Paper 1826-2014

Enhancing SAS® Piping Through Dynamic Port Allocation

Piyush Kumar Singh, TATA Consultancy Services Ltd. Indianapolis, IN.

Gerhardt M Pohl, Eli Lilly and Company, Indianapolis, IN.

ABSTRACT

This paper explains how a port (in Linux SAS environment) can be chosen automatically for SAS® parallel processing on remote server. Using traditional method we can assign a fixed port number for parallel processing. However, that restricts user's flexibility to choose other available port, if the assigned port is being used by some other service/process. This paper will describe how a Linux shell script can be called within SAS code to ascertain available ports that can be further used in SAS code for parallel processing.

INTRODUCTION

Multi-process Connect® (MP CONNECT®) is a powerful feature provided by SAS for parallel processing. It not only helps in processing SAS Programs in parallel but also reduces overall execution time. To use MP CONNECT on remote server, we need to mention the port in the SAS Libname statement. We can give the fixed port in a Libname statement but sometimes it becomes confusing as the given port might be used by some other service/ process. Hard coding of port number in SAS code may lead to port conjunction and even job failures (following error message may appear on the log file: *ERROR: Communications error on pipe*) until other jobs are not executed completely. This problem increases with the increase in user count.

With automation of port selection, users are given the flexibility to select any available (not being used by any other process) port from a given pool of ports. By utilizing an unused port, users can work independently without being impacted by other user's jobs.

PIPELINE PARALLELISM

Pipeline parallelism, an extension of MP CONNECT, is an effective way to speed processing. Piping allows the typical programming sequence of DATA step followed by PROC to be executed in parallel rather than in sequence. In traditional sequential processing, the PROC would not initiate until the entire DATA step is complete. Furthermore, the results of the DATA step would typically be stored in a temporary SAS file increasing the system input/output (I/O) load, since the data is both written to disk by the DATA step and read back from disk for the PROC.

Instead, piping uses TCP ports to pass individual records directly from the DATA step to the PROC as soon as processing of each individual record is complete. The DATA step in effect becomes a data transformation filter for the PROC, running in parallel, consuming no additional disk space, and incurring less I/O lag. Establishing a pipe with MP CONNECT typically requires specifying a physical (hardware) TCP port number to be used by the writing and the reading processes. Coding in this style is somewhat dangerous as users may inadvertently request ports required by other services on the machine; or ironically, multiple users using the same code template with an explicit port numbers may conflict with each other generating system errors and having their sessions aborted.

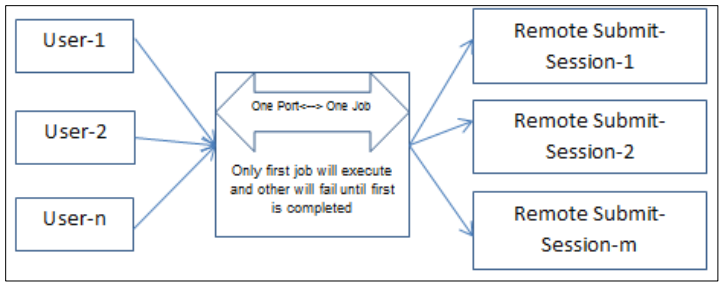

Figure 1. Multiple users establishing pipe using same or any random port

Implementing the SAS metadata server allows one to allocate ports dynamically, that is, users can use a symbolic name for the port with the SAS server thus dynamically determining an unused port to be temporarily assigned to the SAS job. While this capability is very attractive, running SAS metadata server on a system which does not use any of other SAS BI technology can be inefficient from a cost perspective. To enable dynamic port allocation without the added cost, we created a Linux script that can be called from within SAS to ascertain which ports are available at a given point of time. The script returns a list of available ports which are captured in a SAS macro variable and subsequently used in establishing pipeline parallelism.

Below is the piece of SAS code which is calling the *"/export/home01/sasadm/port_check.sh"* Linux shell script. This script is placed on Linux SAS servers to fetch available ports and sent it remotely to SAS code. Complete shell script is given on next page.

```
filename ps_list pipe "/export/home01/sasadm/port_check.sh";

data Port_Check;
infile ps_list truncover;
  length Av_Port $5;
  input Av_Port $5.;
run;

data _null_;
set Port_Check;
  call symput('Avail_Port',Av_Port);
run;

%put "&Avail_Port";
%let Port_Macro=%sysfunc(compress(&Avail_Port));
%put "&Port_Macro";
```

This script gets executed when the SAS code runs and fetches the available ports. The complete shell script is given below.

This step captures the ports returned by the above Linux script into a SAS dataset.

This step converts the available ports into a SAS macro variable to be used in the next SAS steps.

This step displays the available port in the log.

```
16    %let Port_Macro=%sysfunc(compress(&Avail_Port));
17    %put "&Port_Macro";
"4026"
```

Port **4026** is available for use

The above log message shows that port 4026 was selected as an available port which can be further used in SAS code.

The complete Linux shell script (*port_check.sh*) is given below.

```
#!/bin/bash

KILL_ALL='no'
RETURN_HOW_MANY=10

DEBUG='no'
END=1
if [ "${DEBUG}" == 'yes' ];then
    END=10;
fi

if [ "${KILL_ALL}" == 'yes' ]; then
    if [ "${DEBUG}" == 'yes' ];then
        pgrep -fl "nc -l";
        pkill -f "nc -l";
    else
        pkill -f "nc -l" >/dev/null 2>&1;
    fi
fi
## RETURN_HOW_MANY=$((RETURN_HOW_MANY+1));
for n in $(seq ${END});do
    int=0;
    for a in {4015..4080};do

        if [ "${KILL_ALL}" == 'yes' ]; then
            pkill -f "nc -l" >/dev/null 2>&1;
        fi

        is_open=$(nmap -p ${a} localhost 2>/dev/null | awk -v a=${a} '{if($1~a) print
$2}')
        if [ "${is_open}" == 'closed' ];then
            if ! pgrep -fl "nc -l ${a}" >/dev/null 2>&1;then
                nc -l ${a} & echo ${a} |nc localhost ${a};
                if [ $? -eq 0 ];then
                    pkill -9 -f "nc -l ${a}" >/dev/null 2>&1;

                    int=$((int+1))
                    if [ "${RETURN_HOW_MANY}" -le "${int}" ];then
                        if [ "${DEBUG}" == 'yes' ];then
                            echo ${int};
                        fi
                        break;
                    fi
                fi

                if [ "${DEBUG}" == 'yes' ];then
                    pgrep -fl "nc -l";
                fi

                sleep 0.5;
            fi
        fi
    done;

    if [ "${DEBUG}" == 'yes' ];then
        echo ---- ${n} -----;
    fi

done;

if [ "${KILL_ALL}" == 'yes' ]; then
```

1 - END parameter is for checking the number of available ports in a given slot.

2 - In this case user needs 10 available ports, but it doesn't mean all will be used.

3 - Given range for availability check. In this example 'given range' is containing ports that are already used by other services.

```
    pkill -f "nc -l" >/dev/null 2>&1;
fi
```

In the given sample code ports 4017 and 4018 are already being used by other services on the server. So these ports are being discarded by the script and are not being captured. The following output Window depicts a SAS table that has been created using the output from the Linux script as input. This dataset contains the ports given by the Linux script. There is no row corresponding to 4017 and 4018.

Note: The scope of the Linux script has been limited to searching for a total of 10 available ports at a time.

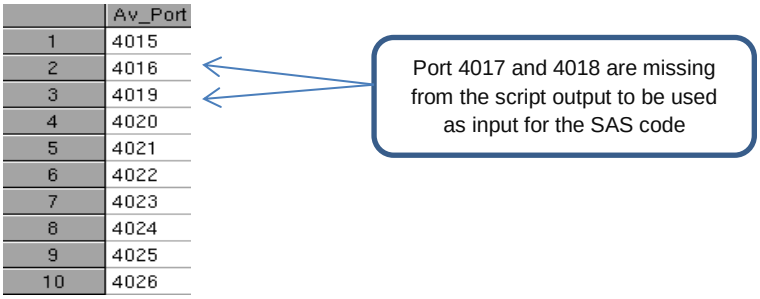

Figure 2. Enhancing SAS Piping Through Dynamic Port Allocation

Given below is the sample code that shows how the available port from shell script has been captured into SAS macro variable and resolved at run time. In this sample code, only one port was selected to be used in SAS Libname statement.

The macro variables highlighted in ‘yellow’ displays the port number flow from shell script to the SAS code.

```
filename ps_list pipe
"/…/test/port_check.sh";

 data Port_Check;
    infile ps_list truncover;
    length Av_Port $5;
    input Av_Port $5.;
 run;


 data _null_;
 set Port_Check;
 call symput('Avail_Port',Av_Port);
 run;

 %put "&Avail_Port";

 %let
Port_Macro=%sysfunc(compress(&Avail_Por
t));
 %put "&Port_Macro";

 /*** Pass the last available port to
below pipe-port      */


 %let Port_Pip=%str(&Port_Macro);
 %put &Port_Pip;

 signon p1 sascmd="!sascmd";
 %syslput Port_Pip=&Port_Pip;
 %put &Port_Pip;
 rsubmit wait=no;
    libname outlib sasesock
":&Port_Pip";
    data outlib.pipe1;
       do i=1 to 500;
          a='abc';
          x=i*2;
          output;
       end;
    run;
endrsubmit;

libname inlib  sasesock ":&Port_Pip";
data x;
  set inlib.pipe1;
run;

waitfor p1;
rget p1;
signoff p1;
```

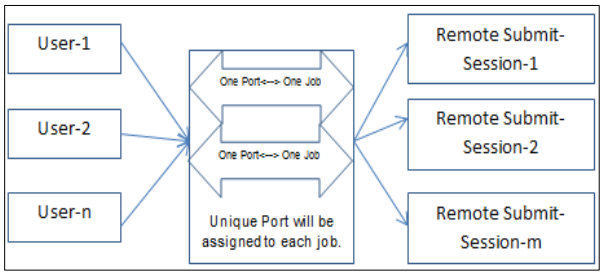

Figure 3. Parallel processing using unique unused port

CONCLUSION

To conclude, piping through dynamic port allocation for SAS parallel processing not only provides flexibility to select any available (not being used by any other process) port from a given pool of ports but also reduces users overthinking regarding port availability on SAS servers.

With the change of parameter in the shell script, multiple ports can be fetched in SAS code and can be used if SAS code needs more than one port. Thus, depending on user's requirement adjustments can be made in the shell script to obtain desired output.

ACKNOWLEDGMENTS

I would like to thank my friends Ed Johnstone and Ritika Singh for their support and valuable inputs.

REFERENCES

http://en.wikipedia.org/wiki/Parallel_computing
http://en.wikipedia.org/wiki/Parallel_processing

CONTACT INFORMATION

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

Piyush Kumar Singh

TATA Consultancy Services Ltd.

639 S Delaware St.

Indianapolis, IN 46285

317-487-9139

piyushkumar.singh@tcs.com