

40,153,273,652

Abstract:

40,153,273,652 is a large number to most people (even congressmen.) If it is the number of bytes of SMF data produced by a single complex in a single day, it can be frightening to think about processing. How long will it take and are there ways to reduce the time?

Chuck Hopf chuck@mxg.com
Barry Merrill barry@mxg.com

Introduction:

As data centers grow ever larger and transaction volumes increase, the amount of data generated for performance measurement, capacity planning, and accounting increases at the same or a higher rate. This increase in data volume can cause the construction of the Performance Data Base (PDB) to become a significant problem. Starting at midnight and finishing this process before 6:00AM is no longer possible but is instead running into the peak processing periods of the day. Building the PDB is becoming a performance problem.

On one recent day at a large banking site during the Christmas shopping period, a peak of 37.39GB of SMF data was generated in a single day. Although this staggering volume should have caused the processing of the SMF data to run well into the next day, it was completed by 8:00 on the following morning.

Accomplishing this was a long-term effort at tuning the CPE application (in this case MXG) that can be applied to most other CPE software with varying levels of effort. This effort had a goal of completing MXG processing by 8:00AM and used three approaches:

- Ensure the software was as optimized as possible
- Ensure any possible improvements using system software were applied
- Increase the parallelism of the workload

Sound like an application tuning effort? MXG is an application and like any other application, it can be tuned. Should we not apply the lessons learned tuning the applications of others to our own applications? Or are we the cobbler's children?

Identifying the Problem:

So where is all of this SMF data coming from? The first thought was CICS since it was known that CICS transaction volume was in the area of 60M transaction segments/day but a detailed analysis was done to be sure. The results shown in figure 1 confirm that, in fact, 30.1GB (91.2%) of the data was from CICS transaction segments with another 1.3GB (3.3%) from DB2. No other data sources exceeded 400MB (1%.) The CICS transaction segments so dominate the totals, it is almost impossible to see how large they are so figure 2 excludes CICS transactions to give visibility to the remaining data.

The main online system fills a full 3390-3 (roughly 2.8GB) volume with SMF data in just about 1 hour during the peak processing periods of the day and volumes increase significantly during the peak period of the year. Obviously (as shown in figure 3,) the main online system is SYS2 with a smaller load from SYS1 and the other systems almost

SMF Statistics — 27JUL98

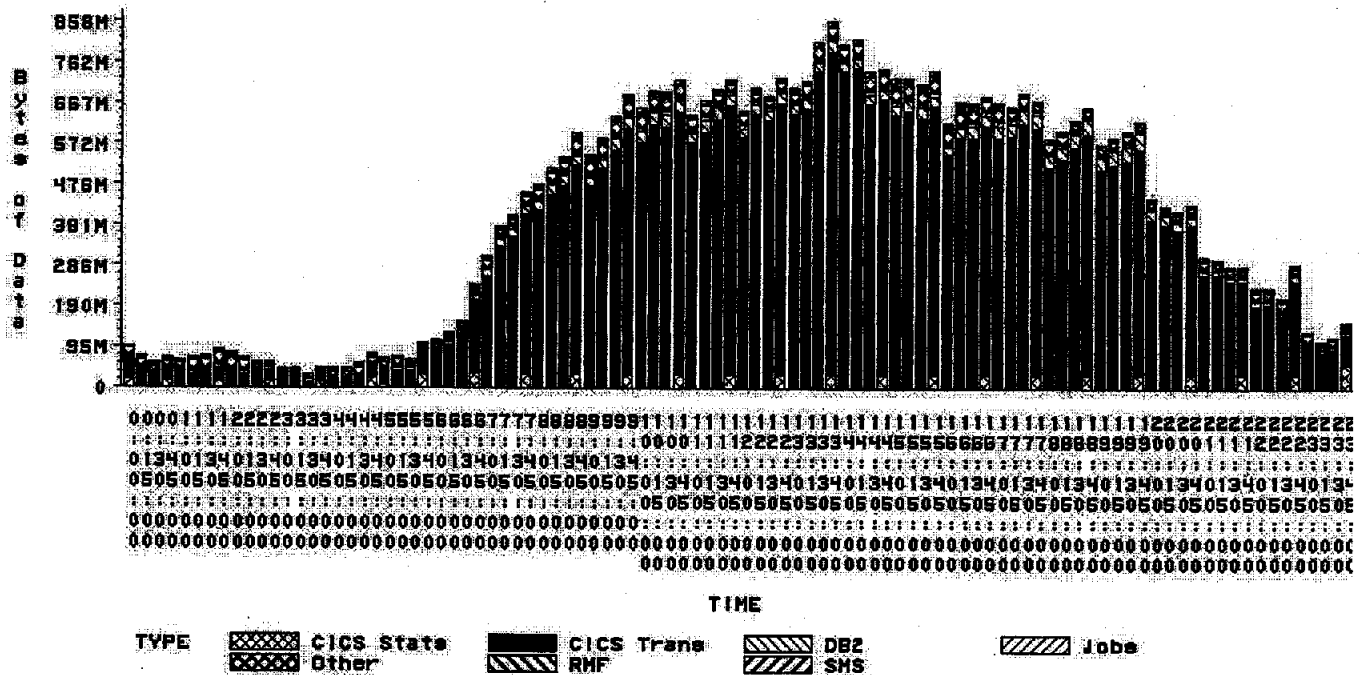


Figure 1 - SMF Bytes by Type of Data

SMF Statistics — 27JUL98

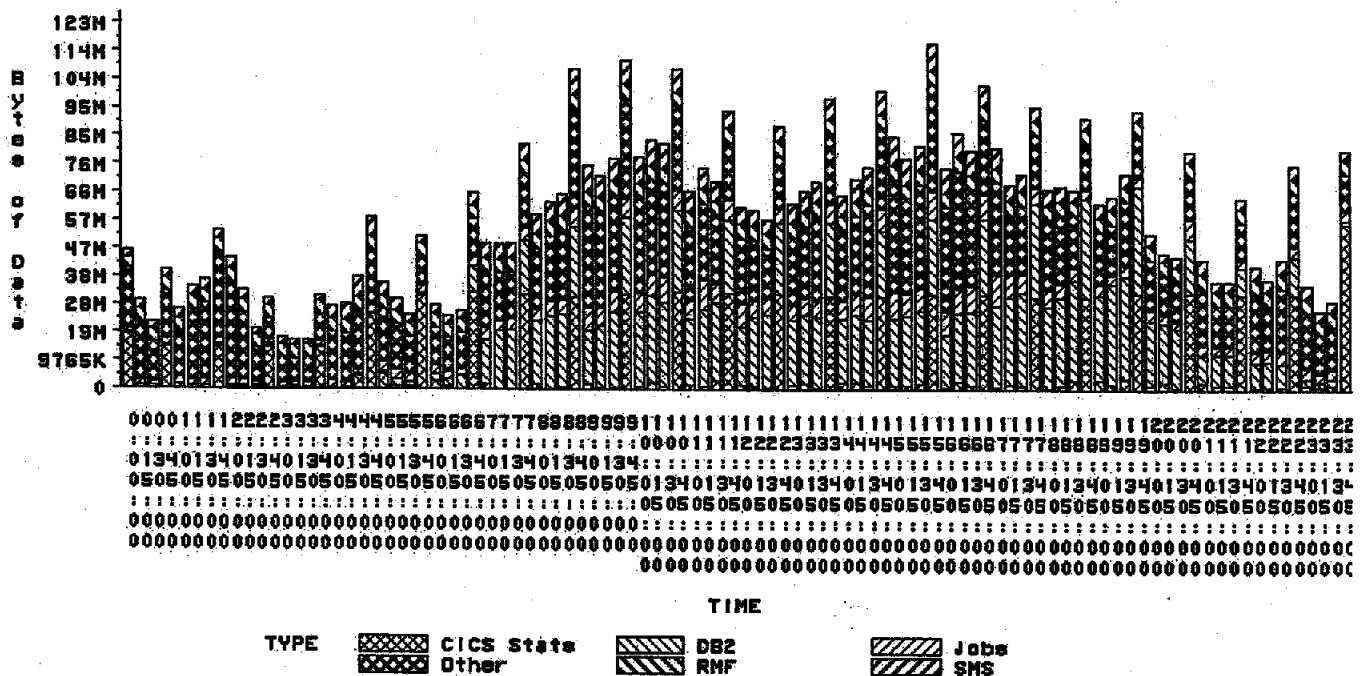


Figure 2 – SMF Bytes by Type of Data Excluding CICS Transactions

SMF Statistics — 27JUL98

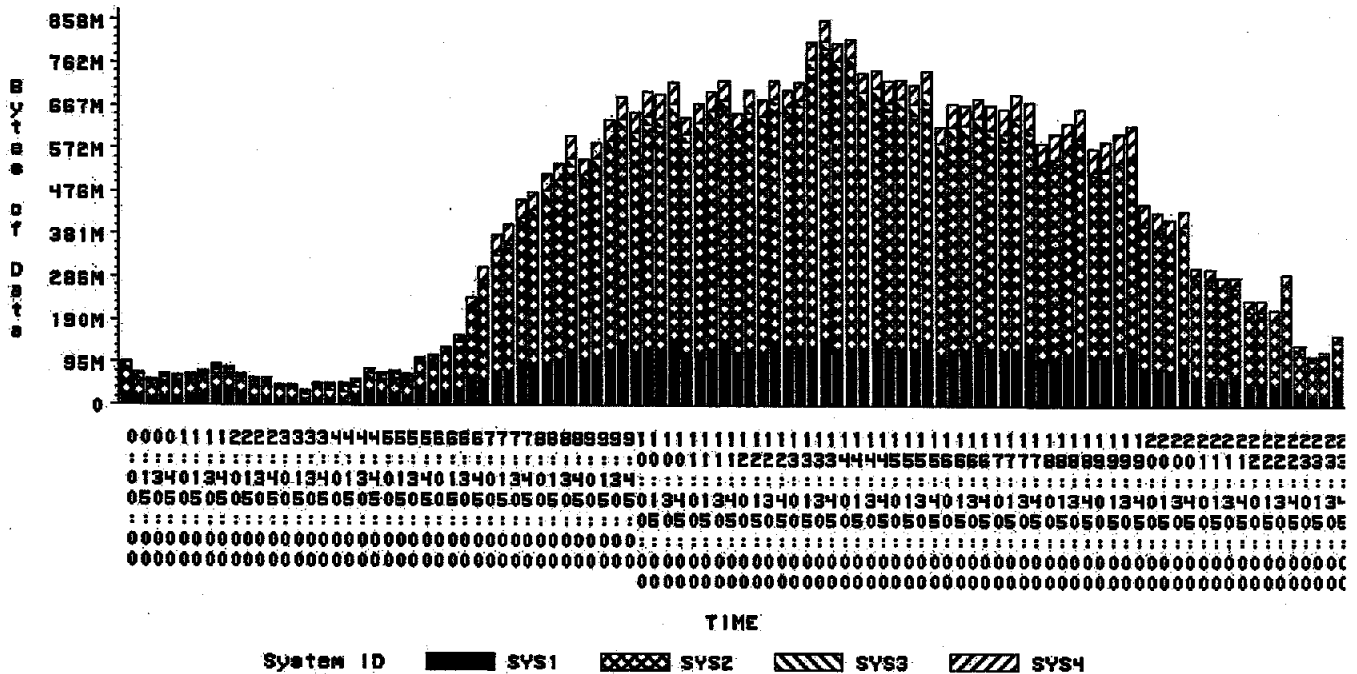


Figure 3 – SMF Bytes by System

disappearing in the overwhelming data volumes coming from these two systems.

To understand the scope of the problem in processing the data, it is first necessary to understand exactly how much of this data must be processed to meet the reporting needs of the installation. In this case the requirements are:

Database	Data sources
Base PDB	Jobs, steps, RMFINTRV, IPLS, TAPES, TYPTTMNT, TYPETALO,
Type 14 15	Non-VSAM IO at close
Type 64	VSAM at close
Type 61 65 66	ICF Catalog Activity
Type 42	SMS IO Statistics
HSM	DFHSM Migration Activity
HSC	STK Silo activity
TMC	Tape Management System
NETSPY	Network Performance

Hiper-Cache	Automated Buffering Tool
Service Center	Problem/Change Management
Type 39	
NDM	Network Data Mover
Type 41	
Type 91	
WSF	Letter Writing Software
SPMS	

In other words, not only was very little SMF data not to be processed, but a number of non-SMF data sources were being added to the mix further aggravating the problem.

Assuming all 37.39GB of data must be processed in a serial process at 10MB/second, we can quickly come to the conclusion that it will take a minimum of 1 hour and some minutes just to read the data. If we want to actually do something with the data, the time stretches considerably. But how much time is it really? To answer that question, we took the data for slightly more than 1 hour (a single MANx¹ dataset on the busiest system containing 1,814MB of SMF data in 254,130 logical records) and processed the data through a standard MXG BUILD PDB process.

This first run of this data consumed 19.4 CPU minutes (on an Amdahl 8-way Millenium CMOS processor) and 624,601 EXCPs in 60.3 minutes. So, processing just over one hour's data on this particular system took just over one hour. At this rate, a daily run would take more than a day and we would never have any hope of catching up. This is BEFORE we add in all of the extra data sources required by this installation. Clearly, we have to find a better solution.

Looking at the detail in the SAS log for this run, we find that about 75% of the total elapsed time is in the first data step that processes the raw SMF data. This is where we decided to concentrate our initial tuning efforts. For this run, the first data step consumed 17:31.44 seconds of CPU time, 545,190 EXCPs, and 45:42.77 of elapsed time. The DASD space required with the CICSTRAN dataset going to tape was:

DDNAME	Track s	Cyls	Mbytes
WORK	1,322	88	70.4
PDB	1,829	122	97.8
SPIN	78	6	4.8

The WORK file is reduced in size since some of the DB2 datasets go directly to the PDB DD. Extrapolating to a daily run, the DASD requirements for a 24 hour period would be:

DDNAME	Track s	Cyls	Mbytes
WORK	31,728	2,116	1,693
PDB	43,896	2,926	2,342

¹ Isn't it time in the interests of equal rights that this becomes a SYS1.PERSONx dataset?

SPIN	78	6	4.8
------	----	---	-----

To run MXG's base BUILD PDB without modification would require (essentially) 1 3390-3² for WORK, 1 3390-3/day for the daily PDB, multiple (6 were used) 500 cylinder SORTWK DDs, plus many tapes for the CICSTRAN dataset (in this run with 2.9 million CICS transactions, the CICSTRAN dataset was three 3490 volumes.) If a weekly PDB is to be constructed and maintained online, then we will have to utilize multi-volume SAS datasets during the WEEKBLD process since the aggregate space required for some of the datasets (notably the STEPS dataset and the TYPE74 dataset) exceed the capacity of a single 3390-3 volume.

Optimizing the Data Step:

Examination of the first problem data step shoes the following structure:

```
_VAR0
_VAR6
_Var...
_SMF
_CDE0
_CDE6
```

...

Knowing the each of the _CDE macros contains the basic structure of:

```
IF ID=x THEN DO;
    .processing for the SMF type
END;
```

it becomes obvious for each and every record read, the IF statement for every other record type in the step is executed. This might not be a problem for a small volume of data but when there is such a significant skew to a single record type (110) and the volume of data is as high as in this example, perhaps the data step needs to be optimized to be:

```
_VAR0
_VAR...
_SMF
_CDE0
ELSE IF ID=6 THEN DO;
    _CDE6
END;
ELSE IF ID=26 THEN DO;
    _CDE26
END;
ELSE ...
```

in order to gain the efficiency of not executing the IF statement unless required. This was first implemented in MXG version 15.08 (change 15.329) and a test run with this in place yielded a CPU time of 17 minutes, EXCPs of 545,070, and elapsed time of 39.5 minutes. This

² For reference, a 3390-3 is assumed to be 3300 cylinders each containing .8GB of data in half-track blocks.

represented an improvement of 2.5% in the CPU time, .1% in the EXCP count, and 13.6% in elapsed time³. This does not represent enough improvement but, looking more closely, this change had problems of its own. For each record type to be processed, the IF ID=x statement is executed twice. Once inside the BUILD606 code (the basis for the 1st data step in BUILDPDB) and once inside the _CDE code for that record type. The code was modified to the form:

```
_VAR0  
_VAR6  
_VAR...  
_SMF  
_CDE110  
ELSE  
_CDEDB2  
ELSE  
_CDE...
```

so this problem was eliminated. Now the IF ID= statement is executed only once for each record type. Note the sequence of the _CDE macros was also altered to match the expected frequency of execution. This should minimize the path length through the first data step.

Results of this test were CPU time of 17.1 minutes, EXCPs of 545,064, and elapsed time of 38.3 minutes. This was still not enough improvement in run time to be able to process a day's data in a day so the next step was to look at ways to improve the throughput using the operating system.

Table 1 shows the results from these three tests for the first data step:

Although the gains were only modest, these changes were the right thing to do. If the situation is encountered where there is a large skew to a single record type, it will not cause huge unnecessary increases in CPU or elapsed time. Besides how can we talk to our customers about optimizing code if our own is not optimized?

Exploiting MVS:

Having had less than stellar success at improving the run time by optimizing the software, we moved on to attempts to exploit features of the operating system to make the improvements we sought.

By default, MXG uses two buffers on files. While this minimizes the amount of memory consumed by the BUILDPDB process the current MEMSIZE of 64M was not considered excessive and the statistics from SAS showed a peak of about 34M used during the BUILDPDB process. What would happen if the number of buffers were increased? Since, in most cases, QSAM will not use more than 5-6 buffers we set BUFNO to 6 on the SMF DD and BUFNO=6 in the SAS options and ran another test.

³ These tests were run on a loaded system where elapsed times may not be good indicators of performance. In this instance, since the time gained in the first data step carried through the entire job, it may be an indication of an improvement.

The results of this test were still disappointing with CPU time of 17.2 minutes, EXCP counts of 544,753, and elapsed time of 40.5 (tape mounts were slower for some reason.) The memory required for the step increased to 42,847K. We are consistently driving the EXCP count down but not by enough to make a difference.

The concept of striped sequential datasets has been around for some time and was the next avenue we pursued. First, the SMF data was copied to a disk file with a single stripe. Then the test was repeated using default buffers for striped datasets.

The default buffering for a single-stripe dataset is not a great deal different from the default buffering for QSAM⁴ and the results reflect this. The CPU time was 17.8 minutes, the EXCP count dropped to 544,684, and the Elapsed time was 39.3 minutes. Memory utilization was now at 42,843K.

One of the advantages of QSAM-E over QSAM is the ability to use many more buffers than standard QSAM. What would happen if we increased the buffers to 15 (one cylinder or about .8MB of data?)

At last, significant improvement! This test showed marked improvement with CPU time of 17.1 minutes, EXCP counts of 544,681, and elapsed time of 27.4 minutes. The memory utilization increased to 42,883K. Still, if we multiply by 24, the daily run would take almost 12 hours and is still unacceptable.

If one stripe with lots of buffers is good, perhaps more stripes with lots of buffers is even better. For the next test, we copied the SMF data to a dataset with 4 stripes and used 15 buffers for the SMF DD. (Or at least the BUFNO parameter in the JCL said BUFNO=15.) What really happens with striped datasets is not as obvious nor is it noted in any of the messages on the log. For our 4-stripe test, the 15 buffers we specified in the JCL would have been rounded up to 16 so that there were an even number of buffers/stripe. Since the number of buffers was actually about the same as in the prior test, so were the results with CPU time of 17.3 minutes, EXCP count of 544,688, and elapsed time of 29.5 minutes. Memory utilization stayed at 42,883K.

Next, we tried BUFNO=60 on the SMF DD. This should cause QSAM to read a full cylinder for each stripe. Results here are encouraging and it may be that further increases in buffers would yield more improvement but we stopped at 60 buffers. The results were CPU time of 17.2 minutes, EXCP count of 544,691, elapsed time of 28.3 minutes, and memory usage of 43,064K.

But if striped data is good for the SMF dataset, what about the SAS datasets? The problem is that striping will not work for EXCP level code such as the SAS access method. But what about the CICSTRAN dataset that is being sent to tape? Can we 'lie' to SAS and

⁴ Non-extended sequential datasets default to BUFNO=5 while striped (extended sequential) datasets default to 2*the number of blocks/track*the number of stripes. In this example, the default buffers for a single striped dataset are 4.

Table 1 - 1st Data Step Optimization Results

Statistic	MXG 15.06	MXG 15.08	MXG 15.08+
CPU Time	00:17:31. 44	00:17:05.1 8	00:17:07.43
EXCP	545,190	545,070	545,064
Virtual Memory	32,983K	33,135K	35,203K
Elapsed Time	00:45:42. 77	00:39:32.3 2	00:38:23.04

send it to disk as a striped dataset? The answer is yes. So long as the DD is changed to something like TAPECICS, SAS will use the sequential engine of SAS and create the data as if it were on tape. This is the same technique used in MXG for the MONTHBLD process to avoid tape mounts but here we are using it to speed up IO operations.

The first test of this technique was run using a BUFNO of 60 on both the SMF and TAPECICS (CICSTRAN) DDs. This was the best test (at least in terms of CPU time) run to date. The CPU time dropped to 16.8 minutes, the EXCP count was 544,988, the elapsed time 28.3 minutes, and the memory utilization was 43,015K.

But, does the BUFNO= specification in the SAS initialization control or does the BUFNO= specification on the TAPECICS DD control? Another test was run where the BUFNO= specification for SAS was increased to 16. This yielded the best results seen with CPU time of 16.6 minutes, EXCPs of 545,770, elapsed time of 27.2 minutes, and memory usage of 39,863K.

Table 2 - MVS Exploitation Test Results

Test	CPU Time	Elapsed Time	EXCP	Virtual Memory
MXG 15.08+	17:07.43	37:07.04	545,064	36,531K
SMF BUFNO=6	17:13.70	37:59.94	544,753	42,847K
SMF 1 Stripe	17:49.15	38:14.13	544,684	42,843K
SMF 1 Stripe BUFNO=15	17:04.33	26:10.79	544,681	42,883K
SMF 4 Stripes BUFNO=15	17:15.90	27:14.32	544,688	42,883K
SMF 4 Stripes BUFNO=60	17:11.01	27:30.03	544,691	43,063K
SMF 4 Stripes CICS 4 Stripes	16:49.95	28:19.37	544,988	43,015K

BUFNO=60				
SMF 4 Stripes	16:36.93	27:12.78	545,770	39,863K
CICS 4 Stripes				
BUFNO=60				
SAS BUFNO=16				

This seemed to be as far as we could go with MXG as it is delivered out of the box and we still have not been able to project meeting the goal of starting MXG processing at 4:30AM and finishing by 8:00AM. The results of our exploitation of MVS features are contained in Table 2. For comparison purposes, the tape mount delays (for those jobs using tapes) have been subtracted from the elapsed times.

Getting Outside the Box:

It now seemed likely that without serious work and modification of the basic structure of BUILDpdb, we were not going to meet the performance objective of completing the BUILDpdb process by 8:00AM given a 4:30AM starting time. The real problem was the sheer volume of data to be moved during the process. It is simply not physically possible (with today's technology) to move this volume of data and process it in so short a time.

It was clear we needed a system that processed different kinds of data in parallel so a multi-headed BUILDpdb process could process all of the data in a timely enough fashion to meet the stated goals. But how should it be divided?

Let's start from the ending and work backwards. When all is done, the minimum databases that must exist are:

- Basic MXG PDB
- Database containing the DSETOPEN dataset (ANALDSET)
- CICS Transactions reduced to Unit of Work
- CICS Statistics
- DB2 Accounting data
- Database containing the DATASETS dataset (DAILYDSN)
- PDB for all TEST systems

This seemed like a good starting point except that the number of type 42 records and type 74 records caused problems in processing this data. The time to sort and summarize these rather large data sources could cause delays. Thus, we decided to add to this breakdown, an IO PDB to supplement the base MXG PDB with IO subsystem data. Further, the DATASETS database can be subdivided in the early stage by having one job that reads the TMC and another that builds and reads DCOLLECT data followed by a third job to put the two back together again.

The jobs and input data sources are contained in Table 3.

Table 3 - MXG Jobs and Data Sources

Job Number	Job Description	Data Sources	Successor Jobs
1	Base PDB	0, 6, 21, 26, 30, 70, 71, 76, 77, 89, TMNT, 39, 41, 91, SPMS, TCP, HIPR, WSF, NDM, NSPY, STC, SVCC	
2	DSETOPEN	14, 15, 30_1, 30_4, 61, 65, 66, 64	
3	CICS Statistics	110 (statistics only)	
4	IO PDB	42, 73, 74, 75, 78, HSM	
5	CICS Transactions	110 (transaction segments only)	
6	TEST Systems	Normal MXG PDB Processing	
7	DB2	100, 101, 102, 30_4	
8	Read TMC	TMC Catalog	10
9	DCOLLECT	Run DCOLLECT and process data	10
10	DATASETS	Merge output of jobs 8 and 9	

Notice that some data is input in more than a single job. It was thought that it would be better to process the same data twice than to have to wait for one job to finish before another could begin. As an added complication, the volume of CICS data is so large it is not practical to process the data for a full day in a single run. For this reason, the CICS transaction data is collected and processed three times per day (12:00, 18:00, and 1:00.)

Sounds simple but what did it take to get from MXG as delivered to MXG unbound?

First, some changes are necessary in the way SMF dumps are processed. Rather than putting all of the data in a single dataset, it should be spread out according to the way it is going to be used. Our choice was to break the data down as follows:

- CICS Transaction Data
- CICS Statistics Data
- DB2 Data
- Everything else

'Everything else' seems like a lot but it is really fairly small compared to the remainder of the data and in those cases where the data is going to be read by more than one job, it is just more convenient. Making this breakdown and managing the SMF dump processing is a paper in itself and we will not spend more time here going into the details. Each of the 4 production systems has three SYS1.MANx datasets and at least two on each system are full 3390-3 volumes (2.8GB) and there are 20-30 dumps on any given day though many are not dumps of full datasets. Managing SMF data is very critical in this installation.

Looking at future volumes and knowing at some point we will want to put the DB2 and CICS data together by Unit of Work, we will ultimately have to put DB2 and CICS on the same three times per day schedule. For now, CICS is the only piece that runs multiple times per day.

The first step in implementing this structure is to build a new SOURCLIB for each of the components. This is necessary because some members (IMACCICS, IMACFILE) will be

needed in multiple places with different values to optimize processing. This makes the structure of the MXG SOURCLIB concatenation at this site look like :

- COMPONENT - SOURCLIB for specific component
- USERID - general USERID SOURCLIB
- CHANGES - CHANGES to MXG since last PRODUCTION install
- PRODUCTION - Latest production release

None of these libraries contains more than a few members but each serves the purpose of suppressing some part of the standard MXG process that would result in unneeded processing in this optimized job stream or is a piece of code unique to this process. Appendix A contains a list of the members in each SOURCLIB and their purposes.

Table 4 gives the timings for test jobs run against the same one hours data used to optimize the first data step.

Table 4 - MXG Unbound Test Run Timings

Job	CPU Time	Elapsed Time	EXCP	Memory
Base PDB	00:02:29.6 4	00:05:04.80	22,413	53,931K
DSETOPEN	00:00:26.4 6	00:01:00.00	12,488	7.250K
CICS Statistics	00:01:42.6 3	00:05:45.60	9,910	16,962K
IO PDB	00:01:22.9 7	00:03:04.80	15,091	16,856K
CICSTRAN ⁵	00:21:55.2	01:06:41.40	451,50	6,264K
Step 1	0	00:21:41.40	6	6,264K
Step 2	00:14:15.6 0	00:45:00.00	434,94 2	4,998K
	00:07:39.6 0		16,564	
Test Systems ⁶	00:04:41.4 9	00:07:34.80	19,290	55,430K
DB2	00:11:48.6 0	00:28:51.00	50,765	17,929K
Read TMC	00:08:56.4 0	00:17:25.20	610,00 6	5,649K
Read DCOLLECT	00:12:15.6	00:38:00.00	73,455	7,647K
Step 1	0	00:21:55.80	49,148	N/A
Step 2	00:04:15.6 0	00:16:04.20	24,307	7,647K

⁵ CICSTRAN jobs run three times/day

⁶ Production run timings used for TEST PDB, TMC, DCOLLECT, and DATASETS jobs

	00:08:00.0 0			
DATASETS	00:17:54.6 0	00:44:33.60	28,552	5,884K

Notice that two of these jobs are multi-step jobs. The job reading DCOLLECT data runs DCOLLECT in the first step and processes the data in the second step. The CICSTRAN job reads the SMF data and creates the CICSTRAN dataset in the first step and runs ASUMUOW in the second. In this case, the two step process was done for recoverability. If the first step completes and the job fails in the second step, the job can be restarted in step 2 so the GDG for the SMF data is not enqueued and the data need not be read a second time.

This job consumes huge amounts of temporary DASD space for SORTWK and WORK. The production version of this job uses 2,000 cylinders (about 1.6GB) of WORK space, 15 500 cylinder SORTWK datasets, and an intermediate output dataset for sorted CICSTRAN data on tape. Even with reduced numbers of variables in the KEEP= list, the CICSTRAN dataset fills 6 3490 volumes 3 times per day. The detail CICS transactions are not retained but only the output of ASUMUOW.

Examining the volumes of data across the day, we can see our one hour of data probably represents about 7% of the total volume of data for the day. From this, we could extrapolate the run times for actual jobs might be about 15 times the run time of the test streams. This means the BASE job should run in about 75 minutes. For the BASE job, this would give a projected time of 1 hour and 16 minutes. In fact, the production version ran in 1 hour and 15 minutes. The entire production implementation begins processing of SMF data at approximately 4:30 each morning and is completed between 7:30 and 8:00 (excluding the CICSTRAN processing.) The longest portion of this process is the DB2 job which will ultimately be run multiple times per day. Excluding this job, all other runs are completed by 7:00 for a total run time of 2.5 hours. Why 4:30? Tape drives for SMF processing become available about 3:00AM and the daily SMF processing completes about 4:30. While tradition dictates a cutoff for SMF processing of midnight, the practicalities of the nightly batch cycle dictate otherwise.

CICSTRAN processing is a seemingly never-ending process. Two of the three daily CICSTRAN jobs run for more than 4 hours and the third for 2.5 hours on most days and longer on heavy days. On the peak day of 37.39GB, the three jobs ran for 3:56, 7:38, and 4:10 for a total of 15:44. More than half of the day there was a CICSTRAN job running! On the same day, the BASE component executed in 1:12.

Rebuilding MXG

The thought of having multiple SOURCLIBs to support many environments caused Barry's stomach to hurt and forced an evaluation of the basic structure of MXG. As a result of Barry's upset tummy, starting with version 16.03 of MXG, it is no longer necessary to have separate SOURCLIBs to run MXG in parallel. In fact, there can be almost no members at all in your USERID.SOURCLIB except those you have written yourself for specialized reporting needs. EVERYTHING in MXG has been externalized (or will be soon) so that you can place all

overrides instream in the SYSIN rather than having to make modifications to MXG supplied members

This major architectural change was accomplished through a combination of substitution macros, macro variable substitution, and macros. For those not familiar with SAS constructs, a substitution macro (or old-style macro) is simple substitution. The string defined by the macro is substituted in place of the macro name in the SAS program. The syntax is very straightforward:

MACRO macroname string of SAS program statements %

The % terminates the MACRO definition. The first 8 characters after the word MACRO become the name of the macro and everything in between the name and the % become the value that will be substituted for the macro name inside of a SAS program.

A macro variable is a more powerful method of substitution in that macro variables can be operated on logically within the macro language. The common method of defining a macro variable is to use the macro command %LET where the syntax is:

%LET varname=string of SAS program statements;

In this case the comma terminates the string and the macro variable definition. To cause the substitution of the value of the macro variable inside of a SAS program, the macro name is preceded by an ampersand (as in ¯oname.)

Timing is a critical consideration. In SAS, the last value placed in either form of substitution before the program containing the substitution 'call' is the value that will appear in the program. Thus, if a macro variable PDB is defined as being equal to the string PDB and a substitution macro is formatted as

```
MACRO _LTY74 &PDB.TYPE74%
```

and the macro _LTY74 is in a member VMAC74 of the MXG SOURCLIB then the substitution of the following would be PDB.TYPE74.

```
%LET PDB=PDB;  
%INCLUDE SOURCLIB(VMAC74);
```

Even though the initial value of PDB may have been something completely different. The last assigned value prior to the compilation of the data step is the value that is substituted. This is true for both substitution macros and for macro variables.

Using this knowledge, it was possible to alter the basic structure of MXG such that virtually program in MXG can be overridden via an instream substitution of a macro variable or substitution macro.

For each dataset created by MXG, there are now new entities that completely externalize the interface so that users can modify MXG without the need to modify source code. These entities are:

Product suffix – the xxxx suffix of the TYPExxxx, VMACxxxx, IMACxxxx, ASUMxxxx, GRAFxxxx, ANALxxxx, TRNDxxxx members of the MXG SOURCLIB. These members define a product as supported by MXG.

Product	Xxxx	Datasets
SMF Type 0	0	TYPE0
SMF Type 30	30	TYPE30_1 TYPE30_2...
RMF Type 74	74	TYPE74 TYPE74CA...
SMF Type 110	110	CICSTRAN CICS CICINTRV CICSxxxx etc
NTSMF	NTSM	PROCESS NTINTRV PROCESOR etc

DATASET – SAS “table name” or SAS DATASET NAME. 8-characters as defined by the L macro corresponding to the dataset in the VMACxxxx member. Examples are TYPE0, TYPE74, CICSTRAN, PROCESS, NTINTRV.

DATASET MACROS – The substitution macros that are defined in the VMACxxxx member for each dataset and that control the destination and contents of the datasets created.

_L – the ultimate destination of the dataset generally created by combining the Pdddddd macro variable with the dataset name described by the _W substitution macro to create a dataset name of the form PDB.DATASET.

_W – the temporary home of the dataset while it is being constructed (usually by the TYPExxxx member. Usually of the form 'dataset'.

_K – the KEEP/DROP list for tailoring the variables in the dataset

_E – the dataset exit MACRO. Generally contains the output statement for the dataset.

_B – the sort list for the dataset. Contains the list of variable by which the dataset is sorted

_S – the sort of the dataset and (if it is moved from WORK to some other destination) the deletion of the dataset from the WORK area.

DATASET SUFFIX – each dataset now has a unique six character suffix (dddddd) that is used in the dataset macros to identify the dataset.

Pdddddd MACRO VARIABLES – each dataset now has a macro variable that designates the LIBNAME to which the final dataset should be sorted.

Using these new features, it is now possible to override anything (or virtually anything) completely within the IMACKEEP member or instream using the MXG GLOBAL MACRO variables MACKEEP and MACFILE.

For example, suppose you want to read the type74 data and you want to send the sorted TYPE74CA dataset to LIBNAME MYDATA. The program to do this would be:

```
%LET PTY74CA=MYDATA; /* SET THE DESTINATION TO MYDATE */
%INCLUDE SOURCLIB(TYPE74); /* READ THE DATA */
_STY74CA ; /* SORT THE DATA */
RUN;
```

Much more sophisticated logic than this can be built and in fact the parallel version of MXG has been successfully built using these new features and will be in an upcoming release of MXG.

Where Do We Go from Here?

It has long been impractical (at this site) to keep a weekly PDB on disk. The size of the dataset has long since exceeded the capacity of the largest available drives and the space that would be required is unacceptably high. The weekly PDB has therefore been built on tape for some time. No effort has been made to create a monthly database at all because of the sheer size. The STEPS dataset alone for a month exceeds 6,000 cylinders.

Weekly data is almost certainly a necessity, the TREND databases are a necessity, and monthly databases may be in certain instances though they are discouraged whenever possible. Given the size of the task, how can this best be accomplished in the most efficient manner possible?

The traditional MXG architecture assumes the existence of a PDB for each day of the week and at the end of the week the daily PDBs are merged into a WEEKLY PDB. At the end of the month, the last 5 weekly datasets (and as many daily datasets as may be necessary based on the day of the week or how many daily jobs have been run since the last weekly job) are merged together usually by the subterfuge of making SAS believe that a disk dataset is really tape and then concatenating the disk dataset onto a tape to avoid repetitive tape mounting. No attempt is made to reduce the number of variables contained in the datasets.

It has been said that the three most likely data needs are yesterday, last week, and last month. This reduces the amount of data that must be kept readily available to a minimum. There may also be some differing levels of detail in the variables needed to look at this data. If we are looking at yesterday's data, we are probably trying to solve a detailed performance problem and all of the variables may be germane. If we are looking at last week's data, there may only be a need for a subset of variables. If we are looking at last months data, we are probably doing accounting or capacity planning and an even more reduced set of variables may be appropriate.

What we propose to do is to build a new set of jobs that will:

- Reduce the number of variables at each level

- Build a Week-to-date PDB
- Build a Month-to-date PDB

Using the facilities available through HSM, each daily PDB is a GDG where the most current generation is kept on disk and other generations are migrated directly to Level 2. In this way, 255 days of detail data are available at any point in time but yesterday's data as well as the week-to-date and month-to-date are available online at any time.

On the first day of each week (which becomes a user definable value rather than the current forced Monday) the week-to-date is rolled to a weekly GDG on tape using a PROC COPY. Similarly, the first day of each month, the previous month-to-date is copied to a monthly GDG on tape. Trending will be done on a daily basis rather than waiting for a weekly run as part of the job that builds the week-to-date PDBs.

Using this architecture, the detail information to research problems is available for 255 days, the weekly data (with reduced variables and datasets) is available for 255 weeks, and the monthly data (with even further reductions in variables) is available for as many months as may be deemed necessary. At this installation, no attempt is made to create monthly SMF datasets. A weekly dataset excluding the CICS transactions is more the 20 volumes of 3490 tape. The time needed to find a specific piece of data for say the 20th day of a month across more than 100 tape volumes is staggering.

The prototype jobs being used to build and clear the week-to-date and month-to-date datasets should be available soon. In each case, the goal was that the resulting datasets for a week-to-date or month-to-date should be no larger than the current days PDB. Thus, the online space required was reduced to that of 3 daily PDBs.

Conclusion:

40,153,273,652 is indeed a very large number. But it is possible to deal with these volumes of SMF data. The combination of optimization of programs, exploitation of operating system features, and parallel processing has enabled us to process more data than we had conceived to be possible or even rational. The PDB now encompasses more data sources than the original design intended making more information available to management and enhancing decision-making capabilities.

Further development is necessary to bring this architecture forward into the processing of WEEKLY, MONTHLY, and TREND processing but this should be trivial in comparison to the effort to date.

Appendix A: Contents of Component SOURCLIBs

BASE Component

BUILDPDB	Modified to remove SORTs of CICS datasets, DIFFDB2, and CICINTRV
IMACCICS	Modified to put all CICS datasets in the WORK DD
IMACDB2	Modified to put all DB2 datasets in the WORK DD
IMACFILE	Modified to delete SMF record types: 73, 74, 75, 78, 100-102, and 110
JCLPDB	JCL to execute

CICSSTAT Component

IMACCICS	Modified to route CICSTRAN to WORK
TYPE110	Modified to remove references to CICINTRV and ASUMCICS
JCLPDB	JCL to execute

CICSTRAN Component

EXCICxxx	Members except EXCICTRN set to null members
IMACCICS	All datasets but CICSTRAN sent to WORK
IMACDB2	DB2 datasets routed to DB2 DD
IMACKEEP	Modified KEEP list on CICSTRAN and elimination of other CICS data
TYPE110	Modified to remove references to CICINTRV and ASUMCICS
JCLPDB	JCL to execute

DATASET Component

BILDDCOL	Code to read DCOLLECT and sort into DCOLLECT PDB
BILDDSNS	Tailored DAILYDSN
BUILD TMS	Code to read TMC and place in DATASETS DB
DCOLLECT	Control cards for DCOLLECT
JCLPDB1	Job that builds TMC data
JCLPDB2	Job that runs DCOLLECT and decodes data
JCLPDB3	Job that merges TMC and DCOLLECT data

DSETOPEN Component

EXTY1415	Modifications described in ANALDSET
EXTY30xx	All but EXTY30_1 and EXTY30_4 modified to NULL members
EXTY664	Modifications described in ANALDSET
EXTY64X	Set to NULL member
IMACKEEP	Modifications as described in ANALDSET
DALYDSET	Modification of ANALDSET to run from WORK rather than tape
JCLPDB	JCL to execute

DB2 Component

DB2PDB	Code to build the DB2 PDB
DB2PLANS	Code that merges the STEP data and DB2 Accounting data
EXTY30xx	All except EXTY30U4 set to NULL members
JCLPDB	JCL to execute

IOSUBSYS Component

BLDIOPDB	Code to build the IO PDB
EXTY74	Modified to output all records
EXTY74CA	Modified to select a single system as the source of CACHE data
JCLPDB	JCL to execute

Appendix B: Typical days SMF Volume Distribution

Job	Input MB	Output MB
Base PDB	4,194	321
IO PDB	1,208	623
Read TMC	193	172
Read DCOLLECT	421	443 ⁷
DB2ACCT	1,753	760
CICS Statistics	455	643
DSETOPEN	2,259	830
Test Systems	167	34
CICS Transaction Run 1	8,001	848 5 tapes*2 ⁸
CICS Transaction Run 2	12,000	1,462 8 tapes*2
CICS Transaction Run 3	9,343	993 6 tapes *2
Daily Total	39,994	7,129 38 tapes

⁷ The total for the DATASETS database includes the data stored in the TMC database

⁸ To allow for recovery, the CICSTRAN data is written to tape and then sorted to tape requiring this number of tapes but allowing for recovery if the job dies for some other reason. The tapes are scratched upon successful completion. All tapes are 'long' 3490 tapes or about 1.6GB each. Note that this is with a severely restricted KEEP= list for ASUMUOW processing.