SCSUG-2016

SASGSUB for Job Workflow and SAS Log Files

Piyush Singh, Prasoon Sangwan
TATA Consultancy Services Ltd. Indianapolis, IN

ABSTRACT

SAS® Grid Manager Client Utility (SASGSUB) is one of the key clients for the users to use SAS® Grid platform. SASGSUB is standard SAS® Grid Client Utility which may not fulfill every use case/business need. This paper describes few techniques to enhance SASGSUB capabilities with the help of shell scripting to enhance the default functionality of SASGSUB. It explains how to achieve the business needs like creating Job workflows for example to execute SAS jobs after the completion of set of running jobs, copy SAS log files from GRIDWORK to specific location without deletion from default location (deletion of GRIDWORK files lead to SASGSUB fail to check the job's status) etc.

INTRODUCTION

SAS® Grid Manager Client Utility (SASGSUB) is one of the key clients for users to use SAS® Grid platform. It plays a vital role in program submission, execution and monitoring. Though it supports ad -hoc executions but is primarily used for batch mode. However, SASGSUB is a standard utility and provides a standard set of functionalities to Grid user. But these functionalities can be well exploited to achieve the business needs.

In this paper, we will discuss two of such techniques to exploit SASGSUB functions to customize to some business needs.

- Execute multiple jobs in sequence.
- Execute a job after finishing the set of jobs.
- Copy SAS Log from Grid work without deleting from SASGRIDWORK directory

Before discussing further, let's discuss the different phases of job execution. These terms will be used frequently in this paper. A job passes through different phases of execution:

1. *Submitted*: - In this state, the job is submitted to grid, but it is still waiting for a compute node for execution. During this period job waits for LSF to assign a specific node for execution. Once compute node will be assigned, status will change to "Running".

2. *Running* - At this state, the job has been assigned to a compute node and has started execution. Until the execution is completed (with or without error), status will remain as "Running".

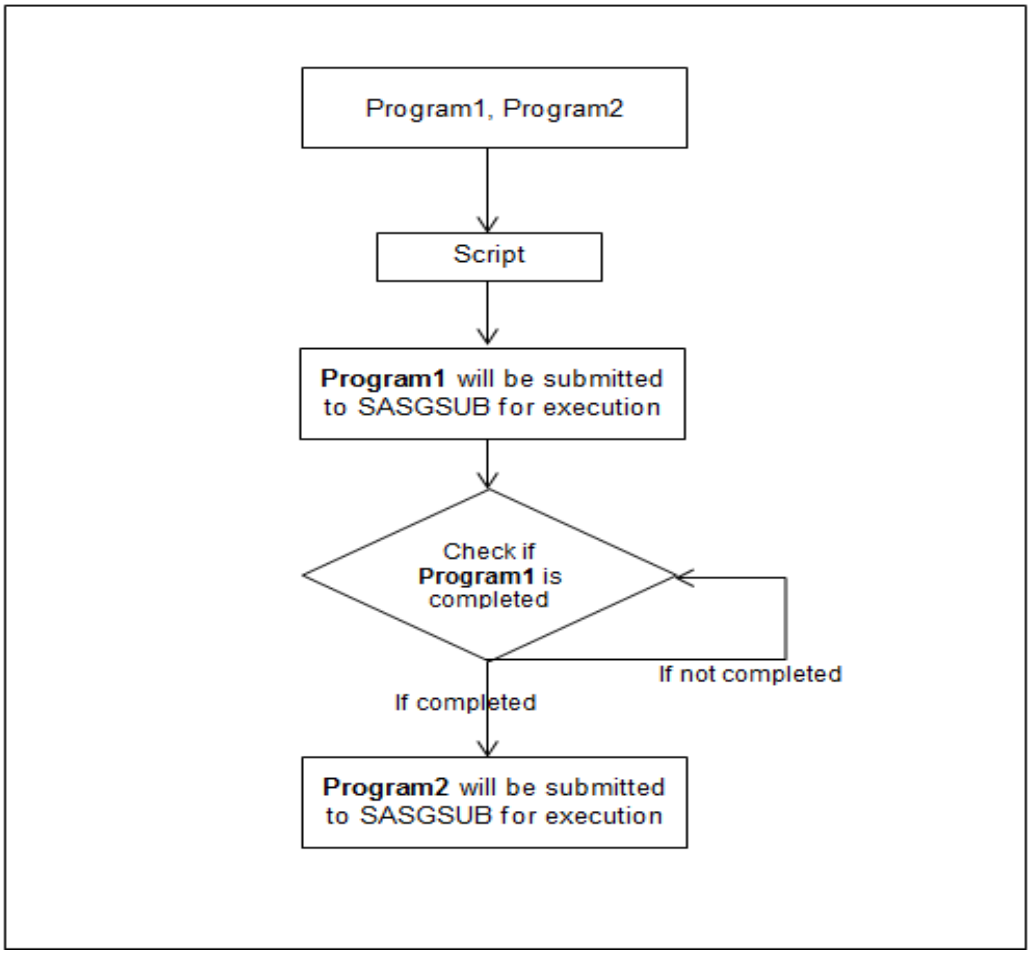
3. *Finished* - When the job status is 'Finished'. it signifies that the job execution has been completed. Finished status means job is completed but it is not suggesting whether it's finished with or without error. The status of a successful completion of a program is verified through RC (Return code) of the output. Any non- zero value of RC signifies that the program has been executed with errors.

1. EXECUTE MULTIPLE JOBS IN SEQUENCE

If we need to run multiple SAS programs one after another in a sequence, then we can create a wrapper the script with SASGSUB. Wrapper script should be written in such a way that it should wait until first program is completed successfully. It will continue to check the status and once the first job is completed,

script will automatically trigger the next job. This facility helps users not to manually wait for job completion for submitting next dependent job.

Below script notify the users if first job was executed successfully and next was submitted. In case, if first job failed then there is no point of triggering next job. In this case the user will be notified that first job got failed and next is not submitted. Below is the flow chart of wrapper script (given at next page) and shows how the script executes at backend. Below script is created to execute two programs in sequence but if users want to run multiple jobs in sequence then this can be further enhanced with same approach.



Below is the wrapper script using SASGSUB. As shown in the below example, the script needs to be executed with two parameters. First parameter will be the first program which needs to be executed before start of next program as second parameter. This script can be further enhanced to increase the functionalities or to change the sequence of execution. There are four main sections in this script as given below:

1. This section contains the variables which need to be changed as per current installation and use by the users. "`sasgsubdir`" is the location where SASGSUB is located and is being executed from. "`execloc`" is temporary location which contains the temp files used during the execution.

2. This submits, the first program using `gridsubmitpgm` and reads the job id from the details returned.

3. We will check the status of the first program through `gridgetstatus` until status gets changed from "`Submitted`" and "`Running`". Here we are checking the execution status every 60 sec. This can be changed as per environment and business need.

4. If status changed to "`Finished`" it will submit the next program and sends an email to user. If status changed to "`Failed`", it will stop the execution and notify user.

```
#!/bin/bash
# Section : 1
sasgsubdir=/sasgsub_root/SASGridManagerClientUtility/9.4
execloc=/top/usr2455/

# Section : 2

$sasgsubdir/sasgsub -gridsubmitpgm $1 > $execloc /wrkflow/nthline.txt
jobstr=$(sed -n '4p' $execloc/wrkflow/nthline.txt | awk '{ print $2 }')
jobid=${jobstr:1:5}
$sasgsubdir/sasgsub -gridgetstatus $jobid > $execloc/wrkflow/status.txt
status=$(sed -n '4p' $execloc/wrkflow/status.txt | awk '{ print $4 }'
| tr -d ":")

# Section : 3

while [ "$status" == "Running" ] || [ "$status" == "Submitted" ]
do
sleep 60
$sasgsubdir/sasgsub -gridgetstatus $jobid > $execloc/wrkflow/status.txt
status=$(sed -n '4p' $execloc/wrkflow/status.txt | awk '{ print $4 }' | tr -d
":")

# Section : 4
if [ "$status" = "Finished" ]; then
$sasgsubdir/sasgsub -gridsubmitpgm $2 > $execloc/wrkflow/nthline2.txt
jobstr2=$(sed -n '4p' $execloc/wrkflow/nthline2.txt | awk '{ print $2 }')
jobid2=${jobstr2:1:5}

echo "The first job <Job ID : $jobid> is completed and the second job <Job ID
: $jobid2> has been submitted."
elif [ "$status" = "Failed" ]; then
echo "The first job <Job ID : $jobid> has Failed, hence the second job is not
submitted for execution."
exit
fi
done
```

Working Example:

```
./wrkflow_job1_job4 "/top/usr2455/cwdtest2.sas" "/top/usr2455/test1.sas"

The first job <Job ID : 25704> is completed and the second job <Job ID :
2805> has been submitted.
. . .
```

2. EXECUTE A JOB AFTER FINISHING THE SET OF JOBS

This section explains, if we need to run a program after finishing a set of jobs. It will wait until all running job get finished. Once all given jobs are finished, it will trigger the next job and notify the user that specified job has been submitted. It will also notify the job id for new job to the user, so that the user can check the status of job. There are main 4 parts of below script as explained below:

1. This is the initial declaration section, which contains the variables that need to be changed as per current installation which will be used in execution. Here we are creating the temporary directory where the temporary files used in this script are stored. `'loc'` variable defines the location of the created temporary directory. `'jobstatus'` is a variable used in this script to assign the status of the input job IDs. `'SASGSUBDIR'` is the location where SASGSUB is located and is being executed from. The LSF profile is also being sourced in this section so that LSF commands can be executed in the script.

2. Here all the validations occur. Initially the Job IDs and the SAS program are separated to different temporary files. This section will provide error messages to the user in scenarios, if the user doesn't provide any input job IDs or SAS program, missing SAS program, missing Job IDs, invalid Job IDs, invalid program, etc.,

3. After the validation step, the status of each input Job IDs is being checked in this section. It checks the jobs which are in "`Running`", "`Submitted`", "`Failed`", "Finished"., etc. status, If the jobs are still running and in either "Submitted" or "Running" status, it waits until all the jobs get completed. If any of the Jobs fail, it shows the message to user that one of the jobs failed and exits without submitting the SAS program for execution.

4. In this final section the SAS program is submitted for execution once all of the input jobs are executed successfully. Also the temporary directory which was created in the first step and the temporary files are deleted. If any of the validation fails above, these temporary directory and temporary files are deleted in the validation step itself after showing the error message to the user.

```
#!/bin/bash
# Section : 1

if ! [ -d /top/usr2455/SASGRIDSCH ]; then
mkdir /top/usr2455/SASGRIDSCH/
chmod 777 /top/usr2455/SASGRIDSCH/
fi
loc=/top/usr2455/SASGRIDSCH
jobstatus=/top/usr2455/SASGRIDSCH/jobinput
SASGSUBDIR=/sasgsub_root/SASGridManagerClientUtility/9.4
. /LSF_TOP/conf/profile.lsf

# Section : 2

if [ "$1" != "" ]; then
for i in $*; do
echo $i
done > $loc/userinputs
cat $loc/userinputs | grep -i "/" >  $loc/proglist
cat $loc/userinputs | grep -v "/" >  $loc/jobids
else
```

```
echo "Missing input parameters. Provide Job IDs & SAS program(Full path) for
execution!"
exit
fi

if ! [ -s $loc/proglist ]; then
echo "Missing input parameters. Provide SAS program(Full path)."
rm $loc/proglist $loc/jobids $loc/userinputs
rm -r /top/usr2455/SASGRIDSCH
exit
fi

if ! [ -s $loc/jobids ]; then
echo "Missing Job IDs. Proceeding with SAS program execution."
fi

for n in `cat $loc/jobids`; do
bhistout=$(bhist -l $n)
if [ "$bhistout" == "No matching job found" ];
then
touch $loc/invalidjobs
echo -n "$n " >> $loc/invalidjobs
fi
done
if [ -f $loc/invalidjobs ]; then
echo "Job IDs <`cat $loc/invalidjobs`> are invalid. Enter Valid Job IDs."
rm $loc/proglist $loc/jobids $loc/userinputs $loc/invalidjobs
rm -r /top/usr2455/SASGRIDSCH
exit
fi

for z in `cat $loc/proglist`; do
fileext=$(echo $z | sed 's/^.*\(....\)$/\1/')
if [ -r $z -a -f $z ]
then
echo "$z is readable" > /dev/null
else
echo "<$z> is either an invalid file or is not readable/accessible."
exit
fi
if [ $fileext != .sas ]; then
echo "<$z> is invalid. Enter a valid SAS program."
exit
fi
done

# Section : 3
for n in `cat $loc/jobids`; do
$SASGSUBDIR/sasgsub -gridgetstatus $n | tail -n +4 |
awk '{print $4}' done >  $loc/jobinput
while :
do
if egrep -q "Submitted|Running" "$jobstatus"; then
sleep 30
for m in `cat $loc/jobids`; do
$SASGSUBDIR/sasgsub -gridgetstatus $m | tail -n +4 |
awk '{print $4}' done >  $loc/jobinput
```

```
else
break
fi
done


for a in `cat $loc/jobids`;do
$SASGSUBDIR/sasgsub -gridgetstatus $a | tail -n +4 |
awk '{print $1, $2, $3, $4}'
done > $loc/failedjob

if grep -q Failed "$loc/failedjob"; then
grep Failed "$loc/failedjob"
echo "One or more of the input jobs failed. SAS program has not been
submitted."
rm $loc/proglist $loc/jobids $loc/validjob $loc/jobinput $loc/failedjob
rm -r /top/usr2455/SASGRIDSCH
exit
else

# Section : 4

for f in `cat $loc/proglist`; do
$SASGSUBDIR/sasgsub -gridsubmitpgm $f
done > /dev/null
echo "SAS program has been submitted for execution."
fi

rm $loc/proglist $loc/jobids $loc/jobinput $loc/failedjob
rm -r /top/usr2455/SASGRIDSCH
```

Working Example:

```
 $./submitgsub.sh 28382 3647 2903 56842 "/top/usr2455/test.sas"

 SAS program has been submitted for execution.
. . .
. . .
```

3. COPY SAS LOG GRIDWORK DIRECTORY

Many times users want to see the SAS log file where they are running SAS program or some other location. SAS Grid execution generates the log file but goes to standard SASGRIDWORK directory and sometime it becomes difficult for users to navigate SASGRID directory after each job execution. There are some SASGRID options to fetch the SAS log file but the problem is, it moves the log file from SASGRIDWORK which will stop –GRIDGETSTATUS option to fetch the status of the specified job. Below script provides the flexibility for user to copy the log file from SASGRIDWORK directory without deleting. In this way –GRIDGETSTATUS will still be running perfectly fine. Below are the key parts of script:

1. This is the initial declaration section which contains the variables that need to be changed as per current installation which will be used by the users. Here we are creating the temporary directory where the temporary files used in this script are stored. `'loc'` variable defines the location of the created temporary directory. `'SASGSUBDIR'` is the location where SASGSUB is located and is being executed from. The LSF profile is also being sourced in this section so that LSF commands can be executed in this script.

2. All the validations occur here. Checking whether the Job ID and the destination location where the log should be copied are entered correctly. After this next validation is for user if there is write access to the destination location or not. If there is no write access, then script exits and notify the user.

3. After the validation step, the status of input Job ID is being checked in this section. It checks, whether the input job is in "`Running`", "`Submitted`", "`Finished`" status. If the job is still running and in either "Submitted" or "Running" status, it waits until the job get completed. If completed, it will move to the next section.

4. In this final section the SASGSUB log is copied to the destination location, the input job has been finished. Also the temporary directory which was created in the first step and the temporary files, are deleted at the end of execution.

```
#!/bin/bash

# Section : 1
if ! [ -d /top/usr2455/SASGRIDSCH ]; then
mkdir /top/usr2455/SASGRIDSCH/
chmod 777 /top/usr2455/SASGRIDSCH
fi

. LSF_TOP/conf/profile.lsf
loc=/top/usr2455/SASGRIDSCH/
SASGSUBDIR=/sasgsub_root/SASGridManagerClientUtility/9.4

# Section : 2
if [ "$1" = '' ]; then
echo "Job Id cannot be left blank."
exit 1;
fi
job_check=$(bhist -l $1)
if [ "$job_check" = 'No matching job found' ]; then
echo "Job doesn't exist."
exit 1;
fi
if [ "$2" = '' ]; then
echo "Location cannot be left blank."
exit 1;
elif ! [ -d "$2" ]; then
echo "Please Enter a valid destination location."
exit
elif ! [ -w $2 ]
then
echo "There is no write permission to destination location."
exit
fi
```

```
# Section : 3
while
do
status=$($SASGSUBDIR/sasgsub -gridgetstatus $1 | tail -n +4
| awk '{print $4}' | tr -d ":")
if [[ "$status" = "Submitted" || "$status" = "Running" ]];
then
sleep 30
else
break
fi
done

# Section : 4
bhist -l $1 | paste -d, -s | tr -d " ," | awk -F "'" '$0=$4' > $loc/path
cp `cat $loc/path`/*.log $2
echo "The Log file has been copied."
rm -rf /top/usr2455/SASGRIDSCH/
```

Working Example:

```
./gsublogcopy.sh 54383 "/top/usr2455"

The Log file has been copied.
. . .
```

CONCLUSION

SASGSUB is a very powerful client to use SAS Grid perhaps there are some limitations. But we can create different kind of wrapper script with SASGSUB to achieve or to increase the functionalities. The scripts given in this paper gives idea and approach. Users can further work on these scripts and modify as per their need and requirements. This will make SASGSUB even more powerful client for SAS Grid.

ACKNOWLEDGEMENT

Authors would like to thank Lisa Mendez, SCSUG 2016 Educational Forum Co-Chair, for all her great help and support during this conference.

REEFERENCE

https://en.wikipedia.org/wiki/SAS_(software)

"Key Requirements for SAS® Grid Users" Proceedings SAS Global Forum 2016, Las Vegas, NV, USA. http://support.sas.com/resources/papers/proceedings16/7140-2016.pdf

"Integrating SAS® and the R Language with Microsoft SharePoint" Proceedings SAS Global Forum 2015, Dallas, TX, USA.
https://support.sas.com/resources/papers/proceedings15/2500-2015.pdf

"Enhancing SAS® Piping Through Dynamic Port Allocation" Proceedings of SAS Global Forum 2014, Washington, DC, USA.
http://support.sas.com/resources/papers/proceedings14/1826-2014.pdf

CONTACT INFORMATION

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

Piyush Singh
piyushkumar.singh@tcs.com

Prasoon Sangwan
prasoon.sangwan@tcs.com