SAS® GLOBAL FORUM 2020 #SASGF

Paper 4764-2020

Managing "IDLE" Grid-Launched SAS® Workspace Servers

Greg Wootton, SAS Institute Inc., Cary, NC.

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

ABSTRACT

In SAS Grid Computing, jobs may be running for hours or even days. It can be difficult to determine if these jobs are active or idle. Batch jobs like those submitted using the SAS Grid Manager Client Utility have their status accurately reflected by the Load Sharing Facility (LSF) as batch jobs do not wait for input during execution. Grid-Launched Workspace Servers however, will appear in a RUNNING status to LSF and its associated commands such as `bjobs`, despite sitting idle waiting on a user's input.

The code and techniques given in this paper will describe how to check for how long a Grid-Launched Workspace Server has been "idle", waiting on user input. Based on the idle time of the SAS session, a decision can be made to kill the respective LSF job id to free up the job slots from SAS Grid Platform. This technique can be even more helpful for the user who forgets to close their SAS® Enterprise Guide session before leaving for the day, keeping a job slot busy throughout the night.

INTRODUCTION

The SAS Grid platform uses the concept of job slots to control the number of concurrently running jobs available to a single user, group, queue or host. Based on the resource, the platform administrator configures the job slots available with an integer value. At any time, this value is the maximum number of processes which can be executed concurrently in the SAS Grid environment. As a lack of available job slots prevents users from accessing the application, it is important for a platform administrator to manage the platform so that the available job slots are being used efficiently as intended.

This paper explains the key reasons a job can occupy a job slot for an extended period, as well as providing SAS code to check the idle time for each job running in the SAS Grid, and how the platform administrator can manage this kind of job.

REASONS JOB SLOTS MAY BE OCCUPIED

There are many reasons a job may hold its job slot for a long time. If a batch job is particularly time consuming, it's expected for the job to occupy the slot until the execution is completed. In many cases however, jobs occupy the slot for hours or days, irrespective of the code processing. In these situations, it's vital to identify the reason for the delay and take the appropriate action to make the job slot available again to prevent impact on other users.

Here are a few reasons why jobs might occupy a slot unnecessarily for a long time:

1. HUNG JOBS

A hung job refers to a job process that is running but is unable to end normally. This could be due to unexpected user input, an infinite loop in the code, abruptly terminated jobs and more.

A platform administrator should check the SAS Grid platform regularly to find out if there are such jobs running.

The **bjobs -u all** command can be used to view a list of currently running jobs, along with their submission time. This can give context on how long a job has been running.

```
[sas@trcv003 ~]$ bjobs -u all
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
8413    sassrv  RUN   normal     trcv003.trc trcv003.trc *62B367672 Dec  7 13:26
8414    sassrv  RUN   normal     trcv003.trc trcv003.trc *7455A56E4 Dec  7 13:26
9301    sassrv  RUN   normal     trcv003.trc trcv003.trc *37956C50C Jan  3 13:33
```

Output 1. List All Grid Jobs

Once a job has been identified as long-running, the **bjobs -l <job_id>** command can be used for additional information on that specific job, including when it was submitted and by which user, as well as what command was submitted to the Grid for execution.

```
[sas@trcv003 ~]$ bjobs -l 8413

Job <8413>, Job Name <SASApp - Stored Process Server_04117130-61C8-3A40-92D1-1C
                     962B367672>, User <sassrv>, Project <default>, Status <RUN
                     >, Queue <normal>, Command </sas94/config/compute/Lev1/SAS
                     App/StoredProcessServer/StoredProcessServer.sh -noterminal
                      -netencryptalgorithm SASProprietary -metaserver trcv003.t
                     rc.sas.com -metaport 8561 -metarepository Foundation -obje
                     ctserver -objectserverparms "sph=trcv003.trc.sas.com proto
                     col=bridge spawned spp=41442 cid=509 pb classfactory=15931
                     E31-667F-11D5-8804-00C04F35AC8C server=OMSOBJ:SERVERCOMPON
                     ENT/A5RWD7FM.AZ00000B cel=credentials lb grid multiuser po
                     rt=8621 saslangrunas=client" -METAUSER '"sassrv@!*(generat
                     edpassworddomain)*!"' -METAPASS 5D2E6E0aFD854843F7F4d74E7e
                     d92545 >, Share group charged </sassrv>
Fri Dec  7 13:26:37: Submitted from host <trcv003.trc.sas.com>, CWD <$HOME>, Sp
                     ecified Hosts <trcv003.trc.sas.com>;
Fri Dec  7 13:26:37: Started 1 Task(s) on Host(s) <trcv003.trc.sas.com>, Alloca
                     ted 1 Slot(s) on Host(s) <trcv003.trc.sas.com>, Execution
                     Home </home/sassrv>, Execution CWD </home/sassrv>;
```

Output 2. LSF Jobs' Execution Information

From the screen shot above, we can see this example of a long running job is a SAS Stored Process Server. As this server is a persistent process, it is expected to run for a long time.

If from this output the administrator determines the job should be killed, the **bkill** command can be used to submit a kill request to the Grid.

```
[sas@trcv003 ~]$ bkill 8413
Job <8413> is being terminated
```

The **bjobs -l <job_id>** output also provides the associated process IDs for a grid job and its execution host. If for some reason a bkill is not successful, it's possible to perform a SIGTERM or SIGKILL on the process ID itself.

```
[sas@trcv003 ~]$ bjobs -l 9344

Job <9344>, Job Name <SASApp - Stored Process Server_C9F0AC3D-76C6-9B45-8FA4-8C
                     BEB4D4D408>, User <sassrv>, Project <default>, Status <RUN
                     >, Queue <normal>, Command </sas94/config/compute/Lev1/SAS
                     App/StoredProcessServer/StoredProcessServer.sh -noterminal
                      -netencryptalgorithm SASProprietary -metaserver trcv003.t
                     rc.sas.com -metaport 8561 -metarepository Foundation -obje
                     ctserver -objectserverparms "sph=trcv051.trc.sas.com proto
                     col=bridge spawned spp=48764 cid=820 pb classfactory=15931
                     E31-667F-11D5-8804-00C04F35AC8C server=OMSOBJ:SERVERCOMPON
                     ENT/A5RWD7FM.AZ000000B cel=credentials lb grid multiuser po
                     rt=8611 saslangrunas=client" -METAUSER '"sassrv@!*(generat
                     edpassworddomain)*!"' -METAPASS 4628A62B92253518F4C64D2B36
                     ecA813 >, Share group charged </sassrv>
Fri Jan  4 15:48:26: Submitted from host <trcv051.trc.sas.com>, CWD <$HOME>, Sp
                     ecified Hosts <trcv051.trc.sas.com>;
Fri Jan  4 15:48:27: Started 1 Task(s) on Host(s) <trcv051.trc.sas.com>, Alloca
                     ted 1 Slot(s) on Host(s) <trcv051.trc.sas.com>, Execution
                     Home </home/sassrv>, Execution CWD </home/sassrv>;
Mon Jan  7 16:14:15: Resource usage collected.
                     The CPU time used is 628 seconds.
                     MEM: 44 Mbytes;  SWAP: 0 Mbytes;  NTHREAD: 29
                     PGID: 26765;  PIDs: 26765 26766 26768 26882
```

Output 3. LSF Jobs' PIDs

A Grid job spawns a total of at least three sub-processes on the compute nodes, as given below:

1. LSF selects the appropriate host and starts the RES (Remote Execution Server) daemon as a child of the sbatchd persistent process on the execution host.
2. RES in turn spawns a shell session calling the environment defined by RES.
3. The shell session starts the SAS process by executing the provided command.

This tree is illustrated in the output of **`ps faux`** on the execution host.

```
root     27727  0.0  0.0  43564  8220 ?        Ss    2018  21:06 /sas94/pss/lsf/10.1/linux2.6-glibc2.3-x86_64/etc/sbatchd
sassrv   26765  0.2  0.0  31660  7260 ?        S1   Jan04  10:20  \_ /sas94/pss/lsf/10.1/linux2.6-glibc2.3-x86_64/etc/res -d /sas94/pss
sassrv   26766  0.0  0.0 113140  1428 ?        S    Jan04   0:00      \_ /bin/sh /home/sassrv/.lsbatch/1546634906.9344
sassrv   26768  0.0  0.4 1841620 74176 ?       S1   Jan04   0:09          \_ /sas94/software/compute/SASFoundation/9.4/sasexe/sas -note
```

Output 4. PIDs on Compute Nodes

All the above PIDs can be obtained by running "**`bjobs -l <job_id>`**" LSF command.

To kill a hung job properly, the platform administrator can check all these sub-processes corresponding to the SAS Grid job and kill all of them using the **`kill`** command or **`kill -9`** if kill without an argument fails to end the processes.

2. SUDDEN RESTART OF PLATFORM LSF

A sudden restart of platform LSF can lead to a situation where the associations between the jobs and SAS Grid are lost. Mostly the child processes for such lost jobs don't end properly and slots associated with such jobs remain occupied even though jobs have finished.

If your Platform LSF daemons are ended unexpectedly, then the platform administrator should check the jobs running on the Grid and decide if there are any of these pseudo jobs present, then kill them to free the associated Grid slots.

3. INTERACTIVE SAS SESSION - SAS ENTERPRISE GUIDE

In clients like SAS Enterprise Guide, a user does not execute the SAS code continuously as would occur with a batch job. Generally, the user connects the Workspace Server and uses executes code as needed. They execute the piece of SAS code and analyze the result, check the logs, and perform other tasks and may come back again to SAS Enterprise Guide for additional code execution. Once the user has established a SAS Enterprise Guide session with the Workspace Server it holds a job slot, regardless of whether any SAS code is being executed from on session or it is left idle.

This can be a key contributor to idle jobs occupying job slots. It is possible to configure a Workspace Server to end after a period of inactivity but depending on your use case this may not be desirable. The session which is being used intermittently still needs to be active. Before killing an idle SAS session, we may wish to make sure these are safe to kill.

As platform administrator you should find the duration for an idle SAS session and if the duration is long, decide on next steps.

4. INTERACTIVE SAS SESSION - SASGSUB

The SAS Grid Manager Client Utility (SASGSUB), like SAS Enterprise Guide, allows users to start an interactive SAS session on the Grid. Users can use the interactive SAS session in display or inline mode from SAS Grid platform. Like SAS Enterprise Guide, these SAS sessions also hold a job slot. If these SAS sessions are left unused for long time, they also will hold a slot unnecessarily. But like SAS Enterprise Guide, it's not easy to say if such SAS sessions are still in use by users or these are left unused and are safe to be killed.

Platform administrators can check in LSF to see how long such SAS sessions have been running. Based on the business rules or conversations with the user, it can be decided if the job can be killed.

FIND IDLE SAS JOBS IN SAS GRID ENVIRONMENT

To decide on whether to end an idle job, we must first generate a list of idle jobs in the SAS Grid environment. Once we have the list, we can decide on the business rule for ending the idle jobs if these are idle for more than a given time. Before going further into the technical aspect of this, we need to understand the basic concept of SAS Job vs SAS Grid Job.

After a Grid Job is submitted successfully, the Grid daemons spawn a SAS session on a Grid compute node. For application servers like the Workspace Server used by SAS Enterprise Guide, the Object Spawner monitors the session's idle time and its process ID, but not the associated Grid job ID. The Grid job ID will allow us to determine on which node the job is executing. If the user has access to a Grid client, the Grid job ID can be used by users to kill

their SAS sessions. We need to find all SAS sessions and their idle time, as well as associate it with the respective SAS Grid Job ID. Having both the idle time and Grid job ID would be helpful for administrators and business users to end the idle sessions with the help of LSF.

IDLE DURATION FOR SAS SESSION

The Object Spawner maintains an idle counter for Workspace Servers that can be queried for each Workspace Server. The SAS program shown below will query each Object Spawner for the idle time of its spawned servers.

This program attempts to connect to all the Object Spawners defined in the Metadata Server.

- If any of your Object Spawners is not running, the below program will throw an error message.
- The idle_time value is in seconds and refers to how long the Workspace Server process has been waiting for SAS code to be submitted for execution.
- An idle_time value of ZERO means it is currently executing code.
- The report from this code does not limit itself to only idle sessions, it will display all Server sessions spawned by the Object Spawner.
- This program may throw a set of three error messages such as "`Error: Requested function is not implemented.`", if a Workspace Server existed when the program queried the Object Spawner initially but did not find it when requesting its attributes.

```
/* Part 1. ************************************************** */
/* Program to extract all workspace servers and associated Idle   */
/* Time values.                                                   */
/* ************************************************************** */

/* Metadata connection information: */

%let metaserve=meta.demo.sas.com;
%let metaport=8561;
%let userid=sasadm@saspw;
%let pass=password;

/* End edit. */

/* Connect to Metadata Server */

options    metaserver="&metaserve"
           metaport=&metaport
           metauser="&userid"
           metapass="&pass"
           metarepository=Foundation
           metaprotocol=BRIDGE;

data work.objspawn;

    keep host_name port; /* Only keep hosts and port for Object Spawners. */
    retain port; /* Keep port for all iterations. */

    /* Declare and initialize variables. */

    length type id objspwn_uri tree_uri mach_uri host_name conn_uri port $ 50;
    call missing(of _character_);
```

```
    /* This is the XML Select query to locate Object Spawners. */
    obj="omsobj:ServerComponent?@PublicType='Spawner.IOM'";

    /* Test for definition of Object Spawner(s) in Metadata. */

    objspwn_cnt=metadata_resolve(obj,type,id);
    if objspwn_cnt > 0 then do n=1 to objspwn_cnt;

    /* Get URI for each Object Spawner found. */

           rc=metadata_getnobj(obj,n,objspwn_uri);

           /* Get associated attributes for the object spawner (connection port and
hosts) */

           rc=metadata_getnasn(objspwn_uri,"SoftwareTrees",1,tree_uri);
           rc=metadata_getnasn(objspwn_uri,"SourceConnections",1,conn_uri);
           rc=metadata_getattr(conn_uri,"Port",port);
           mach_cnt=metadata_getnasn(tree_uri,"Members",1,mach_uri);

           /* For each host found, get the host name and output it along with the
port number to the dataset. */

           do m=1 to mach_cnt;
                  rc=metadata_getnasn(tree_uri,"Members",m,mach_uri);
                  rc=metadata_getattr(mach_uri,"Name",host_name);
                  output;
           end;
    end;
    else put "No Object Spawners defined in Metadata.";
run;

/* WORK.OBJSPAWN now contains a list of hosts running Object Spawners. */

/* Macro below will query each host for the Workspace Servers it has spawned. */

%macro getwkspc;

/* Count how many Object Spawners are defined in WORK.OBJSPAWN as a Macro variable.
*/

proc sql noprint;
    select count(*) into :nobjs from work.objspawn;
quit;

%if &nobjs > 0 %then %do; /* If hosts were found, extract them as macro variables.
*/

proc sql noprint;
    select host_name into:host1-:host%left(&nobjs) from work.objspawn;
    select port into:port1-:port%left(&nobjs) from work.objspawn;
quit;

%end;
%else;

/* Create base tables. */
data work.wkspc;
length SERVERCOMPONENT LOGICALNAME $50 SERVERCLASS PROCESSOWNER SERVERID $36;
call missing(of _character_);
    if compress(cats(of _all_),'.')=' ' then delete;
run;
```

```
data work.wkspcidle;
length SERVERCOMPONENT LOGICALNAME $50 SERVERCLASS PROCESSOWNER SERVERID $36
CATEGORY NAME $ 1024 VALUE $ 4096;
call missing(of _character_);
    if compress(cats(of _all_),'.')=' ' then delete;
run;

/* Connect to each object spawner to get the workspace servers it has spawned,
output them to a dataset. */

%do i=1 %to &nobjs;
    proc iomoperate;
           connect host="&&host&i"
                        port=&&port&i
                        user="&userid"
                        pass="&pass"
                        servertype=OBJECTSPAWNER;
           LIST SPAWNED SERVERS out=wkspc&i;
    quit;

    /* Count the number of total workspace servers were found. */

    proc sql noprint;
           select count(*) into :nwkspc from work.wkspc&i;
    quit;

    /* If any were found, add them to the wkspc dataset. */

    %if &nwkspc > 0 %then %do;
    proc sql;
           insert into work.wkspc
           select * from work.wkspc&i;
    quit;
    %end;

    /* If any were found, gather their IdleTime value. */

%if &nwkspc > 0 %then %do j=1 %to &nwkspc;

proc sql noprint;
    select SERVERID into:server_id1-:server_id%left(&nwkspc) from work.wkspc&i;
quit;
    proc iomoperate;
           connect host="&&host&i"
                        port=&&port&i
                        user="&userid"
                        pass="&pass"
                        servertype=OBJECTSPAWNER
                        spawned="&&server_id&j";
                  LIST ATTRIBUTE Category="Counters" Name="IOM.IdleTime"
out=work.wkspci&j;
    quit;

    /* Add the server ID to the table containing the idle time. */

    data work.wkspci&j;
           set work.wkspci&j;
           server_id="&&server_id&j";
    run;

    /* Join the spawned servers table for the spawner with the idle time. */

proc sql noprint;
```

```
    create table work.wkspcidle&j as select * from work.wkspc&i,work.wkspci&j where
SERVERID=server_id;
quit;

/* Append the new table of server and idle time to a master table. */

    proc sql;
           insert into work.wkspcidle
           select SERVERCOMPONENT, LOGICALNAME, SERVERCLASS, PROCESSOWNER, SERVERID,
CATEGORY, NAME, VALUE from work.wkspcidle&j;
    quit;


    %end;
%end;
%mend;

%getwkspc;

/* Convert the idle time value to a number. */

data work.final;
    set work.wkspcidle;
    keep SERVERCOMPONENT LOGICALNAME SERVERCLASS PROCESSOWNER SERVERID idle_time;
    idle_time=input(VALUE,8.2);
run;


proc print data=work.final; run;
```

Obs	SERVERCOMPONENT	LOGICALNAME	SERVERCLASS	PROCESSOWNER	SERVERID	idle_time
1	SASApp - Stored Process Server	SASApp - Logical Stored Process Server	15931E31-667F-11D5-8804-00C04F35AC8C	sassrv	9B6E312D-9357-4E49-9AE5-A1837956C50C	419156.2
2	SASApp - Stored Process Server	SASApp - Logical Stored Process Server	15931E31-667F-11D5-8804-00C04F35AC8C	sassrv	0AEAA22F-7EFB-8145-B63C-95B7455A56E4	2147520.0
3	SASAppVA - Stored Process Server	SASAppVA - Logical Stored Process Server	15931E31-667F-11D5-8804-00C04F35AC8C	sassrv	BD335134-3BAA-0345-89A2-A89BF5E9C3AB	419491.3
4	SASAppFM - Stored Process Server	SASAppFM - Logical Stored Process Server	15931E31-667F-11D5-8804-00C04F35AC8C	sassrv	C501D8D2-1B47-7844-8989-9BB598E728D5	419446.5
5	SASApp - Stored Process Server	SASApp - Logical Stored Process Server	15931E31-667F-11D5-8804-00C04F35AC8C	sassrv	3B97A1D7-9349-B84B-B048-458E51ABF01D	419178.1
6	SASApp - Stored Process Server	SASApp - Logical Stored Process Server	15931E31-667F-11D5-8804-00C04F35AC8C	sassrv	C9F0AC3D-76C6-9B45-8FA4-8CBEB4D4D408	325561.5
7	SASApp - Workspace Server	SASApp - Logical Workspace Server	440196D4-90F0-11D0-9F41-00A024BB830C	sasdemo	F8D4C876-4141-0945-B517-35FCDCFD0646	0.0

Output 5. SAS Sessions from PROC IOMOPERATE

SAS GRID JOB ID (LSF ID) FOR SAS WORKSPACE SERVERS

PROC IOMOPERATE is the SAS procedure used in the previous section to find the SAS execution related attribute, but LSF Job ID is not an attribute that the Object Spawner can provide through PROC IOMOPERATE. We need some additional SAS coding to associate the LSF job ID with the SAS session from the above SAS code. This would require having SAS parse the output of LSF command **`bjobs`** for the server ID, which the Object Spawner includes in the job name.

A list of all idle SAS sessions launched by the Object Spawner can be merged with the list of jobs from LSF command **`bjobs`** with the help of job name and the server ID. This will provide the list of all jobs launched by Object Spawner and the corresponding Grid job ID, and business users can use these IDs to kill their respective SAS sessions. The Platform Administrator can also take the appropriate action based on the list of such idle jobs to make sure that Grid slots are not occupied unnecessarily.

***Note** - This code's output is limited to idle SAS sessions which are submitted by the SAS® Enterprise Guide client.*

SAS CODE FOR IDLE GRID JOBS

By extending the above program to pull idle time from the Object Spawner with the below code, we create a variable that can be used to match the **bjobs** output with the results of the above program to create a report of the job ID, user, status, queue, job name, idle time, server name, submission host, and execution host.

```
/* Part 2. ************************************************** */
/* To alter the job name from Part1, so that it can be matched    */
/* with LSF job name to retrive the respective LSF JobID.         */
/* ********************************************************** */
data work.clientname;
    set work.final;
    client='SAS Enterprise Guide';
run;

data work.jobname;
    set work.clientname;
    length jobname $ 200;
    jobname = catx('_', client, SERVERCOMPONENT, serverid);
run;

proc sort data=work.jobname out=egjobs;
    by jobname;
run;

/* Part 3. ************************************************** */
/* This portion of code is running "bjobs" LSF command to fetch   */
/* the job id for Workspace Server Session from Part1 of this     */
/* program.                                                       */
/* ********************************************************** */

filename jobsa94 pipe "bjobs -u all -a -w";

data jobsa94;
    infile jobsa94 firstobs=2 dlm=" " truncover;
    length job_id $20. user $20. status $20. queue $20. sub_server $20.
ex_server $20. jobname1 $50. jobname2 $50.
          jobname3 $50. jobname4 $50. jobname5 $50. jobname6 $50. month
$30. day $30. time $30.;
    input job_id $ user $ status $ queue $ sub_server $ ex_server $
jobname1 $ jobname2 $
          jobname3 $ jobname4 $ jobname5 $ jobname6 $ month $ day $ time $;

    *if substr(sasgsub_job_status,1,1)="S" then delete;
run;

proc sort data=jobsa94;
    by user;
run;

data jobsa942;
    set jobsa94;

    if status = 'RUN' and queue = 'normal' and jobname1 = 'SAS';
```

```
run;

data egfrm_lsf (keep= job_id user status queue sub_server ex_server
jobname);
    set jobsa942;
    length jobname $ 100;
    jobname = catx(' ', jobname1, jobname2, jobname3, jobname4, jobname5,
jobname6);
run;

/* Merge the LSF data with data WorkspaceServer session from Part1. */
proc sort data=egfrm_lsf;
    by jobname;
run;

data wrksp_lsfid (drop=serverclass processowner serverid servercomponent
client);
    merge egfrm_lsf (in=a) egjobs (in=b);
    by jobname;

    if a;
run;

data wrksp_lsfid;
    retain job_id user status queue jobname idle_time logicalname
sub_server ex_server;
    set wrksp_lsfid;
run;

proc sort data=wrksp_lsfid;
    by idle_time;
run;

title "All Active Workspace Server Sessions and Corresponding LSF JobId.";

proc print data=wrksp_lsfid;
run;

title;
```

All Active Workspace Server Sessions and Corresponding LSF JobId.

Obs	job_id	user	status	queue	jobname	idle_time	logicalname	sub_server	ex_server
1	9476	sasdemo	RUN	normal	SAS Enterprise Guide_SASApp - Workspace Server_2E6FA7DD-D880-E94D-8EC0-94C445D1678D	0.000	SASApp - Logical Workspace Server	trcv003.trc.sas.com	trcv090.trc.sas.com
2	9477	sasdemo	RUN	normal	SAS Enterprise Guide_SASApp - Workspace Server_98288FED-6900-1A49-90DD-68533C1AFE11	12.194	SASApp - Logical Workspace Server	trcv003.trc.sas.com	trcv003.trc.sas.com

Output 6. SAS® Enterprise Guide Sessions with LSF Job ID

CONCLUSION

In this paper, you have seen how we can use the Object Spawner's spawned server information to identify idle SAS sessions, as well as associating those servers with LSF grid job ids. This empowers the user or administrator to both identify potential problems and take action on them using LSF commands.

REFERENCES

Sangwan, Prasoon, Tanuj Gupta, and Piyush Singh. "Key Requirements for SAS® Grid Users." In *SAS Global Forum*. 2016.
https://support.sas.com/resources/papers/proceedings16/7140-2016.pdf

Singh, Piyush, Sangwan, Prasoon, and Randolph, Steven. "Read SAS® Metadata in SAS® Enterprise Guide" In *SAS Global Forum*. 2017.
http://support.sas.com/resources/papers/proceedings17/1275-2017.pdf

Sangwan, Prasoon, Tanuj Gupta, and Piyush Singh. "Key Requirements for SAS® Grid Users." In *SAS Global Forum*. 2016.
https://support.sas.com/resources/papers/proceedings16/7140-2016.pdf

RECOMMENDED READING

- *SAS® 9.4 Intelligence Platform: Application Server Administration Guide*
- *Base SAS® Procedures Guide*
- *SAS® For Dummies®*

CONTACT INFORMATION

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

Greg Wootton
greg.wootton@sas.com
www.sas.com

Piyush Singh
piyushkumar.singh@tcs.com
www.tcs.com