Paper 037-2009

Scalability of Table Lookup Techniques

Rick Langston, SAS Institute Inc., Cary, NC

ABSTRACT

This paper will examine the issues of memory and disk space consumption and performance considerations for various table lookup techniques using large amounts of data. These techniques include PROC FORMAT in association with the PUT function, the DATA step hash object, DATA step merging, the KEY= option of the SET statement, and PROC SQL. This paper will show performance comparisons of the different techniques as the amount of data increases, and will offer advice on decisions on which techniques to use in various circumstances.

INTRODUCTION

Table lookup is one of the most common needs in SAS programming. By *table lookup*, we mean the mechanism by which a particular key value is matched with some other label. The key might be an account number and the label might be the account information. Or, the key might be a part number and the label might be a part description. SAS provides many different techniques to perform table lookup. These techniques will consume memory, disk space, and CPU time as the number of key/label pairs increases, and some of the techniques might exceed the capacity of these resources depending on the table size. This paper describes five different techniques and how they scale up; that is, how resource utilization changes as the size of the table grows.

THE FIVE TECHNIQUES

For the purposes of comparing resource utilization, we created a SAS data set of key/label pairs that consists of incrementing numbers from 1 to n, with labels being the number using the Z8. format. For example, the key value of 1 had the label 00000001. The base table is the SAS data set that contains the keys to look up, and the lookup table is the SAS data set that contains the key/label pairs. For simplicity, we used the same table for the base table and the lookup table. The base table was sorted using a random value to properly exemplify real-world scenarios where the key is not in sorted order in the base table. For further simplicity, the base table used `start` and `label` as variable names so that it could be treated as a CNTLIN= data set.

Using the base table named `temp`, we tried five standard techniques for table lookup. The following SAS code was used for each technique:

```
/*-----PROC FORMAT and PUT function-----*/
proc format cntlin=temp; run;
data _null_; set temp(rename=(label=shouldbe)) end=eof;
     label=put(start,$testfmt.);
     if label=shouldbe then matched+1;
     if eof;
     if matched=_n_ then put 'all matched';
     else put 'not all matched';
     run;

/*-----hash object-----*/
data _null_; set temp(rename=(label=shouldbe)) end=eof;

     length label $20;
     retain label ' ';

     if _n_=1 then do;
        declare hash ht(dataset:"temp");
        ht.defineKey("start");
        ht.defineData("label");
        ht.defineDone();
        end;
     rc = ht.find();
     if rc = 0 then do;
        if label=shouldbe then matched+1;
        end;
     if eof;
     if matched=_n_ then put 'all matched';
     else put 'not all matched';
     run;

 /*-----merge-----*/
proc sort data=temp out=temp2(drop=random rename=(label=shouldbe)); by start;
run;
proc sort data=temp out=lookup(drop=random); by start;
run;
data _null_; merge temp2(in=want) lookup end=eof; by start;
     if label=shouldbe then matched+1;
     if eof;
     if matched=_n_ then put 'all matched';
     else put 'not all matched';
     run;

 /*-----key= usage-----*/
data lookup(index=(start)); set temp(keep=start label);
     run;
data _null_; set temp(keep=start label rename=(label=shouldbe)) end=eof;
     set lookup key=start;
     if label=shouldbe then matched+1;
     if eof;
     if matched=_n_ then put 'all matched';
     else put 'not all matched';
     run;

 /*-----SQL inner join-----*/
proc sql;
   create table merged as
      select label,shouldbe from temp a inner join temp(rename=(label=shouldbe)) b
         on a.start = b.start;
   quit;

data _null_; set merged end=eof;
     if label=shouldbe then matched+1;
     if eof;
     if matched=_n_ then put 'all matched';
     else put 'not all matched';
     run;
```

TESTING THE TECHNIQUES

Because we wanted to demonstrate scalability, the number of observations in the base and lookup table increased logarithmically: 1000, 10000, 100000, 500000, 1 million, 5 million, and 10 million. All five techniques were tried with the base and lookup table at this observation count.

The tests were run on a 64-bit UNIX system, a 64-bit Windows system with 16 gigabytes of memory available, and a 32-bit Windows system with 2 gigabytes of memory. The results were similar for the two Windows platforms, but the UNIX platform had problems due to greater memory constraints.

For an observation count of up to 500,000, formats and the hash object both outperformed key lookup and SORT+MERGE, but SQL JOIN performed a little better than either formats or hash tables at this level. However, at a count of around 700,000, PROC FORMAT did not have sufficient memory to produce the format on the UNIX platform. The hash object processing was successful up to around 1,900,000, at which point it ran out of memory. The other three techniques continued to function up to 10 million observations, with SQL JOIN out-performing the others. KEY= seemed to be the slowest regardless of the observation count. For both Windows platforms, all operations completed successfully with no memory failures.

Specific numbers recorded from these tests can be seen in the PERFORMANCE STATISTICS section.

RECOMMENDATIONS

Based on the tests we ran, we recommend that either formats or hash objects be used for smaller numbers of observations in the lookup table. For larger numbers of observations, we recommend using SQL JOIN if you are looking for the fastest throughput. However, formats and hash objects are still viable choices as long as you have sufficient memory.

The threshold really depends on the characteristics of your key and label values. The more memory needed to hold your keys and labels, the lower the threshold before memory capacity will be exceeded.

Also, keep in mind your familiarity with the various techniques. Formats have been around for many versions of SAS, but hash objects are relatively new. PROC SQL has been around since SAS 6, but many users might find that their comfort level with SQL is not high enough to use it for extensive SAS coding. However, programmers who are more familiar with SQL than with SAS programming might prefer SQL techniques.

KEY= is certainly one of the easiest to write in SAS applications, but be mindful of the performance differences that we saw. SORT and MERGE are very natural to SAS programmers, and indeed the technique does scale well, although requiring sorted base and lookup tables causes additional overhead that could be avoided by using SQL and JOIN or hash objects.

PERFORMANCE STATISTICS

UNIX	1K	10K	100K	500K	1M	5M	10M
Format execution	0.01	0.02	0.25	1.56	1.96	-	-
Hash usage	0.06	0.05	0.26	1.58	3.61	-	-
MERGE merging	0.04	0.08	0.44	1.63	4.36	21.70	44.09
SQL table creation	0.06	0.05	0.21	1.81	3.88	21.26	45.45
Key lookup	0.01	0.13	1.46	10.56	21.96	129.57	266.95

Windows 64-bit	1K	10K	100K	500K	1M	5M	10M
Format execution	0.00	0.01	0.25	1.79	4.73	30.45	72.90
Hash usage	0.35	0.03	0.28	2.15	5.03	36.34	84.76
MERGE merging	0.05	0.02	0.37	1.33	3.01	25.44	49.60
SQL table creation	0.73	0.01	0.15	0.92	1.90	13.64	32.01
Key lookup	0.01	0.10	1.46	8.92	18.14	98.60	209.01

Windows 32-bit	1K	10K	100K	500K	1M	5M	10M
Format execution	0.01	0.01	0.21	1.68	4.82	27.39	62.84
Hash usage	1.20	0.01	0.26	2.42	4.90	35.68	82.59
MERGE merging	0.10	0.03	0.20	1.47	3.33	16.66	27.49
SQL table creation	0.59	0.01	0.12	0.76	1.68	12.25	27.60
Key lookup	0.07	0.09	1.37	8.14	16.75	93.17	189.50

Note that when comparing the performance of formats and hash objects, only the execution time of the DATA step was considered. The time taken to produce the format via a PROC FORMAT step was not considered, because the format can be reused without having to re-create it. If your applications require you to re-create the format with each execution, the additional overhead of format creation would need to be considered. In those cases, the hash object will out-perform formats due to this additional overhead.

MEMORY CONSTRAINT ISSUES

It is important to keep in mind the memory limitations you face, depending on the operating system you are using. When running SAS on 32-bit Windows, 32-bit Linux, or z/OS, the memory model only allows for a maximum of 2 gigabytes of user memory. Remember that format processing and hash object processing are memory-bound, and if more than 2 gigabytes are needed, you cannot use these methods. The actual amount of memory that is available might be much less. And remember also that certain portions of that memory must be contiguous. You might have a cumulative amount of 1 gigabyte available, but if there is no contiguous segment over 1 megabyte, you might still fail to process a larger number of values. Sixty-four-bit systems do not have the 2-gigabyte memory model limit, but their limits are still controlled by factors such as SAS system options, the available hardware memory, and also by administrator controls.

z/OS users tend to have the smallest amounts of memory available to them. This is more the case for those users running TSO, but typically the amount of memory available to batch jobs might not exceed 64 megabytes. Your system administrators will set those limits based on various criteria, including whether you might be charged for excessive memory usage.

SAS system options can have an effect on the available memory you have for building the format. For example, the MEMSIZE system option places a limit on the total amount of virtual memory that SAS dynamically allocates at any time. To determine the optimal setting of MEMSIZE, run PROC FORMAT with MEMSIZE=0 with the FULLSTIMER option. Note the amount of memory that is used by the process, and then set MEMSIZE to a larger amount. To see all the system options that pertain to memory, run

```
proc options group=memory define; run;
```

and look in the log for the definitions. This diagnostic is applicable regardless of your operating system.

MEMORY COMPUTATION FOR FORMATS

Computing the amount of memory needed to load a format can be determined by the use of the macro that I've written based on requests from users contacting Technical Support. This macro (called get_fmt_memsizes) will compute the memory needed for each format in a given catalog. If instead you want to know if a format you're planning to write can be feasibly loaded into memory, you can create a small sample of the format and run this macro, then extrapolate the memory usage. For example, if your final format will have 1 million ranges, you can create a format containing the first 100 ranges and find out the memory needed for that subset. Then multiply by 10000 to get the approximate memory usage for 1 million ranges.

At the time of this writing, a specific link had not yet been established for this macro. To obtain the macro and its documentation, go to http://support.sas.com and search for get_fmt_memsizes.

Remember also that if you are associating many user-written formats with many variables in a SAS data set, all of these formats are loaded when the data set is accessed in a DATA step, even if you do not explicitly reference the variables. For example, if your data set X has variables X1-X100, and you have a unique format named X1X-X100X that corresponds to each variable, when you run

```
data new; set x; run;
```

all the formats X1X-X100X will be loaded. If these are large formats, they will all need to occupy memory simultaneously. Note that this does not necessarily hold true for procedures, which can decide which of the formats to load.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author:

Rick Langston
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
E-mail: Rick.Langston@sas.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.