Paper 99

The SDQZ-Pipe Technique: Doing More With Less, Faster and Cheaper

Reading UNIX-Compressed SAS® Data Sets Directly in a Data Step

Randy Hirscher, RAND, Washington, DC

ABSTRACT

While hard disk space is amazingly "cheap" nowadays, many of us programmers continue to find ourselves working in environments that have significant hard disk constraints. The 'always-ever-increasing-in-size' data set too quickly eats up the hard disk "real estate" available to us which will give us immediate and direct access to our data. How many of us have found ourselves spending an inordinate amount of time on trivial, mundane, and unwelcomed tasks such as temporarily compressing or moving our data offline, slicing-and-dicing it, or making some other sort of concession with respect to our data just because there was not enough hard disk space for what we needed to do? As a result, in order to save disk space, programmers working in a UNIX environment will typically store SAS data sets in UNIX-compressed format. Unfortunately, SAS is unable to directly read a UNIX-compressed, random access, SAS data set. The typical "INFILE...PIPE" approach does not work with SAS random access (ssd01) files. Programmers are instead forced to uncompress SAS data sets in an entirely separate step prior to reading the data in SAS. The technique presented in this paper however reduces the processing of a UNIX-compressed SAS data set to one step using named pipes and the TAPE sequential engine and, for this reason, it should prove to be a valuable tool for any programmer's toolbox.

INTRODUCTION

In this paper, I present a technique applicable to SAS running under UNIX, however the concept *should* theoretically extend to any operating system that supports the implementation of *named pipes*. Specifically, this technique has been successfully tested and implemented on a Sun Sparc 20 workstation running SunOS and on a Sun Ultra 5 workstation running Solaris 2.6 running SAS 6.11 and above. Using **named pipes** in conjunction with SAS's sequential **TAPE** library engine, this technique allows a programmer to *directly* read/write a UNIX-compressed SAS data set to a named pipe. The neat "trick" about this is that the SAS file is UNIX-compressed (or uncompressed) *on-the-fly* -- WITHOUT having to UNIX-compress (or uncompress) the SAS data file in a totally separate processing step. This technique, however, requires that the SAS data be created as a sequential file using SAS's TAPE engine. Using the typical INFILE…PIPE technique does not work since .ssd01 (SAS default data) files are not sequential files but rather random access files. This technique relies on the SAS data files being stored as sequential files and, for this reason, has some limitations with respect to what can be done with the UNIX-compressed, sequential-format SAS data. However in my estimation, the advantages of employing this technique far outweigh the disadvantages. Of course, this depends on what the user wants to do with the data and how they want to do it – every case is unique. In this paper I present the basic elements of the technique in its simplified form in addition to pointing out several of the advantages and disadvantages of using this technique.

As an introduction, let me explain why I resorted to this technique – describing the constraints of the environment that I was working under and which subsequently motivated the development of this technique. First, I had a limited amount of available hard disk space that I was sharing with multiple users (~16GB). Second, I was regularly and concurrently processing several large data sets (3-6+ GB). Third, I needed to abide by strict confidentiality agreements which required that all data (including temporary data like workspace) be stored on one particular volume. Fourth, as usual, I needed to complete the data development task in as timely a manner as possible. Finally, since I work in a "project environment" where individual projects are assessed data processing, storage, and backup charges based on the amount of data processed, stored, and backed up, to the extent possible, I needed to minimize the costs associated with these particular data processing tasks.

THE SDQZ-PIPE TECHNIQUE

The SDQZ-PIPE technique is a two step process:

- Setting up the named pipe
- Calling the appropriate SDQZ macro

STEP 1: SETUP THE NAMED PIPE

First, you setup the global macro variable, **&PIPENAME (**the named pipe), by issuing a call to the **%SETPIPE** macro. &PIPENAME is composed of the values of the operating system assigned macro variable, **&SYSJOBID**, which is the job id for the current process being executed, and the user-assigned value, **&PIPEDIR**, which is a directory/path, the argument that is passed to the %SETPIPE macro which the user specifies. Thus, the resulting value of the global macro variable, &PIPENAME, is a fully qualified name of the special file -- the named pipe used in the reading/writing of the UNIX-compressed SAS data file. Note that I set the value of &PIPENAME to the current job id so as to preclude the possibility of having an immediate naming conflict with another recently submitted process. [Note: Pipes are a means for processes to communicate with each other. A named pipe, sometimes referred to as a FIFO (first-in-first-out) file, is a combination of a file and a pipe that can be opened, written to, and read from. After a named pipe is opened, data is read/written in a first-in-first-out order].

```
%macro setpipe (pipedir) ;
   %global pipename ;
   %let pipename = &pipedir&sysjobid ;
%mend setpipe ;

example pipe setup:

    %let pipedir = /home/scratch/ ;
    %setpipe(&pipedir) ;
```

SETPIPE Macro Argument:

&pipedir – fully qualified directory/path of named pipe

STEP 2: CALLING THE SDQZ MACRO

Next, you issue a call to the appropriate SDQZ macro (%SAS2SDQZ for writing, %SDQZSAS for reading) passing three arguments: 1) **&ZDIR**, a fully-qualified directory/path of the file; 2) **&ZFN**, a one-level, SAS file name; and 3) **&ZSTMT**, any SAS data set options desired for the file. The syntax for calling the SDQZ macros is:

```
%sas2sdqz (&zdir, &zfn, &zstmt) ;
%sdqz2sas (&zdir, &zfn, &zstmt) ;

example sdqz2sas call:

       %let zdir = /home/myproject/ ;
       %let zfn = sdqzfile ;
       %let zstmt = (obs = 10 keep = id) ;
       %sdqz2sas (&zdir, &zfn, &zstmt) ;
```

SDQZ Macro Arguments:

&zdir – fully qualified directory/path of file
&zfn – one-level, SAS file name
&zstmt – SAS data set options statement
(e.g. keep=, rename=, obs=, where=, etc)

THE SDQZ-PIPE ENGINE

ENGINE OVERVIEW

The engine behind the SDQZ-PIPE technique is essentially made up of four components. These components can be summarized in the following for steps:

- Create a named pipe using UNIX's **MKNOD** command with the p option
- Initiate the UNIX compress/uncompress process, referencing the named pipe, using a standard SAS **DATA…INFILE** statement, forking the process
- Read/write data from/to the end of the named pipe using a standard SAS **DATA…SET** statement
- Delete the named pipe using UNIX's **RM** command

ENGINE COMPONENT 1: CREATE NAMED PIPE

First, the SDQZ engine creates the named pipe, **&PIPENAME** that was previously defined by the user, using a SAS **X** statement which invokes UNIX's **MKNOD** command with the **p option.** The UNIX MKNOD command with p option makes a directory entry for a special file. Note that you *must* specify the *p option* denoting that the file is a special FIFO file (i.e. a named pipe).

```
x "mknod &pipename p" ;
```

ENGINE COMPONENT 2: INITIATE COMPRESS/UNCOMPRESS PROCESS

Next, the SDQZ engine initiates the UNIX compress/uncompress process using the named pipe. The local macro variable, **&EXTN,** which is the file extension specific to the macro being called (in these macros, **sdq01.Z**) is set. [Note: Regardless of the extension that is specified, SAS will create a sequential-format file.] Then the FILEREF to the UNIX uncompress/compress command using the filename pipe is assigned. The process is then initiated and forked via a DATA…INFILE statement, referencing the uncompress FILENAME pipe.

```
%let extn = .sdq01.Z ;
filename ucmpres pipe "uncompress _
>&pipename< &dir&fn&extn &" ;
data _null_  ;
     infile ucmpres ;
 run ;
```

Note the order and direction of the redirection is very important. If this were a compress using a named pipe, the syntax the SDQZ engine would use is as follows:

```
%let extn = .sdq01.Z ;
filename cmpres pipe "compress _
<&pipename> &dir&fn&extn &" ;
data _null_  ;
     infile cmpres ;
 run ;
```

ENGINE COMPONENT 3: READ/WRITE DATA FROM/TO NAMED PIPE

After initiating the compress/uncompress process, the SDQZ engine reads/writes data from/to the end of the named pipe using a standard SAS…SET statement. It first assigns the LIBREF to the named pipe. Then the data being uncompressed by the UNIX uncompress command is read/written from/to the end of the named pipe.

Note that by employing the **&stmt** macro variable (the &zstmt

```
libname saslib "&pipename" ;
data &fn ;
  set saslib.&fn &stmt ;
 run ;
```

macro variable assigned by the user), valid SAS data set options can be employed to control the output of the data set. (I have found these to be very useful.)

Alternatively, when writing a UNIX-compressed SAS file using the SDQZ technique, the syntax is as follows:

ENGINE COMPONENT 4: REMOVE NAMED PIPE

```
libname saslib "&pipename" ;
data saslib.&fn ;
  set &fn &stmt ;
 run ;
```

Finally, the SDQZ engine removes the named pipe that was created earlier for the SDQZ technique using another SAS X statement that invokes the UNIX **RM** command. The UNIX RM command removes directory entries.

ADVANTAGES/DISADVANTAGES

```
x "rm &pipename" ;
```

Several advantages and disadvantages (limitations) to using the SDQZ-PIPE technique are identified below including a brief discussion.

ADVANTAGES

- **Modest *decrease in access/processing times*.** Preliminary unscientific, generalized benchmarks suggest a 5-20% decrease in processing times, with many factors potentially affecting the magnitude of decrease in access and processing times you will encounter. The speed increases that you encounter is primarily due to the fact that the SAS sequential engine (TAPE) requires much *less overhead* than the default engine that creates standard SAS random access files since sequential access is much simpler than random access.

- **Significant *decrease in data storage requirements.*** Preliminary unscientific, generalized benchmarks suggest a 30% decrease in disk space required. Of course this will vary depending on the type of data that you are processing, dependent upon how ‘compressible’ your data is.

- **Reduction in *data management tasks.*** Related to this, the tasks associated with managing your data are significantly reduced. The number of files necessary to have on hand can be limited to one copy. You can avoid the usually unwanted scenario of having compressed and uncompressed versions of the same file. Partial files, test files, sliced-and-diced files, compromised data processing, etc. are all reduced which will translate into more efficient programming.

DISADVANTAGES

- The SAS sequential engine does *not support indexing* of observations

- Cannot use *random data access methods* such as the POINT= data set option

- Can only access one SAS file in a sequential library in a particular DATA or PROC step at a time (i.e. no concurrent access)

- Other limitations apply to:

  - Updating a SAS data set in place (e.g. APPEND procedure, SQL with UPDATE statement)

  - Deleting, renaming, or moving a SAS data set (e.g. COPY procedure with MOVE option, SQL procedure with the DROP statement)

  - Changing a variable’s attributes (e.g. DATASETS procedure with the MODIFY statement, SQL procedure with the ALTER statement)

  - Procedures that use nonsequential access to the data (e.g. CHART, PLOT, RANK procedure with the BY statement)

CONCLUSION

While this technique will not be applicable for every application due to the limitations imposed by the sequential access format, it has proved to be a very useful technique for me given the constraints that my particular data processing environment imposed. Assuming that random access methods are not required, a user can experience the same benefits that I did using this technique, most notably faster processing times, decreased data storage requirements, and most notably, a reduction in the data management tasks that is required to keep ones data organized.

REFERENCES

SAS Institute Inc. (1990*), SAS Language Reference Version 6 First Edition,* Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1993), *SAS Companion for UNIX Environments: Language,* Cary, NC: SAS Institute, Inc.

SAS Institute, Inc. (1990), *SAS Procedures Guide Version 6 Third Edition*, Cary, NC: SAS Institute, Inc.

SAS Institute, Inc. (1991), *SAS Languages and Procedures Version 6 First Edition,* Cary, NC: SAS Institute Inc.

AUTHOR CONTACT INFORMATION

Any comments, questions, or suggestions concerning this technique are valued and encouraged. The full source code for the technique that is presented in this paper will be freely distributed upon request. The author can be contacted at the following:

Randy Hirscher
RAND
1333 H St. NW
Suite 800, DC-11
Washington, DC 20005

Phone: 202.296.5000 x5210
Fax: 202.296.7960
Email: randall_hirscher@rand.org