Large System Implementation in SAS: The Resource Management System Using SAS, SAS/FSP, SAS/GRAPH and the Merrill Database

Dennis Case and Scott McGregor

1.0 ABSTRACT

This paper describes the design and implementation of a large SAS production billing system developed at Hewlett-Packard's Corporate Data Center. The design considerations for implementing such a large system in SAS will be discussed along with advantages of implementing this system in SAS, such as: quick prototyping and implementation, ease of use, adaptability, efficiency and the additional feature of graphics. Also described are a few problems that were encountered and overcome: conversion problems for users, problems with being the first SAS production system, disk capacity problems, SAS limitations, interface problems with the center's HP 3000 minicomputers, and code integrity.

The system implemented was a Computer Resource Management System based on the Merrill CPE database, using SAS as the implementation language and using SAS/FSP and SAS/GRAPH for additional support. This system allows data center staff to perform computer utilization research, as well as capacity management and hardware planning, though its primary function is for user billing.

1.1 BACKGROUND

Hewlett-Packard is a large electronics manufacturer with corporate headquarters and a data center located in Palo Alto, California. The data center runs 2 Amdahl 470 V/8s, an Amdahl 470 V/6-II, and a large network of HP3000 computers. A worldwide communications system is also based at the center. This could be described as a typical large corporate data center.

1.2 THE PROBLEM

The center is funded under a charge-back system, charging users for the cost of resources used, with a few economic incentives to encourage use of certain resources as opposed to the use of less efficient ones. Accounting for these charges was formerly accomplished by a complex string of 38 separate jobs, with over 60 programs using more than 250 files. Maintaining this system was an extraordinary task for one or even two programmers, especially given the constantly-changing computing environment of the data center. The system was showing several signs of approaching the end of the software life cycle.

In addition to the complex billing system, system performance statistics were being gathered with a separate system, licensed from Johnson Systems, Inc. This comprehensive system performed analysis of the SMF/RMF records completely separately from the user billing system.

Data center staff reported to management that a change was necessary. A much simpler system could easily satisfy all user billing and system statistics needs. Several distinct advantages would also be gained. Most significantly, the new system would be much simpler and at the same time more flexible, making it less costly to run and maintain. Thus began the analysis and design phase for the Resource Management System (RMS).

2.0 DESIGN CONSIDERATIONS

The design of the Resource Management System had to respond to difficulties encountered with its predecessor. Listed below are a few of the design criteria and the problems which inspired them:

The new system should be written entirely in one language.

The language should be carefully chosen to ensure it is powerful enough to handle the variety of functions required of the system, yet simple enough to easily allow for programming support.

The predecessor to RMS was written in Assembler, COBOL and PL/I, requiring an extraordinary programmer to maintain it all. SAS was selected as the programming language because it was deemed powerful enough to handle all the varied functions, but simple enough to be easily maintained in a changing environment.

The system should make maintenance simple.

The major maintenance problem with the old system was the modification of files and the subsequent modification of programs. With SAS it is much less a problem since, as mentioned before, SAS databases are self-defining. If a file format changed (because of the addition or deletion of a variable) most programs usually needed no modification.

All the data for the system should be in a unified database.

The previous system consisted of literally hundreds of files, each containing bits and pieces of information. If a slightly unusual request came along, it could require the matching and merging of many, many files. SAS provided the capability of handling a unified database of immense size and complexity with relatively simple programming techniques.

There should be a standard variable naming convention throughout the system.

Again, the old system was maintained by several different people throughout its lifetime, each with his/her own ideas on naming conventions. SAS enforces naming conventions by the fact that variables are named on the database itself. No opportunity is afforded to change names halfway through a program, minimizing confusion when system modifications are made.

The billing database(s) should be self-descriptive, not requiring a variety of special format descriptions to access.

The formats of the 250 or so files in the old system were similar but different and each required a separate format description. SAS, of course, eliminates the problem because SAS datasets are self-defining. Formats are of no concern.

Above all, the replacement system must not have a complex design or have complex program steps.

This required a completely new design for RMS. A straight replacement of the old system (which had grown to be very complex) would not be sufficient.

Each of the problems described above could probably have been solved with a programming language other than SAS, but with all things considered as a whole, SAS appeared to be the best solution.

2.1 ADVANTAGES

There are several advantages unique to SAS in an application like this in addition to the solutions to those problems just mentioned. Among these are:

Quick Prototyping and Implementation. SAS saves time. Large, complex systems can be developed very quickly in SAS. SAS supports concepts of structured programming and SAS control structures make the structure of a program very visible, rather than burying it amidst unimportant detail lines, such as format statements.

Ease of use. The SAS database is the key to adaptability: SAS allows code to be changed and modified with minimum effort. Since the SAS database takes care of all variable naming, typing and formatting, the maintenance programmer doesn't have to worry about that. This way, many of the problems with data compatibility between program versions are eliminated, greatly simplifying the programmer's task.

Also, since SAS programs always run from the source, there is never any worry about matching source versions with object versions. You are always certain about what code is being executed.

Efficiency. Machine efficiency of SAS programs is considered to be a little less than programs coded in other languages. SAS, in order to preserve its simplicity, has had to presume the general case in most instances. Specialized routines for specialized tasks are more efficient than the generalized routines in SAS.

But machine efficiency is becoming less and less important. While machine costs are falling rapidly, people costs are going up. SAS programs have great people efficiency. In SAS, complex tasks can be completely programmed and debugged in short order. The same efficiencies apply to program maintenance.

Graphics. Graphics were not considered in the original design of the replacement system. However, as design and programming progressed, it became clear that graphics would be a great aid in management's understanding of the morass of information this system would output.

The fact that SAS/GRAPH is completely integrated with SAS made the addition of graphic output a trivial task.

2.2 WHY SAS?

SAS was initially chosen as the source language for RMS, for the following reasons:

It has all the capabilities of a complete programming language. PROCs are set up for doing most common tasks, and specialized routines can be written in DATA steps. Additionally, PROCs can be written in-house to perform tasks which would otherwise require assembly language.

it is easy to learn and teach. It is not uncommon for a user to learn the fundamentals of SAS in 2 or 3 days, and to become proficient in a few weeks.

Non-programmers can understand SAS code. They can tailor SAS programs to fit their own needs or write their own simple SAS programs.

SAS is simple to maintain. It encourages structured programming styles. Modular approaches to problem-solving work well in SAS.

SAS has many specialized PROCs which simplify the kinds of processing required by RMS. Table handling is simplified and report generation is trivial. Also, graphics are easily produced.

SMF analysis routines are already coded in SAS. The Merrill SMF routines eliminate tedious analysis and interpretation of SMF records. MICS, another SMF/RMF analysis package, is also coded in SAS.

2.3 ADDITIONAL FEATURES

In addition to the advantages listed above, SAS has a couple of additional features and products which made it even more attractive to this sort of production system.

SAS/FSP: This relatively new SAS product finally made direct updates to SAS databases easy for the non-SAS programmer to do. PROC EDITOR was too difficult for a non-SAS expert to handle with confidence. SAS/FSP is used to update rate tables and small SAS datasets. We have implemented SAS/FSP as an option of our TSO/SPF main menu.

SAS/GRAPH: Graphics were not part of the original system design, but an afterthought. Since one of the functions of the system was management information, and graphics are a most effective means of communicating large amounts of information to management, graphics were a natural extension for RMS. SAS/GRAPH made this all very easy. Graphs illustrating trends in system utilization, system performance and budget performance were easily produced from the RMS databases.

With all these advantages to using SAS in the new Resource Management System, management decided SAS was the optimal language for this application. They were impressed with the low cost figures for development and maintenance. Some concerns were aired over compatibility and other problems with the language itself, but these were minor, and eventually, management approved the project.

3.0 IMPLEMENTATION CONSIDERATIONS

Management approval of the project was not the last of the problems, however, as any large-scale project is bound to have a few headaches. This one was no exception.

Most of the problems with the Resource Management System stemmed from the incompatibility of SAS with anything previously developed at the data center. No one had ever developed anything in SAS or any language like it before. Specifically the problems were:

SAS does not have object or load modules. SAS is executed from source each time.

This is both an advantage and a disadvantage. There is never any problem with version control between source and executable code, as the program is executed from source each time. But it also gives greater opportunity for error. Load modules have an advantage in that they have already been checked for syntax errors. SAS programs cannot be compiled without execution (with real files, which contain real data). A lot of errors tend to pop up at 2:00 a.m. without careful testing. Also, since no dataset security system was installed at the time, raw SAS code was very vulnerable. The security people in our data center were very nervous about that.

The syntax problem was solved by setting up a completely duplicate system with a small amount of data for testing.

Also a security system was installed to protect datasets against unauthorized access.

SAS doesn't have a good interface to minicomputers. Many users had previously used their HP3000 minicomputers to access and massage Amdahl system data. With SAS, this became impossible to do directly.

For these users, we created a sequential file of a standard format and let them access that. The form of the data was different, so their old data analysis programs didn't work on it. This required much rewriting of programs.

SAS has a annoying requirement that SAS disk datasets reside entirely on one pack. When working in a production environment with massive amounts of data (in our case, reducing a month's worth of SMF data down to useful information) it can be difficult to store all that data on one pack. Space problems may occur during peak periods on work packs.

This was resolved by pre-allocating necessary space on work packs or on available private packs when several hundred cylinders or so were not available on any one work pack. In a production environment, it is often difficult to justify reserving several hundred cylinders all month for just 3 hours' use. So, in addition, several little tricks, such as PROC DELETEing everything, subsetting and summarizing data early in the programs, and moving SAS datasets into separate databases (on separate packs) helped a great deal. We experimented and found moving work datasets to tape to be unacceptable as an alternative.

A minor bug was discovered during our work with PROC SUMMARY which we reported to SAS, and should be fixed in SAS82. Briefly, we found it was impossible to use PROC SUMMARY with a BY statement on multi-volume SAS datasets. This problem seriously affected development. It required a lot of work to write around. But, as mentioned before, the problem should be fixed in SAS82.

The problems listed above are mostly concerned with difficulties we encountered in using SAS. By far the most stymying of the problems was parallel testing RMS with the old system.

In many cases, the Merrill database and RMS had new capabilities which were not possible in the old system. For instance, remote output and I/Os to dynamic DDnames were not detected on the old system. These figures were difficult to prove correct. Several other things, including several rounding errors by the old system, all produced significant differences in the figures.

Strangely enough, the simplicity and accuracy of SAS had created its own problem. In many cases, we had to reproduce old system errors in SAS in order to obtain comparable figures for verification. This sometimes was very difficult to do. It required research into many of the old programs to find their errors and omissions, and often required some very convoluted coding in SAS in order to reproduce the error.

Finally, after several months of verification work, the new Resource Management System was certified functional and accurate. It was adopted as the new billing instrument of HP's Corporate Data Center.

4.0 THE RESOURCE MANAGEMENT SYSTEM

The Resource Management System can best be described by an analysis of it's functions. There are three of them: The daily system data collection, the foreign data collection, editing and consolidation, and the monthly adjustments, summarization and reporting.

4.1 DAILY DATA COLLECTION

This consists mostly of collecting the SMF records with the Merrill system (soon to be replaced with MICS). Disk VTOCs are also surveyed for space information. All this information is placed on a daily database and added to a month-to-date database. A summary report is generated for Administration.

4.2 FOREIGN INPUT AND CONSOLIDATION

All during the month, several foreign inputs to RMS are generated. Among these are COM, Keypunch and HP3000 charges. These are collected and edited at the end of the month and consolidated with the entire months system data. Another summary report is generated for Administration showing preliminary totals for each type of record for the month.

4.3 FINAL ADJUSTMENT,SUMMARIZATION AND REPORTING

The totals from the summary report are reviewed by Administration each month. If total revenues do not match expenses (as is often the case), manual adjustments (called "Rebates") are entered into the system. One final job consolidates these rebates into a final, adjusted database for the month. This job also produces tapes for Corporate Accounting along with several reports. One of these reports is the user's bill for system services.

4.4 MORE INFORMATION

If the reader is interested, more detailed information on this application is available from the authors. They can be contacted at the address given in the Proceedings.

5.0 CONCLUSIONS

SAS can be a good tool in a production environment, but it requires some work. The applications programmer must be willing handle the differences in non-traditional ways. These problems, however, are not over-powering.

Overall, the RMS development has been a learning experience and is regarded as a positive step in the development of flexible and maintainable production systems. In other words, it works, and when applications arise in the future where SAS is a possible choice, it will be given full consideration.