

Analyzing Mainframe Resources using Starting and Ending Time Stamps

By Richard Banks, Texas Comptroller of Public Accounts

Abstract: One approach to analyzing resource usage is to plot the number of concurrent tasks, such as the number of active TSO sessions, over some time period. This paper shows how to use starting and ending time stamps to produce such plots, showing the number of concurrent processes versus the time-of-day. This particular study used data collected both from the MXG¹ and MICS² computer performance database products. These data, consisting of the starting and ending time stamps for several types of IBM mainframe tasks, were "counted" in SAS DATA steps using SAS arrays. Then, the summarized data were analyzed using PROC GPLOT. SAS macros were used to generalize the SAS programs used for the data reduction and the analysis.

Computer Performance Evaluation

The computer performance analyst often is concerned with the amount of computer activity over some period of time. Because large mainframe computers are expensive, businesses who must buy and maintain them insist on getting as much work as possible processed by their machine before the inevitable day when the mainframe must be replaced -- by an even larger mainframe. Computer performance analysts provide business managers with information about the various workloads processed on the mainframe. Performance analysts also use information about these workloads to manage them and to troubleshoot system performance problems.

Because of the importance of large mainframe computer installations, an entire industry has grown up around the monitoring and reporting of computer performance. IBM mainframes and plug compatibles that run the MVS operating system are configured with an internal software monitor called the System Management Facilities (SMF). Many important system events are recorded in the SMF: for example, each time a user signs on or off, each time a job calls for a tape to be mounted, and each time a dataset is deleted from or added to the catalog, an appropriately formatted record is written to the SMF. In addition to important system events, hardware utilization statistics are written to the SMF. This information, written at specific intervals, is recorded by the Resource Measurement Facility (RMF).

In addition to the SMF and RMF, which are actually a part of IBM's MVS operating system, a number of third party software vendors market software for monitoring mainframe work. Some of these, the "real time" monitors, provide information about system events as they occur and are useful for troubleshooting problems or performance bottlenecks. Other monitors simply write additional information to the SMF, information that can be summarized and stored for later use. Another type of monitoring software is the performance database. A performance database summarizes SMF and RMF data and stores it in a form that performance analysts and capacity planners can use.³ The MICS and MXG performance databases consist of SAS files, making them easily available for data analysis in SAS. The performance graphics used here were produced from information collected by MXG and MICS.

This paper grew out of some work done at the Comptroller's office in 1990, where the Systems Support Group needed to know how many batch jobs were running in a particular workload grouping or "job class" at different times during the day. Since the starting and ending times of every batch job are maintained in the performance database, it occurred to me that I could use this information to build a record of activity for a particular job class. I will describe how we used starting and ending time stamps to analyze these batch workloads and then show how this technique can be applied to other kinds of system activity.

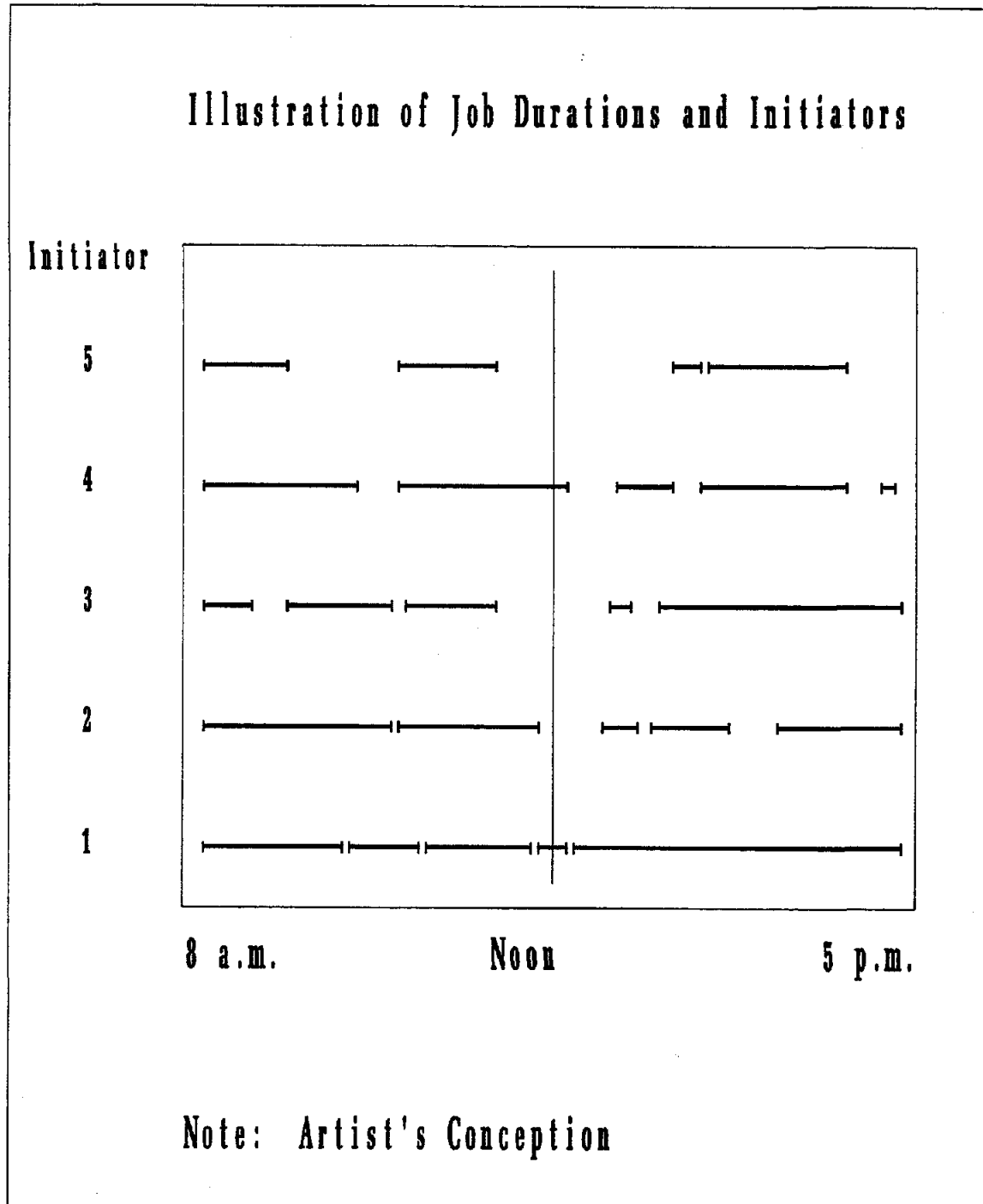
Analyzing Time Stamped Events

This analysis of time-stamped events began as an attempt to determine how many batch jobs in certain job classes were executing during the course of the business day. The MVS operating system will only allow a specified number of jobs in a given class to start. Before a job can start, it must be the highest priority job in its class, and there must also be an available "initiator" in that class.⁴ The number of jobs running in a job class is equivalent to the number of initiators in use for that class.

The purpose of our original study was to determine how many available initiators there were in the two "user" job classes at the Comptroller: the "U" class for long-running jobs and the "V" class for shorter jobs that did not require any access to data on tape. There was no way to determine this directly since we had no monitor that reported initiator usage. But knowing how many initiators are in use is equivalent to knowing how many "U" jobs and how many "V" jobs were in the system, and we did have SMF records for the start time and end time and the job class of each job. The data reduction program described below used the start and end time for each job to build an array of counters. Each element in the array represented one second and was used to count the number of jobs running at that second. The array could represent whatever time interval we were interested in, depending on how many elements (seconds) it included. We wanted to look at the period of time between 8 a.m. and 5 p.m., a duration of 32,400 seconds.

Figure 1 attempts to convey the idea of a number of concurrently running batch jobs over time. At any time there are zero to five initiators available for additional jobs to start. By moving a vertical line across the graph from left to right, one can see some number of intersections with horizontal lines (batch jobs). Each one of these intersections represents an initiator in use at that moment. For example, the vertical line represents a view of initiators in use at noon. At this moment two initiators are in use and three are available for additional jobs. A job submitted at this moment would find an available initiator and would start right away, assuming it was able to "allocate" or secure access all of the other resources it needed.

Figure 1: Illustration of Job Duration and Initiators



The SAS program described below performs an analogous function to the vertical line in figure 1. For each second of the business day, 8 a.m. to 5 p.m., the program counts the number of jobs running in some job class. This amounts to reducing the data to a form where it can be analyzed by SAS procs, specifically PROC FREQ and PROC GPLOT. Figure 2 lists the SAS DATA step where the counting takes place.

The input to this DATA step consists of a SAS dataset where each observation represents a job that ran during the previous day. Each observation contains two variables, STARTTS and ENDTS, representing the starting and ending times for the job. The two variables are numeric and were originally read using the informat SMFSTAMP8., meaning the variables contain both a date and a time. We are interested only in the times for this DATA step (the dataset having already been subsetting by date in a previous step). The start or end times can be extracted from the variables by using the TIMEPART function.

Figure 2: Data Reduction Routine

```
DATA OUT.RUNTIME;
  SET CLASSU END=DONE;

  ARRAY MU { * } MU00001 - MU32400;
  LENGTH MU00001 - MU32400 2;
  TIME 4 STARTSEC ENDSEC 5 CLASSU 2;
  RETAIN MU00001 - MU32400 0 GRAFDATE "&RUNDATE";
  RETAIN STHR 8 ENHR 16;
  KEEP CLASSU TIME GRAFDATE;
  FORMAT TIME TIME8.;

  IF HOUR(TIMEPART(STARTTS)) LT STHR THEN STARTSEC=1;
  ELSE STARTSEC=(TIMEPART(STARTTS) + 1) - (STHR * 60 * 60);
  IF HOUR(TIMEPART(ENDTS)) GT ENHR THEN ENDSEC=32400;
  ELSE ENDSEC=(TIMEPART(ENDTS) + 1) - (STHR * 60 * 60);

  DO CURSEC=STARTSEC TO ENDSEC;
    MU{CURSEC} + 1;
  END;

  IF DONE THEN DO INDEX=1 TO 32400;
    CLASSU=MU{I};
    TIME=INDEX + (STHR * 60 * 60);
    OUTPUT;
  END;
```

The method by which the routine counts the number of jobs is to create an array of seconds in the day, each cell in the array representing one second. For every job, the starting and ending second of the job's run correspond to the beginning and ending cell in the array to be incremented. So, for example, if we have limited the program to jobs running between 8 a.m. and 5 p.m. and it encounters a job which ran from 8:00:00 a.m. through 8:05:00 a.m., then the program will increment cells 1 through 301 hundred (the first 301 seconds in the 8 a.m. to 5 p.m. business day).

The array is defined and initialized in the ARRAY, LENGTH and RETAIN statements:

```
ARRAY MU( *) MU00001 - MU32400;  
RETAIN MU00001 - MU32400 0 . . . ;  
LENGTH MU99991 - MU32400 2 . . . ;
```

The RETAIN statement is necessary since we want SAS to retain the value of each cell for the duration of the DATA step. The LENGTH statement limits storage to two bytes per cell; SAS would otherwise allocate 8 bytes of storage for each cell in the array. Under MVS, a length of two bytes is sufficient to store integers as large as 255.

Before the routine can increment its array cell counters, it must first handle two exceptional conditions: these are cases where (1) the job began prior to the beginning of the measurement period (8 a.m. in our example; (2) the job continues to run beyond the end of the measurement period (5 p.m. in this case). The following code sets the STARTSEC variable to the minimum value if the job began earlier than 8 a.m.; it sets the ENDSEC variables to the maximum value if the job continues to run past 5 p.m.

```
IF HOUR(TIMEPART(STARTTS)) LT STHR THEN STARTSEC=1;  
ELSE STARTSEC=(TIMEPART(STARTTS) + 1) - (STHR * 60 * 60);  
IF HOUR(TIMEPART(ENDTS)) GT ENHR THEN ENDSEC=32400;  
ELSE ENDSEC=(TIMEPART(ENDTS) + 1) - (STHR * 60 * 60);
```

Failure to handle these exceptions would lead to array subscripts out of range.

After the STARTSEC and ENDSEC variables have been determined, the corresponding array cells are incremented:

```
DO CURSEC=STARTSEC TO ENDSEC;  
  MU(CURSEC) + 1;  
END;
```

After all jobs have been counted, a new SAS dataset is created. Into this dataset one record is output for each cell in the array.

```
IF DONE THEN DO INDEX=1 TO 32400;  
  CLASSU=MU(I);  
  TIME=INDEX + (STHR * 60 * 60);  
  OUTPUT;  
END;
```

This new dataset will be used for subsequent analysis using PROC FREQ and PROC GPLOT.

Having explained the routine for reducing the start and end time stamps of batch jobs to an array representing initiator usage over a time period, I will show how to integrate this routine into a SAS program. The complete SAS program is shown in figure 3.

Figure 3: SAS Program to Read and Format Time Stamp Data

```

OPTIONS NODATE NONUMBER LS=72 SYMBOLGEN;
DATA DATE;
  CLASDATE=DATE() - 1;
  CALL SYMPUT("RUNDATE",PUT(CLASDATE,DATE7.));
RUN;

%MACRO CLASSNUM(CLASS=,BEGIN=,END=,STHR=,ENHR=,OUTFILE=);
  %LOCAL ARRAYSIZ;
  %LET ARRAYSIZ=%EVAL((((&ENHR + 1) - &STHR) * 3600));
  DATA CLASS&CLASS;
    SET IN.BATJOB01
        IN.BAT_JS01;
    IF PUT(DATEPART(&BEGIN),DATE7.)="&RUNDATE";
    IF HOUR(TIMEPART(&BEGIN)) GT &ENHR OR
       HOUR(TIMEPART(&END)) LT &STHR THEN DELETE;
    IF JOBCLASS="&CLASS";

  DATA OUT.&OUTFILE;
    SET CLASS&CLASS END=DONE;
    LENGTH M&CLASS.00001 - M&CLASS&ARRAYSIZ 2;
    TIME 4 STARTSEC ENDSEC 5 CLASS&CLASS 2;
    RETAIN M&CLASS.00001 - M&CLASS&ARRAYSIZ 0 GRAFDATE "&RUNDATE";
    KEEP CLASS&CLASS TIME GRAFDATE;
    FORMAT TIME TIME8.;
    ARRAY M&CLASS ( * ) M&CLASS.00001 - M&CLASS&ARRAYSIZ;
    IF HOUR(TIMEPART(&BEGIN)) LT &STHR THEN STARTSEC=1;
    ELSE STARTSEC=ROUND(TIMEPART(&BEGIN) + 1) - (&STHR * 60 * 60);
    IF HOUR(TIMEPART(&END)) GT &ENHR THEN ENDSEC=&ARRAYSIZ;
    ELSE ENDSEC=ROUND(TIMEPART(&END) + 1) - (&STHR * 60 * 60);
    DO CURSEC=STARTSEC TO ENDSEC;
      M&CLASS(CURSEC) + 1;
    END;
    IF DONE THEN DO INDEX=1 TO &ARRAYSIZ;
      CLASS&CLASS=M&CLASS{I};
      TIME=INDEX + (&STHR * 60 * 60);
      OUTPUT;
    END;

  PROC FREQ DATA=OUT.&OUTFILE;
    TITLE1 "Distribution of Jobs for Class &CLASS";
    TITLE2 "&RUNDATE";
    TABLES CLASS&CLASS;
  %MEND;

%CLASSNUM(CLASS=U,BEGIN=STARTTS,END=ENDTS,STHR=8,ENHR=16,OUTFILE=RUNTIME);

```

This program includes 4 parts: (1) an **OPTIONS** statement to set up the runtime environment; (2) a **SAS DATA** step to determine the date needed for date selection and then pass this date to subsequent steps as a **MACRO** variable; (3) a **MACRO** definition, including the definition and initialization of a **LOCAL** variable called **ARRAYSIZ**, a **DATA** step to specify the data, a **DATA** step to populate the array that will be used in the analysis and a **PROC FREQ** to perform the initial data analysis.

The **MACRO** variable **ARRAYSIZ** is used to define the number of array elements. It corresponds to the number of seconds to be analyzed and is derived from the **STHR** and **ENHR** variables in the **MACRO** definition. I should point out that the program, as presented here, creates an array of 32,400 elements. This implementation, using 32,400 individually named elements, is relatively close to the SAS limit (under MVS, at least) of 32,767 variables. Selecting a time interval much larger than 9 hours produces the following SAS runtime error:

ERROR: Too many variables defined for file OUT.RUNTIME. This file may not have more than 32767 variables.

The first DATA step in the MACRO selects the observations using the date provided by the external DATA step, the beginning and ending hours and the job class specified in the MACRO invocation.

The DATA step used for creating the array is similar to the one shown in figure 2. It differs primarily in its use of macro variables where hard coded values were used in figure 2. Notice also the elimination of the RETAIN STHR 8 ENHR 16; statement. This statement is no longer necessary since these values are supplied by the STHR and ENHR MACRO variables.

The PROC FREQ is used to create a frequency distribution of "U" initiators. In job class "U" there are 15 initiators. Figure 4 shows that all 15 initiators were in use for almost 25% of the business day. Another interesting outcome is that for 56 seconds, there appear to have been 16 initiators in class "U". This is misleading. The apparent "extra" initiator is explainable only as the case where one job ends and the next job starts during the same second. This anomaly could be prevented by decreasing the size of reporting interval below one second. Some will argue that the data are pretty grainy already, without increasing their granularity by looking at fractions of seconds. But it could be done (for intervals small enough to avoid exceeding the 32,767 variable limit). The upward bias introduced into the analysis by the phenomenon of one job starting immediately after another ends can also be controlled by another method that will be explained shortly.

Figure 4: Output from PROC FREQ

Distribution of Jobs for Class U 21JUL92				
CLASSU	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	285	0.9	285	0.9
2	527	1.6	812	2.5
3	759	2.3	1571	4.8
4	2673	8.3	4244	13.1
5	2643	8.2	6887	21.3
6	3666	11.3	10553	32.6
7	3320	10.2	13873	42.8
8	2057	6.3	15930	49.2
9	1971	6.1	17901	55.2
10	1673	5.2	19574	60.4
11	1593	4.9	21167	65.3
12	1267	3.9	22434	69.2
13	1010	3.1	23444	72.4
14	995	3.1	24439	75.4
15	7905	24.4	32344	99.8
16	56	0.2	32400	100.0

Graphical Analysis

The data lend themselves quite nicely to graphical analysis in SAS. The following SAS program, listed in Figure 5 is one approach to analyzing the data graphically, producing a scatter plot created with **PROC GPLOT**.

Figure 5: SAS Program for Plotting Batch Initiator Data using PROC GPLOT

```
GOPTIONS VPOS=100 HPOS=100 DEVICE=GDDMPCG NODISPLAY FTEXT=SWISSX;
LIBNAME CLASSES 'UTIL.S96.SAS.CLASSES' DISP=SHR;

PROC DATASETS LIB=GRAPHCAT MT=CATALOG;
  DELETE CLASSES;
RUN;

%MACRO CLASS(CLASS=,COLOR=,IN=,TO=,BY=,TICK1=,TICK2=,TICK3=,TICK4=,TICK5=,N=);
  %GLOBAL DATE;
  DATA ALL;
    SET CLASSES.&IN END=EOF;
    IF EOF THEN CALL SYMPUT("DATE",GRAFDATE);
  RUN;

  PROC GPLOT DATA=ALL GOUT=GRAPHCAT.CLASSES;
    SYMBOL1 I=JOIN C=&COLOR H=1.0;
    PLOT CLASS&CLASS*TIME /
      NAME="CLASS&CLASS&N" VM=0 HM=3 HAXIS=AXIS2 VAXIS=AXIS1;
    AXIS1 ORDER=(0 TO &TO BY &BY) COLOR=WHITE
      LABEL=( JUSTIFY=RIGHT H=3PCT 'Jobs')
      VALUE=(H=3PCT TICK=1 JUSTIFY=R "&TICK1"
        H=3PCT TICK=2 JUSTIFY=R "&TICK2"
        H=3PCT TICK=3 JUSTIFY=R "&TICK3"
        H=3PCT TICK=4 JUSTIFY=R "&TICK4"
        H=3PCT TICK=5 JUSTIFY=R "&TICK5")
      MINOR=(NUMBER=1 COLOR=WHITE);
    AXIS2 ORDER=(28800 TO 61200 BY 3600) COLOR=WHITE
      LABEL=( JUSTIFY=CENTER H=3PCT ' ')
      VALUE=(H=3PCT TICK=1 JUSTIFY=C " 8 a.m."
        H=3PCT TICK=2 JUSTIFY=C " "
        H=3PCT TICK=3 JUSTIFY=C " "
        H=3PCT TICK=4 JUSTIFY=C " "
        H=3PCT TICK=5 JUSTIFY=C "Noon"
        H=3PCT TICK=6 JUSTIFY=C " "
        H=3PCT TICK=7 JUSTIFY=C " "
        H=3PCT TICK=8 JUSTIFY=C " "
        H=3PCT TICK=9 JUSTIFY=C " "
        H=3PCT TICK=10 JUSTIFY=C "5 p.m. ");
    TITLE1 H=2.5 F=SWISSX C=BLUE "Comptroller of Public Accounts";
    TITLE2 H=3.5 F=SWISSX C=BLUE "Jobs Running in Class U";
    TITLE3 H=2.5 F=SWISSX C=RED "&DATE";
  RUN;
%MEND;

%CLASS(CLASS=U,COLOR=GREEN,IN=RUNTIME,TO=16,BY=4,N=R,
  TICK1=0,TICK2=4,TICK3=8,TICK4=12,TICK5=16);

PROC GREPLAY IGOUT=GRAPHCAT.CLASSES GOUT=GRAPHCAT.CLASSES
  TC=GRAPHCAT.TEMPLATE NOFS;
  TEMPLATE TEMPLA1S;
  TREPLAY 1:CLASSUR;
RUN;
```

The program begins with a **GOPTIONS** statement to initialize the environment and a **LIBNAME** statement pointing to the SAS library where the data reside. These are followed by a **PROC DATASETS** to delete an entry in a graphics catalog. I like to keep sets of graphs in separate catalogs. I usually create whole sets at one time. SAS will not write over existing entries in its graphics catalogs, appending numbers to the entry

names. To avoid this, I always delete the graphics *catalog* and recreate the set of graphs that reside in that catalog.

Next is a **MACRO** definition for creating the graph set. This particular program will create only one graph in the set. The **MACRO** primarily consists of a **DATA** step, which has the *potential* to do some additional formatting or subsetting of the data, and a **PROC GPLOT** for producing a scatter plot of the data.

The **PLOT** statement will produce a plot of the number of jobs executing (variable **CLASSU**) versus time (variable **TIME**), with **VM=** and **HM=** specifying the number of minor ticks to be placed on the vertical and horizontal axes, and **HAXIS=** and **VAXIS=** specifying that additional axis definition statements will be used.

The appearance of the plot itself is controlled by the **SYMBOL** statement. The plot symbol is the default, the plus (+) sign. In fact, the plot is a number of tiny plus signs joined to one another by the **I=JOIN** interpolation option.

One noticeable aspect of this plot, shown in figure 6, is that, as jobs begin and end, there are brief periods where the number of jobs increases by one for one second and then drops back down. Because of this choppy appearance and also because of the upward bias of the data, I took steps to correct the appearance and accuracy of the plot by averaging data points.

Figure 7 shows the previous SAS program, now modified by the addition of additional summarizing logic. With the modification, **DATA ALL** has been modified to include two new variables, **hour** and **minute**. These two new variables make it possible to summarize the data at the level of minutes instead of seconds. This summarization takes place in the **PROC SUMMARY** that follows. A final data step takes a subset of the output produced by **PROC SUMMARY** and rounds the values of the analysis variable -- the number of jobs running in the system -- to whole numbers. This effectively eliminates any upward bias caused by one job finishing and another job (waiting for a job to end) starting during the same second.

Figure 8 shows the output from **PROC GPLOT** using the data that have been summarized using the code shown in figure 7. In addition to the elimination of the upward bias on the data, the graph itself has been smoothed out.

Figure 6: Plot of Jobs Running in Class U

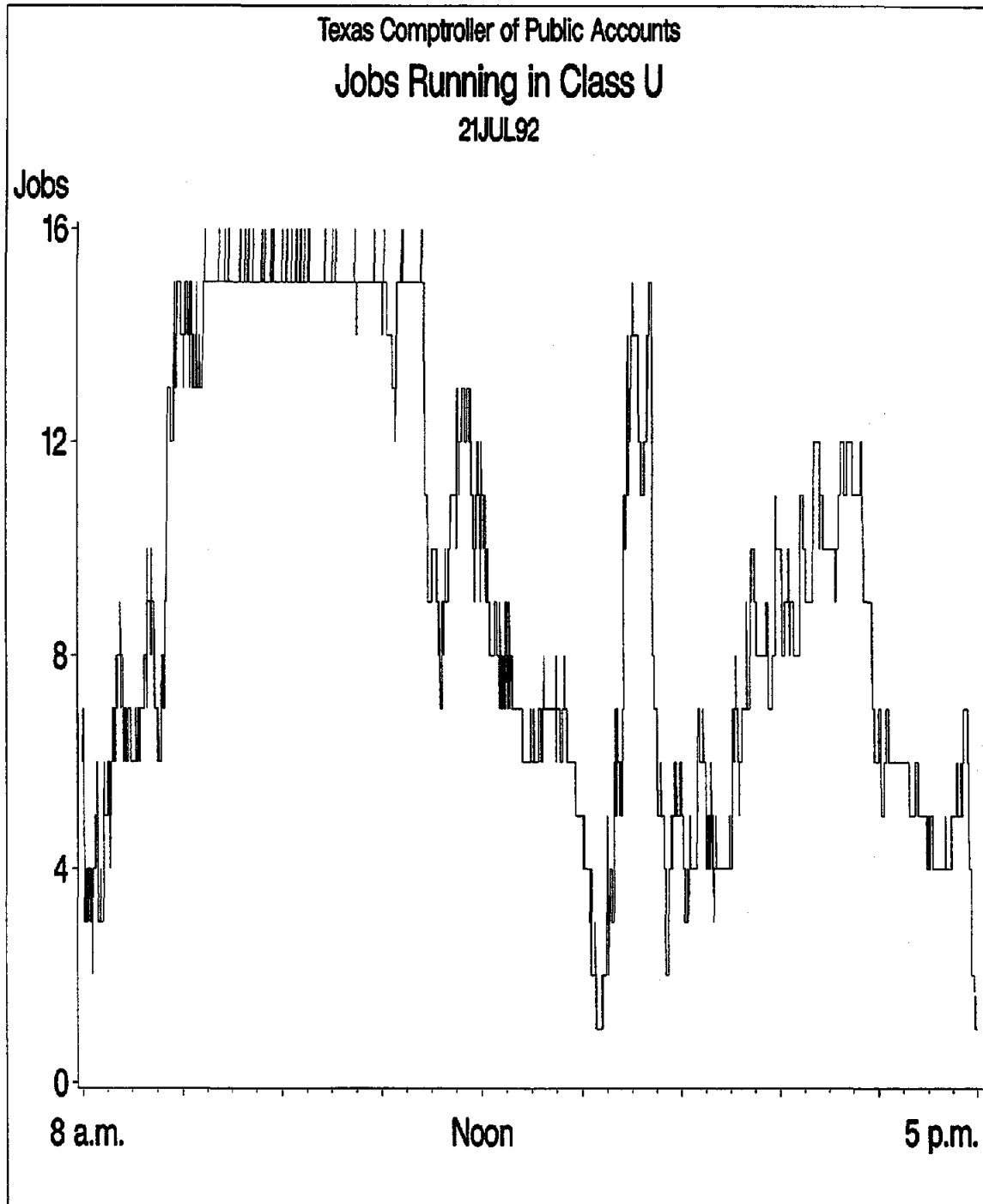


Figure 7: Plotting Batch Initiator Data using PROC GPLOT(Averaged)

```
GOPTIONS VPOS=100 HPOS=100 DEVICE=GDDMPCG NODISPLAY FTEXT=SWISSX;  
LIBNAME CLASSES 'UTIL.S96.SAS.CLASSES' DISP=SHR;
```

```
PROC DATASETS LIB=GRAPHCAT MT=CATALOG;  
  DELETE CLASSES;  
RUN;
```

```
%MACRO CLASS(CLASS=,COLOR=,IN=,TO=,BY=,  
  TICK1=,TICK2=,TICK3=,TICK4=,TICK5=,N=);  
  %GLOBAL DATE;  
  DATA ALL;  
    SET CLASSES.&IN END=EOF;  
    IF EOF THEN CALL SYMPUT("DATE",GRAFDATE);  
    MINUTE=MINUTE(TIMEPART(TIME));  
    HOUR=HOUR(TIMEPART(TIME));  
  RUN;
```

```
PROC SUMMARY;  
  CLASS HOUR MINUTE;  
  VAR CLASS&CLASS;  
  ID TIME;  
  OUTPUT OUT=ALL MEAN=;
```

```
DATA ALL;  
  SET ALL;  
  IF _TYPE_=3;  
  CLASS&CLASS=ROUND(CLASS&CLASS);
```

```
PROC GPLOT DATA=ALL GOUT=GRAPHCAT.CLASSES;  
  ....
```

Figure 9 shows what can be done with PROC GREPLAY. Here a number of graphs, Jobs Running and Jobs Waiting for classes "U" and "V" reveal that jobs are waiting to start in one user job class while a number of initiators are available in another class. If the jobs waiting to start in class "U" are short jobs, requiring no access to tape, then there is no need for them to wait. This graphic strongly suggests that some users are unaware of the opportunity to run jobs in class "V".

Figure 10 represents the number of ROSCOE users compared with the number of TSO users during a normal business day at the Texas Comptroller's Office. The graph shows a morning peak and an afternoon peak in the number of people signed on for both workloads though the number of ROSCOE users exceeds the number of TSO users at all points.

Figure 11 is an intriguing look at the number of CICS transactions executing during each second during a normal business day. The data show a major clustering of transactions between about 10 and 30 transactions each second during most of the day (with the characteristic dip during the noon hour).

But there also appears to be another minor clustering of data points at around 50 transactions. I cannot explain this secondary "mode" in the data. I can only speculate that periodically CICS is given additional resources by the System Resource Manager (SRM) that allows it to process more transactions. Nor can I explain the spikes in the data. The most prominent spike occurs at about 3:45 p.m. Both the spikes and the primary/secondary clustering occurred on several days of data although the spikes appeared at different times. Both phenomena are an indication of something occurring in the system (but what?)

Figure 8: Plot of Batch Initiator Data using PROC GPLOT (Averaged)

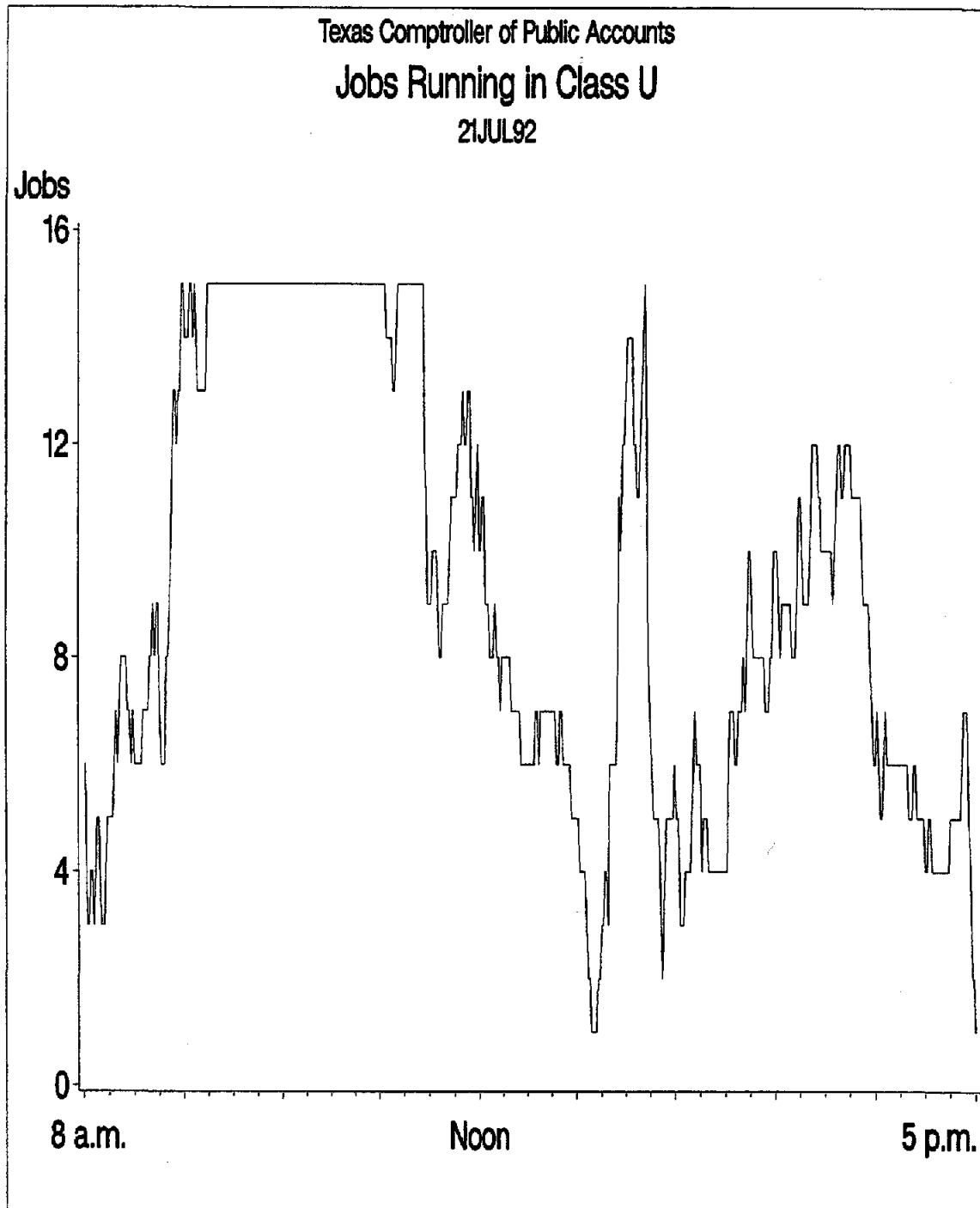


Figure 9: Plot of Jobs Running versus Jobs Waiting using PROC GREPLAY

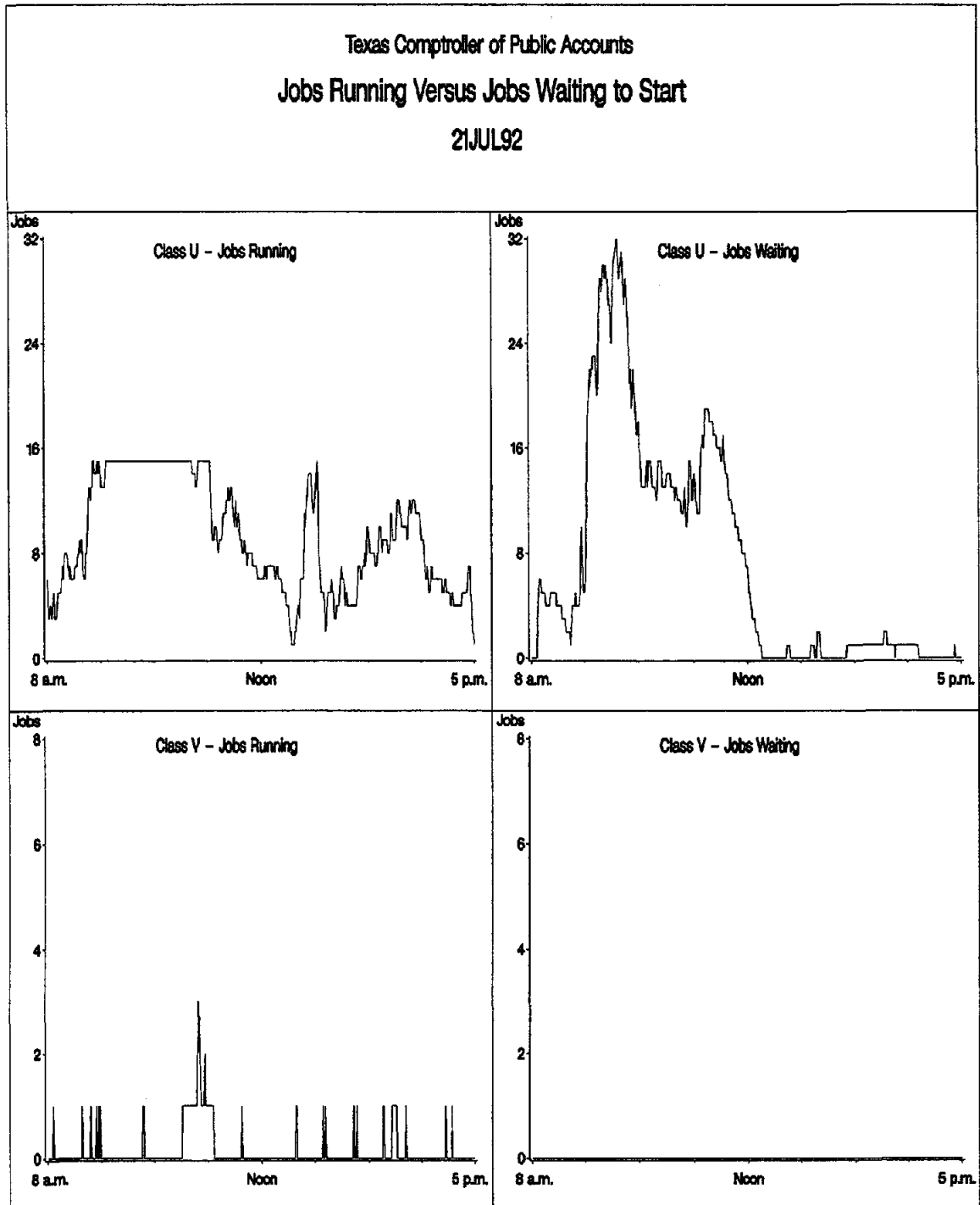


Figure 10: Plot of the Number of TSO and ROSCOE Users

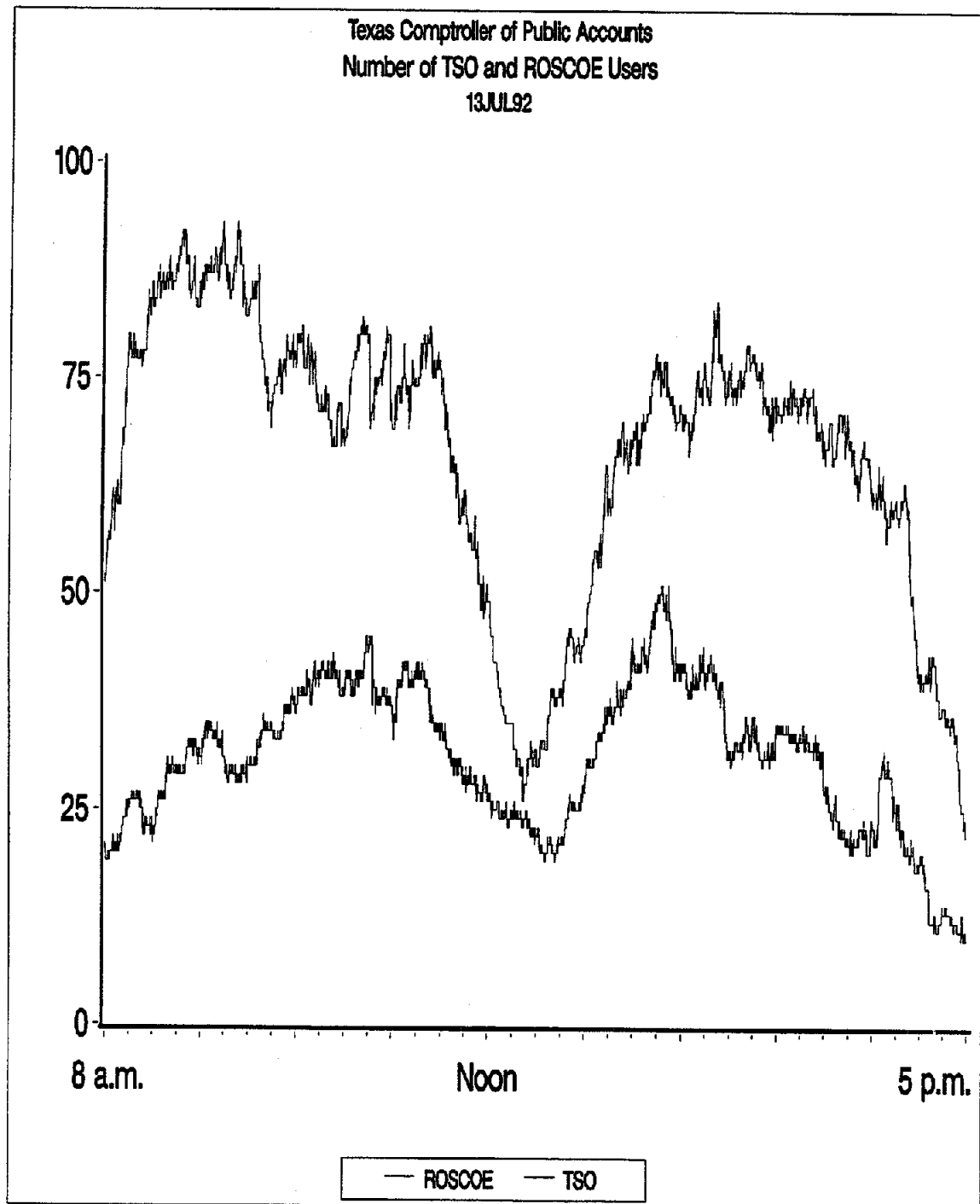


Figure 11: Plot of CICS Transaction Load During Prime Shift

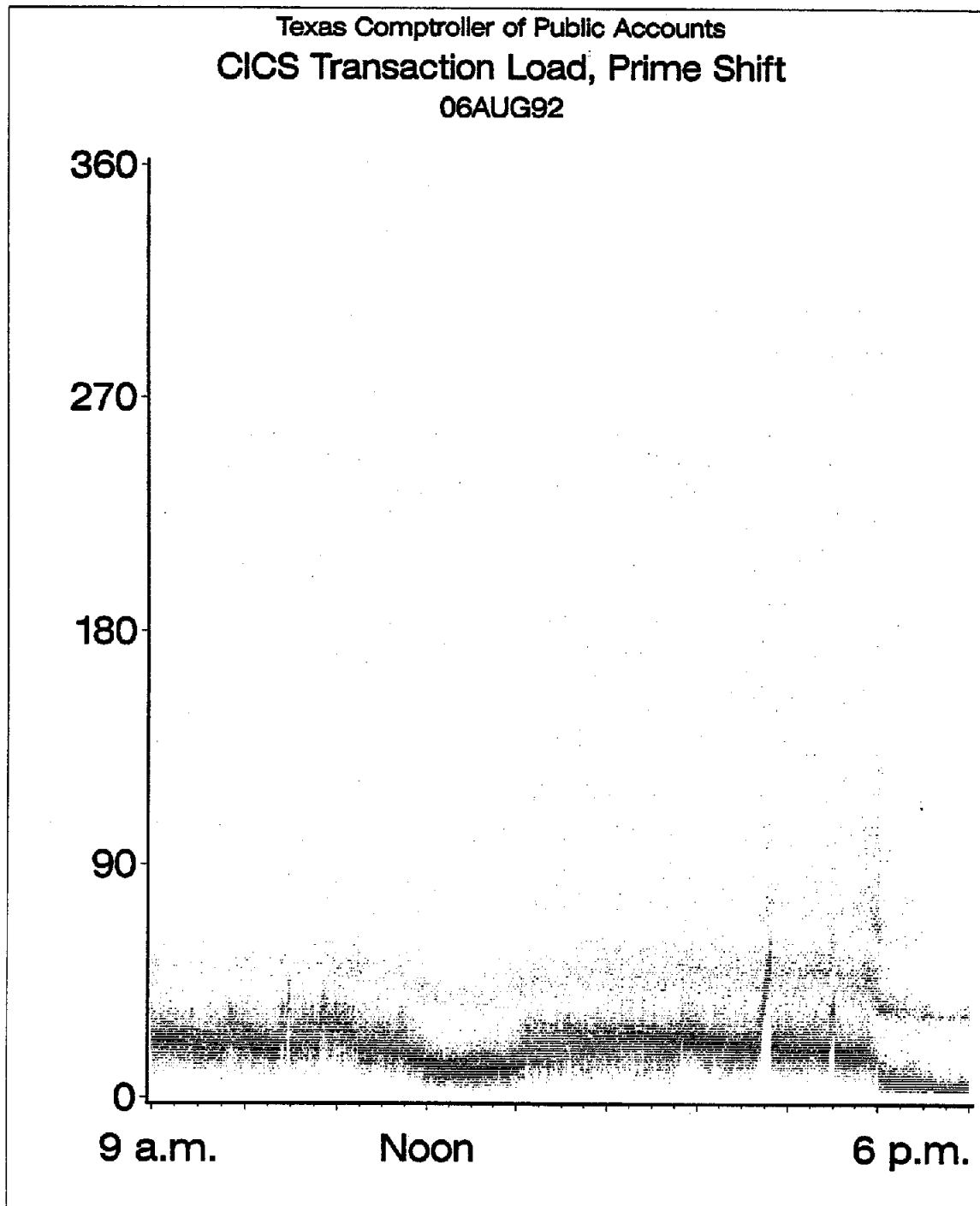
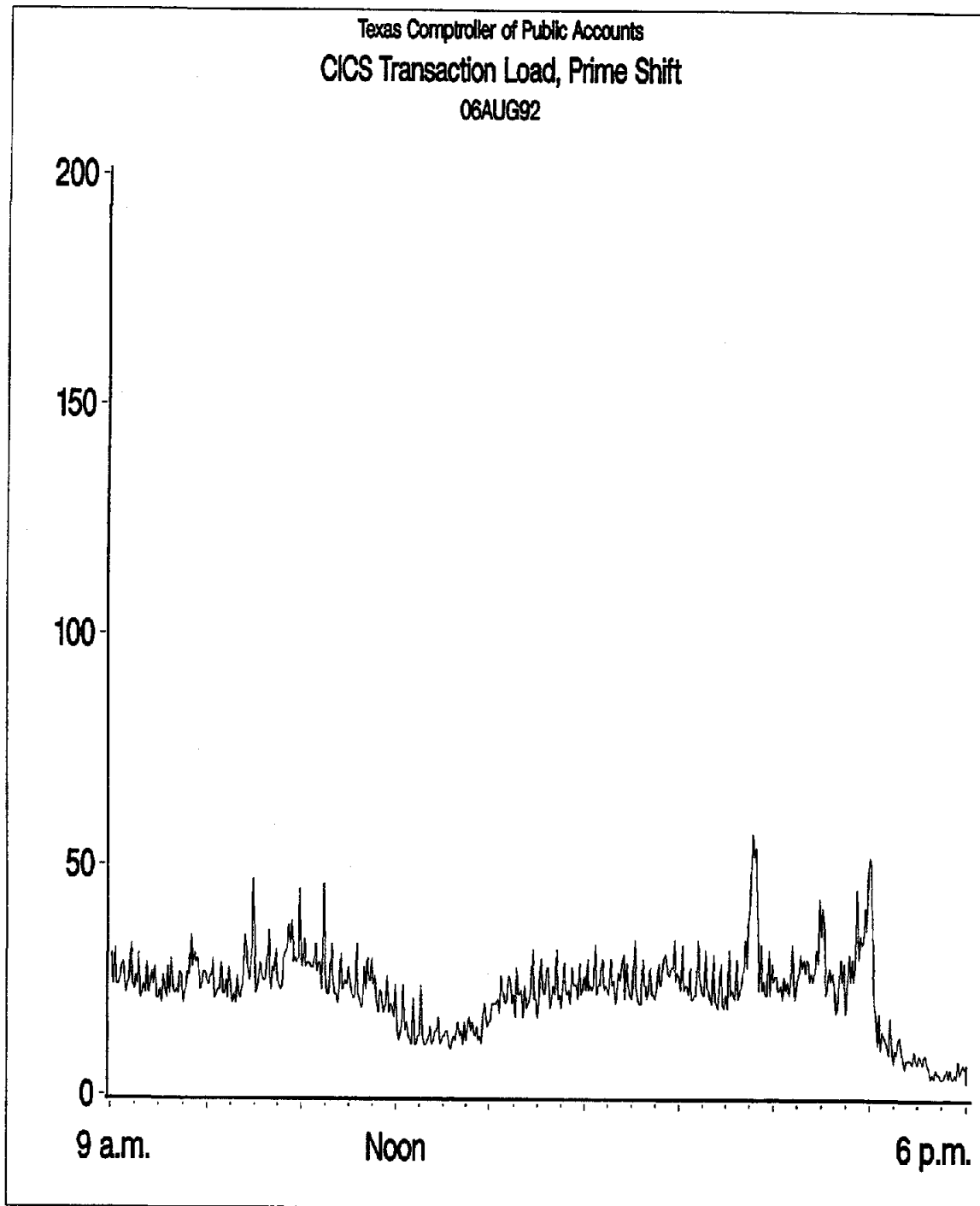


Figure 12: Plot of CICS Transaction Load During Prime Shift (Averaged)



There is a quite noticeable drop in the number of transactions in the system at 5:00 p.m. corresponding to the close of the normal business day. Finally, the data show isolated seconds where the number of transactions jumps to over 300.

I am mystified by the apparent horizontal lines running through the scatter plot. These appeared on the display monitor and also on the printed and plotted graph, using several different device drivers. There really should be a solid mass of points; there is nothing in the data to produce these lines. This is a good question to take up with the SAS/GRAPH support people.

Figure 12 shows the same CICS data averaged by minute. This graph has a more civilized look to it, but it hides much more than it reveals. It loses the primary/secondary clustering of the data and also the appearance of order amid chaos. It retains the spikes. Averaging this CICS transaction data seems to be a transformation that serves only to obscure any useful analysis.

Conclusion

A need to monitor initiator usage led to the development of an analytical approach that can be applied to other kinds of processes. While the resources required for this type of analysis are significant (2 megabytes of storage below the line and 12 megabytes of storage above the line on an MVS/XA or ESA system) there was no problem running the listed programs using the default storage at our shop for SAS release 6.06 or 6.07. For smaller systems or for SAS 5.18, where storage constraints may more severely limit array sizes, it is possible to break the data reduction into smaller time intervals and then merge these time intervals for later graphical analysis.

Where it is important to see a high level of detail, the technique I have described for analyzing time-stamped data can be quite useful. In the case of batch job initiators, the technique revealed patterns of usage that suggest ways to increase job throughput by running jobs in other initiator classes. In the case of CICS transaction data, the approach showed the extreme variability of the transaction processing environment and revealed a number of patterns, some of which were unexpected and not easily derived by other methods. The technique also was useful for comparing TSO and ROSCOE usage. Averaging the data improved the appearance of the batch data but obscured important information about the CICS data.

¹MXG is a product of Merrill Consultants.

²MICS is a product of the Legent Corporation.

³Several vendors offer performance databases made up of SAS datasets and SAS data libraries. Both MICS and MXG are SAS databases. In fact, many of the Computer Performance Evaluation (CPE) products for MVS systems are based on SAS.

⁴Barry Merrill, *Merrill's Expanded Guide to Computer Performance Evaluation Using the Sas System*, pp. 39-47, SAS Institute, Inc., 1984.