Exploring Efficient Ways to Collapse Variables

Yubo Gao, University of Iowa Hospitals and Clinics, Iowa City, Iowa

Abstract

There are many situations where we would want to find the frequency distributions of discrete results. Usually, more than one method is available in SAS. With today's computer technology, data files under a million records are not a problem. But if the data file is much larger than that, such as those present in major credit card/retailers companies and university/hospitals, efficient methods should be sought in order to reduce resource utilization, such as time expended. Based on a frequency distribution question posted at the SAS-L community, this paper first reviewed the methods suggested by the SAS-L subscribers, and then proposed two other methods. Next, a performance comparison among these methods in terms of CPU time usage was made by solving an expanded example, the comparison showed that the ratios of CPU time usage between the slowest method and the fastest one is 12, and the result may serve as a benchmark when solving similar problems.

Introduction

The SAS-L community is a web-based SAS user forum located at the University of Georgia, where SAS users can post questions and while other users can provide ideas or solutions. One day, a person called J posted a question titled "a better way to collapse variables?" as follows.

A dataset has 4 variables (q1-q4). To count the total frequency distribution of the answers (A,B,C,...,F) of all 4 questions, I use proc transpose to collapse these 4 variables. See below.

```
data old;
  input index q1 $ q2 $ q3 $ q4 $;
        /*do i = 1 to 200000;
                id+1;
                output;
        end;*/
  datalines;
 1 A B C D
 2 E F A E
 3 C B B A
 4 B A D E
 5 E F A B
 6 A A A C
 7 F E A E
 ;
run;

proc transpose data=old out=temp(rename=(col1=q));
  by index;
  var q1-q4;
run;

Proc freq data=temp;
  tables q;
run;
```

So the partial output looks like below, which is what I want.

q	A	B	C	D	E	F
frequency	9	5	3	2	6	3

Finally, he asked if anyone please offer a more resource-efficient way to achieve his goal. Since in reality he said his data has more than 16 million records ("index" variable runs up to 16 million +).

SAS-L is a really good helping site where many experts are eager to help. Very soon, several methods are proposed. DN provided with two methods: one is an array based method and the other proc summary based. JC suggested a divide and conquer method. YH presented a view plus proc freq based method. In the following sections, first we will

review each of them, next introduce two others, and then compare their performances in terms of CPU time by solving the expanded problem listed by J.

Method reviews

(1). J's method

The method by J consists of three parts, a data step, a transpose step, and a frequency step. See above. J did not have the do-end part and variable id in his original sample question. He used by variable index in the transpose step. The purpose of including do-end part and variable id here is to expand the file size for later performance comparison, and we will use by variable id rather than index then. No missing value is assumed in this paper.

(2). DN's array's method has a clever data step and a print step, see below. Array _l[*] contains the category variables. The inner do-end iterations for _n_ uses an index function to check the values of variables q1-q4 with string 'ABCDEF', and the index function will return a positive position number if there is a match. Based on the returned position number, the corresponding variable in array _l[*] will increase value by one. The data step does not output anything in the middle process but only the final frequency after the final record has been checked. This implicitly saves lots of time. Since the print step needs no time, this method is supposed to be very fast. The do-end iterations for k are used for later comparison.

```
Data DN1;
  infile cards eof=eof;
  input index (q1 q2 q3 q4) (:$1.);
  array _l[*] A B C D E F;
  array _q[4] q1-q4;
    do k=1 to 200000;
      do _n_ = 1 to dim(_q);
          _l[index('ABCDEF',_q[_n_])] + 1;
      end;
    end;
  keep A B C D E F;
  return;
eof: output;
datalines;
1 A B C D
2 E F A E
3 C B B A
4 B A D E
5 E F A B
6 A A A C
7 F E A E
;
run;

proc print data=DN1;
run;
```

While DN's proc summary based method has two data steps, two proc summary steps, see below. Since proc summary needs a relatively long time, it is conjectured that this method would need more time. The **ways** statement, according to the online documentation for SAS 9.1 3, specifies the number of ways to make unique combinations of class variables. Function coalescec accepts one or more character arguments. The coalescec function checks the value of each argument in the order in which they are listed and returns the first non-missing value. If only one value is listed, then the coalescec function returns the value of that argument. If all the values of all arguments are missing, then the coalescec function returns a missing value.

```
proc summary data=old;
   class q1-q4;
   ways 1;
   output out=oneways(drop=_type_ rename=_freq_=freq);
run;

data oneways;
   set oneways;
   answer = coalesceC(of q1-q4);
run;

proc summary data=oneways nway;
```

```
   class answer;
   freq freq;
   output out=freq(drop=_type_);
run;

proc print data=freq;
run;
```

(3). JC's divide and conquer method is to write out separate output datasets for each variable from a proc freq, then combine them and use the count field as a weight for a second proc freq. Since there is only one record in each of files freq1-freq4, data step new and second proc freq needs no time.

```
proc freq data = old;
  tables q1 / out=freq1 (drop=percent rename=(q1=Q));
  tables q2 / out=freq2 (drop=percent rename=(q2=Q));
  tables q3 / out=freq3 (drop=percent rename=(q3=Q));
  tables q4 / out=freq4 (drop=percent rename=(q4=Q));
run;

data new;
  set freq1 freq2 freq3 freq4;
run;

proc freq data = new;
  tables q;
  weight count;
run;
```

(4). YH's view plus proc freq method has two steps: one is to save data step view, and the other is to retrieve the view and to make frequency. YH thought that his method should be faster than the summary method. Since the view-based method is not often used, its performance needs to be determined.

```
data vold/view=vold;
  set old (keep=q1)
      old (keep=q2 rename=(q2=q1))
      old (keep=q3 rename=(q3=q1))
      old (keep=q4 rename=(q4=q1));
run;

proc freq;
  table q1;
run;
```

Two other methods

First, we may simply use the proc freq by eliminating the view in YH's method. We included this one in our comparison. In addition, we can have the following two more methods.

(1). It is much more common to use proc freq rather than proc chart to obtain frequency results. The appealing point to use proc chart is that it can give both numeric and graphic results at the same time. First concatenate the separated column based datasets after appropriate dataset options keep/rename implemented. It is noted that this will increase the records size by number of variables minus one. Then use proc chart.

```
data combine;
  set old (keep=q1)
      old (keep=q2 rename=(q2=q1))
      old (keep=q3 rename=(q3=q1))
      old (keep=q4 rename=(q4=q1));
run;

proc chart data=combine;
  hbar q1;
run;
```

(2). If-Then (select)

Now we use if-then block to replace the inner do-end block in DN's array method. This needs a counter initialization process when doing the first iteration and a retain statement to keep the counters values current. The do-end iterations for k are used for later comparison.

```
Data old2;
   infile cards eof=eof;

   input index q1 $ q2 $ q3 $ q4 $;
            array q[4]q1-q4;
            retain a1 b1 c1 d1 e1 f1;
            if _n_=1 then do;
                        a1=0;b1=0;c1=0;d1=0;e1=0;f1=0;
                        end;
            do k=1 to 200000;
              do i=1 to 4;
                        if q[i]='A' then a1=a1+1;
                        else if q[i]='B' then b1=b1+1;
                        else if q[i]='C' then c1=c1+1;
                        else if q[i]='D' then d1=d1+1;
                        else if q[i]='E' then e1=e1+1;
                        else if q[i]='F' then f1=f1+1;
              end;
            end;
            keep a1 b1 c1 d1 e1 f1;
            return;
            eof: output;
   datalines;
 1 A B C D
 2 E F A E
 3 C B B A
 4 B A D E
 5 E F A B
 6 A A A C
 7 F E A E
 ;
run;

 Proc print data=old2;
 run;
```

Similarly, we can replace if-then block with select-end block to achieve the same goal.

```
Data old3;
   infile cards eof=eof;
   input index q1 $ q2 $ q3 $ q4 $;
   array q[4]q1-q4;
   retain a1 b1 c1 d1 e1 f1;
   if _n_=1 then do;
                        a1=0;b1=0;c1=0;d1=0;e1=0;f1=0;
                        end;

   do k=1 to 200000;
          do i=1 to 4;
             select(q[i]);
                when('A')a1=a1+1;
                when('B')b1=b1+1;
                when('C')c1=c1+1;
                when('D')d1=d1+1;
                when('E')e1=e1+1;
                when('F')f1=f1+1;
                otherwise;
             end;
          end;
   end;
   keep a1 b1 c1 d1 e1 f1;
   return;
```

```
   eof: output;
datalines;
 1 A B C D
 2 E F A E
 3 C B B A
 4 B A D E
 5 E F A B
 6 A A A C
 7 F E A E
 ;
run;

Proc print data=old3;
run;
```

Compared the array based method with if-then/select-end block based methods, if-then block makes several comparisons and therefore uses most time, and the array based method uses the least due to its internal computation mechanism. The following test confirms this.

Performance Comparison

We test the above methods in terms of CPU time using the expanded data above after including do-end iterations for k. Therefore, the data contains 1.4 million records in total. Our operating system is Windows Professional Version 2002 with Service Pack 2. Our SAS version is 9.1.3. The result is listed in the table below. Time is in seconds.

	Transpose	Array	JC	Summary	View+freq	Freq	If-then	Select	Chart
Time	4.02	0.32	0.86	2.35	2.54	1.84	1.14	1.06	2.2

We can see that the differences are huge. The slowest method (transpose based) uses as 12 times time as the fastest one (array based) consumes for this 1.4 million records example. Most time spent in transpose based method is in transpose part, while most time spent in summary based method is in summary parts. The use of view is with a cost of saving it first and retrieving it later, especially it takes a lot of time when retrieving it. Proc freq based one saves the retrieving time by removing the view. Proc chart displays more information while using more time. Among all data steps, the second data step in proc freq based uses more time since it put 1.4 million records together. This tells that when processing dataset the best way is not to increase the record size.

Conclusions

Based on a frequency distribution question posted at SAS-L community, this paper first reviewed the methods suggested by SAS-L subscribers, and then proposed two other methods. Also, we made a performance comparison in terms of CPU time usage among these methods while solving an expanded example. From this comparison we can learn something even in some unpopular method, e.g., proc summary. The result may serve as a guideline when solving similar problems. For other possibilities, we may expect that the HASH/SQL guys will have faster solutions and we look forward to them.

References

http://www.listserv.uga.edu/cgi-bin/wa?S2=sas-l&q=&s=a+better+way+to+collapse+variables?
SAS OnlineDoc 9.1.3, SAS Institute Inc., Cary, NC, USA.

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

Yubo Gao
Department of Orthopaedics and Rehabilitation
University of Iowa Hospitals and Clinics
200 Hawkins Dr.
Iowa City, Iowa 52242
yubo-gao@uiowa.edu
(319)356-1674