

Parallel Process Querying on Multisource Retail Data

David Malerich
Best Buy Inc.
Richfield, MN

Presented to Twin Cities SAS Users Group *T.C.A.S.U.G.*
September 14, 2009



Marketing



- Finding Customers
- Identifying their Needs

- Communicating Solutions
- Measuring Effectiveness

Overview

- Batch programs can run for hours if required data for analysis is coming from multiple sources which are queried sequentially
- If no underlying reason exists for sequential querying, parallel processing can cut execution time significantly

Topics highlighted

- Writing subprograms
- Creating and executing script file
- Utilization of work folder for dataset cleanup via the [getoption](#) function and command-line parameter passing

System Requirements

- Ability to utilize parallel processing is dependent on your computing system
 - Batch SAS program execution
 - Background execution
 - Scripting environment
 - Command-line parameter passing
- Best Buy – SAS 9.1.3 in a SunOS UNIX environment with KSH scripting

Setting up the parallel process – Step 1 – Review your existing program

Example Code – parent program.sas

```
%let userid=xxxxxxx;  
%let password=yyyyyyyy;  
  
proc sql;  
  connect to oracle (user=&userid. password="&password." path=myspath);  
  create table work.custdata1 as  
  select * from connection to oracle  
  (  
    select * from source_table_1  
  );  
  create table work.custdata2 as  
  select * from connection to oracle  
  (  
    select * from source_table_2  
  );  
  disconnect from oracle;  
quit;  
  
data work.combined_data;  
  merge work.custdata1 work.custdata2;  
  by customer_id;  
run;
```

These two CREATE TABLE statements are going to be run sequentially. But do they really need to be, or can we run them in parallel to reduce execution time?

Setting up the parallel process – Step 2 – Create subprograms



sub_program 1.sas

```
%let userid=xxxxxxx;
%let password=yyyyyyy;
libname temp './tempdir';
proc sql;
    connect to oracle (user=&userid. password="&password." path=myspath);
    create table temp.custdata1 as
    select * from connection to oracle
    (
        select * from source_table_1
    );
quit;
```

sub_program 2.sas

```
%let userid=xxxxxxx;
%let password=yyyyyyy;
libname temp './tempdir';
proc sql;
    connect to oracle (user=&userid. password="&password." path=myspath);
    create table temp.custdata2 as
    select * from connection to oracle
    (
        select * from source_table_2
    );
quit;
```

Okay, now we have the queries set up in subprograms. How do we get them to run from the parent program before the DATA step merge?

Setting up the parallel process – Step 3 – Revising the parent program

Example Code – parent_program.sas

```
%let userid=xxxxxxx;
%let password=yyyyyyy;
libname temp './tempdir';
%sysexec(rm ./child_script*);
%put ***** system return code from rm command: &sysrc;
%put &sysmsg;

data _null_;
  file './child_script' mod; * script file *;
  put '#!/usr/bin/ksh';
  put './opt/bbc/bin/t47bbyenv.ksh -a cir -e prod';
  put 'export SASV8_CONFIG=/data/ci/mktinfo/sasv8.cfg';
  put "sas sub_program_1.sas %nrstr(&)";
  put "sas sub_program_2.sas %nrstr(&)";
  put "wait";

run;
** make the script executable;
%sysexec(chmod ug+x ./child_script);
** execute the script;
%sysexec(noexec ./child_script);

data work.combined_data;
  merge temp.custdata1 temp.custdata2;
  by customer_id;
```

Add a LIBNAME statement to read the output of your subprograms

Delete the previous version of the script before writing new

Create a text file that will act as a script.

Make sure the shell realizes it's a script.

Make sure the system's environment variables are set.

Launch the subprograms in the background

Wait for subprograms to finish before completing script execution.

Make the script an executable file and launch it.

Parallel process dataset management

– Utilize your WORK directory

- One of the downsides of using parallel processing is that datasets from the child programs have to be written to “permanent” directories to be accessed by the parent program.
- **Or do they?**
- Through the `getoption` function and parameter passing, we can keep the output datasets from the child program in our temporary WORK directories.

Parallel process dataset management – More revisions to the parent program

Example Code – parent_program.sas

```
%sysexec(rm ./child_script*);  
%put ***** system return code from rm command: &sysrc;  
%put &sysmsg;  
  
%let workdir=%sysfunc(getoption(work));  
  
data _null_;  
  file "./child_script" mod; * script file *;  
  put '#!/usr/bin/ksh';  
  put './opt/bbc/bin/t47bbyenv.ksh -a cir -e prod';  
  put 'export SASV8_CONFIG=/data/ci/mktinfo/sasv8.cfg';  
  put "sas -sysparm &workdir.sub_program_1.sas %nrstr(&)";  
  put "sas -sysparm &workdir.sub_program_2.sas %nrstr(&)";  
  put "wait";  
run;
```

Assign the location of the parent program's WORK directory to a macro variable

Pass the value of the macro variable to the child programs through the SYSPARM option

Parallel process dataset management – Using the passed parameter



sub_program 1.sas

```
%let userid=xxxxxxx;  
%let password=yyyyyyy;  
libname temp "&sysparm."; ←  
proc sql;  
  connect to oracle (user=&userid. password="&password." path=myspath);  
  create table temp.custdata1 as  
  select * from connection to oracle  
  (  
    select * from source_table_1  
  );  
quit;
```

sub_program 2.sas

```
%let userid=xxxxxxx;  
%let password=yyyyyyy;  
libname temp "&sysparm."; ←  
proc sql;  
  connect to oracle (user=&userid. password="&password." path=myspath);  
  create table temp.custdata2 as  
  select * from connection to oracle  
  (  
    select * from source_table_2  
  );  
quit;
```

Now we change the children programs so that the location of the output datasets will take that passed parameter. That puts the datasets in the WORK directory of the parent program!

Parallel process dataset management – Back to the parent one more time



Now that the child programs are writing to the parent program's WORK directory, we can change our DATA step MERGE statement back.

Example Code – parent_program.sas

```
%let userid=xxxxxxx;  
%let password=yyyyyyy;  
libname temp './tempdir';  
  
.  
  
.  
  
.  
  
.  
  
.  
  
data work.combined_data;  
    merge work.custdata1 work.custdata2;  
    by customer_id;  
run;
```

We no longer need the LIBNAME for this permanent directory for the child program output datasets...

... and we can reference those datasets as coming from the WORK library!

Thank You!

David Malerich
Lead Analyst – Marketing Services
Best Buy Inc.
david.malerich@bestbuy.com

Acknowledgement

Thank you to Rick Langston from SAS Institute for his tip on using the SYSPARM command line option. That trick has been invaluable on a large number of SAS programs!