Using SAS® software to shrink the Data used in Apache Flex® Application
Ahmed Al-Attar, AnA Data Warehousing Consulting LLC, McLean, VA

Abstract
This paper discusses the techniques I used at the Census Bureau to overcome the issue of dealing with large amount of data while modernizing some of their public facing web applications by using Service Oriented Architecture (SOA) to deploy SAS powered Flex web applications. Techniques that resulted in reducing 142,293 XML lines (3.6 MB) down to 15,813 XML lines (1.8 MB) a 50% size reduction on the server side (HTTP Response), and 196,167 observations down to 283 observations, a reduction of 99.8% in summarized data on the client side (XML Lookup file).

Introduction
When the Dissemination Internet Staff (DIS) team at the Census Bureau decided to update some of their existing web applications and modernize them, they had to decide on a technology that could

- Integrate with SAS/IntrNet® in phase I
- Get embedded in HTML page
- Operate without Java Application Server
- Integrate with SAS Enterprise Business Intelligence (EBI) Platform in phase II

Based on the above, Adobe/Apache Flex® application framework was selected to provide the required Rich Interactivity, and Integration with SAS® 9.2 through Web Services standard protocols such as REST and SOAP.

While the client application (Flex) can communicate with SAS via submitting either HTTP request or SOAP envelop over HTTP, the response (data) is always returned as XML stream. That’s where we started to have issues when dealing with large data!!

SAS continues to enhance and simplify the creation of web services and data conversion to XML via the XML Libname engine. Here is an example of exporting a SAS data set from one library to XML document in another directory/library.
Using SAS® software to shrink the Data used in Apache Flex® Application, continued

LIBNAME sesug14 XML92 "C:\Projects\SESUG14";

PROC COPY IN=work OUT=sesug14;
  SELECT sample;
RUN;

The resulting XML would look like this:

```
<?xml version="1.0" encoding="windows-1252"?>
<TABLE>
  <SAMPLE>
    <CODE>AA</CODE>
    <GEOAREA>Not specified</GEOAREA>
    <REFDATE>1994</REFDATE>
    <SHEET>Female Prostitute</SHEET>
    <POPCODE>P</POPCODE>
    <SEXF>Female</SEXF>
    <AGE>18+</AGE>
    <SOURCEID>DN178</SOURCEID>
    <SAMPLESIZE>131,991</SAMPLESIZE>
    <#Obs>131,991</#Obs>
    <#Columns>23</#Columns>
    <#Lines>3,299,778</#Lines>
  </SAMPLE>
</TABLE>
```

Examining the resulted output indicated “Total Resulted lines” = \((#Obs \times #Columns) + (#Obs \times 2) + 3\)

<table>
<thead>
<tr>
<th>Table Name</th>
<th>#Obs</th>
<th>Size in MB</th>
<th>#Lines</th>
<th>Size in MB</th>
<th>Real time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGF2014.SAMPLE</td>
<td>131,991</td>
<td>120</td>
<td>3,299,778</td>
<td>125</td>
<td>56.68</td>
</tr>
</tbody>
</table>

Note: While different SAS data set sizes and structures could result in different storage footprint and ratios, the formula used to generate the XML lines always the same!

That’s where this formula started causing issues especially when we had to transfer the resulted XML data across the network. While all the examples illustrated in the referenced papers used this standard XML output format, we had to find an alternative. Upon further investigation and research, I discovered Flex’s ability to consume XML Attributes

---

1 XML Attribute: “A markup construct consisting of a name/value pair that exists within a start-tag or empty-element tag”. http://en.wikipedia.org/wiki/XML
Using SAS® software to shrink the Data used in Apache Flex® Application, continued

FILENAME smplxml "C:\Projects\SESUG14\sample_cust.xml";

%util_reformatTable(p_inDsName=work_r.sample, p_newFormat=xml, p_outFileRef=smplxml, p_dispNumObs_yn=N , p_numObs=, p_dispByteSize_yn=N, p_byteSize=);

The resulting XML would look like this:

Here is how this new approach compares to the XML Libname engine.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>SAS Data Set</th>
<th>XML92 Engine</th>
<th>Custom XML</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Obs</td>
<td>Size in MB</td>
<td>#Lines</td>
</tr>
<tr>
<td>SGF2014.SAMPLE</td>
<td>131,991</td>
<td>120</td>
<td>3,299,778</td>
</tr>
<tr>
<td></td>
<td>131,994</td>
<td>78</td>
<td>131,991</td>
</tr>
<tr>
<td>SGF2014.SAIPESD</td>
<td>196,167</td>
<td>120</td>
<td>1,373,172</td>
</tr>
<tr>
<td></td>
<td>196,170</td>
<td>15.2</td>
<td>196,170</td>
</tr>
<tr>
<td>SGF2014.SAIPESN_COUNTRY</td>
<td>56,528</td>
<td>120</td>
<td>282,643</td>
</tr>
<tr>
<td></td>
<td>56,531</td>
<td>3</td>
<td>56,531</td>
</tr>
</tbody>
</table>

Now that I have got this issue addressed on the server I had to deal with large data on the client side.

2 XML Element: “A logical document component either begins with a start-tag and ends with a matching end-tag or consists only of an empty-element tag”. http://en.wikipedia.org/wiki/XML
All deployed Flex web applications

- Follows the same layout which comprised of two parts
  - Data Filters Part: Two or more data selectors widgets with data driven XML lookup files
  - Data Viewer Part: Data visualization widget, such as Data Grids, Charts and Maps
- Performed client side data selection validations to avoid Zero result sets returned.
  This proved to be troublesome and required some attention and thinking outside of the box!

I have always used Proc Summary/Proc Means to figure out the unique combinations of variables values, but when the final number of unique combinations exceeds the hundreds and starts to range in the thousands, tens of thousands, and hundreds of thousands, it starts to fail to load during the client application initialization, which in turn causes the application to crash!!

Here are few examples of the variable combinations I had to deal with

<table>
<thead>
<tr>
<th>Variables Set</th>
<th># Unique Combinations</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year, State, County</td>
<td>56,528</td>
<td>Counties vary from one year to another</td>
</tr>
<tr>
<td>Year, State, School District</td>
<td>196,167</td>
<td>School District vary from one year to another</td>
</tr>
<tr>
<td>Year, State, County Flag, Age Category, Race Category, Gender Category, Income Category</td>
<td>71,196</td>
<td>Certain category values vary across years</td>
</tr>
</tbody>
</table>

I had no chance loading such amounts of unique combinations, and even if I could, processing them at run time, would have resulted in a very unsatisfactory user experience!

I had to find an alternative approach to the traditional OLAP approach in order to reduce the size of the combinations without affecting the integrity of the data and the relationship amongst the values of the variables. This is where the power of the SAS language came to the rescue, and provided me with straight forward processing techniques allowed me to achieve my goal.

Having
- SAS supports long character strings (32,767 chars)
- SAS provides first. & last. processing
- All the variables I had to deal with have relatively short code values
- Custom developed data combinations class in Adobe ActionScript

Allowed me to transpose and collapse particular variable values into a single space delimited string, and find unique combinations based on the newly created string value.

The following screen shots illustrate the data transformation.
Using SAS® software to shrink the Data used in Apache Flex® Application, continued

Standard Proc Summary output data set with 196,167 observations.

By transposing the Year values while maintaining the State and School District values, I got the following output:

This resulted into reducing 196,167 observations down to 14,772 observations.
Taking this one step further, by transposing School District values while maintaining the State and Year gave me this output:

```
This resulted in reducing 14,772 observations down to 283 observations only. Bingo!
```

Applying the same techniques against the other combination tables yielded the following results:

<table>
<thead>
<tr>
<th>Variables Set</th>
<th># Unique Combinations</th>
<th># Unique Combinations after Transposing Values</th>
<th>%Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year, State, County</td>
<td>56,528</td>
<td>62 [64 XML lines -25K]</td>
<td>99.89%</td>
</tr>
<tr>
<td>Year, State, School District</td>
<td>196,167</td>
<td>283 [285 XML lines-134K]</td>
<td>99.86%</td>
</tr>
<tr>
<td>Year, State, County Flag, Age Category, Race Category, Gender Category, Income Category</td>
<td>71,196</td>
<td>328 [330 XML lines-84K]</td>
<td>99.54%</td>
</tr>
</tbody>
</table>

With combinations in such low numbers, we were able to maintain optimum application initialization and run time processing.
Conclusion

Working with large data sets often requires adoption of alternative techniques beyond compression and other standard out of the box functionalities provided by SAS.

I would strongly encourage the reader to think outside of the box and find ways to innovate. After all, developing custom solutions can sometime be frustrating and demanding, but when they work, they can be very rewarding.

References


Contact Information

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

Ahmed Al-Attar
AnA Data Warehousing Consulting, LLC.
McLean, VA 22101
Cell Phone: 703-477-7972
E-mail: ahmed.al-attar@anadwc.com
Web: www.anadwc.com

Trademarks

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.
APPENDIX 1

%MACRO shrinkCombos(
    p_inDsName= /* Input Source Data Source */
   , p_outDsName= /* Output Data Set */
   , p_byClause= /* Classification Variables Set */
   , p_classVar= /* Class Variable to Transpose */
   , p_prntClassVar= /* Preceding Class Variable */
   , p_aggrVarLen=10 /* Variable length of the new transposed values */
);

    /* Summarize & Sort */
    PROC SUMMARY DATA=&p_inDsName NWAY;
        CLASS &p_byClause;
        OUTPUT OUT=&p_outDsName;
    RUN;

    /* Process the data and generate the transposed value */
    DATA &p_outDsName(DROP=&p_classVar RENAME=(aggr=&p_classVar));
        SET &p_inDsName;
        BY &p_byClause;
        LENGTH aggr $&p_aggrVarLen ;
        RETAIN aggr ;
        IF (FIRST.&p_prntClassVar) THEN
            aggr='';
        aggr = catx(' ',aggr,&p_classVar);
        IF (LAST.&p_prntClassVar) THEN
            OUTPUT;
    RUN;
    %MEND shrinkCombos;

    /* Usage Examples */
    %shrinkCombos( p_inDsName=saipe.Saipeschldstrct, p_outDsName=work.saipesd_combo
       , p_byClause=%str(state district year), p_classVar=year, p_prntClassVar=district, p_aggrVarLen=120);

    /* Max District Count by State = 1045, --> 1045*8=8400 */
    %shrinkCombos( p_inDsName=work.saipesd_combo, p_outDsName=work.saipesd_combo2
       , p_byClause=%str(state year district), p_classVar=district, p_prntClassVar=year, p_aggrVarLen=8400);

    filename saipesd "C:\Projects\SESUG14\saipesd.xml";

    /* Export to XML*/
    %util_reformatTable(p_inDsName=work.saipesd_combo2, p_newFormat=xml, p_outFileRef=saipesd
       , p_dispNumObs_yn=N, p_numObs=, p_dispByteSize_yn=N, p_byteSize=);