Paper 210-2009

Using SAS® to Analyze System Performance Metrics

Chris Helwig, PhD Student UW-Milwaukee, Phoenix, AZ

ABSTRACT

This paper explains how Microsoft Management Console (MMC) 2.0 can generate flat files containing system metrics, such as memory, disk space, and CPU. System metrics can then be used in SAS® to build data sets to analyze computer performance data. SAS® functions, including PROC SQL, PROC MEANS, PROC FREQ, and PROC TTEST are used. The data can be graphed using PROC CHART and by generating XML that can be displayed on the Web using Flash charts with XML/SWF Charts technology.

INTRODUCTION

This project involved using Microsoft Management Console 2.0 (Perfmon) to generate flat files containing system metrics, including memory, disk space, and percent CPU utilization, which were then used in SAS(R) to build data sets in order to analyze the performance data. SAS(R) functions including SAS(R) proc SQL, proc means, proc freq, proc ttest and proc reg were utilized. Graphing of the data was performed, both using proc chart and by generating XML that was displayed on the web using XML/SWF Charts Flash technology.

Perfmon was used to pull data files, collecting data at 5 second intervals for 30 minutes. Four such files were pulled, one with the system idle, one with McAfee virus scan software running, one with Dragon voice recognition software running, and one with Netflix software playing a movie. The system used for the testing was a Pentium 4, 2.39 GHz processor, with 768 MB RAM, running Windows XP SP2, with a 37.2 GB hard drive.

METHOD

Perfmon is launched by opening a command window and entering perfmon on the command line. The console root of the performance monitoring tool has two tabs, one is the system monitor and one for performance logs and alerts. Under performance logs and alerts, counter logs was highlighted, then by right-clicking on it a new log settings was selected, then under Properties | General the counters were selected.

The first task in terms SAS(R) programming was to read the data into a data set. Because the Perfmon data was in tab delimited format in a text file it was possible to go to the file menu and use the import option on the file menu of SAS(R) and follow the wizard to import the data and create a new data set, after first using FTP to transfer the .txt files from Windows to Unix.

Another task is to use the SAS(R) proc means function in order to generate summary statistics for disk space, CPU utilization, and memory utilization. The summary statistics include minimum and maximum values, standard deviation, and mean values.

The Perfmon file had 153 columns of data. A proc contents command was first used to obtain the names of the variables of most interest. Then the new data set named metrics was initialized and the variables renamed to CPU, disk_space, and memory in order to simplify the analysis. A proc print command was also run to visually inspect the data.

Before running the proc means function it was necessary to convert the character data into numeric format with the input command. The perfmon data had an initial row with column header information which was in text format, therefore SAS(R) initialized all the data in the column as character formatted. After converting the data the proc means function was run and generated the summary statistics shown below.

```
 data metrics;
set metrics_temp;
keep  __DFRP4M31_Processor__Total___
      __DFRP4M31_LogicalDisk__Total__
      __DFRP4M31_Memory___Committed_B
;
rename __DFRP4M31_Processor_0____Proce  = cpu
       __DFRP4M31_LogicalDisk__Total__   = disk_space
       __DFRP4M31_Memory___Committed_B = memory
;
```

```
run;

data metrics_numeric;
set metrics;
cpu_numeric = input (cpu, 3.6);
disk_space_numeric = input (disk_space, 3.6);
memory_numeric = input (memory, 3.6);
run;

proc means data=metrics_numeric;
var cpu_numeric disk_space_numeric memory_numeric;
output out=testout;
run;
```

System Idle Data					
	N	Mean	Std Dev	Minimum	Maximum
cpu_	367.0	1.8	8.2	0.0	99.0
disk_space_	367.0	12.0	0.0	12.0	12.0
memory_	367.0	46.7	0.4	46.0	48.0
Virus Scan Data					
	N	Mean	Std Dev	Minimum	Maximum
cpu_	366.0	58.4	25.6	0.0	99.0
disk_space_	366.0	12.0	0.0	12.0	12.0
memory_	366.0	54.4	1.1	49.0	59.0
Dragon Voice Recognition Data					
	N	Mean	Std Dev	Minimum	Maximum
cpu_	414.0	10.3	20.4	0.0	99.0
disk_space_	414.0	12.0	0.0	12.0	12.0
memory_	414.0	60.3	0.7	58.0	62.0
Netflix Data					
	N	Mean	Std Dev	Minimum	Maximum
cpu_	362.0	39.2	17.7	0.0	99.0
disk_space_	362.0	11.3	0.5	11.0	12.0
memory_	362.0	56.6	0.6	53.0	57.0

Figure 1. Summary Statistics

RESULTS

What do these summary statistics tell us? Not surprisingly CPU is lowest when the system is idle (1.7%), the virus scan is the most CPU intensive application at 58% followed by Netflix at 39% and voice recognition software at 10%. Also each data category had instances where CPU spiked to 99%, this is a significant data point because applications can behave erratically when CPU is maxed out.

For memory usage when the system was idle memory usage was the lowest at 47%, the highest usage was seen with voice recognition software at 60% followed by Netflix at 57% and voice recognition at 54%. The highest spikes of up to 62% were seen with Dragon voice recognition software.

Disk space was flat at 12% free space because these applications did not write any files to disk, with the exception of Netflix, which wrote some (from 11% to 12%) data to disk. The highest standard deviation of 26 was seen with CPU during the virus scan indicating that CPU usage during the virus scan application was most variable. Also interesting was the fact that memory was at a minimum of no less than 46% even when the system was idle, indicating that the operating system and background processes consume 46% of available memory all by themselves.

In order to visualize the data more clearly it is helpful to graph it. In order to generate graphs of our data SAS(R) chart commands were used, as well as XML files using SAS(R) data, which was then displayed graphically on the web using flash technology.

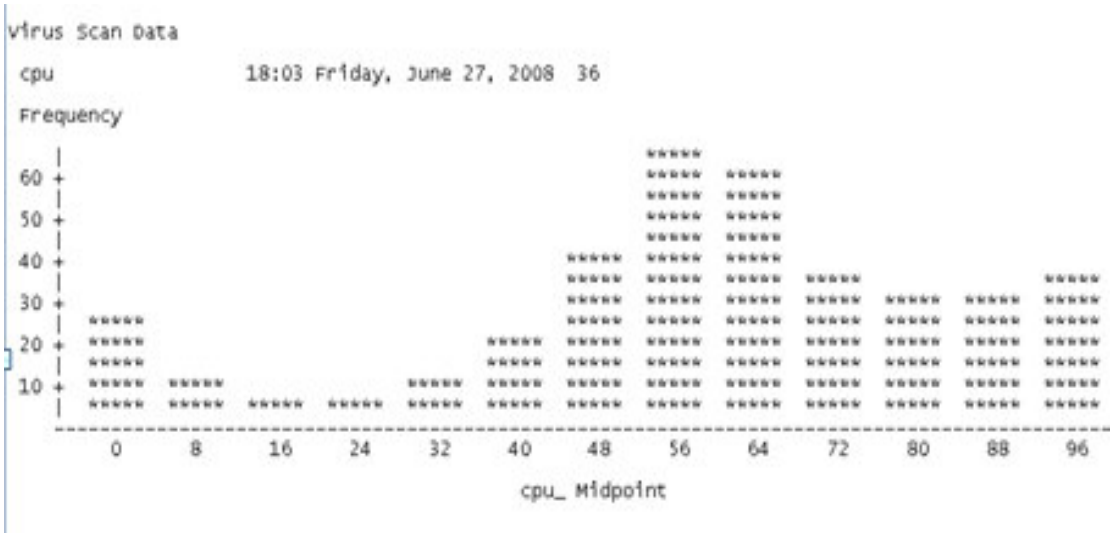

Figure 2. CPU Levels While Rrunning Virus Scan

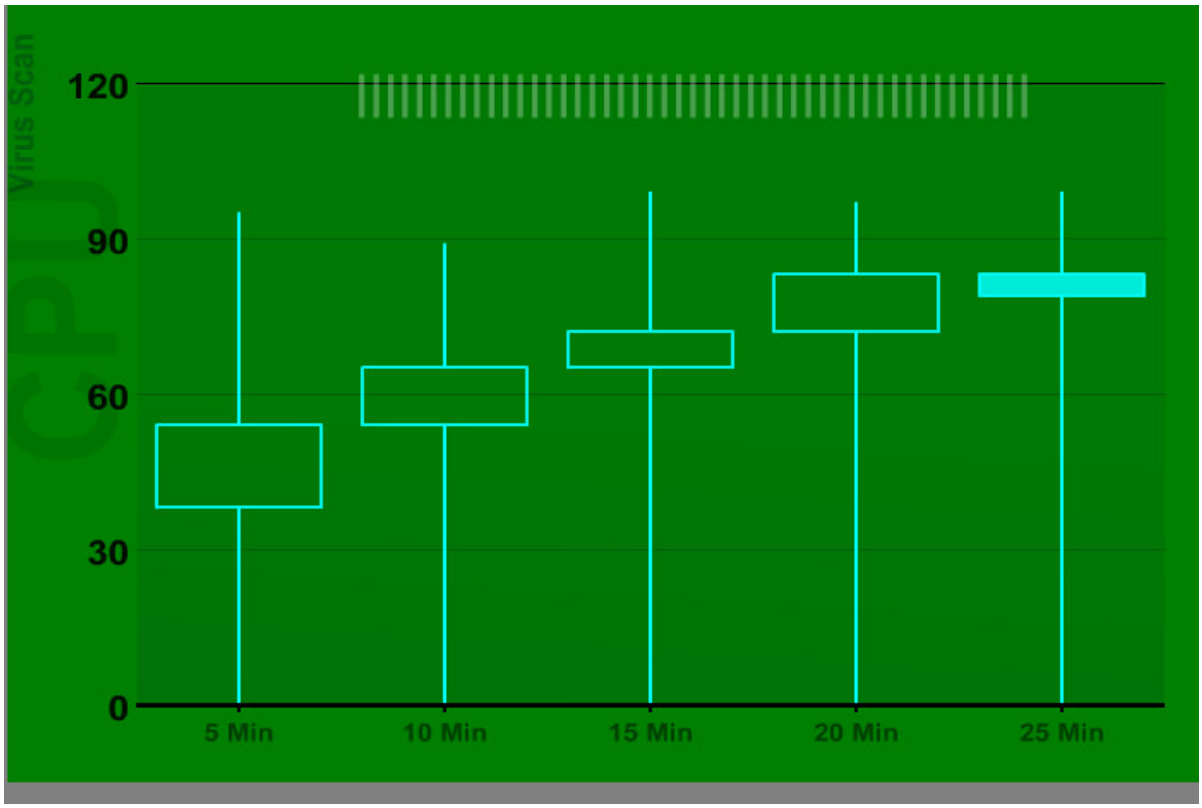

Figure 3. XML/SWF Charts Web Graph Showing CPU Utilization During the Virus Scan

In order to generate readable graphs it was not be possible to graph every data point since our data was pulled at five second intervals, this would create graphs that were too messy. The following code was used to pull out data points needed to create candlestick graphs. These graphs produce candlestick shaped graphs at each five minute time window, each one displaying data pulled 2.5 minutes before and after the five minute window number displayed on the x axis of the graph.

```
data gr;
set one_;
drop apu_;
retain max_1 max_2 max_3 max_4 max_5 ;
if _n_ > 30 and _n_ < 90 then
  if cpu_ > max_1 then max_1=cpu_;
if _n_ > 90 and _n_ < 150 then
  if cpu_ > max_2 then max_2=cpu_;
if _n_ > 150 and _n_ < 210 then
  if cpu_ > max_3 then max_3=cpu_;
if _n_ > 210 and _n_ < 270 then
  if cpu_ > max_4 then max_4=cpu_;
if _n_ > 270 and _n_ < 330 then
  if cpu_ > max_5 then max_5=cpu_;
 run;
data gr2;
set one_;
drop apu_;
retain t30 t90 t150 t210 t270;
retain min_1 min_2 min_3 min_4 min_5;
if _n_ = 30 then min_1=cpu_;
if _n_ = 30 then t30=cpu_;
if _n_ > 30 and _n_ < 90 then
if cpu_ < min_1 then min_1=cpu_;
if _n_ = 90 then min_2=cpu_;
if _n_ = 90 then t90=cpu_;
if _n_ > 90 and _n_ < 150 then
if cpu_ < min_2 then min_2=cpu_;
if _n_ = 150 then min_3=cpu_;
if _n_ = 150 then t150=cpu_;
if _n_ > 150 and _n_ < 210 then
if cpu_ < min_3 then min_3=cpu_;
if _n_ = 210 then min_4=cpu_;
if _n_ = 210 then t210=cpu_;
if _n_ > 210 and _n_ < 270 then
if cpu_ < min_4 then min_4=cpu_;
if _n_ = 270 then min_5=cpu_;
if _n_ =270 then t270=cpu_;
if _n_ > 270 and _n_ < 330 then
if cpu_ < min_5 then min_5=cpu_;
run;
proc print data=gr2;
run;
```

DISCUSSION

The XML/SWF Charts package utilizes an XML configuration file as well as a template HTML file (sample files reproduced with full presentation) to generate a variety of graphs that can be displayed on the Internet. It is not necessary to run your own web server to use it, it can be used by copying over SWF library files to your ISP's user web space. The output of SAS(R) code written to extract maximum and minimum values was used to configure an XML file to generate the candlestick graphs reproduced above.

The candlestick graph is used primarily for charting stock price movement over time but it is also a useful graph for analyzing system performance. The vertical line at each time interval represents the low and high values while the box represents the starting and ending values for each time window. If the box is shaded the value moved down, if un-shaded it moved up during that time interval.

T TEST

Another thing we can do is to analyze what level of sampling frequency would be sufficient to make sure we accurately capture the mean values for CPU, Disk Space, Memory, and other metrics. The following SAS(R) code was used to run a T test to determine whether a 1 minute sampling interval would be sufficient, comparing that to the 5 second rate we started with.

```
proc sort data=one_;
by cpu_;
run;
data cpu_ttest;
set one_ (keep=cpu_);
by cpu_;
retain total_over_90;
if MOD (_n_, 12) = 1 then every_minute=1 ;
else every_minute=0;
if cpu_ > 90 then over_90_flag = 1;
else over_90_flag=0;
total_over_90 = sum(total_over_90, over_90_flag,0);
percent_over_90=total_over_90 / _n_;
run;

proc print data=cpu_ttest;
run;

data cpu_ttest;
set cpu_ttest 2;
if cpu_>90;
run;

proc ttest data=cpu_ttest;
class every_minute;
var cpu_;
run;
```

This code generated the following output:

Obs	cpu_	total_ over_90	every_ minute	over_90_ flag	percent_ over_90
361	99	33	1	1	0.09141
362	99	34	0	1	0.09392
363	99	35	0	1	0.09642
364	99	36	0	1	0.09890
365	99	37	0	1	0.10137
366	99	38	0	1	0.10383

Statistics

Variable	every_minute	Lower CL N	Mean	Upper CL Mean	Lower CL Mean	Std Dev	Std Dev
cpu_	0	335	55.743	58.483	61.223	23.698	25.493
cpu_	1	31	47.859	57.677	67.496	21.39	26.767
cpu_	Diff (1-2)		-8.645	0.8059	10.257	23.868	25.6

	Upper CL Std Dev	Std Err
cpu_	27.585	1.3928
cpu_	35.779	4.8076
cpu_	27.606	4.806

T-Tests

Variable	Method	Variances	DF	t Value	Pr > |t|
cpu_	Pooled	Equal	364	0.17	0.8669
cpu_	Satterthwaite	Unequal	35.2	0.16	0.8730

The TTEST Procedure

Equality of Variances

Variable	Method	Num DF	Den DF	F Value	Pr > F
cpu_	Folded F	30	334	1.10	0.6591

The PR > F value of .6591 indicates that the equal variance assumption is met and the Pr > |t| value of .8669 indicates that we cannot reject the null hypothesis that the means of the two data sets are equal. If the value had been less than .05, we would conclude that our sampling frequency was too long to accurately reflect the population.

PROC REG

The SAS(R) proc reg command was used to run a regression analysis. One column of data was CPU and a second column showed whether or not the virus scan was running. This produced a reasonably accurate R square value of .81. This gives us a way to estimate the impact on CPU of running the virus scan software.

```
proc reg data=d;
M1:  model cpu=flag / P R;
run;
```

Number of Observations Read	731
Number of Observations Used	731

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	1	710564	710564	3112.42	<.0001
Error	729	166431	228.29978		
Corrected Total	730	876994			

Root MSE	15.10959	**R-Square**	0.8102

Dependent Mean	32.68553	**Adj R-Sq**	0.8100
Coeff Var	46.22716		

Parameter Estimates

Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	**1**	1.55054	0.78979	1.96	0.0500
flag	**1**	62.35528	1.11770	55.79	<.0001

The REG Procedure
Model: M1
Dependent Variable: cpu

Output Statistics

Obs	Dependent Variable	Predicted Value	Std Error Mean Predict	Residual	Std Error Residual	Student Residual	-2- 1 0 1 2	Cook's D
1	0.3115	1.5505	0.7898	-1.2390	15.089	-0.0821	| | |	0.000
2	0.6250	1.5505	0.7898	-0.9255	15.089	-0.0613	| | |	0.000
3	0.6250	1.5505	0.7898	-0.9255	15.089	-0.0613	| | |	0.000
4	0.6250	1.5505	0.7898	-0.9255	15.089	-0.0613	| | |	0.000
5	0.3125	1.5505	0.7898	-1.2380	15.089	-0.0820	| | |	0.000

[additional records omitted]

368	45.9375	63.9058	0.7909	-17.9683	15.089	-1.191	| **| |	0.002
369	75.9375	63.9058	0.7909	12.0317	15.089	0.797	| |* |	0.001
370	53.1250	63.9058	0.7909	-10.7808	15.089	-0.714	| *| |	0.001
371	45.6250	63.9058	0.7909	-18.2808	15.089	-1.212	| **| |	0.002
372	33.4375	63.9058	0.7909	-30.4683	15.089	-2.019	| ****| |	0.006

Figure 4. Output of Proc Reg

CONCLUSION

The CPU_system_idle graph shows that starting and ending values were always zero. There is actually a hidden horizontal line at each time interval where a mouse over reveals the zero value. During the 15 minute time window CPU rose to as high as 74%.

The CPU for the virus scan shows CPU consistently having a wide range spanning from zero to over 90 for the starting and ending values. Also CPU values are increasing, starting at the five minute window with 38% as the starting value and 54% for the ending value for that time window. However at the 25 minute interval the shaded box indicates the starting value was 83% while the ending value was 79% for that time window.

The Dragon software started and ended mostly under 40% CPU, but in two time windows spiked to over 75%. Netflix exhibited CPU spikes of over 90% in two time windows, but mostly started and ended each time window in the 30% to 60% range.

Disk space was flat for all applications tested, showing a horizontal line at 12 for each time window for each application, except Netflix, which shows a downward trend for available disk usage starting at 12.71% available and ending at 11.11%.

The memory graphs show memory hovered around 46 to 47 when the system was idle, ran at about 54 for the Virus

Scan with spikes as high as 59, ran even higher for voice recognition in the 60 to 61 range, and a little lower at 56 to 57 for the Netflix software.

In general the graphs are useful for helping visualize the data, assist with spotting trends, and help identify spikes that may warrant further investigation.

REFERENCES

- Adobe Flash, previously called Shockwave Flash and Macromedia Flash, is a set of multimedia technologies developed and distributed by Adobe Systems and earlier by Macromedia. http://en.wikipedia.org/wiki/Adobe_Flash See also http://www.maani.us/xml_charts/.
- For SAS(R) coding, the textbook Sharpening Your SAS(R) Skills, by S. Gupta, 2005, was referenced.
- SAS(R) Software 123 by Joanne Peng, IU Press 1998.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: Chris Helwig
Enterprise: Student, UW-Milwaukee
Address: 26905 N. 22nd. Ave.
City, State ZIP: Phoenix, AZ
Work Phone: 623-242-7648
E-mail: chelwig1@cox.net
Web: http://www.cs.uwm.edu/~cchelwig/