Paper 160-29

Improved Methods for Early Fault Detection in Enterprise Computing Servers Using SAS® Tools

Kenny C. Gross, Sun Microsystems Inc, San Diego, CA
Kesari Mishra, Duke University, Durham, NC

Abstract

Advanced telemetry systems are being developed to collect and archive hundreds of system performance, throughput, quality-of-service (QoS), and physical variables for the purpose of enhancing the reliability, availability, serviceability, scalability, and security of business-critical enterprise computing servers. **SAS®** software was chosen for this project because of the language's powerful coding features and its ability to model prototype systems quickly and cost-effectively. SAS macro language was exploited in a parametric sensitivity study to explore and optimize clusters of signals that provide high sensitivity for early fault detection, but with a good avoidance of false alarms, for subsequent post-processing with an advanced statistical pattern-recognition surveillance system called MSET. Finally, **PROC GPLOT** and **PROC G3D** proved indispensable for displaying results of our many-variable sensitivity investigation. SAS programs developed for this investigation are generic rather than operating system specific. The results presented in this paper used SAS 8.2 for Unix environments operating on Solaris 8 platforms. The coding practices discussed in this paper are aimed at users with an average **SAS/GRAPH** experience and an average proficiency with SAS macro language.

Introduction

Fault detection in complex systems typically requires costly on-line monitoring and expertise. Conventional approaches to identifying faults, combining event correlation and threshold-based rules, have proven inadequate in a variety of safety-critical industries with complex, heterogeneous subsystem inputs not dissimilar to those from enterprise computing. Fundamentally, while many high-end computing servers are already rich in instrumentation, the data produced by the instrumentation are complex, non-uniform, and difficult to correlate. Pattern recognition technology, coupled with continuous system telemetry, offers key enabling technology that can help:

- identify incipient faults proactively
- reduce the expertise required to identify correlations
- reduce the compute cost of monitoring
- reduce the probability of false alarms
- increase the accuracy and timeliness of root cause analysis
- help eliminate “No-Trouble-Found” (NTF) events that can drive up warranty & serviceability costs for a server vendor

The effectiveness of using pattern recognition to discern incipient faults in noisy process data coupled with continuous system telemetry is gated by the quality of information available from instrumentation. This investigation was undertaken to extract, pre-process, and provide analytical resampling for a vast variety of performance, quality-of-service, and system load metrics for

large enterprise computing systems. The project encompasses improvements to existing instrumentation, addition of supplementary external instrumentation (in controlled laboratory experiments), and improved analytical methods for normalization, validation, and application of statistical pattern recognition of the time series signals from an enterprise server telemetry harness. **SAS®** software tools **[1]** were instrumental in this project for the steps of extracting and pre-processing large volumes of digitized signals generated by the server instrumentation. Signals that are pre-processed using the tools developed herein are then suitable for analysis with dynamic system characterization methods **[2]** and with advanced proactive fault monitoring such as that available in the Multivariate State Estimation Technique (MSET) **[3]**.

The continuous system telemetry harness (CSTH) devised for this investigation is illustrated schematically in Figure 1.

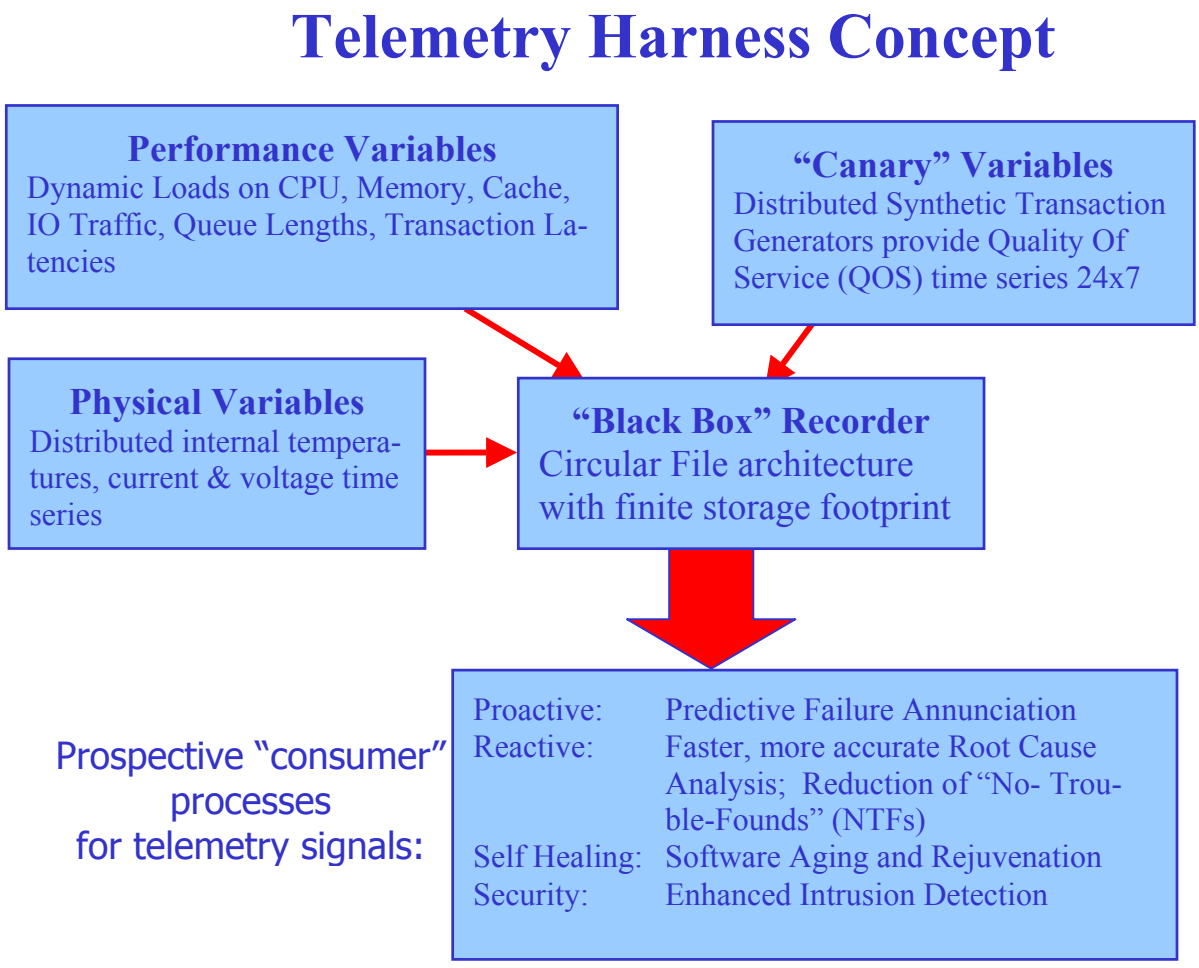

Figure 1 Telemetry Harness Concept

The telemetry variables from large, multi-CPU enterprise servers may be partitioned conveniently into three categories as illustrated in the figure. The monitored variables include "performance variables" (parameters having to do with throughput, transaction latencies, queue lengths, loads on the CPU & memory modules, IO traffic, bus saturation metrics, FIFO overflow

statistics, etc.); "canary variables" (distributed synthetic user transactions that give user QOS metrics 24x7); and physical variables (distributed internal temperatures, currents, voltages, time-domain reflectometry readings, relative humidity, vibration, acoustics, etc).

For the box labeled “Performance Variables,” the “virtual sensors” used for this project are embodied in a tool called the SEToolkit **[4].** The SEToolkit primarily consists of an interpreter for a programming language (called SymbEL), which is a subset of both C and C++, and its library files. Primitives for accessing performance data within the operating system are inherent in SymbEL so that instead of writing tedious code to access them, just defining the variables suffices. The interpreter reads the values and places them into the variables. A major advantage of SEToolkit that led to its selection for this project is that it extracts performance data from the system without becoming one of the performance problems that needs to be investigated. The access methods used are quite lightweight and execution is fast.

To access and analyze different types of time series originating from physically distributed sources, it was necessary for this investigation to develop an analytical resampling program that synthesizes data from multiple, disparate-format sources and can be used to produce synchronized data streams suitable for use in the design, testing, and performance evaluation of proactive fault monitoring tools for catching incipient problems in enterprise computing servers. SAS software was chosen for this project because of the language's powerful coding features and its ability to model prototype systems quickly and cost-effectively. **SAS Macro** language was exploited in a parametric sensitivity study to explore and optimize clusters of signals that provide high sensitivity for early fault detection, but with a good avoidance of false alarms, for pattern-recognition surveillance systems. Finally, **PROC GPLOT** and **PROC G3D** proved indispensable for displaying results of our many-variable sensitivity investigation.

Analytical Resampling Program Implementation

A plethora of transducers and software-implemented “virtual sensors” used to collect system parameters and performance variables in executing servers lead to a rich storehouse of system data, but with incompatible signal formats, dissimilar timestamp format, and disparate sampling rates. The surveillance system that “consumes” these signals needs for the input data from all these sources to be available in a uniform format and aligned with a common sampling frequency. This provided a functional requirement for the Analytical Resampling Program (ARP) that was developed in SAS.

The process flow chart for this interpolative online normalization consists of three broad steps as outlined in the following paragraphs.

Timestamp Format Conversion

The first step is converting the different timestamp formats associated with parameters from each source to a common format (the SAS format in units of seconds since Jan 1, 1960). From some of the sources, the data were already in the form of time series and could be converted easily to units of seconds and translated to a common reference date of 1/1/60. From some of the sources, the timestamps did not conform to any of SAS Date-Time informats. A few observations from one such source have been reproduced in Table 1.

Month	Day of Month	Hour	Minute	Parameter Value
7	26	14	29	2.46
7	26	14	30	1.88
7	26	14	31	1.42

Table 1. Sample input data from one of the sources

The macro used to convert the data from this source to timestamp in SAS format is reproduced in Table 2.

```
*SPECIFYING THE FILE NAMES;

FILENAME IN1  ('~/INPUTFILE.DAT');
FILENAME TEMP ('~/TMPFILE.DAT');
FILENAME OUT ('~/OUTPUTFILE.DAT');

*SETTING THE YEARCUTOFF TO BE 2000 SO THAT THE YEAR 02 MEANS 2002 ;

OPTIONS YEARCUTOFF=2000;
DATA RDDATA;
FILE TEMP;
INFILE IN1;
INPUT MNTH DT HR MIN LTNCY;

*PUTTING THE DATE INTO ddmmyy FORMAT ;

MM = MNTH*100;
DD = DT*10000;
REALDT=DD+MM+2;
PUT REALDT HR MIN LTNCY;

DATA GETDT;
INFILE TEMP;
FILE OUT;
INPUT DATT DDMMYY6. HR MIN LTNCY;

*THE ddmmyy FORMAT GIVES NUMBER OF DAYS SINCE JAN 1, 1960 ;

TMSTMP = DATT*24*3600 + HR*3600 + MIN*60;
PUT TMSTMP LTNCY;
RUN;
```

Table 2. SAS code for Timestamp Conversion

This macro converts the data in table 1 to the format of Table 3.

Timestamp	Parameter Value
1343312940	2.46
1343313000	1.88
1343313060	1.42

Table 3. The output from Timestamp Conversion of input data in Table 1.

Similar code is used for conversion of timestamps from each of the data sources to a common format (Table 3).

The Synthesis and Alignment of multi-sourced data

The next step after timestamp conversion is integration of data from multiple sources, now with a common timestamp format, into one data file. An empty times series, ranging from the first to the last timestamp of one of the inputs, is generated and merged with that input, using **PROC MERGE,** which is a part of the **Base SAS®** Software (Table 4). This gives a time series with missing parameter values corresponding to timestamps that were absent in the original time series. The dataset thus generated is merged with all the other inputs and sorted by the timestamp using **PROC SORT**. This results in a dataset with timestamps at irregular intervals and missing parameter values. In order to eliminate the missing values, we interpolate the data using **PROC EXPAND**, which is a part of **SAS/ETS** software **[5].**

Cubic spline interpolation was developed for this application. A polynomial fitted to many data points could exhibit erratic behavior, but the cubic spline has the advantage of being continuous across sampling intervals and having a low order of three or less at the same time. Moreover, it possesses the desirable property of being the smoothest of all possible interpolating curves as its first and second derivatives are continuous and it minimizes the integral of the square of the second derivative. The option **METHOD=SPLINE** in **PROC SPECTRA** develops a cubic spline to fit the data.

A portion of the SAS code that merges and interpolates disparate data from five of the inputs has been reproduced in Table 4.

```
* READ FROM FILE AND STORES THE   LAST TIMESTAMP;

DATA FNDEND;
RETAIN TMSTMP;
INFILE IN1 EOF=JMPD; FILE TEMP1;
INPUT TMSTMP C2-C50;
RETURN;
JMPD:
PUT TMSTMP;

*STORE THE FIRST TIMESTAMP;

DATA FNDSTRT;
INFILE IN1; FILE TEMP2;
INPUT TMSTMP C2-C50;
IF _N_ = 1 THEN PUT TMSTMP;

*STORE FIRST AND LAST TIMESTAMP;

DATA JSTIME;
INFILE OUT2;
INPUT FSTVAL;
INFILE OUT1;
INPUT LSTVAL;

*GENERATE THE EMPTY TIME SERIES ;

DATA EMPTIM(KEEP =TMSTMP);
SET JSTIME;
DO WHILE(TMSTMP <  LSTVAL);
            TMSTMP = FSTVAL+IT;
             IT+5;
* 5 SECOND IS THE DESIRED SAMPLING PERIOD HERE;

OUTPUT;
END;
```

```
*READ INPUT FROM OTHER SOURCES;

DATA SRC1; INFILE IN2;
INPUT TMSTMP VAL1;

DATA SRC2; INFILE IN3;
INPUT TMSTMP VAL2;

DATA SRC3; INFILE IN4;
INPUT TMSTMP VAL3;

DATA SRC4; INFILE IN5;
INPUT TNSTMP VAL4;

*SORT ALL THE INPUTS;
PROC SORT DATA=SRC1; BY TMSTMP;
PROC SORT DATA=SRC2; BY TMSTMP;
PROC SORT DATA=SRC3; BY TMSTMP;
PROC SORT DATA=SRC4; BY TMSTMP;
PROC SORT DATA=FNDSTRT; BY TMSTMP;
PROC SORT DATA=EMPTIM; BY TMSTMP;

*MERGE DATA FROM ALL THE SOURCES;
DATA ALTGTHR;
MERGE FNDSTRT EMPTIM SRC1 SRC2 SRC3 SRC4;
BY TMSTMP;

*INTERPOLATE TO FILL IN THE MISSING VALUES;
PROC EXPAND DATA=ALTGTHR OUT=INTRPLTD METHOD=SPLINE FROM=SECOND;
CONVERT C2-C50 VAL1 VAL2 VAL3 VAL4/OBSERVED = TOTAL;
RUN;
```

Table 4. Part of SAS code for synthesizing and interpolating data.

Resampling for Synchronization

The previous step does the job of synthesizing outputs from all the sources and interpolating them. The resulting dataset has timestamps at uneven intervals. The "index" time series that is used to define the equal sampling intervals forms the backbone of the resampling process insofar as the timestamps corresponding to the index time series will be used to define the desired time-stamps. Subsequently, interpolation fills up all the missing parameter values to produce a dataset with "row and column" signal observations synchronized to a desired sampling rate. The portion of the SAS code for resampling has been reproduced in Table 5 and demonstrates at once both the power and the simplicity of the SAS language for time series manipulations.

```
DATA RESMPL;
FILE OUT;
RETAIN K 0;
SET INTRPLTD;
IF _N_ = 1 THEN DO;
        K=TMSTMP;
        PUT TMSTMP C2-C50 VAL1-VAL4;
       END;
ELSE
IF MOD ((TMSTMP- K), 5) = 0 THEN DO;
PUT TMSTMP C2-C50 VAL1-VAL4;
RUN;
```

Table 5. Part of SAS code for resampling

The output of the above code segment is now in the desired data format, aligned and synchronized to a common sampling rate. Figure 2 plots four example system telemetry parameters after

analytical resampling. Our technique of analytical resampling took care of their disparate sampling rates as well as different start times.

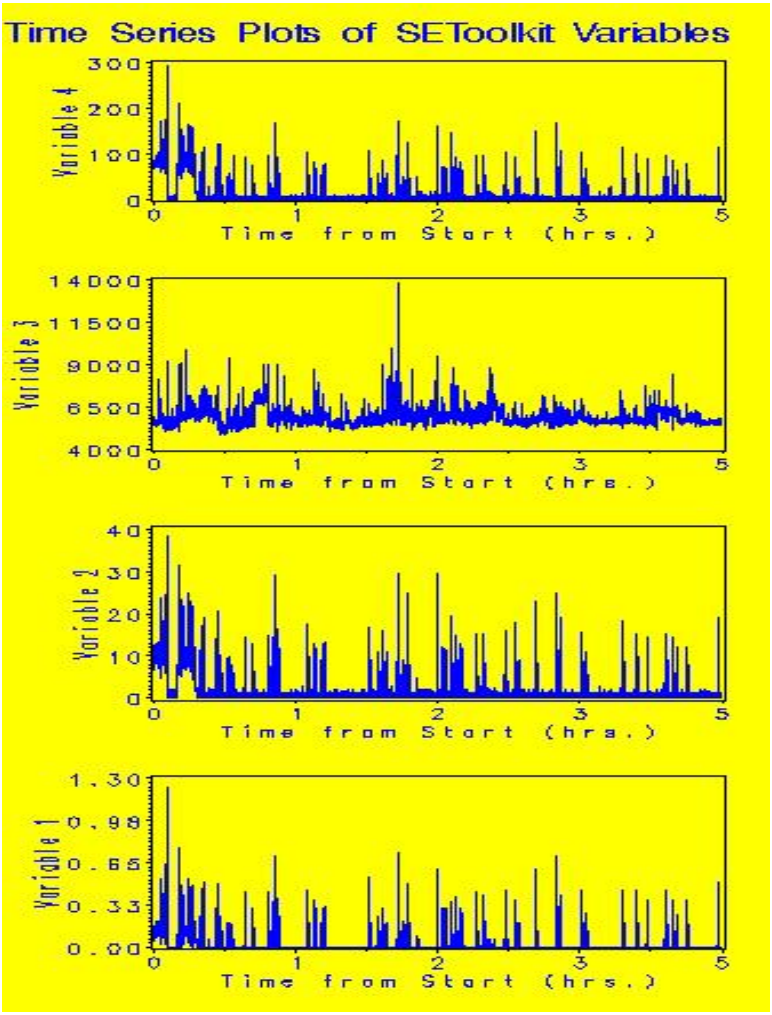

Figure 2. Time Series plots of four of the system variables after analytical resampling

A further advantage of this technique evinces itself in the process of aligning the user response time canary variables. The canary variables are response times of synthetic client processes that launch themselves periodically and perform some typical user transaction to enable transaction-latency evaluation around the clock. These canary-test processes are not synchronized and additionally may have different sampling intervals. Moreover, occasional speeding-up or slowing-down of the processes, caused by the variations in the system load, lead to time-varying phase shifts between the parallel time series. Figure 3 plots the time series of three of the Canary Variables.

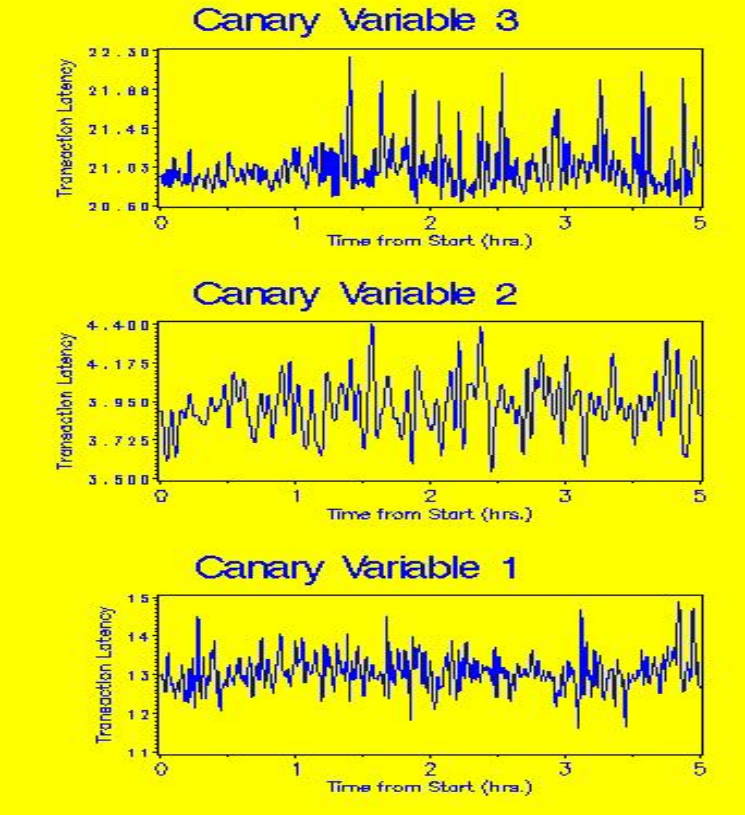

Figure 3. Time series plot of three of the canary variables after analytical resampling

Both the figures have been produced using **PROC GPLOT**, which is a part of **SAS GRAPH** software. The multiple plot formats were facilitated by calling the procedure **PROC GREPLAY**. Single plots were first produced and stored in a catalog. These catalog entries are then replayed onto a template and displayed as a single graph.

Summary:

A continuous system telemetry harness (CSTH) has been devised for monitoring performance, throughput, QoS, and physical parameters in executing computing systems. Use of system telemetry can bring many benefits to high-end enterprise computing, including enhancements to availability, serviceability, performance, capacity planning, quality of service, and security. However, considerable pre-processing of signals is necessary to enable analysis of data streams by commercially available pattern recognition system tools. For this project, it was demonstrated that the features of SAS can be exploited to produce an Analytical Resampling Program (ARP) that brings together signals having disparate formats and nonuniform (and possibly time varying) sampling intervals to produce synchronized time series that may be subsequently displayed with a variety of 2-D and 3-D color graphic formats. Moreover, the signals that have been pre-processed with the ARP presented here are in a format that facilitates analysis with "downstream" consumer processes such as the Multivariate State Estimation Technique (MSET).

References:

[1] SAS Institute Inc. (1999), *SAS User's Guide Version 8*, SAS Institute Inc., Cary, NC.

[2] K. C. Gross, W. Lu and K. Mishra (2003), "Spectral Decomposition of Performance Variables for Dynamic System Characterization of Web Servers", **Proc. of SAS Users Group Intn'l Conference**, (SUGI-28) pp. 206-218, Seattle, March 2003.

[3] "MSET Performance Optimization for Detection of Software Aging," K. Vaidyanathan and K. C. Gross, ***Proc. 14th IEEE Intn'l. Symp. on Software Reliability Eng. (ISSRE'03),*** Denver, CO (Nov. 2003).

[4] A. Cockroft and R. Pettit, http://www.setoolkit.com

[5] SAS/ETS User's Guide Version 8, SAS Institute, Inc., Cary, NC, 1999.

Contact Information

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

Kenny C. Gross, Ph.D.
RAS Computer Analysis Laboratory
Sun Microsystems, Inc.
9525 Towne Centre Drive, USAN10-103
San Diego, CA 92121
Kenny.Gross@sun.com