

Efficient Techniques for Random Access to SAS Datasets

David Sommer, DynaMark Inc., Arden Hills, MN

Abstract

The standard technique for random access in SAS® DATA step coding is the use of the KEY= option of the SET statement to read observations from an indexed dataset. Also well known as a random access technique is the use of formats created by the FORMAT procedure. But when millions of accesses must be made, the indexed dataset technique is often very slow. And although very fast in execution, the format technique is tedious to implement.

This paper gives performance results on the indexed, format, and a hybrid technique, with some guidelines as to when each is appropriate. Macros presented here make the use of any of the three techniques easy and as efficient as each can be.

Introduction

This paper deals with the following problem: A very large dataset is to be read. As each observation is read, information from a random access file must be added to it. For example, the very large file may be a name and address file having 10 million observations, and as each observation is read, we must access demographic information that is stored by zip code or census tract.

Modern operating systems use read-ahead techniques that make sequential processing quite efficient. But when a file must be accessed randomly, these techniques are of little use. If each access requires 25 milliseconds, then 10 million accesses take $0.025 \times 10,000,000 = 250,000$ seconds = 69 hours!

As will be seen, no one technique works always, but knowing the techniques, and having some macros available to ease the work of implementation, can make it possible for you to choose the best method for your needs.

Alternatives

There are three well-known methods for random accessing:

- Sort both files by the key, e.g. zip code, merge, then sort back to desired order.
- Use the SAS indexed file technique.
- Use the FORMAT procedure to create a format containing the needed information.

Sort and Merge

This is satisfactory if neither the large sequential file nor the small random access file changes very often compared to the number of times the large file must be read. Under this condition, the large amount of processing to sort, merge, and sort again is easily "paid back" by the resulting efficiency the many times the result file is read.

SAS Indexed File using KEY=

This can fail spectacularly, as indicated in the Introduction. If the random access file is sufficiently small, the operating system's buffering techniques may end up reading the entire file into RAM, and then access will be quite efficient. So sometimes this works well. But we will see in the Performance section that when indexed files work well, the third technique (use of PROC FORMAT, etc.) works even better.

FORMAT Method

The method consists of three steps:

1. Encode the desired information into the "label" portion of a format. (Use PROC FORMAT for this.)
2. Use a PUT(index_value, \$format_name) to get the "label".
3. Decode the label into the original variables.

As will be seen, this method is by far the most efficient *when it can be used*. When the format is active in a dataset, it resides entirely in RAM, and SAS uses an efficient binary search technique to find the desired value. There are two main objections to its use:

- It can be tedious to set up the format.
- The memory requirements may be too large and execution aborts.

The first objection is met by using a macro such as described later in this paper to perform the three steps outlined above. The second objection is real, but its incidence is minimized by doing a good job of encoding, which the macro does quite well.

Efficient, Easy Techniques

From here on, we drop discussion of the "Sort, merge, ..." technique. Instead we focus on cases where random access is really needed. And in addition to the Indexed Dataset and Format methods, we introduce a "hybrid" technique. This third technique will be shown to give greater efficiency than the standard Indexed technique in those instances where a Format technique cannot be used. The "hybrid" technique builds a format containing *only the observation number* of the desired information. This is used with the POINT= option of the SET statement to access the needed observation.

In summary, the efficient approach to random access is as follows:

1. Extract only the variables needed from the random access dataset. That is, don't keep any variables not actually required in the run.
2. Use a macro to build a Format in which the desired information is encoded.

3. OR use a macro to build a "point" format (the hybrid technique discussed above).
4. OR use a macro to build an indexed dataset.
5. And then do random access with code generated by any of the three macros involved in steps 2, 3, or 4.

Performance

To test the three techniques, we built random datasets of size 500, 5,000, 25,000, 50,000, 75,000, and 100,000. All are 99% dense, meaning that there are observations corresponding to 99% of all index values that will be tried. (In order to assure that all three techniques handle "misses" properly, we have to arrange for a few misses!)

Each of these six observation sizes are tried with 4, 12, 32, and 62 variables to be retrieved. This is intended to cover a realistic range of data width.

The three techniques used are: Format, Hybrid, and Indexed. All runs were done on SAS version 6.08 for OS/2®, running on a Gateway 2000® 4DX2-66V computer with 16 MB RAM and 100 MB free disk space.

The timing method uses a DO-loop in which random index values are generated and the data is retrieved by whichever of the three methods is being tested. Here is an example using the Format technique on retrievals of four variables (actually just three are retrieved – the fourth is the index variable itself). The index variable is NINDEX, and the other three variables are CHR1, CINDEX, and NUM1. The code is generated by the %ACEXER macro which has parameters SECONDS and METHOD. Here, SECONDS=100 (meaning the timing is for 100 seconds or more), and METHOD=F (the Format method).

%ACEXER(SECONDS=100, METHOD=F) * Here is what MPRINT shows;

```
DATA NULL ;
  START = DATETIME();
  DO UNTIL( (DATETIME() - START) GE 100 );
    DO I = 1 TO 1000;
      NINDEX = CEIL(500*RANUNI(0));
      * Next 6 lines are expansion of &COMMANDS;
      LENGTH HIT $1 XLABEL $19;
      XLABEL = PUT(PUT(NINDEX, RB4.), $GETIT.);
      HIT = SUBSTR( XLABEL,1,1);
      CHR1 = SUBSTR( XLABEL,2,8);
      CINDEX = SUBSTR( XLABEL,10,6);
      NUM1 = INPUT(SUBSTR( XLABEL, 16, 4),
        RB4.);
      HITS + (_HIT EQ 'Y');
    END;
    ITERS + 1000;
  END;
  SECONDS = DATETIME() - START;
  TPIM = 1000*SECONDS/ITERS;
  HITPCT = 100*HITS/ITERS;
  PUT ***** METHOD=F SECONDS=100 * ITERS= 'HITSX='
    HITPCT BEST5. * Time/Iter(ms)= TPIM BEST6.;
  STOP;
RUN;
```

We now give several tables showing performance results. Columns in each table are:

- Variables: the number of variables.
- Total Width: the number of bytes required for the data in each observation of the random access file.
- Technique: F for Format, H for Hybrid, or I for Indexed.
- Two columns labeled "msec". This gives the milliseconds per random access. The two columns give the extremes of our results – the first column is the smallest result; the second is the largest.

Subtitles give the number of observations in the test dataset, starting with 500.

These tables are tedious to read in detail. But scan them looking for evidence of these main points:

- The Format technique is always best (when it can be used).
- Except for some anomalies in intermediate file sizes, the Hybrid technique outperforms Indexed. In those exceptions, the Format technique should be used anyway.

Performance: 500 observations

Variables	Total Width	Technique	msec	msec	
4	22	F	0.176	0.182	Format always fastest
		H	0.595	0.617	
		I	0.625	0.725	
12	70	F	0.291	0.303	
		H	0.746	0.763	
		I	0.870	0.877	
32	190	F	0.699	0.719	
		H	1.423	14.125	Outlier!
		I	1.149	1.250	
62	370	H	1.530	4.208	Outlier! (no Format because Width > 200)
		I	1.667	1.724	

Note the "Outlier!" comments. For some reason, the operating system buffering didn't work well on one of the tests, and we got a very large value. There is also a comment in the Variables=62 case: the Total Width of 370 prohibits the Format technique since the "label" cannot exceed 200 bytes. (Of course, two formats could have been used, but we didn't extend our macro to the handling of this case.)

Performance: 5,000 observations

Variables	Total Width	Technique	msec	msec	
4	22	F	0.156	0.163	Faster than 500 observations! (Why this happens I have no idea.)
		H	0.840	0.901	
		I	1.515	1.603	
12	70	F	0.256	0.262	
		H	11.556	12.875	Time is approximately one disk access
		I	1.563	1.639	Outperforming HI A surprising result, but as we get to larger tests this will not continue. Of course, Format outperforms both Hybrid and Indexed and should be used.
32	190	F	0.662	0.667	
		H	10.	12.875	
		I	3.704	5.722	
62	370	H	23.6	24.	
		I	19.333	20.2	Indexed time is creeping up...

Performance: 25,000 observations.

Variables	Total Width	Technique	msec	msec	
4	22	F	0.148	0.149	Faster than 5000 observations!
		H	13.625	14.571	
		I	14.857	15.571	
12	70	F	0.263	0.265	
		H	15.714	16.429	
		I	23.2	23.2	
32	190	F	0.676	0.685	
		H	22.2	23.	
		I	41.	42.	From here on, H always better than I
62	370	H	28.25	30.25	Looks like 2 disk accesses.
		I	47.333	63.5	Approximately 4 disk accesses.

Performance: 50,000 observations

Variables	Total Width	Technique	msec	msec	
4	22	F	0.201	0.211	
		H	15.	17.667	
		I	31.25	51.	Outlier!
12	70	F	0.303	0.326	
		H	16.429	17.833	
		I	37.667	49.333	
32	190	F	0.719	0.735	
		H	23.4	23.8	
		I	44.333	47.333	
62	370	H	31.5	35.667	
		I	51.5	62.	

Performance: 75,000 observations

Variables	Total Width	Technique	msec	msec	
4	22	F	0.171	0.184	
		H	15.857	16.143	
		I	39.	65.	
12	70	F	0.265	0.278	
		H	17.333	19.333	
		I	40.333	43.	
32	190	F			SASLOG reported "Memory capacity exceeded after 66271 ranges"
		H	23.8	24.	
		I	59.5	65.5	
62	370	H	32.	33.333	
		I			SASLOG reported "Insufficient Resources"

Performance: 100,000 observations

Variables	Total Width	Technique	msec	msec	
4	22	F	0.154	0.215	
		H	16.286	17.667	
		I	41.667	65.	
12	70	F	0.267	0.320	
		H	19.333	21.2	
		I	43.667		
32	190	F			
		H	27.5		
		I	50.5		
62	370	H			Insufficient resources
		I			Insufficient resources

The Macros

Here we give brief summaries of each macro. See Author section for information on getting copies of the macros themselves. The default values of the parameters allow easy tests without specifying much.

%ACSTRIP(DATA=WORK.TEMP, OUT=WORK.TEMP1, VAR=NINDEX CINDEX NUM1 CHR1) Selects specified variables from any dataset, and writes resulting dataset with desired name.

%ACSNOOP(DATA=WORK.TEMP1, VARS=SNOOPY, PRINT=NOPRINT)

Uses PROC CONTENTS to get length and type information about all variables in a dataset. The information is put in a global variable. This is used by ACFORM1 and ACFORM2. The macrovariable consists of tokens separated by spaces, where each token is of the form name/length/type. name=name of variable, length=its length, and type=1 if numeric, 2 if character. Uses temporary dataset _TEMP.

%ACFORM1(DATA=WORK.TEMP1, INDEXVAR=NINDEX, VARS=SNOOPY, CMDS=COMMANDS, FMT=GETIT, XLABEL=_XLABEL, HITVAR=_HIT)

Builds a format from a dataset, using specified variables as index, and other variables as the formatted value. INDEXVAR is a list of index variables separated by spaces. VARS is the name of a macrovariable of the same form that macro ACSNOOP creates. CMDS is a name for a global macrovariable which will contain the commands to take apart the formatted value. FMT is the name of a format to be used for the access. XLABEL is a variable that will be used when accessing the format. HITVAR is the name of a variable to be used for the Hit flag. It will be 'Y' if there is a hit, 'N' otherwise.

%ACFORM2(DATA=WORK.TEMP1, INDEXVAR=NINDEX, VARS=SNOOPY, CMDS=COMMANDS, FMT=GETIT, XLABEL=_XLABEL, POINT=_POINT)

Builds a format from a dataset, using specified variables as index, where the formatted value points to the record. INDEXVAR is a list of index variables separated by spaces. VARS is the name of a macrovariable of the same form that macro ACSNOOP creates. CMDS is a name for a global macrovariable which will contain the commands to take apart the formatted value. FMT is the name of a format to be used for the access. XLABEL is a variable that will be used when accessing the format. POINT is the name of a variable to use for a POINT value in a SET statement. When the format returns the point variable = 0, there is no match and no SET should be done.

%ACINDEX(DATA=WORK.TEMP, OUT=WORK.TEMP1, VAR=NINDEX CINDEX NUM1 CHR1, INDEXVAR=NINDEX)

Selects specified variables from any dataset, and writes resulting dataset with desired name and with indexing. If a composite index is wanted, give all index variable names in INDEXVAR=... and then follow that with a "/" and the name for the composite index name. DATA can be either a one-part or two-part name, as can OUT.

Test macros: The following are descriptions of macros used to build test data and exercise the three techniques.

%ACBUILD(OUT=WORK.TEMP, SIZE=500, NUMERIC=2, CHAR=2, DENSITY=99)

Generates a dataset of any desired size, with any number of numeric (length 4) and character (length 8) variables, plus a numeric index (length 4) and a character index (length 6). The dataset is not in sort

by the index variables. The density may be any percent (given by the DENSITY parameter). The prime number 17389 is used as offset from one index to the next, so avoid near multiples or submultiples of this number to get good dispersion. The actual size of the dataset will be SIZE*DENSITY/100.

```
%ACEXER(SECONDS=100, METHOD=F,  
DATA=WORK.TEMP1, KEY=NINDEX,  
CMDS=COMMANDS, SIZE=500)
```

This exercises any of the three ACRANDOM access methods. SECONDS is a lower bound on how long to exercise. METHOD is F for Format only, P for format and Point, or I for Indexed. If METHOD is I, the DATA parameter gives the dataset. If METHOD is F or P, CMDS gives the global commands sets. If Method is I, KEY gives the index variable. The SIZE parameter gives the SIZE value as in ACBUILD. Output is to the SASLOG, and gives the Method, Seconds of testing, Iterations done, Hits percent, and the Time/Iteration in Milliseconds (as "Time/Iter(ms)"). It is assumed that _HIT is the name of the HIT variable for METHOD=F, and that _POINT is the POINT variable for METHOD=P.

Conclusions

- Use Format when you can.
- Otherwise, use the Hybrid technique.
- Experimentation in your own operating environment is certainly worthwhile.
- Using the macros makes any of the three techniques easier.

Acknowledgements

My employer, DynaMark, Inc., provided the time and inspiration to write the macros, test the techniques, and prepare this paper. The macros are available with the permission of DynaMark, Inc.

Author

David Sommer
Dynamark, Inc.
4290 Fernwood St.
Arden Hills, MN 55112
612/486-1805
Fax: 612/481-8077

For a copy of the macros, send a 3.5" formatted diskette with a self-addressed, stamped mailer to the author.

SAS is a registered trademark or trademark of SAS Institute, Inc. in the USA and other countries. OS/2 is a registered trademark or trademark of International Business Machines Corporation. Gateway 2000 is a registered trademark or trademark of Gateway 2000, Inc.

• indicates USA registration.