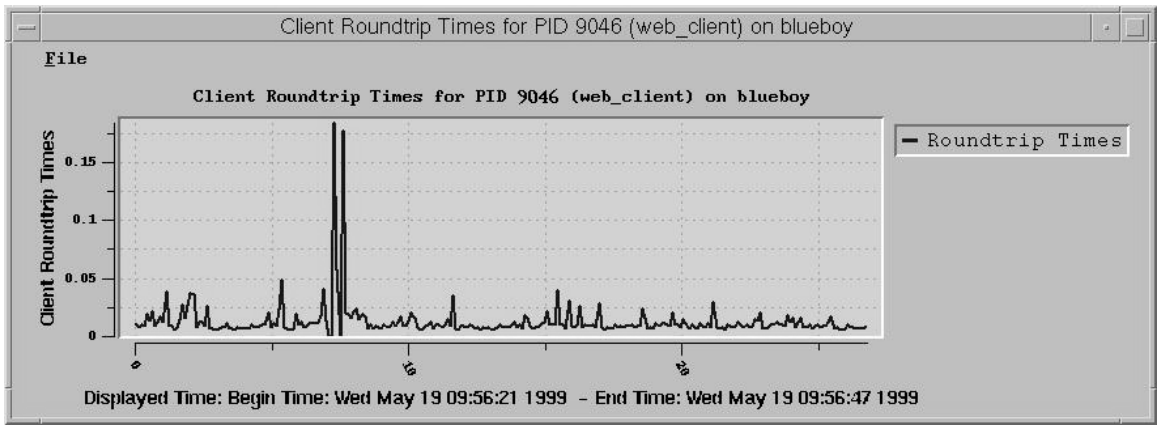

Figure 13

This plot shows the client transaction round trip time as an average. This statistic is computed by summing the three components of the client request-response measurements and dividing by the number of completed transactions in each interval of the data. The three components of the request response measurements are plotted by selecting Client Times and are shown in Figure 14.

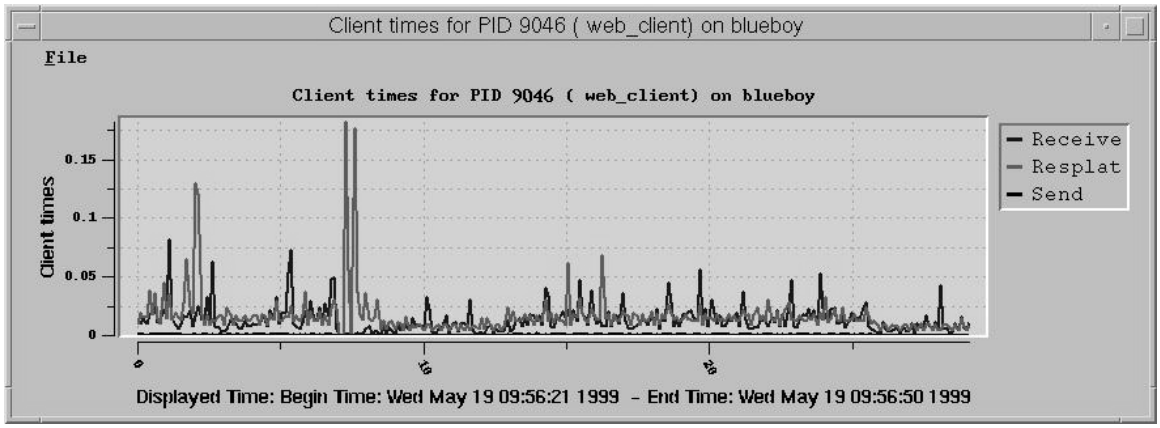

Figure 14

The points in this plot are spaced at .1s. The peaks around 7.5 seconds are in response latency which represents the time from sending the last request message to receiving the first request message. The other components of the plot are send time and receive time. It is clear that for the client the other dominant component of round trip delay is the receive time which shouldn't be surprising based on the report output we generated earlier. In that data we saw an average request size of 116 bytes and an average response size (RESPSIZE) of 6974 bytes. To look at the amount of data being transferred by sockets we activate the socket bytes selection for plotting which produces the plot in Figure 15. This plot indicates that the higher receive times in Figure 14 are correlated to the higher receive data amounts shown in Figure 15. The data being fetched is composed of both html and gif files and so varies in size. This is fairly normal behavior and within what we would expect. However, we still have a spike in latency and round trip time

that isn't explained by any data we have yet plotted. We would like to know what happened to the transaction rates around the 7.5s time as well.

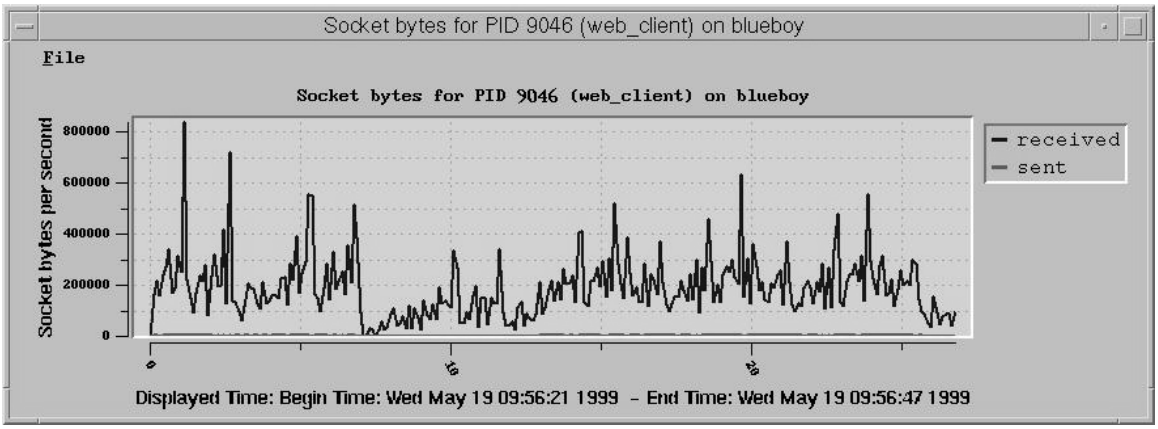

Figure 15

To see the transaction behavior of the process we activate the Transaction Rates button which produces the plot shown in Figure 16. This plot shows the rate at which client and server transactions are completed as well as the rate at which server requests are posted. The web_client process only shows client transactions and since the plotting grid is .1s a 10/s rate corresponds to 1 transaction completed in that reporting interval. What we see is that right about the same time the response latency is high, the transaction rate is zero implying that we are spending more than .1s without completing any transactions.

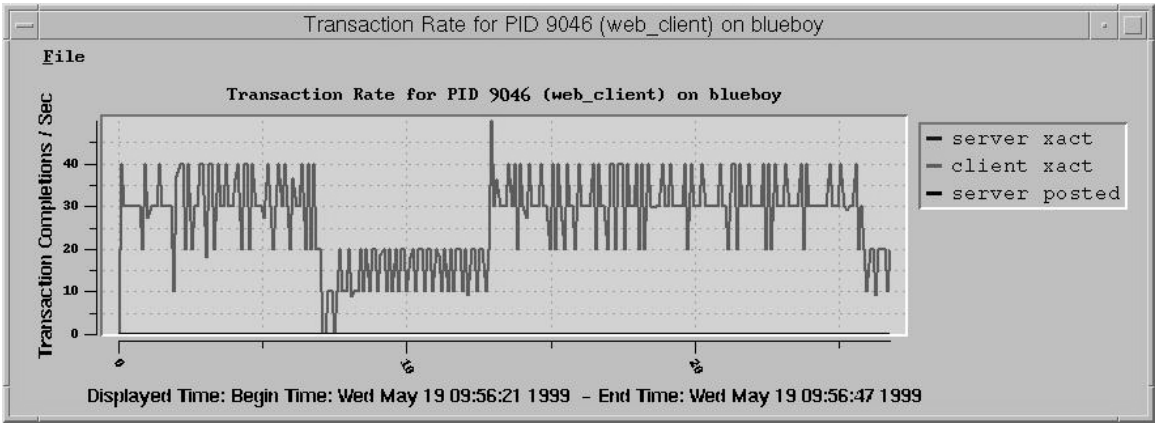

Figure 16

At this point we need to consider that the hindrance to completing transactions and the reason for the long round trip times is outside of the web_client application. We can plot server times for the ns-httpd on bigblue but since the report showed that the total time spent on that system was around 2ms it is better to look further on blueboy. What we want to consider is if CPU is a limitation and then if not consider the network. To plot system CPU time we select Total under

the CPU Plots part of the System Plots control window. This produces the plot shown in Figure 17 from which we see additional CPU activity but no limitation. So now we plot the ethernet interface packet info from the System Plot Window as shown in Figure 18.

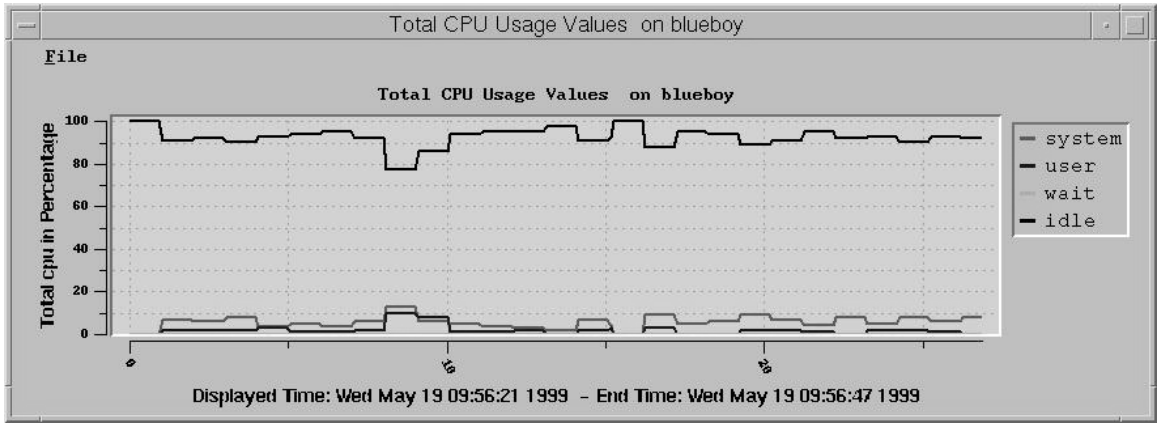

Figure 17

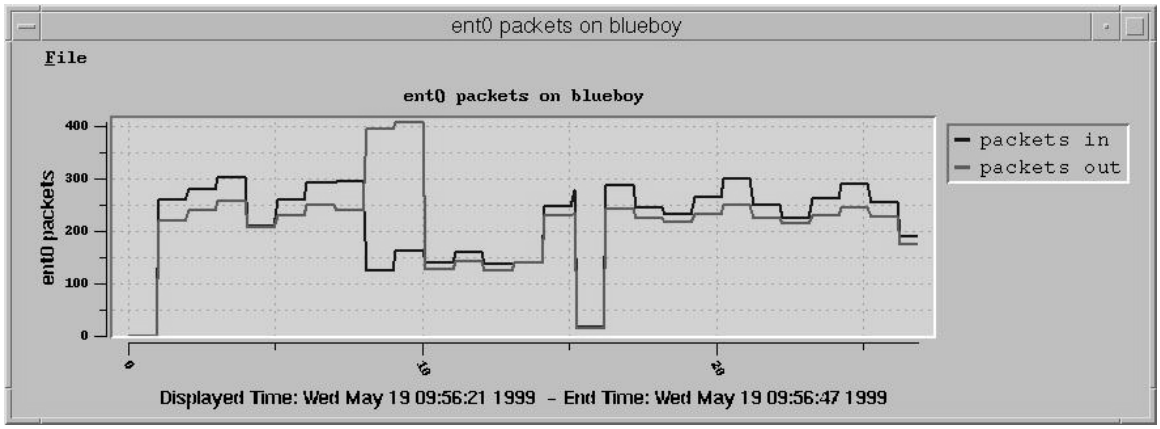

Figure 18

This plot tells us we are getting somewhere with this search. It looks like the number of packets per second increases around the 8s mark. This is correlated to 7.5s data in the process by the following reasoning. The system data is only updated once per second which is why the values have the flat tops shown in the plot. The increase in activity in second 7 which was around 7.5s couldn't show up in the system data plot until second 8. This is commensurate with the additional CPU. What we conclude is that some other process was competing for the network interface. Now we only have data on the drmd but we know two things about it. Number one is that we are collected the data file on bigblue and so there is network traffic associated with collecting this data. The second thing is that this spike looks like the footprint of a 5 minute history dump which can happen due to another user or the Capacity Data Collector getting a 5 minute summary of the activity on blueboy. We want to go back to the main window and clear the process name field and then go to the File Control window and

activate the Read button. Here we want to select Both and when the process selection window appears we double click the drmd. Once the data is available we plot the socket bytes on the drmd which is shown in Figure 19.

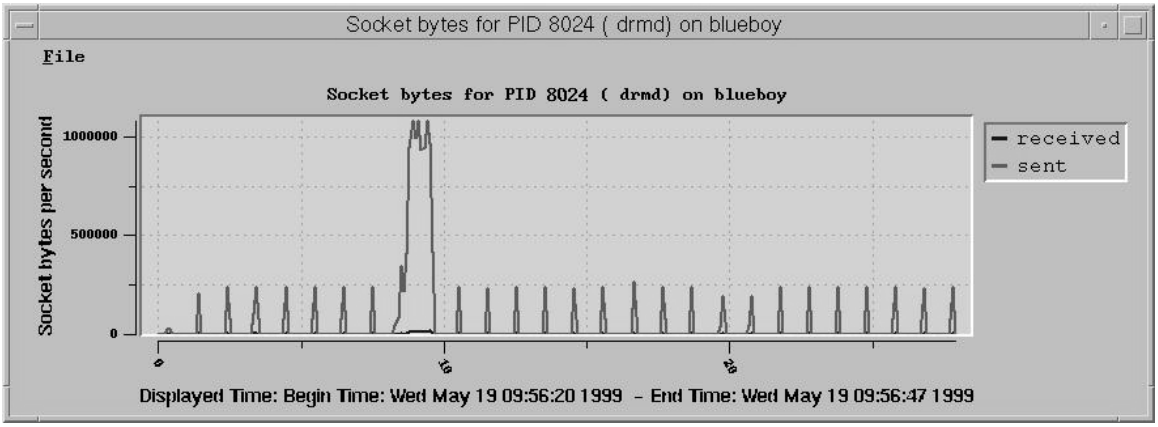

Figure 19

This is what we were looking for, a large burst of activity. Notice that the regular sampling that created the blueboy data is spaced at one second intervals as we previously described. The large burst is between 8 and 10 seconds. Remember that the plots for the web_client showed the long response times at 7.5s so is there an error? No, this is the way the plots were made, the web_client data is plotted from 9:56:21 to 9:56:47 while this plot is from 9:56:20 to 9:56:47. This means that the relative time shown on the x-axis is one second later on this plot than on the web_client plots so the burst overlays correctly with the long round-trip times.

Conclusion

This guide has demonstrated how to use the drm.gui to create a file and analyze it's contents. We hope that you can see the power of granularity of information and our transactional data in this example. There are command line interfaces for creating and post-processing files but we will not address them in this manual. These interfaces are described in the Users Guide provided with the product. A list of the metrics is provided in the final section of this document.

Appendix – Metric Descriptions (labels from dstat)

pri - priority of the main thread in the process for a single threaded process this is the process priority. The schedulable entity in AIX and Solaris is a kernel thread. In AIX there is a one-to-one mapping between kernel and user level threads. In Solaris this can vary but isn't important as far as priority is concerned to DRM.

syscpu - This is the amount of CPU consumed by all threads within the process while running in kernel mode. Kernel mode is when the process has made a system call or is in a device driver.

usercpu - This is the amount of CPU consumed by all threads within the process while running at user level. User level is when the program is running it's own logic or code in a third-party or system library.

NOTE ON CPU:
On AIX this measurement is only an estimate because the CPU accounting in the kernel by the operating system is sampled. Each time the system timer fires (100hz on AIX) the thread/process on the CPU gets one tick (.01s) added to either it's user cpu or system cpu based on what protection environment the thread is running in (user or kernel). On Solaris this measurement is exact because the DRM forces each thread in the system into microstate accounting mode which adds a small amount of extra overhead to make high-resolution time-stamps each time the processes changes mode and accumulate the differences at each mode switch. This can prove to be 30% to 100% more accurate for some types of processes and systems.

vmuse - This is the amount of page space consumed by the process private address space segments. This includes segment 2 on AIX and any additional big-data model segments, which by definition includes bss,stack and any data that isn't read-only(predefined strings are an example of read-only data). On Solaris this includes the bss,stack and data segments.

rmuse - This is the amount of real memory consumed by the same address space as in vmuse plus the private copy of the data section of each shared library the process is using. In addition, text segment real memory for the first instance of a process is added in here. The difference between this and PS is that PS will add text real memory for all reported instances of a process even though it is really shared among all instances (i.e. ksh).

shrvmuse - This is the amount of page space consumed by all shared segments that the process is using. An example of this is anonymously mapped memory that is mapped shared.

shrrmuse - This is the amount of real memory consumed by all shared segments that the process is using. This is reported for all processes sharing the segment. The first instance of a process gets this shared memory usage added to it's real memory count, subsequent instances do not.

pfaults/physio - This is resolved addressing faults that require physical I/O to populate the data in the memory associated with the addressing fault. Page space page-ins would be an example as would be accesses to mapped file regions that are not in memory or have been paged out. This number also includes read ins by the file system when the process performs low-level I/O and the file pages being accessed are not in memory, however, since he file system performs read-ahead, a single page-fault will cover more than a single page fill. The file system read-ahead can read up to 32kb or 8 pages in a single page fault. For writes, this number will vary depending upon the size and rate of the write operations.
Conclusion: USE THIS NUMBER JUDICIOUSLY

reclaim - Resolved addressing faults that are satisfied from the free-list and require no I/O. This includes zero fill, file remapping and so on. This is almost always a one-to-one mapping between faults and page fills.

ciokb - This is the amount of I/O reads and writes through character oriented interfaces using read and write system calls, in kilobytes. A list of some but not all of those interfaces includes file I/O, socket I/O, tty I/O, pty I/O and device driver I/O as well as raw device I/O on disks through the raw logical volume interface.

NOTE: THE REST OF THE METRICS ARE CREATED BY WHAM Engineering & Software and will be the same from platform to platform. These metrics will not be available in any other vendors product.

msgsnd - This is the number of messages sent through the socket level interfaces and also including reads and writes on sockets. A message corresponds to one call of the socket level interface such as send,sendmsg,sendto,and write.

msgrcv - The analog of the messages sent for the corresponding interfaces

bytein/byteout - The number of bytes transferred by the msgrcv and msgsnd stuff.

kthrd – The number of threads available to run on processors as represented in the kernel. The actual number of user threads in the program is usually greater but this represents the greatest number of processors the application could use.

cpua – The number of threads actually running on a processor when this sample was taken. On Solaris this is exact but on AIX there is some uncertainty in this since the state of a thread is less exact than in Solaris. For AIX this is at least the number of threads within the process that have need of a CPU, when this is consistently larger than the number of CPUs on the system, the process will be concurrency limited by CPU's.

nsock – The number of socket cycles i.e. a create and close is a cycle.

actsk – The number of active sockets

nread – The number of read system calls

nwrite – The number of write system calls

cbytein – The number of character io bytes read

cbyteout – The number of character io bytes written

nopen – The number of open system calls

nclose – The number of close system calls

dincount – The number of reads at the vnode interface for file vnodes

doutcount – The number of writes at the vnode interface for file vnodes

dinlat – The amount of time in microseconds for all reads at the vnode interface for file type vnodes. This is a 32 bit counter and will roll over after approximately 4200 accumulated seconds of read time. It cannot be assumed for long running processes to accumulate all read latency.

doutlat – The amount of time in microseconds for all writes at the vnode interface for file type vnodes. This is a 32 bit counter and will roll over after approximately 4200 accumulated seconds of read time. It cannot be assumed for long running processes to accumulate all read latency.

cbytein/out – The number of bytes read/written at the vnode interface from file type vnodes.

wlat/rlat – The write and read latency at the socket interfaces. If a process does blocking socket reads the read latency can be quite high.

svrlat – The amount of time a process spends reading request messages from client requests. This is a 32 bit counter with resolution to microseconds.

svstime – The total service time accumulated for all transactions performed as a server by a process. This is a 32 bit counter with resolution to microseconds. On an individual transaction this is measured as the time from the last message received in a request to the sending of the first message of the response.

svrlat – The amount of time a process spends writing response messages for client requests. This is a 32 bit counter with resolution to microseconds.

smsend/recv – The overall number of messages sent and received by the process as it processes requests from it’s clients.

svxact – The number of transactions processed by a process as a server. A transaction is defined to be a half duplex conversation on a tcp socket connection that starts with a message(s) received by the server and terminates with a message(s) sent to the client. Multiple transactions may take place on a single connection or as in the case of HTTP 1.0, there is only one transaction per connection. For database servers, connections are held open for long periods and usually each transaction will correspond to a specific SQL query.

clmsend/recv – The overall number of messages sent and received by the process as it processes requests made to server(s).

clsrlat– The total time taken sending client side request messages.

clrlat– The total time taken after sending the last client side request message but before receiving the first response message. This is actually the request latency and includes server side time as well as network time. It is useful for determining if long round-trip time is a local or remote phenomenon.

clrtime– The total time taken receiving response messages. This can be a significant component of long round trip times when responses are large.

clxact– The number or client type transactions

System Level Statistics

Disk Information:

Bytes In: bytes written to the disk
Bytes Out: bytes read from the disk
Time Active: Time in ticks the disk was busy -- a tick is .01s

Network Information:

Packets Out: Packets sent on the network throught the device driver interface on AIX and through the network interface on Solaris
Packets In: Packets received by the network through the device driver interface on AIX and through the network interface on Solaris
Bytes Out: Bytes sent on the network throught the device driver interface on AIX and through the network interface on Solaris
Bytes In: Bytes received by the network through the device driver interface on AIX and through the network interface on Solaris
Errors: Specific to the interface type

CPU Informantion:

Per CPU
System CPU: Number of ticks this CPU was doing work in the kernel protection domain. This will include interrupt handling and address fault handling

User CPU: Number of ticks this CPU was doing work in the user protection domain.

I/O Wait: Number of ticks this CPU had available ticks but some thread was blocked waiting on I/O and no others were available for it to run. It is conceivable that some CPU's could be busy and others have I/O wait time.

Idle: Number of ticks this CPU ran the kernel level wait process i.e. doing nothing useful except waiting for a runnable thread.

Memory Information:

Real

Total: The amount of memory available for processes and files and the kernel, this isn't the physical memory size.

Used: The amount used by processes, files and the kernel

Swap:

Total: The total amount of swap space on the system

Used: The amount currently used
Pageins: Page Space page-ins
Pageouts: Page Space pageouts(writes)
Steals: Number of pages stolen by the VMM

Faults:

Major - Address Faults requiring I/O
Minor - Address Faults not requiring I/O