Paper 246

Using SAS Software to Analyze Sybase Performance on the Web

Joseph Mirabal
America Online Inc.
12100 Sunrise Valley Dr.
Reston, VA 20191

Zhengyu Wang
Sybase Inc.
6550 Rock Spring Drive
Bethesda, MD 20817

***Abstract**: This paper provides a web-based system using SAS, HTML and CGI/PERL to provide rudimentary and complex Sybase DBMS performance metrics for Unix based system operations. Sybase SQL Server performance data is collected by Sybase Historical Server allowing for the collection of performance information with minimal impact on the server. The SAS System (Base SAS, Macro, STAT and SAS/Graph) is especially well suited to analyzing and visualizing these data. Its graphical, data mining and statistical capabilities coupled with its ability to access data from a multitude of formats, including DBMS, PC and ASCII files, provides an all in one performance analysis toolkit. In this paper, we present a World Wide Web interface to exploit the power of SAS processing for open system delivery of information. The SAS system processes calculations much faster than the functions supplied with DBMS technology. SAS Access to Sybase SQL Pass through facility allows the use of DBMS data access technology. Therefore, this approach exploits the strengths of two technologies to optimize information delivery. The skill level for SAS programmers requires an intermediate to advanced programmer track.*

Introduction

The primary objective of the Web based performance analysis system is to provide a centerpiece where users can make information requests on the performance of a system and derive historical information by specifying the criteria and level of detail. Since the user community of this system can be varied from the system administrator and database administrator ranks to the managerial level, the system must be able to deliver from the lowest level of granularity to the highest. Mature systems would perhaps even require years of information to better plan capacity for their large system applications.

Providing performance information in the relational database environment is increasingly becoming critical for information technology decision support. With the explosion of large system applications in the client-server computing centers and the advent of the web as a point of delivery, scalability of resources and transaction response time become a major concern for the information based support.

Every relational database vendor has their own performance management tools with varying degrees of information depth. Sybase Inc., with its Sybase SQL Server® and the more recent Adaptive Server Enterprise® RDBMS technologies, provides two facilities for gathering and analyzing performance of transactions, stored procedures and database objects to better understand the utilization of resources and the impact failures have on the overall user community. The two Sybase performance tools are *sp_sysmon* and Historical Server®. The latter of these is the least intrusive as it polls Sybase SQL Monitor Server® and minimally uses resources to gather data for performance reporting. The *sp_sysmon* stored procedure must run on the server in question and although very informative could adversely affect processing resources for the application of concern. This makes the use of Historical Server indispensable to an operation that delivers information 24 hours a day every day of the year.

SAS® is a Statistical Analysis System that provides a multitude of utility procedures with the SAS language to allow data mining, warehousing and reporting as well as graphical presentation facilities. SAS reads a number of formats and is platform independent so it is well at home in the client-server (Unix system) environment as well as the Desktop or NT Server environment. The SAS/ACCESS® component provides a SQL Pass Through facility, which serves as the access point between SAS and the database information we use to report DBMS performance. In our processing model, SAS allows us to use the relational database's data access facility and employ SAS for performing the calculations and analyses with Historical Server data. It is a most formidable tool to achieve the goals of performance assurance since it provides us a means to correlate the information we present with the operating system performance layer of the analytical spectrum. In addition to the SAS graphics drivers allowing the presentation of charts and graphs, SAS renders its users free web publishing tools that allow programmers to simply create HTML tables and blend the graphical and tabular forms of presentation on the web.

The web program is written in CGI/PERL. This language proves to be the most versatile for the objectives needed for this application. The PERL language allows us to place controls on the programs executed depending on users' selection. PERL is especially well suited for CGI development and was used here to control the variables passed to the SAS system as well as using its interface to the Unix operating system environment to uniquely name the output files for presentation.

Architecture

The SAS System runs on an HP K460 Kitty Hawk Server running HP-UX 10.20 and has an Apache web server software installed for web information delivery. All Sybase SQL Servers in our environment run several versions of 11.X and more recently we have upgraded to Sybase Adaptive Server Enterprise 11.5.X. The SAS version is 6.12 TS020 and the PERL version installed is 5.004. In essence, we use Sybase Historical Server to gather the data, Sybase SQL Server to store and manage the data, SAS to process and generate visual presentations of database performance and the web to deliver the application.

The infrastructure of our web-based Sybase SQL Server performance analysis system is composed of three tiers. The first tier is performance data collection by Sybase Historical and Monitor Server. The second tier is the SAS interface and analytical reporting facility. It selects the data as requested by the user and processes the data to provide the graphical representation. The final tier is the interactive component allowing the user to describe the selected data through web browsers. The Process Flow Diagram (Figure 1) shows the architecture and data flow from generation to delivery of information on the web as a reporting tool.

Sybase SQL Monitor Server, combined with Sybase Historical Server, provides Sybase SQL Server performance data in today's enterprise-wide client-server computing environment. Sybase SQL Server saves performance data in a shared memory area where Monitor Server reads in a manner that doesn't impact the server performance. Sybase Historical Server captures the performance information from Monitor Server and saves it in files for future analysis. Users can specify in views the performance data and time period to collect. Each view is composed of several performance data items and statistics. The statistic can be defined as absolute counts, average counts or rates of activities. Filters are defined on a per-data-item basis to limit the amount of monitoring data that Historical Server records. In our environment, we are interested in collecting performance data on CPU utilization, process status, server status, device activity, cache activity, object activity and procedure execution. A view is created for each performance criterion by combining related data-items. For example, a "device activity" view consists of device name, number of hits and misses, number of i/o operations plus the number of reads and writes. A filter is created for the "number of hits" to filter out 0 hits during the collecting period. In this way we can limit the number of insignificant rows of data stored in the performance database. Sybase Historical Server defines over 150 performance metrics providing differential performance perspective on Sybase SQL Servers.

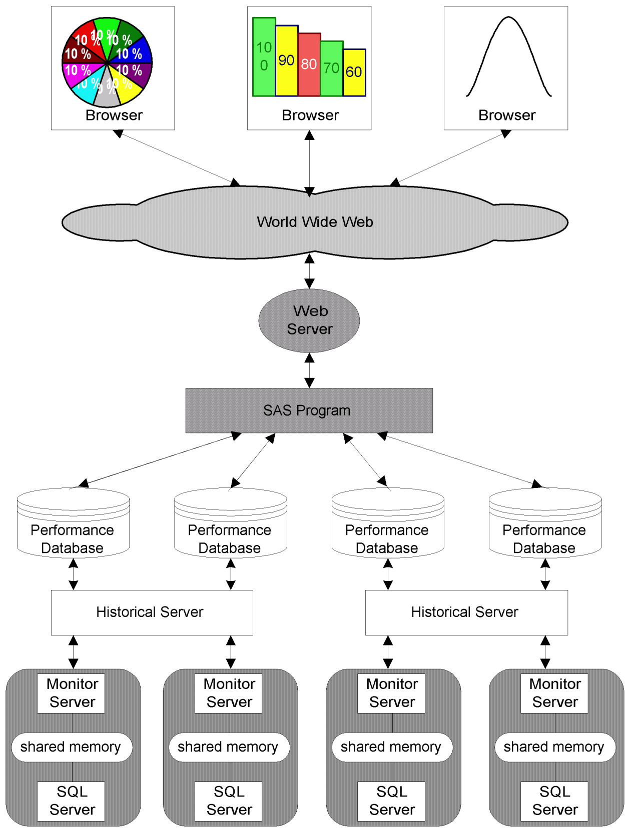

Figure 1. Web-based Sybase Performance Analysis System Architecture

Sybase Historical Server stores performance data (defined via data views) in flat files. These flat files can be processed by SAS. However, it is not always practical since in the processing of these flat files the entire file must be read to extract a subset of data. Also, the size of flat files may also grow very rapidly when we collect a large amount of performance data in short time intervals. Therefore, it made sense to build a relational database on a Sybase SQL Server to keep these performance data. Additionally, SQL Server provides data indexing and built-in storage management facilities. These features present definite advantages by limiting the data brought in for processing and increasing the speed of delivery on the web. Each view corresponds to a performance table, and a bulk copy utility (bcp) provided by Sybase SQL Server is used to parse the data stored in flat files and insert them into tables.

Once the detailed performance data is stored in the database, a summary of the data at various levels of granularity (for example daily, weekly, monthly or yearly) can be generated via SAS. Summarized historical performance data is useful for observing trends in resource usage. Capacity planning, trend analysis, resource reporting, benchmarking and even troubleshooting are activities that can benefit from historical performance data stored in the database. In our environment, we have found situations where historical performance trend analysis helps prediction and preparation for future events. Detailed historical performance data is useful for tracing the causes of intermittent or recurring problems, for keeping watch over the general performance of Sybase SQL Server, and for creating benchmark analysis to anticipate future performance. Detailed performance data is a very useful supplement to our existing troubleshooting tools. Multiple views in Sybase Historical Server provides different system performance perspectives captured right before any problem happens. System administrators are able to compare the existing system status with any historical system status stored in the database to help diagnose problems

A graphical view of the performance data stored in the databases assists system administrators to analyze symptoms of the problem as well as the problem itself. Through the review of patterns and relationships among key system activities, preventive measures can be implemented. SAS is a proven powerful tool to analyze data and generate vivid graphs in a reasonable amount of time. As the performance data grows steadily with daily data collection, SAS has scaled very well with the data size.

Since system administrators do not usually include SAS in their set of tools, the SAS programmer's limits are pushed as programs are requested in increasing frequency and thus information delivery is delayed. Also, making minor modifications on SAS programs based on the requester's requirements can be burdensome for the SAS programmer. At times these modifications are so minimal (such as server name, performance data type and date) that the time it takes to generate can only be described as inefficiency. The solution of building a front-end web delivery program to provide users the ability to select key parameters and generate answers to trivial questions leaves the statistician with the ability to spend more time on complex issues. The technician can thus gain a greater understanding of system behavior and provide a basis for better planning.

The web browser grants users easy access to performance data on PC or UNIX workstations without installing additional software. The Common Gateway Interface grants us the ability to make the data available to users in a form they can choose (graphical or tabular). The web program was kept simple in this instance since our intent was to serve the introductory population. However, the web program could be built in such a way that colors, graphical procedures as well as font selection could be selected by the user and passed to SAS as a macro variable and used throughout this presentation. The method to accomplish this is discussed in the SAS Processing portion of this paper.

Users request an electronic document on the web browser by specifying an URL (universal resource locator). The browser connects to the web server over the internet to access and retrieve documents. Web servers serve up documents and web browsers retrieve and display the documents. Figure 2 shows the initial screen on the web browser, where users can make selections and inputs in the system. The screen is written in HTML (Hypertext Markup Language). On the screen, users can choose applications, servers and performance data (engine statistics, disk activity, object activity and cache activity). They can also input the time frame when the performance data are to be extracted from database and graphed by SAS. Summarization level (detail, daily, weekly and monthly) can be selected to choose the granularity performance level of the data being analyzed. When the “Submit” button is clicked on the web browser, all the values entered by users on the screen are sent to the web server. The web server executes a program written in PERL. The PERL program makes the interface between the SAS programs and the web content. The program parses the user input and calls the proper SAS program dependent upon performance data and granularity level requested by the user. Server name and time frame requests are passed to the SAS program that generates an image file in GIF format. The PERL program opens the image files and sends the images back to the web browser, which displays the image to the user.

Each component of the system is designed and developed for flexibility so new features and functionality can be accommodated easily. For example, when additional performance data is required to be collected from Sybase SQL Server, new views can be created for Sybase Historical Server and new tables can be created in the performance database to load the data. An item can be added to the selection list on the screen, and the PERL program can be modified to call additional SAS programs for new performance concerns. We are now planning to add new performance analysis graphs to the system, including server performance comparison and granting output options to the user such as graphical, tabular or both formats.

In the following section, we focus on the SAS program and PERL program that represent the major programming components of our system. First we discuss a description of how to apply the SAS language for web CGI, including accessing the Sybase database, defining the variables passed by the web program to SAS and delivering the graphical output to a file used directly in information delivery on the web. The Web processing discussion focuses on the PERL program that is used to process the parameter list the user selects. This is done to secure that SAS receives the appropriate information to process the user’s request by pointing to the appropriate program and selecting the summary level and server desired.

We elected not to provide the HTML code because it truly is a matter of the user’s preference and is relatively simple to write. Also, with the expansion of HTML development tools like AOL press, Microsoft FrontPage and Netscape Composer among others, it seems the reader can benefit more from the functional design than from coding the web page. We have, however, included a view of the home page of our web presentation so the user can see how it was applied in this instance (Figure 2).

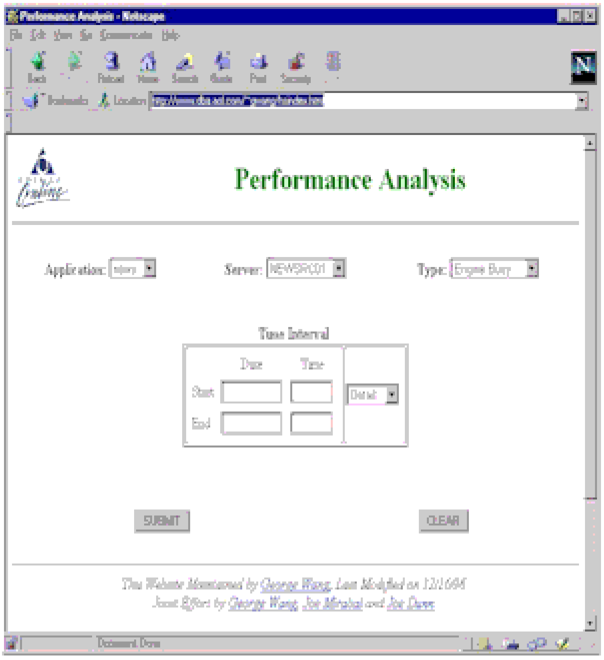

Figure 2. Sybase Performance Analysis System Web Page

SAS Processing

The SAS programming in this system involves the use of SAS macros in addition to Base SAS and the SAS/Graph component. As previously mentioned, the SAS/Access to Sybase (SQL Passthrough facility) is our access point to the database. Common Gateway Interface (CGI) programming requires the definition of *name=value* pairs that are part of the interactive portion of web application. The *name=value* pairs in SAS are passed using the macro facility’s “*call sympu*t” function. In our web interface application (Figure 2), there are five parameters that are passed to SAS from the web program. These are the server name, the desired start time, the desired end time, the unique name of GIF file (supplied by the web program) and finally the table or information desired (categorized as *Type* in the HTML presentation). In the *NULL* program a variable is created to derive a unique identifier for the *sql view.* The SAS guideline requires that variable names be limited to 8 characters. Therefore, it is necessary to trim the table name selected by the user (displayed in the *Type* field) to where it becomes unique (the first 5 characters mostly) and select the first three characters of the server name. Since Historical Server maintains a database for each server there is no risk of providing information for a non-requested server. SAS keeps account of the active view. If the user chooses a different table or server and does not replace the view, there is risk the user will not get the requested information. When SAS Version 7/8 becomes a reality this requirement will become unnecessary since we shall be able to name views beyond the 8 character limit. The *fileview* definition and the *&fview* macro variable defined in the “*call sympu*t” provide for the unique definition of the view created for each dynamic operation request from the web.

The SAS data step reads the standard input as follows:

```
Data _NULL_ ;
 infile stdin delimiter=’ ‘ missover ;
 informat db $16. time1 $16. time2 $16. gifile
$15. type $24. ;
```

```
 input db $ time1 $ time2 $ gifile $ type $ ;

file1=substr(type,1,5) ;
file2=substr(server,1,3) ;
fileview=trim(left(file1))||trim(left(file2)) ;

call symput ('server',trim(left(db))) ;
call symput('stime',trim(left(time1))) ;
call symput('etime'trim(left(time2))) ;
call symput('gifname',trim(left(gifile))) ;
call symput('fview',trim(left(fileview))) ;
   run;
%put &server &stime &etime &gifname &fview ;
```

This model can be used with any data passed from a local web server to the SAS engine (in our case both engines share the same host). Obviously, we can add more parameters or manipulate any data passed from the web in this "*DATA NULL*" step. These parameters provide all the input data sources for *name=value* pairs needed for CGI processing. This step merely reads the input and creates a macro definition with *call symput* command to redefine the user provided input for dynamic processing in SAS. Therefore our SAS program will process the macro variable to provide the user the desired output.

The macro for the Engine_Stats table would be defined as follows:

```
%macro engstat(db,time1,time2,gifname,fileview);

……….SAS Data step and Procedure Code

%mend engstat ;
%engstat(&server,&stime,&etime,&gifname,&fview)
<no semicolon in a macro call>
```

Note that the values defined in the macro definitions are the standard input variables. However, when we call the macro program in *%engstat*, the positional variables reflect those defined by the "*call symput*" function.

The SAS Program

The following pieces are the SAS program embedded in the macro statements as shown above. The first part of the program creates the view into Sybase by connecting to the historical server data using the SQL Pass through facility. Beyond the user name and password, the server and the database need to be specified. The "*%let*" macro declaration command is used to define the static SQL Server name and the dynamic database name which is dependent on the server of interest by the user. Therefore, the server macro variable name passes the macro value "*&server*" defined by the user on the HTML page. The reader may find that the historical server data for all servers may be defined in a common database. In this case the database may be the same but the server must be specified in the where clause (eg. Select * from Engine_Stats where Server="&server"). Remember to always use double quotes around each macro call as in this example.

```
/*  The SQL Pass Through to Sybase */

%let sqlsrvr=HISTSERVER01 ;
%let server=&server ;

options dbdebug ;
libname sybsasdb '/saspdb/views/hssybase';
proc sql ;
  connect to sybase(user="sasguru"
        password="urdbest" database="&server"
        server="&sqlsrvr") ;
  create view sybsasdb.&fview as
    select * from connection to sybase
    (select Timestamp "datetime",
       EngineNumber_ValSmp "engno",
       CPUBusyPercent_ValSmp "pctcpbsy"
      from  Engine_Stats) ;
   disconnect from sybase ;
run; quit;
```

This segment is the SQL pass through portion of our program. The select statement that appears in parentheses is the query processed in Sybase SQL Server. A clustered index on the "Timestamp" field will significantly speed up access processing. This is accomplished by adding the following SQL code within the Sybase processing portion between parentheses:

```
where Timestamp >= <start date>
  and Timestamp <= <end date>
```

To extract the start and end date from the detail level request simply strip the date from the start and end datetime and add another "*call sympu*t" to process the macro variable as follows:

```
date1= datepart(time1); date2 = datepart(time2);
call
sympūt('sdate',trim(left(put(date1,mmddyy8.))));
call
sympūt('edate',trim(left(put(date2,mmddyy8.))));
```

The date format selected must be compatible with Sybase query processing syntax. This is the reason for the selection of the "mmddyy8" format. At this point the where clause can be applied to the Sybase SQL Pass through query and the program simply accesses data only for the dates requested, significantly speeding the response delivery of the CGI application. The where clause would thus be written as follows:

```
where Timestamp >= " &sdate"
  and Timestamp <= " &edate"
```

The "*Data Step*" portion simply calls the data and subsets it for the specific requirements of the web user making that request. The generation of a "date" and "hour" variable are included in this program for example but are not applied in this case since the level of granularity desired is "Detail" (5 minute interval information). The period requested by the user is defined in the "*if*" statement using &stime and &etime macro variables specified in the *NULL* step of our program.

The procedure section of the data step simply sorts and summarizes the data. For the detail level of granularity the summary step is not necessary as the request does not require summarization. Though the statistical procedure is superfluous in this case the reader can see how a higher summary level (eg. by day or hour) can be generated. Typically for this CGI the date and hour would have to be joined for processing multiple days if the user so desires. In this case, we simply define datetime to a format of datetime10. To achieve this we simply define a new variable as in the following example:

```
datehour=put(datetime,datetime10.);.

/* The Data Step Portion of the Program */

data test ;
set sybsasdb.&fview ;

date=datepart(datetime);  hour=hour(datetime) ;

/* Subset the date and time */
if datetime>="&stime"dt
   and datetime<="&etime"dt ;
proc sort data=test ;  by datetime ;
proc means data=test noprint ;
   var  pctcpbsy  ;   by datetime  ;
   output out=output mean=avgcpbsy ;
```

The graphical portion defines the device driver. In this case the “gif733” driver is used because it provides the highest resolution level provided in a SAS graph output driver. The graphical program has options and functions that define the way the graph is presented. The plot is a standard GPLOT with the title identifying the server and period of time of interest.

The rationale behind the graphical options “*goptions*” selected in this exercise is as follows:
RESET=ALL – Clears all previously generated goptions from the SAS system.
DEVICE=GIF733 – Sets the device driver to be used in the GIF generation. Other drivers in SAS 6.12 include GIF, IMGGIF, IMGJPEG (check *PROC GDEVICE* for details).
GSFNAME=GOUT – Points the output to the location where the gif file will be available to the web server for delivery to the user in the World Wide Web.
GSFMODE=REPLACE – Secures that each generated graph points to the new specifications. SAS keeps a graphics catalog that without the replace will provide the first generated graph.
LFACTOR=1 – Provides for the line thickness and ranges from 0 to 5 accepting fractions of each integer (e.g. lfactor=1.45).

```
/*  The Graphical Portion of the Program */
filename gout "/htdocs/sasgrafs/.www/&gifname" ;
   goptions reset=all device=gif733 gsfname=gout
   gsfmode=replace lfactor=1;

symbol1 value=star color=red interpol=spline ;
symbol2 v=circle c=blue i=spline ;
```

The symbol statements provide the character image used to plot datapoints, the interpolation algorithm to connect the data points and define the color of the line. In this case the SPLINE algorithm which uses statistical smoothing to express the trend line could be replaced with a JOIN interpolation algorithm which simply connects the points without using a smoothing calculus.

```
proc gplot data=output ;
  plot avgcpbsy * datetime ;
  label avgcpbsy="Average CPU Busy" ;
  format datetime datetime13. time time5. ;
  title "CPU Performance for &server to User
Specifications on CGI" ;
  title2 " Timeframe under study: from &stime to
&etime " ;
  footnote1 "This job ran on &SYSDAY &SYSDATE at
&SYSTIME" ;
  footnote2 c=red "AOL Confidential" ;
 run; quit;
```

The GPLOT procedure uses the Y * X approach, so on the vertical axis the avgcpbsy(average cpu busy) will be plotted for each instance of time. The label statement is supplied to replace the variable name on the graph and the format provides the horizontal axis tick mark definitions as “ddmmmyy:hh:mm”. The title and footnote are text lines presented at the top and bottom of the charts respectively. An example of the output on the web appears below.

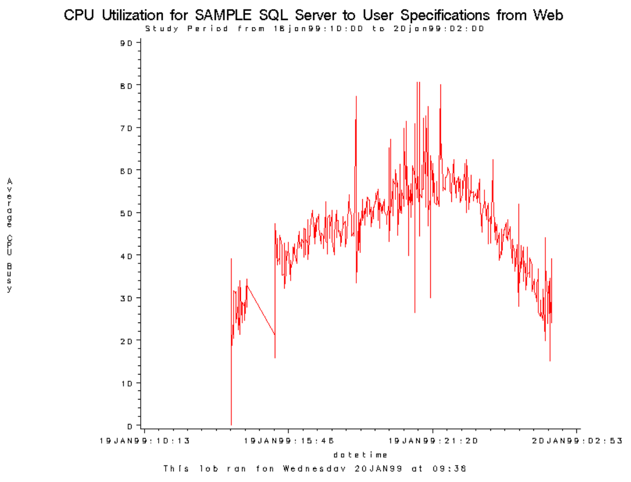

Figure 3. A Sample GIF file Appearing on the Web

Multiple programs would be generated for each summary level so the template presented above could be saved as daily and the programmer can simply change the procedure sort and means to reflect the new sort and summary requirements.

Web Processing

The Common Gateway Interface (CGI) is the part of the web server that can communicate with other programs running on the server. With CGI, the web server can call up a program that passes user-specific data requirements. The program processes the data and the server passes the program’s response back to the web browser. Although there are many programming languages supporting CGI capability, we chose PERL because of its powerful features and portability.

In our system, users supply information on the screen by clicking the desired application server, granularity level and entering the time period. By clicking “Submit” on the form, users input information that is sent to the web server. The web server calls the following PERL program and passes the users input to it. The PERL program first parses the input string and extracts the elements, such as server name, time period and granularity level. It also transforms the date from user input format to a SAS recognizable format. It then decides the right SAS program to call depending on the performance data type and granularity level that the user selected on the web page. The PERL program passes the server name, starting and ending date and time, and the image file name to the chosen SAS program. The image file will store the performance graph in GIF format generated by SAS. After the SAS program generates the image file, the PERL program opens it and sends every byte to the web browser. Before sending the image content, the PERL program tells the web browser that the following information will be an image by specifying an image content type. The web browser will display the information as an image on its screen.

PERL Program

```
#!/usr/local/bin/perl

parse_form_data(\%form_data);
$pid = $$;  $gifname = "perf_" . $pid . ".gif";

#generate SAS recognizable date format
@month = qw(jan feb mar apr may jun jul aug sep
oct nov dec);
```

```
($mon, $day, $year) = split /\//,
        $form_data{'startdate'};
$startdatetime = "$day"."$month[$mon-1]"
        ."$year".":$form_data{'starttime'}";
($mon, $day, $year) = split /\//,
        $form_data{'enddate'};
$enddatetime = $day"."$month[$mon]"
        ."$year".":$form_data{'endtime'}";

#select SAS program
$cmd = "echo $form_data{'server'} $startdatetime
        $enddatetime $gifname |
        /usr/local/sas/sas611/sas
        /home/username/\.www/";
if ($form_data{'type'} eq "Engine Busy") {
    %eng_cmd = ("Detail", "engine_stats_detail",
                "Hourly" , "engine_stats_hourly",
                "Daily",   "engine_stats_daily");
    $cmd .= $eng_cmd{$form_data{'interval'}};
}

#call SAS program
system("$cmd");

#deliver GIF image generated by SAS to browser
$fullname = "/home/username/\.www/".$gifname;
if (open(IMAGE, "<" . $fullname)) {
    $no_bytes = (stat ($fullname))[7];
    $piece_size = $no_bytes / 10;

    print "Content-type: image/gif", "\n";
    print "Content-length: $no_bytes", "\n\n";

    for ($i=0; $i<=$no_bytes; $i+=$piece_size) {
        read(IMAGE, $data, $piece_size);
        print $data;
    }
    close(IMAGE);
}
else {
    print "Content-type: text/plain", "\n\n";
    print "Sorry! I cannot open the file
        $fullname!", "\n";
}
system("/usr/bin/rm -f $fullname");
```

This application's design uses multiple programs that are selected in the CGI process on the web. This was done for two reasons. One was in order to supply multiple summary granularity levels it is more efficient to access a program that can exploit the indexing capabilities of the RDBMS for speed in the data extraction process. The other is the ability to apply modifications to a program without affecting the other. If a program fails, the application is not completely down or unavailable, other portions can continue to perform the information retrieval/delivery process while the failing program is repaired.

Conclusion

In this paper we have provided an approach to devise a web interface to SAS program and present the results generated by SAS on the web. Our system hosts both the SAS System and the web server but these two could be segregated using a socket program between two hosts. Users with SAS/Intrnet can use the principles presented here with minor modifications. The objective sought with this application was to provide the user/managerial/administration community with a means to access information in various forms. This provides a mechanism to quickly view various summarization levels of metrics and leads users to study a specific area of performance concern so that they can better formulate the strategy for improvement or problem resolution. The SAS programs can be written independently and easily applied to the PERL script. The PERL script is modular enough so that it does not require much PERL programming experience to make the necessary adjustments. Since the means to define a CGI/PERL script in an HTML page is readily available, this is left as an exercise for the user. However, we have presented the concept, approach and an example of the essential code to make the performance analysis system work. This paper focused on the delivery of database performance from Sybase Historical Server. However, the principles and approaches can be applied to any source of information. The data could be in a SAS dataset so the access to Sybase is not necessary. Any data that SAS can process can apply these principles.

Those who know PERL can additionally use its power of extraction and manipulation of data to exploit SAS's exceptional speed in calculation and statistical processing. We have found some of PERL's features to work exceptionally well as a front end to SAS especially in extracting data from logs and text based data. The implementation has afforded us time to add features to the application rather than the daily drive to answer basic questions for operational management meetings or to serve as justification for expansion of capacity.

Trademarks

Contact Information

The authors may be contacted at

Joseph Mirabal
America Online, Inc.
12100 Sunrise Valley Drive
Reston, VA 20191
(703)265-4620
FAX: (703)265-4020
Email: jmmirabal@aol.com

Zhengyu Wang
Sybase Inc.
6550 Rock Spring Drive
Bethesda, MD 20817
(301)896-1127
FAX: (301)896-1602
Email: zhengyu.wang@sybase.com