Paper 277-28

Accelerating Performance of SAS® Applications via Rapid Extraction and Multiprocessing

John M. LaBore, Eli Lilly and Company, Indianapolis, IN
Fred R. Forst, Eli Lilly and Company, Indianapolis, IN

ABSTRACT

All large commercial operations generally face the same problems: massive amounts of data; hardware/software constraints; and tight deadlines to generate analyses and reports. Handling these situations has always been a SAS software strongpoint. However, even within SAS, determining the best (i.e., fastest) method is not always easy, since SAS typically provides multiple ways to accomplish the same task. This is further complicated by the common situation within companies of having multiple hardware platforms, multiple datasources, and a vast array of reporting requirements and needs. Today's rapid rate of mergers and acquisitions, the creation of joint ventures, and the evolution of all types of working alliances/collaborations further increase the likelihood of such complex computing environments. In this paper, we examine real-time performance across a variety of hosts, clients, and datasources using different approaches to extract and process data.

INTRODUCTION

There is a wide array of platforms, datasources, extraction, and processing possibilities across the computing spectrum, and often even within organizations. It would be difficult to assemble a comprehensive set of test cases, so we restricted our test cases to the platforms, client/server structures, processing methods, and forms of distributed processing of immediate interest. Our test cases comprised a matrix of:

Platforms:
- OS/390
- Sun/Solaris Unix
- Windows 2000 (PC)

Client/server:
- Clients: OS/390, PC
- Hosts: OS/390, Unix, PC

Processing:
- Synchronous
- Asynchronous (MP)*

Distributed Processing
- Remote compute services (RCS)
- Remote library services (RLS)

Data Size
- Medium
- Large

**Asynchronous processing is also known as parallel processing; SAS uses the term multiprocessing (MP) to describe asynchronous/parallel processing.*

Unix was not used as a client since the available system did not have a defined TCP/IP service to send requests (e.g. SIGNON, RSUBMIT, etc) to the 'listening' TCP/IP service (spawner) on OS/390. The lack of a connection between Unix and OS/390's spawner prevented parallel signons to OS/390.

Our interest was to evaluate real-time performance of various permutations of this matrix and develop guidelines for end users. Various authors have addressed aspects of this issue, notably Bentley (2000, 2002), Theuwissen and Croonen (2002), and Brinsfield (1999). As Brinsfield stated, "Most software applications can be tuned by optimizing data access, data processing, and data display. With client-server systems, you must also try to optimize machine-to-machine transfer rates, location of processing, and location of data." Additionally, he states "You must strive to reduce the transfer of data across the network or modem and attempt to give the work to the machine with the muscle. The basic concept is quite simple, but applying the idea often requires some extra time and analysis."

We also wanted to evaluate the balance between 1) Speed of a particular machine (the 'muscle'), 2) The 'penalty' associated with data transfer across a network, and 3) The ability/simplicity to take advantage of multi-processing. For example, perhaps the speed of a machine was not so important if we could more easily utilize parallel processing across several slower machines. Or perhaps there is no advantage to parallel processing when there are small data sources. Perhaps, even no amount of CPU speed or parallel processing could overcome the inherent slowness associated with RLS. We sought to answer this empirically. The following test cases represent our efforts to answer these questions.

TESTING ENVIRONMENT

The following chart shows performance parameters of the hardware used for these test cases. This table does not show all possible platforms for which SAS is available; however, these were the platforms available when testing commenced, and represent a relatively common set (individually and collectively) in use across many organizations. These numbers merely give a basis of the differences between our platforms. It is not intended to be a comparison between, say, Unix and OS/390 simply because our Unix processors and our OS/390 processors on the test systems are not necessarily from the same era.

Platform	Speed of CPUs	Elapsed time to execute a large DATA step (450K obs) (CPU intensive work)	Elapsed time to read 145Mb file (I/O intensive work)
OS/390	150 MIPS (570 Mhz)	19 minutes	16.4 seconds
Solaris Unix	900 Mhz	7 minutes	15.2 seconds
Windows 2000	733 Mhz	40 minutes	11 seconds

SAS Version 8 was the SAS version used on each of the computing platforms during the execution of the test cases.

DATA SIZES

In our testing, we elected to assess only medium (M) and large (L) data sizes. These two sizes were defined:

Category	Observations	Size
Medium	50,000 observations	15 Mb
Large	1,600,000 observations	502 Mb

Small data sizes were left out 1) to reduce the number of test cases, and 2) because when dealing with a small amount of data, any asynchronous method will be slower than the synchronous method due to the overhead cost of creating the parallel sessions. Each creation of a parallel task takes a few (4-10) seconds for the sign-on.

TEST CASES

For each data size (medium and large) there were six datasets created (MEDIUM0-MEDIUM5 and LARGE0-LARGE5). These datasets were replicated into SAS libraries on the PC, Unix, and OS/390 platforms. The goal of each test case was to execute the sugi28 macro (see next section) six times, once for each dataset of the same size (e.g. MEDIUM0-MEDIUM5) and to record the time from start to finish. Some test cases ran all six tasks synchronously and others took advantage of asynchronous (e.g., SAS MP Connect) methods to use additional CPUs on the same system (commonly referred to as "scaling-up"). Some test cases took advantage of asynchronous execution to use additional CPUs on remote hosts (referred to as "scaling-out"). Details regarding each test case are provided in Appendix A.

THE SUGI28 MACRO

A "sugi28" macro (see Appendix B) was created that performs a mixture of DATA steps and PROCs. These DATA steps and PROCs simulate common SAS tasks performed against medium to large datasets in a commercial setting. Tasks were weighted heavily (in terms of CPU time used) toward DATA steps, with significant weighting also given to PROC SORTs and PROC FREQs, and some weighting given to PROC PRINTs and PROC CONTENTS.

TEST CASES 1 THROUGH 6: NATIVE SYNCHRONOUS METHODS

In these cases all six tasks were performed synchronously on a single platform where the data resided on the same platform. SAS/Connect, RCS, and RLS were not used. These results provide a benchmark against which the other methods can be compared.

Test Case	Description	Data Size	Elapsed Time
1	Native OS/390	M	0 min 30 sec
2	Native OS/390	L	20 min 0 sec
3	Native Unix	M	0 min 24 sec
4	Native Unix	L	6 min 48 sec
5	Native PC	M	0 min 42 sec
6	Native PC	L	40 min 25 sec

TEST CASES 7 THROUGH 12: OS/390 AS CLIENT

In these cases the OS/390 system functioned as the client and either accessed (Cases 7 & 8) a Unix system to perform asynchronous processing, or accessed native OS/390 CPUs (Cases 9 & 10) to perform asynchronous processing via SAS MP Connect, or accessed data using RLS from a Unix system (Cases 11 & 12) to perform synchronous processing on OS/390.

Test Case	Description	Data Size	Elapsed Time
7	Async Unix RCS	M	0 min 17 sec
8	Async Unix RCS	L	4 min 7 sec
9	Async OS/390 MP	M	0 min 43 sec
10	Async OS/390 MP	L	8 min 10 sec
11	Sync Unix RLS	M	3 min 6 sec
12	Sync Unix RLS	L	87 min 14 sec

TEST CASES 13 THROUGH 20: PC AS CLIENT

In these cases a mixture of synchronous and asynchronous methods combined with either RLS or RCS was used. In Cases 13 & 14, PC SAS executes the six tasks synchronously on the PC using RLS to read the data from OS/390. In Cases 15 & 16, the same process is used, but with the data being read from Unix. RCS is used in Cases 17 & 18 to asynchronously RSUBMIT to OS/390, which then uses RLS to read data from Unix. Cases 19 & 20 represented an effort to determine the effect on elapsed time from a process that used asynchronous RCS from the PC to OS/390 that was combined with the use of FTP and SAS hiperspaces on OS/390.

Test Case	Description	Data Size	Elapsed Time
13	Sync OS/390 RLS	M	5 min 1 sec
14	Sync OS/390 RLS	L	200 min 0 sec
15	Sync Unix RLS	M	1 min 21 sec
16	Sync Unix RLS	L	59 min 32 sec
17	Async OS/390 RCS + Unix RLS	M	2 min 3 sec
18	Async OS/390 RCS + Unix RLS	L	58 min 54 sec
19	Async Special Case 1*	M	4 min 1 sec
20	Async Special Case 2*	L	120 min 33 sec

*Async Special Case 1 and Special Case 2: PC as the client executes six tasks asynchronously each of which uses Remote Compute Services (RCS) to Remote Submit the sugi28 macro to OS/390. The OS/390 tasks acquire medium or large size datasets from Unix using FILENAME FTP to PROC COPY each dataset from a Unix SAS transport file to an OS/390 hiperspace file, where the sugi28 macro processes it.

TEST CASES SUMMARY

it is readily apparent that use of Remote Library Services (RLS) degrades performance across all these cases. Degradation is substantial when data sets are large. This is not unexpected, since RLS pulls data from the remote system record by record.

The speed boost provided by asynchronous processing is readily apparent when comparing Test Cases 1 & 2 with Test Cases 9 & 10. Although the performance degraded slightly for the medium data sets, in the large data set cases the elapsed time dropped by more than half.

Perhaps the most surprising result was the performance achieved in Test Cases 7 & 8. The elapsed time for the asynchronous execution of tasks on Unix via RCS from an OS/390 client actually beat the performance of asynchronous execution of tasks via other CPUs on the same OS/390 system. In other words, "scaling out" of the tasks to another system with faster CPUs was better, in both the medium and large data set scenarios, than "scaling up" the tasks on the same system.

CONCLUSION

The twenty test cases described in this paper permit several conclusions:

- Avoid RLS at almost all costs, especially in cases of large data sets
- There is a benefit to performing tasks using asynchronous methods

- There can be a benefit to asynchronous "scaling-out" of tasks to faster CPUs on remote systems
- The benefit in performance using asynchronous methods (whether scaling-up or scaling-out) can be particularly significant in cases involving large datasets

REFERENCES

Bentley, John E., (2000) "An Introduction to Parallel Computing", Proceedings of the Twenty-Fifth Annual SAS Users Group International Conference, 1513-1522.

Bentley, John E. (2002) "Using SAS to Extract From a Parallel RDBMS", The SESUG Informant, Vol. 4:2, 16-18.

Brinsfield, Eric (1999) "SAS Client-Server Development: Through Thick and Thin and Version 8", Proceedings of the Seventh Annual Conference of the SouthEast SAS Users Group, 427-433.

Garner, Cheryl, (2000) "Multiprocessing with Version 8 of the SAS System", Proceedings of the Twenty-Fifth Annual SAS Users Group International Conference, 95-100.

SAS Institute Inc., Multiprocessing with SAS Software Course Notes, Cary, NC: SAS Institute Inc., 2002.

Theuwissen, Henri, and Nancy Croonen, (2002) "SAS Accessing DB2 Data: a Performance Victory Away", Proceedings of the Twenty-Seventh Annual SAS Users Group International Conference, 1-9.

ACKNOWLEDGMENTS

The authors would like to acknowledge the assistance of Thomas H. Burger and John Krogulecki for reviewing technical aspects of this manuscript, and Linda E. LaBore for editing assistance.

CONTACT INFORMATION

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

John M. LaBore
Lilly Corporate Center
Eli Lilly and Company
DC 1745
Indianapolis, IN 46285
Work phone: 317.277.6387
Email: jml@lilly.com
Web: www.lilly.com

Fred R. Forst
Lilly Corporate Center
Eli Lilly and Company
DC 2254
Indianapolis, IN 46285
Work phone: 317.276.0842
Email: frf@lilly.com
Web: www.lilly.com

APPENDIX A

Description of Test Cases

Test Case 1:
Native MVS: All six iterations of the sugi28 macro executed synchronously against medium size datasets.

Test Case 2:
Native MVS: All six iterations of the sugi28 macro executed synchronously against large size datasets.

Test Case 3:
Native Unix: All six iterations of the sugi28 macro executed synchronously against medium size datasets.

Test Case 4:
Native Unix: All six iterations of the sugi28 macro executed synchronously against large size datasets.

Test Case 5:
Native PC: All six iterations of the sugi28 macro executed synchronously against medium size datasets.

Test Case 6:

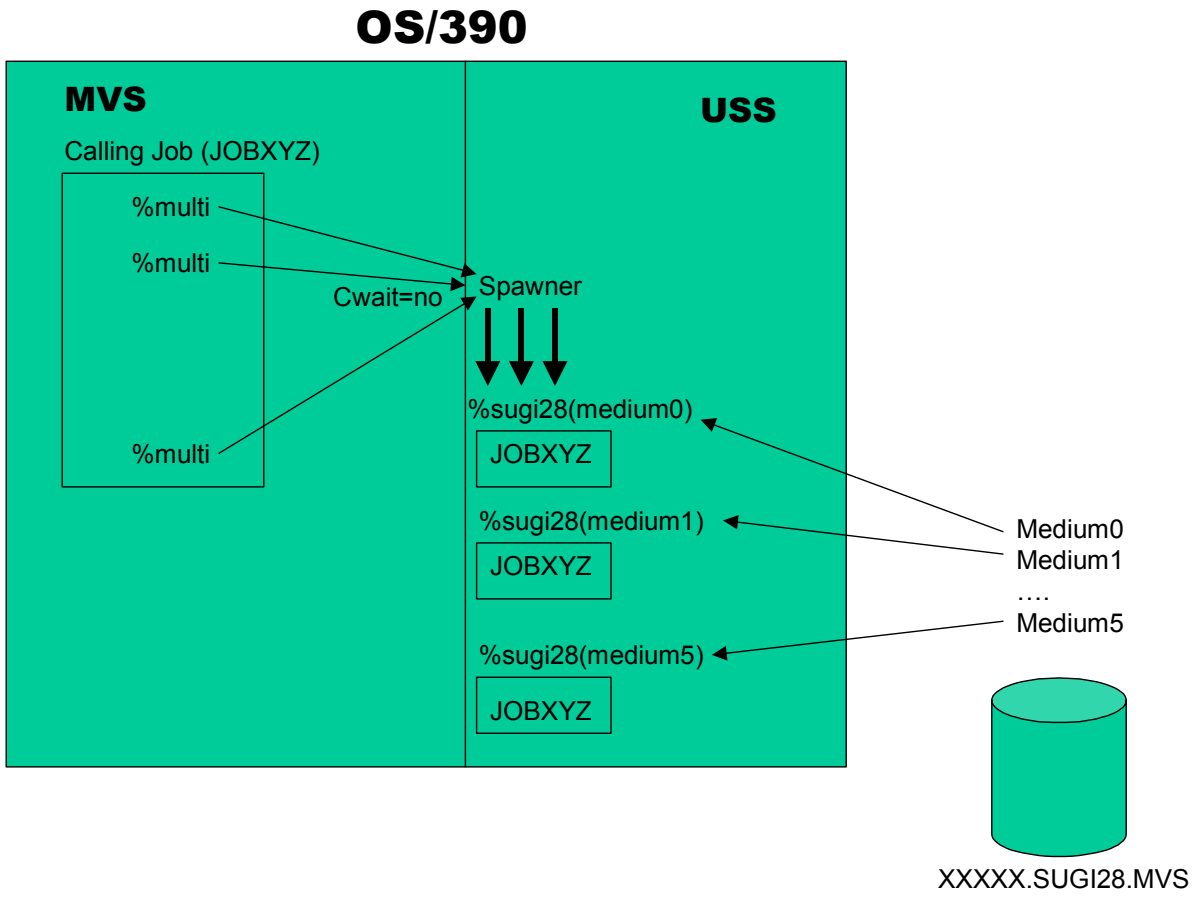
Native PC: All six iterations of the sugi28 macro executed synchronously against large size datasets.

Test Case 7:
OS/390 as the client, with Unix as the remote system. All six iterations of the sugi28 macro executed asynchronously on Unix against medium size datasets on Unix via SAS/Connect RCS (Remote Computing Services) between OS/390 and Unix.

Test Case 8:
OS/390 as the client, with Unix as the remote system. All six iterations of the sugi28 macro executed asynchronously on Unix against large size datasets on Unix via SAS/Connect RCS (Remote Computing Services) between OS/390 and Unix.

Test Case 9:
OS/390 as the client Remote Submits all six iterations of the sugi28 macro asynchronously (against medium size datasets) on different CPUs on OS/390. See the figure (below) for a graphical illustration of the process.

Note: USS stands for Unix System Services, which is the Unix subsystem of OS/390 that allows Unix processes to run on OS/390. The OS/390 Web server is a USS task, SAS/Intrnet has USS pieces, FTP is a USS task, etc. The SAS/CONNECT relevance comes into play because with the availability of USS and the spawner, a user can fire up a SAS/CONNECT session from their PC and signon to a USS task instead of TSO. There is no TSO script file involved (thus signon is slightly faster) and the real advantages are 1) Multiple sessions can be signed on, 2) TSO is still available for other non-SAS processes.



Test Case 10:
OS/390 as the client Remote Submits all six iterations of the sugi28 macro asynchronously (against large size datasets) on different CPUs on OS/390.

Test Case 11:
OS/390 as the client executes all six iterations of the sugi28 macro synchronously, using Remote Library Services (RLS) to read medium size datasets from Unix.

Test Case 12:

OS/390 as the client executes all six iterations of the sugi28 macro synchronously, using Remote Library Services (RLS) to read large size datasets from Unix.

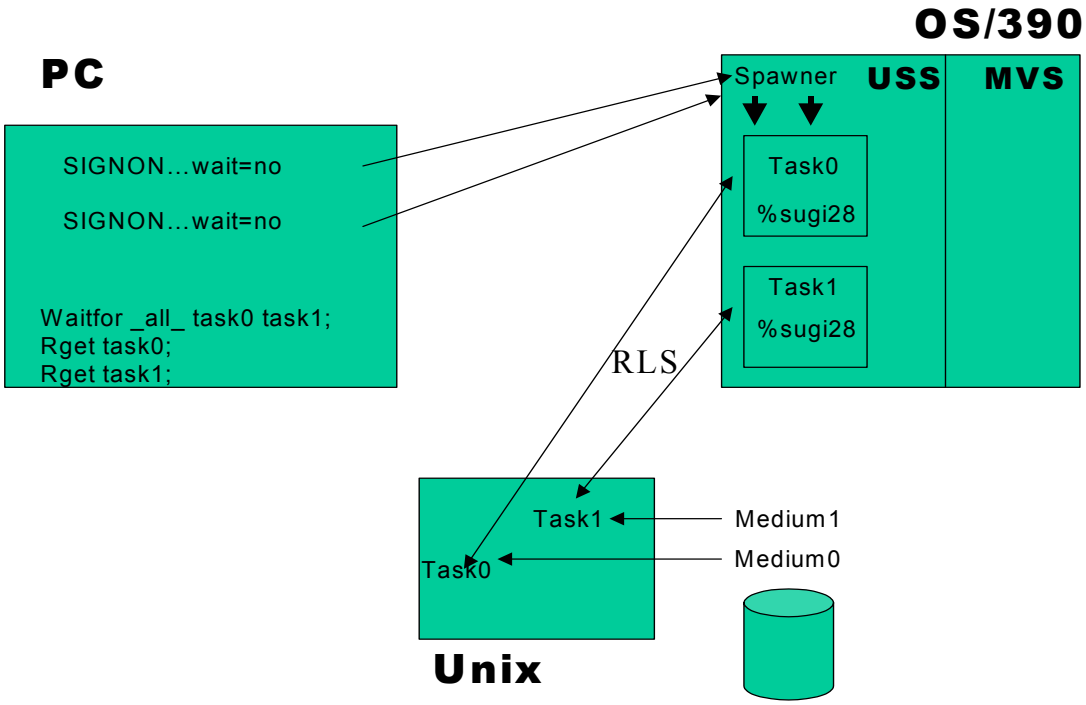
Test Case 13:
PC as the client executes all six iterations of the sugi28 macro synchronously, using Remote Library Services (RLS) to read medium size datasets from OS/390.

Test Case 14:
PC as the client executes all six iterations of the sugi28 macro synchronously, using Remote Library Services (RLS) to read large size datasets from OS/390.

Test Case 15:
PC as the client executes all six iterations of the sugi28 macro synchronously, using Remote Library Services (RLS) to read medium size datasets from Unix.

Test Case 16:
PC as the client executes all six iterations of the sugi28 macro synchronously, using Remote Library Services (RLS) to read large size datasets from Unix.

Test Case 17:
PC as the client executes six tasks asynchronously on the PC, each task uses Remote Compute Services (RCS) to Remote Submit the sugi28 macro asynchronously to OS/390. The OS/390 task then uses Remote Library Services (RLS) to read medium size datasets from Unix. See the figure (below) for a graphical illustration of the process. Note: for simplicity, only two concurrent tasks are illustrated in the graphic; the test case actually executed six concurrent tasks.



Test Case 18:
PC as the client executes six tasks asynchronously on the PC, each task uses Remote Compute Services (RCS) to Remote Submit the sugi28 macro asynchronously to OS/390 . The OS/390 task then uses Remote Library Services (RLS) to read medium size datasets from Unix.

Test Case 19:
PC as the client executes six tasks asynchronously each of which uses Remote Compute Services (RCS) to Remote Submit the sugi28 macro to OS/390. The OS/390 tasks acquire medium size datasets from Unix using FILENAME FTP to PROC COPY each dataset from a Unix SAS transport file to an OS/390 hiperspace file, where the sugi28 macro processes it.

Test Case 20:
PC as the client executes six tasks asynchronously each of which uses Remote Compute Services (RCS) to Remote Submit the sugi28 macro to OS/390. The OS/390 tasks acquire large size datasets from Unix using FILENAME FTP to PROC COPY each dataset from a Unix SAS transport file to an OS/390 hiperspace file, where the sugi28 macro processes it.

APPENDIX B

SUGI28 Macro

```
%MACRO SUGI28(SIZE=&SIZE,I=&I);
   DATA &SIZE.&I;SET in.&SIZE.&I in.&SIZE.&I;
     LENGTH X $ 10. ;
     PCCPUBY=INT(PCCPUBY);
     INAVG=INT(INAVG);
     IORATE=INT(IORATE);
     BACHAVG=INT(BACHAVG);
     READYMAX=MAX(OF READY00-READY15);
     READYMIN=MIN(OF READY00-READY15);
     IF 0  LE PCCPUBY LE 10 THEN X='SLOW';
     IF 10 LT PCCPUBY LE 50 THEN X='MEDIUM';
     IF 50 LT PCCPUBY LE 80 THEN X='BUSY';
     IF       PCCPUBY GT 80 THEN X='VERY BUSY';
     WHERE (MACHINE='SYSA' OR MACHINE='SYSB')
       AND DATE GE '01JAN95'D;;

   PROC SORT DATA=&SIZE.&I;BY MACHINE HOUR
     TSOMAX BATMPL BACHAVG;
   DATA TEMP&I;SET &SIZE.&I;
     BACHAVG=INT(BACHAVG);
     BACHMAX=CEIL(BACHMAX);
     BACHMIN=INT(BACHMIN);
     BACHRAT=INT(BACHRAT);
     BATMPL=CEIL(BATMPL);
     BATRESP=INT(BATRESP);
     BATSUSEC=CEIL(BATSUSEC/1000);
     BATSVC=SQRT(BATSVC);
      IF INMAX GE 5 THEN DO;
        %DO K=10 %TO 15;
         READY+READY&K;
        %END;
      END;
      IF TSOMAX GE 5 THEN DO;
        MAXINIT=MAX(BACHMAX,TSOMAX);
        MININIT=MIN(BACHMIN,TSOMIN);
      END;
      IF TSOAVG GE 2 THEN DO;
        RESP=MEAN(BATRESP,TSORESP);
      END;

      WHERE MACHINE='SYSA' ;
   DATA _NULL_;
   SET TEMP&I ;
      DO I=1 TO 200;
        BURN=RANUNI(3993030);
      END;
   PROC SORT DATA=TEMP&I ;
     BY X MACHINE READY DATETIME;
   PROC SORT DATA=TEMP&I OUT=JUNK&I;
     BY DESCENDING DATETIME BATSVC BATTRANS;
   PROC FREQ DATA=TEMP&I ORDER=FREQ;
     BY X  MACHINE;
     TABLES HOUR INMIN INMAX INAVG TSOAVG TSOMIN
       TSOMAX BAT: /
     NOPRINT OUT=FREQ&I;

   PROC UNIVARIATE NOPRINT DATA=&SIZE.&I;
   BY MACHINE HOUR;
     VAR INAVG INMAX INMIN PCCPUBY IORATE;
     OUTPUT OUT=OUT&I MEAN=INAVG INMAX INMIN
     PCCPUBY IORATE ;
```

```
   PROC PRINT DATA=OUT&I;
     TITLE1 "OUT&I";
   PROC PRINT DATA=&SIZE.&I(OBS=1000);
     TITLE1 "&SIZE.&I";
     VAR HOUR MACHINE INAVG PCCPUBY IORATE DATE
     BACHAVG;
   PROC CONTENTS DATA=WORK.OUT&I.;
   PROC DATASETS LIBRARY=WORK;
        MODIFY OUT&I;
            INDEX CREATE HOUR;
        MODIFY TEMP&I;
            INDEX CREATE READY HOUR MAXINIT
            MININIT;
        DELETE TEMP&I JUNK&I FREQ&I;

%MEND SUGI28;
```

APPENDIX C

An example of SAS code used to invoke asynchronus multiprocessing for one of the test cases. This instance illustrates the launching of six threads on an OS/390 system to process large datasets using the “multi” macro (see appendix D for an example of the multi macro).

```
   OPTIONS LS=80 NOOVP FULLSTATS SGEN MPRINT
      sasautos=('xxxxxx.sas.macros' sasautos);

   *-------------------------------------------;
   *                                           ;
   * SIZE=MEDIUM OR LARGE                      ;
   * I=0 THROUGH 5                             ;
   *                                           ;
   *-------------------------------------------;

   *-------------------------------------------;
   *                                           ;
   *  GET START TIME                           ;
   *                                           ;
   *-------------------------------------------;

   DATA _NULL_;
    TIME=TIME();
    CALL SYMPUT('START',TIME);
     RUN;

   *-------------------------------------------;
   *                                           ;
   * submit tasks to remote platform           ;
   *                                           ;
   *-------------------------------------------;
   %let mvs0=mvs.spawner;
   %let mvs1=mvs.spawner;
   %let mvs2=mvs.spawner;
   %let mvs3=mvs.spawner;
   %let mvs4=mvs.spawner;
   %let mvs5=mvs.spawner;

   %multi(SIZE=MEDIUM,I=0,platform=MVS,remote=mvs0);
   %multi(SIZE=MEDIUM,I=1,platform=MVS,remote=mvs1);
   %multi(SIZE=MEDIUM,I=2,platform=MVS,remote=mvs2);
   %multi(SIZE=MEDIUM,I=3,platform=MVS,remote=mvs3);
   %multi(SIZE=MEDIUM,I=4,platform=MVS,remote=mvs4);
   %multi(SIZE=MEDIUM,I=5,platform=MVS,remote=mvs5);
     RUN;

   waitfor _all_ mvs0 mvs1 mvs2 mvs3 mvs4 mvs5;

   rget mvs0;
   rget mvs1;
   rget mvs2;
   rget mvs3;
   rget mvs4;
   rget mvs5;
```

```
*-----------------------------------------;
*                                         ;
*  GET END   TIME AND CALCULATE ELAPSED TIME;
*                                         ;
*-----------------------------------------;
DATA _NULL_;
  FORMAT START 8.;
  TIME=TIME();
  START=SYMGET('START');
   DURATION=TIME-START;
   PUT '****** START TIME= ******' START    TIME12.5;
   PUT '****** END   TIME= ******' TIME     TIME12.5;
   PUT '****** DURATION  = ******' DURATION TIME12.2;
   RUN;
```

APPENDIX D

An example of the "multi" macro executed by the code in Appendix C.

```
%MACRO multi(SIZE=,I=,platform=,remote=,);

 /*    MVS  EXECUTION    */

  %IF &platform=MVS     %THEN %DO;
    OPTIONS autosignon=yes cwait=no                                        /* async signons          */
     sascmd="x: comamid=xmsn SASAUTOS=('XXXXXX.sas.macros' sasautos    /* sascmd implies MP      */

          MAUTOSOURCE" ;
     SIGNON &remote ;                                                      /* signon to mvsn.spawner */
     %syslput size=&size ;                                                 /* pass local macro vars  */
     %syslput i=&i;                                                        /* into MP session        */

     RSUBMIT  process=&remote;                                             /* rsub to mvsn.spawner   */
        LIBNAME in     'XXXXXX.SUGI28.MVS'  DISP=SHR;
        %sugi28(size=&size,i=&i) ;                                         /* Run for specific size  */
        run;                                                               /* and I                  */
     endrsubmit;
  %END;

 /*    unix execution    */

  %else %IF &platform=SunOs %THEN %DO;
     FILENAME RLINK 'XXXXXX.CONNECT.RLINK(TCPUNIX)' ;
     OPTIONS REMOTE=&REMOTE COMAMID=TCP;
     SIGNON &remote ;
     %syslput size=&size ;
     %syslput i=&i;
     RSUBMIT wait=no;
       FILENAME RLINK '/home/xxxxxx/mvs1' ;
       OPTIONS mautosource SASAUTOS=('/home/XXXXXX/macros' )
               REMOTE=MVS comamid=tcp SET=TCPTN3270 1;
      /* LIBNAME in     '/home/xxxxxx/data'    */ ;
      SIGNON MVS;
      LIBNAME in REMOTE 'XXXXXX.SUGI28.MVS' SERVER=mvs;
        %sugi28(size=&size,i=&i) ;
        run;
      SIGNOFF MVS;
     endrsubmit;
  %END;

 /*    pc execution    */

   %ELSE %IF &platform=WIN_PRO %THEN %DO;
     LIBNAME in      'c:\documents and settings\jl86201'  ;
   %END;
%mend multi;
```