Development of a Data Acquisition Program for the Purpose of Monitoring Processing Statistics Throughout the BaBar Online Computing Infrastructure's Farm Machines

Paul Stonaha

Office of Science, Energy Research Undergraduate Laboratory Fellowship

Alfred University

Stanford Linear Accelerator

Menlo Park, California

August 11, 2004

Prepared in partial fulfillment of the requirements of the Office of Science, Department of Energy Research Undergraduate Laboratory Fellowship under the direction of S. Luitz and M. Wittgen in SLAC Computing Services (SCS) at Stanford Linear Accelerator.

Participant:

Paul Stonaha _____

Research Adviser:

Matthias Wittgen _____

Research Adviser:

Steffen Luitz _____

Contents

1	Introduction	1
2	Materials and Methods	2
3	Results	3
4	Discussion and Conclusions	4
	References	6
A	Acknowledgments	6
B	Figures	7

Development of a data acquisition program for the purpose of monitoring processing statistics throughout the BaBar Online Computing Infrastructure's farm machines. Paul Stonaha (Alfred University, Alfred, NY 14802) Steffen Luitz(SLAC, Menlo Park, CA, 94025) Matthias Wittgen(SLAC, Menlo Park, CA, 94025)

Abstract

A current shortcoming of the BaBar monitoring system is the lack of systematic gathering, archiving, and access to the running statistics of the BaBar Online Computing Infrastructure's farm machines. Using C, a program has been written to gather the raw data of each machine's running statistics and compute various rates and percentages that can be used for system monitoring. These rates and percentages then can be stored in an EPICS database for graphing, archiving, and future access. Graphical outputs show the reception of the data into the EPICS database. The C program can read if the data are 32- or 64-bit and correct for overflows. This program is not exclusive to BaBar and can be easily modified for any system.

1 Introduction

The BaBar experiment at Stanford Linear Accelerating Center (SLAC) is a collaboration of approximately 500 scientist from 12 countries. These scientists work together to study CP-violation by observing the decay of B-mesons and $\bar{B}$ -mesons [1]. Along with B and $\bar{B}$ pairs, other interesting results can be generated from the electron $\leftrightarrow$ positron collisions at SLAC.

The 180 computers in the BaBar Online Computing Infrastructure are constantly sending information such as available memory, CPU cycles, incoming bytes, etc. from clients on each computer to a central monitoring server. In addition to receiving the information regarding the running conditions of the clients, the monitoring server must also archive the data for future reference.

As the clients are sending data pertaining to their vital statistics, the monitoring server is receiving the information in 5 second intervals.

Currently, there is no systematic archiving of the data being carried out; the data that are being collected from the clients is being sent to a central monitoring system. These raw data (available memory, CPU cycles, data transfered, data read/written, etc) can be used to compute rates and percentages such as cycles per second, bytes sent per second, and percentage of memory remaining, to name a few. Currently, there is no tool to display these rates and percentages.

To correct this, the incoming data can be used to compute these various rates and percentages of data. The computations can be done the moment that the information is collected. After the calculations are made, the data will be put into an Input/Output Controller (IOC), where each client will be represented by one channel within the IOC, which runs EPICS (Experimental Physics and Industrial Control System) software [2]. From there, the data can be accessed using the EPICS display interface.

These data of the systems, CPUs, networks, and disks (and access thereto) are important for the BaBar experiment to maintain its data storage capabilities as luminosity levels as SLAC continue to improve output. If a client is sending erroneous data, receiving/sending no network information, soon to have no more free memory, or any other problem, that must be known so that proper action can be taken. This project will allow easy and efficient access to the data in question.

2 Materials and Methods

A program was written in C [3] to gather data from the clients. Along with gathering the data, it is also in the C program that the data are used to compute various rates and percentages. After the C program makes these computations, it sends the data to an EPICS database, where the data are accessed for viewing and stored for archiving.

To begin, the program opens the server port by address and binds the data socket to that port [4]. After the port is opened, the program begins looping through all of the servers connected the the socket, receiving data from each one every five seconds.

The first event to occur with any one server is the retrieval of the name of the server sending the data. If the program cannot find the name of the client, or it finds that the client doesn't have a name, the program will loop back to the beginning, and start with a new client. Once it is known that the client's name in fact exists, the program first compares that client's name to two separate arrays. The first array is a list of clients to whom the program can send data (CLIENT LIST); the second is a list of hosts from which the program can retrieve data (HOST LIST). If the name does not compare to both lists, the name is first compared to a file with acceptable names for the CLIENT LIST, and if the name matches, it is added to the CLIENT LIST and the HOST LIST. If the client's name does not match the acceptable list of names for CLIENT LIST, the program will simply ignore it and start anew with the next client. Once the client has been added to both lists, a channel name is created for each datum that will be computed. These channels are used to directly translate the data into the EPICS database.

Information from the client can pertain to the system, CPU, network, or disk (available memory, cycles, I/O bytes and packets, and bytes and packets read and written). The information is collected in the order listed above and used for computations immediately after its collection. The information is **scaler** values, which are values that have been integrated over the entire time since the system was booted. For example, to compute the rate of incoming bytes, the difference between the current total number of bytes received and the last total number of bytes received is divided by the amount of time passed. These computations are then stored as elements of an array. The

last operation done by the C program is to transmit the data from the array of computed rates and percentages to the EPICS database; once completed, the program loops back to the beginning and selects a new client.

Once the information is transferred to the EPICS channels, further computations can be easily done. From the rate of incoming bytes and packets, for example, one can compute the number of bytes per packet. BaBar specific software then extracts these computed values from their respective channels and stores them in a database for future archiving. In addition to archiving, the data can also be used to produce graphic representations of the computing processes using the StripTool program.

3 Results

In this project, the data from the clients could be successfully obtained, and the various rates and percentages therefrom could be properly calculated. Figure 1 shows the four different structures from which raw data are obtained, as well as the computations that are done from those incoming data. Many of these numbers, such as the header, average number of running processes in the past 1, 5 and 15 minutes, incoming and outgoing network errors, and collisions are not used in any computations.

Figure 2 shows a tree diagram of the process through which the program runs. The program will exit on certain conditions—if the port cannot be bound or if the file with a list of client names cannot be opened.

Figures 3 and 4 display the typical output of the program once it has received the raw scaler values and after the data from those values are converted into rates and percentages and sent to their proper channels, respectively.

4 Discussion and Conclusions

In this project, it was found that the data from the clients were not all sent as exclusively 64-bit or 32-bit; network data were 32-bit, while system, CPU and disk data were 64-bit. This can be a serious problem, because an event will occur when the amount of data exceeds $2^{32} - 1$ bits. It does not take long for data from the network to exceed this number (approx. 4.295×10^9), and when it does, the information will be reset to zero and continue from that point. When computing a rate, it will appear as though the current value is less than the previous value, and the computation $value - value_{last}$ will be less than zero. Hence, the following corrections must be included:

$$\begin{aligned}
 & if(val < val_{last}) \\
 & if(abs(0xffffffff - val_{last}) < 2000000000) \\
 & \quad val = 0xffffffff - val_{last} + val \\
 & \quad val_{last} = 0
 \end{aligned} \tag{1}$$

This ensures that if the last datum caused an overflow and was less than 2 Gigabytes, the recorded value would be set equal to the amount that the raw datum changed, while the last value would be set to zero. The data that are set to 64-bit most likely never will experience this problem because of the sheer size of 64 bits; however, I have included a similar correction factor in case such an event does occur.

Figure 1 displays the format in which the data are obtained, and the rates into which those data are calculated. In the 'system' structure, *freemem* and *maxmem* are used to compute available physical memory, *freeswap* and *maxswap* are used to compute available swap memory, and all four are used together to compute available virtual memory. The data in the system structure are not scalars, unlike the other three structures. From the 'CPU' structure, all three elements are used to compute the percentage of all cycles in the last 5 seconds that had been system, user, or idle cycles.

The network structure uses *ibytes*, *ipackets*, *obytes*, and *opackets* to compute the rates of incoming KBytes and packets and outgoing KBytes and packets, while the other elements are not used. For the 'disk' structure, all elements are used in the computations.

In addition to computing rates such as bytes per second in the C program, the EPICS software computes statistics such as CPU usage, which is calculated with $CPU = 100 - cycles_{idle}$, bytes per packet, and the asymmetry of a system. In a two CPU system, the asymmetry, A_s , follows the equation $A_s = \frac{CPU_1 - CPU_2}{CPU_1 + CPU_2}$. When this unit-less number is zero, the two CPUs are running at equal load. (In a system of more than two CPUs, the numerator must be set equal to the sum of all possible differences.)

Figure 5 shows the reception of network data from 8 clients into the EPICS database. The green line, which begins at 3.4 minutes, corresponds to a client that was added to the graph after the others. Figure 6 reveals a graph of the number of KBytes read from and written to a clients disk. During the graphed process, the machine was writing data and then read the data back to confirm the process. The last graph, figure 7, displays the CPU usage for two CPUs in one system, with the asymmetry for those two CPUs graphed also. When the two CPUs are running at around the same load, the line graph of asymmetry should be around 0, which, because of a change of scales, would appear as 50 on the graph. These graphs were all created by the StripTool program.

The program I have written for this project is not limited to the use of the BaBar team alone. This program can be easily modified to accommodate any series of clients. It's primary use will be to identify bottlenecks and systems that are running low on memory, and beyond that it can be used for packet size analysis, asymmetry analysis, and other operations. If I had had more time on this project, the only additional modification I would make is for an alarm to sound when memory becomes too low or when network traffic produces a collision.

References

- [1] B. Aubert *et al.* [BABAR Collaboration], “The BaBar detector,” Nucl. Instrum. Meth. A **479**, 1 (2002)
- [2] J. Anderson, M. Kraimer, and P. Stanley *Record Reference Manual*, The University of California, 1995.
- [3] H.M. Deitel and P.J. Dietel C++ How to Program Upper Saddle River, NJ: Prentice Hall. 1998
- [4] W.R. Stevens UNIX Network Programming Englewood Cliffs, NJ: Prentice Hall. 1990

A Acknowledgments

I would like to thank the DOE for allowing my participation in this year’s SULI program at Stanford Linear Accelerating Center. I have gain immense knowledge from the outside activities that the department has funded, such as the tours and lectures.

I would also like to thank my mentors Steffen Luitz and Matthias Wittgen for their support and guidance in this project. Before beginning this project, I had never once written a program in either C or C++, but now I feel I can easily handle myself in those computing enviroments.

Thirdly, I would like to thank Helen Quinn and Pauline Wethington for their work in organizing the Summer Student Program in which I was able to take part.

And last, I would like to thank Roberto Vega for his time and energy in ensuring that the students in the SULI program had a safe and enjoyable time.

B Figures

Figure 1: Structure of the incoming data and the calculations therefrom

Figure 2: Flowchart of the C program

-----SYSTEM-----		-----NET-----	
header	: 4058174404	ipackets	: 967498670
timestamp	: 1091817661	opackets	: 1708331709
boottime	: 1084297811	ierrors	: 0
nproc	: 97	oerrors	: 0
ncpu	: 2	ibytes	: 3546501509
nnet	: 2	obytes	: 661073230
ndisk	: 2	collisions	: 0
freemem	: 27832	-----NET-----	
maxmem	: 1030248	ipackets	: 2551184009
totalswap	: 2096376	opackets	: 418363552
freeswap	: 2091304	ierrors	: 136214
avenrun_1min	: 0	oerrors	: 0
avenrun_5min	: 0	ibytes	: 1943834976
avenrun_15min	: 0	obytes	: 1596916210
-----CPU-----		collisions	: 0
idle	: 631991964	-----DISK-----	
user	: 100164994	read	: 262324328
system	: 19822947	write	: 864570422
-----CPU-----		reads	: 1426031
idle	: 620125545	writes	: 2694057
user	: 118311814	-----DISK-----	
system	: 13543952	read	: 262036838
		write	: 864570422
		reads	: 1426403
		writes	: 2694570

Figure 3: Sample output during raw data retrieval

```

received data from bfo-farm102
loop number 19
server number 20
time step : 10
Data put      bfo-farm102:NCPU          2
Data put      bfo-farm102:NNETWORK      2
Data put      bfo-farm102:NDISK          2
Data put      bfo-farm102:MEM_USAGE      2.70149   %
Data put      bfo-farm102:SWAP_USAGE     99.7581    %
Data put      bfo-farm102:TOTAL_USAGE    67.7771    %
Data put      bfo-farm102:IDLE1          100        %
Data put      bfo-farm102:USER1          0          %
Data put      bfo-farm102:SYS1           0          %
Data put      bfo-farm102:IDLE2          100        %
Data put      bfo-farm102:USER2          0          %
Data put      bfo-farm102:SYS2           0          %
Data put      bfo-farm102:IPACK1         0.5         Pckt/s
Data put      bfo-farm102:OPACK1         0.8         Pckt/s
Data put      bfo-farm102:IBYTE1         0.036        KByte/s
Data put      bfo-farm102:OBYTE1         0.1548       KByte/s
Data put      bfo-farm102:IPACK2         0.6         Pckt/s
Data put      bfo-farm102:OPACK2         0          Pckt/s
Data put      bfo-farm102:IBYTE2         0.0649       KByte/s
Data put      bfo-farm102:OBYTE2         0          KByte/s
Data put      bfo-farm102:RBYTE1         0          KByte/s
Data put      bfo-farm102:WBYTE1         0.0024       KByte/s
Data put      bfo-farm102:RPACK1         0          Pckt/s
Data put      bfo-farm102:WPACK1         0.2         Pckt/s
Data put      bfo-farm102:RBYTE2         0          KByte/s
Data put      bfo-farm102:WBYTE2         0.0024       KByte/s
Data put      bfo-farm102:RPACK2         0          Pckt/s
Data put      bfo-farm102:WPACK2         0.2         Pckt/s
stat number 283

```

Figure 4: Results of computations on raw data

Figure 5: Graph of Incoming bytes for 8 farm machines

Figure 6: Graph of KBytes written and read for 1 farm machines

Figure 7: Graph of CPU usage and asymmetry for 1 system, 2 CPU's