HOF JUNE 22, 2016

# COURSE OF PBS ACTIONS AND WEB BASED MONITORING VIA PHP

MASTER THESIS

MASTERS IN SOFTWARE ENGINEERING FOR INDUSTRIAL APPLICATIONS

SUBMITTED TO: PROF.DR.MR.MARTIN THOST

SUBMITTED BY:SHRUTI MAHESH KULKARNI

MATRIKELNUMMER:00369013

## Abstract

Most recent developments, such as high performance and parallel computing offerings are having rapidly growing requirements for homogeneous cluster solution approaches. High Performance Computing adds power to computation by providing a cluster solution for industrial applications using simulations, modeling, analysis in industries such as pharmaceuticals, CAD/CAM, ecommerce, life sciences, meteorology,aerospace,oil and gas exploration.Small and medium sized company businesses have a necessity for coordination between their data hence they need to be connected together with a centralized cluster. This centralized cluster brings all together by virtually or physically close such that it is easy to manage, monitor and account the users. It help the users to perform calculations, simulations,etc.

The project is being developed for the enterprise users where job schedulers are widely used to handle the calcutions in manufacturing various product designs. This application is utilized where there is a necessity of high performance performance.

On server side the most basic requirement of the system is to install or configure PBS Environment such that the web application understands the commands to let it perform client operations. The PuTTY interface is just an medium to communicate with the server and execute the commands installed on server.

The solution is to provide the PBS functionalities to the users on web based monitoring application taking into consideration the authentication and authorization of the user. It is simpler for user to perform PBS operations and also will no longer need another interface such as PuTTY to perform PBS operations.

The myJAM application developed in University of Düsseldorf is referred for basic implementation of the project. Though the concept of implementation is similar but the ideology of implementation makes the difference. The concept of web based monitoring via PHP is new in using of functionality for different PBS commands and application helps user by no more wasting timing in getting knowledge on PBS commands to perform operations on jobs.

## Abbreviation

| | |
|---|---|
| PBS | Portable Batch System |
| MOM | Message Oriented Middleware |
| HTTP | HyperText Transfer Protocol |
| TCP | Transmission Control Protocol |
| IP | Internet Protocol |
| LAMP | Linux Apache MySQL PHP |
| MySQL | Structured Query Language |
| AJAX | Asynchronous JavaScript and XML |
| URI | Universal Resource Identifier |
| URL | Universal Resource Locator |
| HTML | Hyper Text Markup Language |
| CSS | Cascading Style Sheets |
| JSON | JavaScript Object Notation |
| JS | JavaScript |
| API | Application programming interface |
| Config | Configuration |

## List of figures

## List of tables

## Preface

On this note I would like to thank and express deep sense of gratitude to my guide Prof.Dr Martin Thost for his consistent guidance, inspiration and sympathetic attitude throughout the study over thesis. I also owe you for your understanding and patience when everytime I required support.

I owe my a big thanks to my friends, family and colleagues for supporting me throughout my life and inspiring me to achieve my objectives in life.

I would like to thank all the people who have provided their valuable time in listening to my ideas and guided me finding a thesis work.

## Glossary

Linux- Linux Redhat is the most preferable operating system used in enterprise because the platform is built in such a way that it can handle enterprise workloads

Apache- Apache HTTP Server is opensource and most widely used web servers. Apache HTTP server is most important component for hosting a web application. The Apache webserver using the http protocol works with client and server architecture.

MySQL- The information is stored relational database tables in database. The database view concepts is used for merging different database tables entities as per the retrival of data

PHP- PHP is well known language for web based application and compatible with various databases

PBS- the portable batch system (PBS) plays a vital role as a resource management system.

AJAX- AJAX which is used to create asynchronous web applications on client side development which can send and retrieve server data asynchronously.

JSON- JSON is language independent and lightweight text based interchange format so it used with PHP and AJAX.

Css- Cascading Style Sheets is used to writ markup language as base to HTML.

HTML- HyperText Markup Language used to create web pages and can be embedded in PHP

SSH- Secure Shell provides a secure channel over unsecured network

PuTTY- PuTTy is a free SSH,Telnet and Rlogin client for Windows system.

## Table of Contents

# Chapter 1

## 1 Introduction

Most recent developments, such as high performance and parallel computing offerings are having rapidly growing requirements for homogeneous cluster solution approaches. High Performance Computing adds power to computation by providing a cluster solution for industrial applications using simulations, modeling, analysis in industries such as pharmaceuticals, CAD/CAM, ecommerce, life sciences, meteorology,aerospace,oil and gas exploration.

Small and medium sized company businesses have a necessity for coordination between their data hence they need to be connected together with a centralized cluster. This centralized cluster brings all together by virtually or physically close such that it is easy to manage, monitor and account the users. It help the users to perform calculations, simulations,etc.

Basically, the high performance computer comes into picture and plays a curial role by connecting the user worstations with network. On the other hand, organizing it centrally with a web based software application. The platform required for such kind of should consist of fast access, with strong functionality carring the feature of avaliablity to the user. Suppose, a user performs operations then the user accepts to have high accessibilty hence it should be a fast process not leting the user to wait for each job.

In case the feature of availability to the user, the operations should switch within different servers with no time consumption.The cluster provides a solution to mass production processing networks by the means of parallelization of web based applications present on the server. Batch systems used to distribute the application uniformly over the resources of the cluster.The submission of user process is nothing but the user jobs to the batch system hence, the PBS i.e portable batch system acts as a batch system whereas the web based application is the PHP web application deployed on Linux Redhat.

Linux Redhat is the most preferable operating system used in enterprise because the platform is built in such a way that it can handle enterprise workloads. Linux if several lines of code in various modules in parts. RedHat creates a single functional system by integrating the modules which is useful for handling enterprise workload.

To handle the management and monitoring of the computational workload on clusters, the portable batch system (PBS) plays a vital role as a resource management system. To this management system the users submit jobs where they are queued up until the system is ready to run them. PBS selects which job to run and decides when and where to run the job in order to balance competing user needs and to increase efficient use of the cluser resources.

A control script file which includes the script to execute the operation is submitted to the PBS server by a PBS command to run on the HPC cluster. The control script is a shell script containing the set of commands which the user wants to run on cluster computer nodes. The system that allows usage of PBS actions like job controlling (qsub,qdel,qrls) and job monitoring (qstat,qselect) on web based monitoring platform so that it is user friendly by automating via PHP.

The PBS system processes the user process by standard specified commands which are known as the PBS commands are also are called as standard TORQUE commands.The user calculations are written in a program and the TORQUE commands are used at the shell command line that runs the user program.For an instance, we can say that the user performs some calculations and the user performs the submission of user process i.e submission of job by TORQUE command *qsub*. To be more simpler

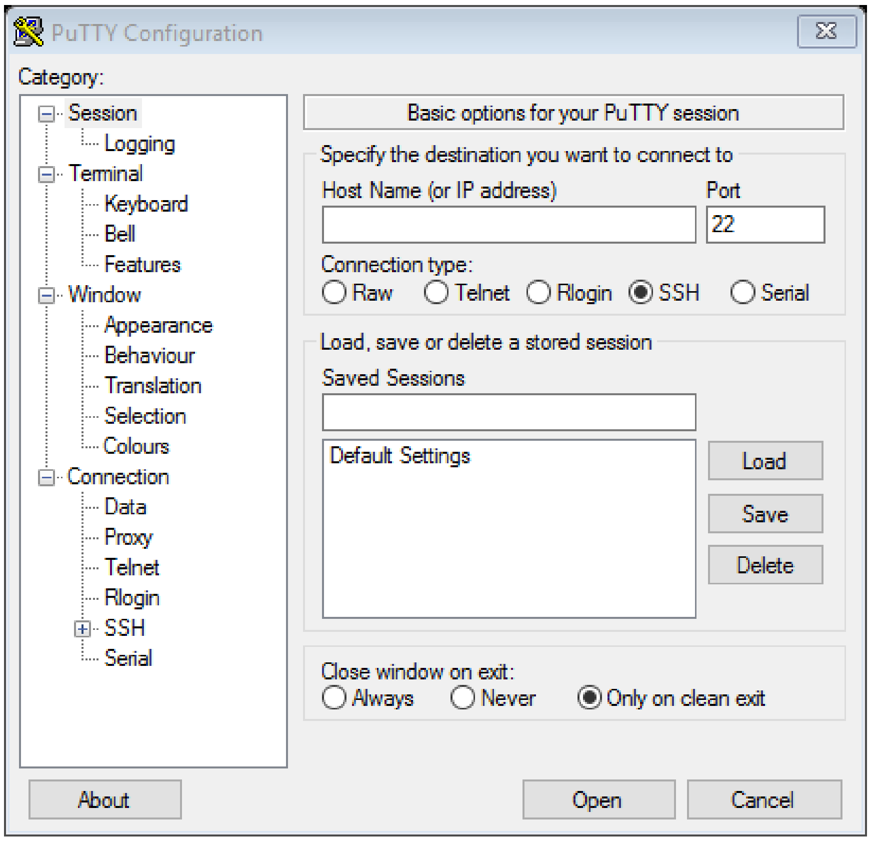
*qsub jobscript*

Assuming the user has developed a logic in the form of job and well now the job script is submitted it on the server to get output of the user calculations which can be executed with the help of Torque command *qsub.*

The software application gives the possibility of uniformly and provision of availability to execute the user processing threads. The batch system abstracts the technical details like CPU,memory space used by the job, disk i/o information on each node and also the information regarding the nodes which are free for the users to perform their calculations on the web based application.

Currently, the user's use PuTTY as a platform to execute the commands present on the server. PuTTy is a free SSH,Telnet and Rlogin client for Windows system. PuTTY is command line interface to users to use TORQUE commands via SSH, Telnet and Rlogin network protocols configured on the server. These protocols login from one multi user system to another system by the means of network or over the network. PuTTy supports many variations on the secure remote terminal, and provides user control over the SSH encryption key and protocol version, alternate ciphers such as 3DES, Arcfour,Blowfish, DES and Public key authentication. It can also be used with local serial port connections. PuTTY comes bundled with command line SCP and SFTP clients, called “pscp” and “psftp” respectively.

To achieve a connection to server, the user connects to server via remote login and submits their calculations to specific the server as a pbs job. Main task to automate the process and provide it from frontend by which the motive to avoid PuTTY can be achieved.



Specify the destination you want to connect to

Host Name (or IP address):

Port: 22

Connection type: Raw Telnet Rlogin [✓ SSH] Serial

Close window on exit: Always Never [✓ Only on clean exit]

**FIGURE 1: USER LOGGING VIA PUTTY**

Figure 1 gives a ideology how PuTTY makes it easy for user to perform connect to a particular server and perform various operations using PBS commands.The PBS commands can allocate resources for each job submitted where the PBS optimizes the resource utilization with taking into consideration to balance the job load. If the current resources does not meet the job requirements then the job is allotted to another server which is available. Another possibility for the PBS will wait in queue and pick a job in the queue,whose job requirements can be satisfied by the current resources for execution. The user can monitor and view the status of the job. It can schedule a job for a duration to be executed and manage its execution.Suppose, the job_script is embedded with the PBS commands

*#PBS -l walltime=9:00:00*

Here the job is given an walltime of 9:00:00 hours i.e the duration a particular job needs to be executed would be dependent on 9:00:00 hours.The most commonly used commands used on PuTTY are the job controlling and job monitoring commands such as *qsub,qstat,qdel,qhold qrls.*

| | |
|---|---|
| *-qsub* | submit a pbs job |
| | i.e qsub[job_script] |
| *-qstat* | shows status of a pbs job |
| | i.e qstat[job_script] |
| *-qdel* | deletes pbs job |
| | i.e qdel[job_script] |
| *-qhold* | helds a pbs job |
| | i.e qheld[job_script] |
| *-qrls* | release helded pbs job |
| | i.e qrls[job_script] |

Enterprise servers need to be excellent,secured, inexpensive,supportive.

My contribution to the project was to provide the PBS commands actions by only by one keypress and make it userfriendly on monitoring and viewing platform without the user to use another rlogin and perform PBS actions. If incase the user is new to the system then the user has the necessity to educate regarding usage of PuTTY and following with PBS commands. Therfore, the application would be an advantage and used as an industrial purpose.

The applicaton is known to be userspecific application because only the authorized user can perform these operations and to whom the authentication is granted. I have provided the functionalitity to the application taking into the note that the logged user can perform his own operations. The logged user can montior the other user jobs but cannot perform operations to it. Only the root or the application administrator can terminate all jobs or other user jobs.

Using one of the PHP program execution functions exec()[1]. The exec() is utilized for execution of an external program or an command which appends the command or a program until end.

[1] http://php.net/manual/de/function.exec.php

# Chapter 2

## 2 PBS Operations

### 2.1 CREATING WEB BASED PBS OPERATIONS VIA PHP

To create the web based PBS operations via PHP, it's important to be acknowledge about PBS workflow consisting it's components, it's distribution, load management methodology, resource utilization, queuing theory for PBS, schedule the PBS jobs, monitor and perform PBS operations on jobs on client side.

On server side the most basic requirement of the system is to install or configure PBS Environment such that the web application understands the commands to let it perform client operations. The PuTTY interface is just an medium to communicate with the server and execute the commands installed on server which is direct connection to the command shell terminal to PuTTY. To monitor the data the output of *qstat* command is returned in a normal text file and from where the data is read. The returned output is formatted and with the contribution scripting languages and style sheets made it visible on front end such that the users can perform concerned operations on the jobs they had submitted on the system. To make the web application work with the interface the enitre workflow is important to understand.

PBS is a worload management system which includes the feature of queuing. Queueing is basically to queue the submitted jobs until the managing system is ready to run the jobs. During the process of queuing several factors are taken into consideration such as service time distribution i.e the time required by the user, total number of servers for *n* number of customers, the maximum capacity a queue in the resource managing system.

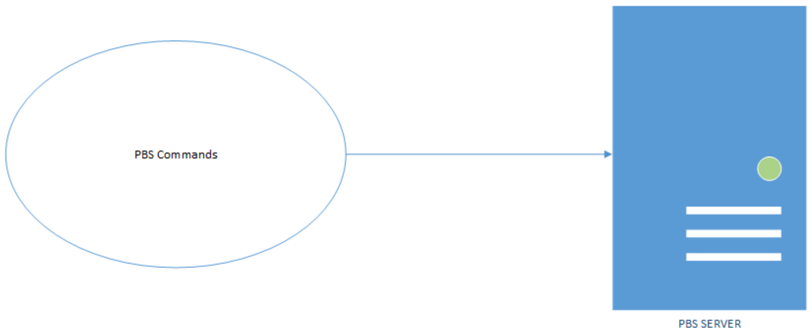


FIGURE 2: PBS COMMANDS TO JOB SERVER

PBS Components are of two types such as user level and system service. The workflow of PBS Components consists of PBS Commands, PBS Server, PBS Monitoring, Job Scheduler. The PBS Commands used on command line and on a graphical interface to perform PBS operations like submit, delete, hold etc. The user commands can perform its own PBS operations but the user should have account i.e the user should be a valid user. The root user of PBS server has the privileges to perform PBS operations in common and also has the privilege to undo specific user operations or perform particular PBS operation in similar to overall jobs. The PBS Server plays a important role within the components by running jobs, allocation of resources, workload management, queue up the PBS jobs. For web based PBS action execution is also performed on the PBS Server. The transmission between the user specific PBS commands and server is carried on network.

The jobs created by the PBS commands by the user to the server is monitored by MOM i.e Message Oriented Mini-server[2]. MOM is platform where the outputs are returned and after viewing the outputs the web based PBS operations can be executed by the user from the monitoring.When the web based operations are performed the user login privileges are same on the server by which makes the server execution easy, efficient and possiblity of jobs.

[2] http://www.pbsworks.com/documentation/support/PBSProUserGuide11.2.pdf

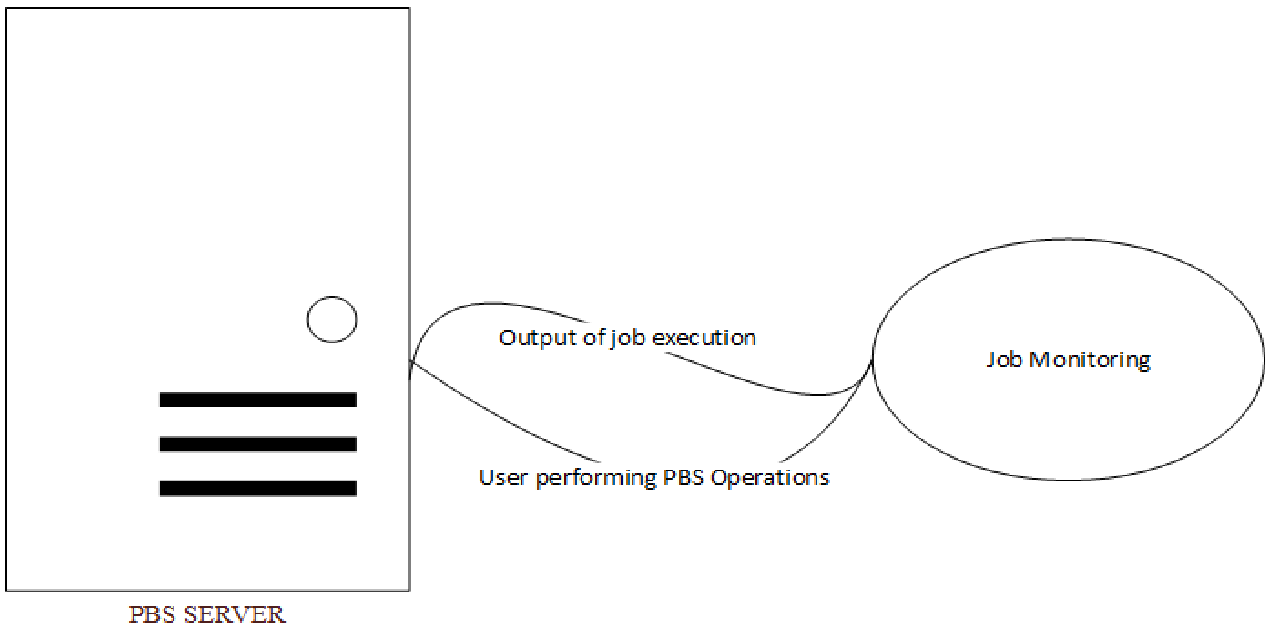


FIGURE 3: WEB BASED PBS OPERATIONS

## 2.2 WHAT MAKES PBS OPERATIONS WEB BASED?

The Job Monitoring which is built on web interface in here is hosted by PHP. The most of the knowledge is from the open source project <myJAM> for "Job Accounting and Monitoring"[3]. Various related to <myJAM> system were known from the project downloaded from internet because it was been utilized as a reference project to achieve the base of the job monitoring. Further the userfriendly operations on the jobs is most important part however to make it web based operations.

To create such kind of an web development interface PHP is most suitable while PHP is tightly coupled,supports server scripting and command line scripting which is better works with job schedulers. PHP comes with easy installation via LAMP[4],WAMP[5], etc. This feature is a great advantage to make the PBS commands execution worthfull. Another feature of PHP to embed

[3] https://www.dfn.de/fileadmin/3Beratung/DFN-Forum2/115.pdf

[4] https://en.wikipedia.org/wiki/LAMP_(software_bundle)

[5] http://www.wampserver.com/

HTML using PRINT or ECHO tags.These tags make it manageable for job to be monitored and implementation of shell commands. However, to make PBS operations web based an executable batch scipts which include the script of authentication and autorization as well as the executable commands with their concerned attributes.

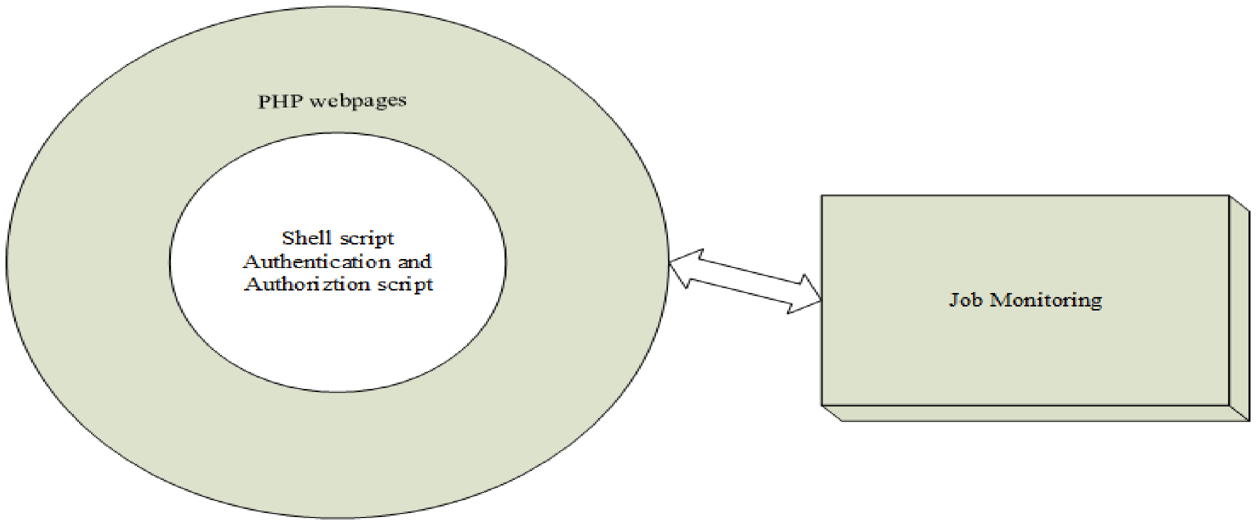


FIGURE 4: WHAT MAKES PBS OPERATIONS WEB BASED?

## 2.3 PBS JOBS

The general senario of PBS jobs is the calcuations and the computation performed on structure of an prototype by the user on user workstation. This job is made up of several task which needs to be perform processing, testing and various phases of product life cycle. The file could be a script file with tasks or an executable file generated by various product life cycle tools at each phase. Different processing tools generate different file extensions depending on the phase tool acceptance of file type.

The job scheduler is used to schedule the submitted jobs and with its exact credentials and resources required. There are various ways of submitting jobs single jobs, multiple jobs, parallel jobs, dependancy jobs, interactive jobs, etc.

## 2.4 PBS SHELL COMMANDS

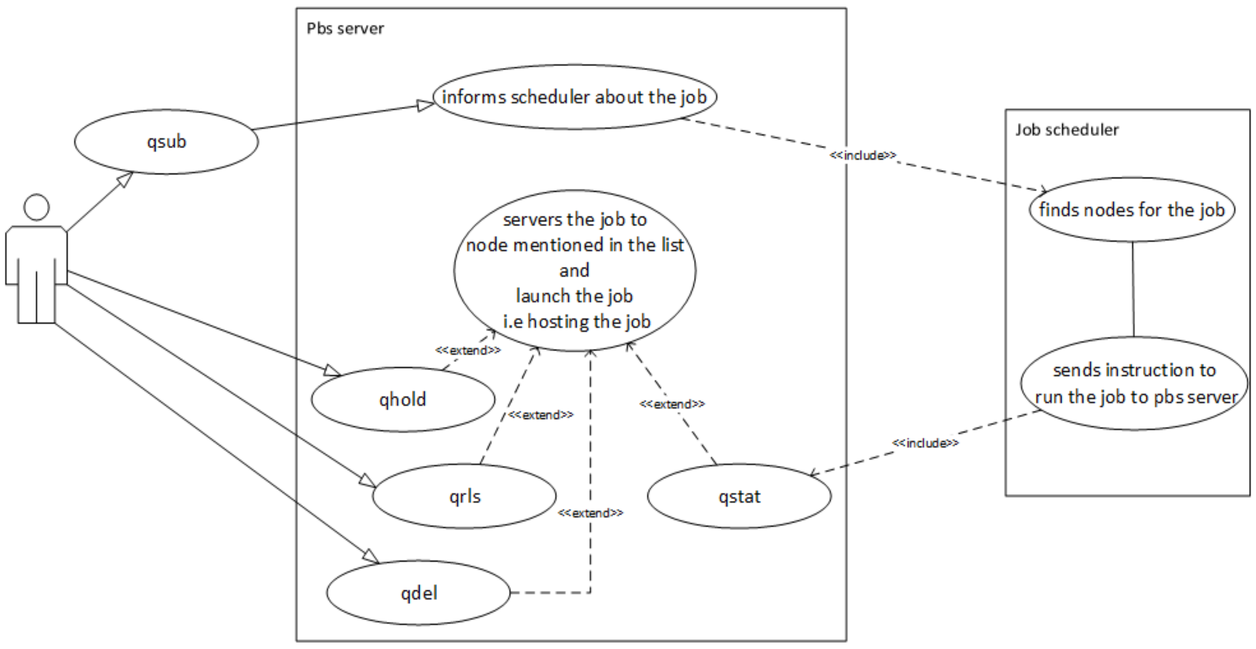

Pbs server
informs scheduler about the job
qsub
<<include>>
Job scheduler
finds nodes for the job
servers the job to node mentioned in the list and launch the job i.e hosting the job
<<extend>>
qhold
<<extend>>
<<extend>>
sends instruction to run the job to pbs server
<<include>>
qrls
<<extend>>
qstat
qdel

FIGURE 5: WORKFLOW/ USECASE OF THE WEB BASED PBS COMMANDS

To build the PBS commands and let them be usable from monitoring platform it is important to understand the standard PBS commands which are registered trademark of Altair Engineering,INC[6]. The PBS commands for user on web based operations are mentioned below with their following attributes

[1] qsub - submitting jobs

[2] qstat - monitoring jobs

[3] qalter - altering a job

[4] qhold - helding a job

[5] qrls - releasing the qhold jobs

[6] www.altair.com

[6] qdel - deleting jobs

[7] qorder - reordering the jobs

[8] xpbs - support to web based pbs operations

## 2.5 PBS SHELL COMMANDS IN DETAIL

[1] **qsub** - Submit an executable script to PBS server to create a job.

Usage: *qsub [-a date_time] [-A account_string] [-b secs] [-c checkpoint_options] [-C directive_prefix] [-d path] [-D path] [-e path] [-f] [-F] [-h] [-I] [-j join] [-k keep] [-l resource_list] [-m mail_options] [-M user_list] [-n] [-N name] [-o path] [-p priority] [-P user[:group]] [-q destionation] [-r c] [-S path_to_shell(s)] [-t array_request] [-u user_list] [-v variable_list] [-V] [-W additional_attributes] [-x] [-X] [-z] [job_script]*[7]

TABLE 1: QSUB ATTRIBUTES

| Option | Name | Description |
|---|---|---|
| -a | date_time | Use this time to calculate the eligibility for execution Format: [[[[CC]YY]MM]DD]hhmm[.SS] where *CC* is the first two digits of the year, *YY* is the last two disgits of the year, MM is the two digits for the month, *DD* is the day of the month, *hh* is the hour, *mm* is the minutes, and the optional *SS* is the seconds. |
| -a | account_string | Use this attribute to associate account string with the job. Format: <String> |
| -b | Seconds | Timeinterval to block qsub (is case of server failure) in seconds. |
| -c | Checkpoint_options | Use this attribute to define checkpoints for the job. Valid options: • **None** – no checkpoints • **Enabled** – Specify the checpointing is allowed but must be explicitily invokes by either the **qhold** or **qchkpt** commands. • **Shutdown** – Specify checkpointing to be done on the job at pbs_mon shutdown. • **Periodic** – Specify that periodic checkpoinint is enabled. (Default interval is 10 minutes) |

[7] http://docs.adaptivecomputing.com/torque/4-0-2/Content/topics/commands/qsub.htm

| | | |
|---|---|---|
| | | • **Interval=minutes** – Checkpointing is an interval of minutes (*walltime>0*)<br>• **Depth=number-** specifies number of checpoint images<br>• **dir = path-** directory path of checkpoint |
| -C | directive_prefix | Used to define the prefix that declares a directive to qsub command within the script file. -C option is is declared then *qsub* will not scan the script file for directives |
| -d | path | Used to define working directory path which is to be used by the job. |
| -D | path | Used to define the root directory for the job. |
| -e | path | Used to define the path to be used for the standard error st of the batch job.<br>Format: *[hostname:]path_name*<br>Where hostname is the name of a host to which the file will be returned, and path_name is the path name on the host in the POSIX syntax. |
| -f | | Use this option to make job fault tolerant.<br>Job running on multible nodes are polled by mother superior periodically. If one of the node fails to report, the job is canceled by the mother superior and a failure is reported. If a job is fault tolerant, it will not be canceled based on the failed polling. |
| -F | | Use to specify the argument that will be passed to the job script when the script is launched.<br>Format: *qsub -F "myarg1 myarg2 myarg3=myarg3value" myscript2.sh* |
| -h | | Use to specify that a user hold be applied to the job at submission time. |
| -I | | Used for interactive jobs |
| -j | Join | If error stream and output stream of job is mergered |
| -k | Keep | used when output or error will be retained the execution host |
| -l | Resource_list | Used to define the required resources for the job<br>For instance CPU time. |
| -m | Mail_options | Mailing messages regarding job |
| -M | User_list | User list of the mail sent. |
| -n | Node-exclusive | Node allocation request |
| -N | Name | Name for the job, the option is used as the base name of the job script file specified on the command line. |
| -o | Path | Used to define the path to be used for the standard ouput stream of the batch job.<br>Format: *[hostname:]path_name* |

| | | |
|---|---|---|
| | | Where *hostname* is the name of a host to which the file will be returned, and the *path_name* is the path name on that host. |
| -p | Priority | Used to define the priority of the job. The priority argument must be a integer between -1024 and +1023 inclusive. The default is 0 (no priority). |
| -P | User[:group] | Use to submit a job as another user. |
| -q | Destination | Defines the destination of the job. The destination names a queue, a server, or a queue at a server.<br>Format:<br>• *queue*<br>• *@server*<br>• *queue@server* |
| -r | y/n | Declares the job is rerunable. If *"y"*, the job is rerunable, if *"n"* the job is not rerunable. Default is *"y"*. |
| -S | Path_list | Declares the path to the desired shell in the job.<br>Format: *path[@host][,path[@host],…]* |
| -t | Array_request | Specifies the task ids of a job array. |
| -u | User_list | Use to define the user name under which the job is to run on the execution system.<br>Format: *user[@host][,user[@host],…]* |
| -v | Variable_list | Use this attribute to expand the list of environment variables that are exported to the job. |
| -V | | Use this attribute to declare that all the environment variables that are exported to the job. |
| -W | Additional_attributes | Use this attribute to specify additional job attributes.<br>Format:<br>*-w attr_name=attr_value[,attr_name=attr_value…]*<br>PBS currently supports the following attributes within this option:<br>• **depends=dependency_list** – Define the dependency between this and other jobs.<br>Format:<br>*type[:argument[:argument…][,type:argument…]*<br>The argument is either a numeric count or a PBS job id according to type.<br>• **Group_list=g_list** – Defines the group name under which the job is to run on the execution system.<br>Format: *group[@host][,group[@host],…]*<br>Only one group name may be given per specified host.<br>• **Interactive=true** – if the interative attribute is specified, the job is an interative job. |

| | | |
|---|---|---|
| | | • **Job_radix=<int>** - use this with parallel jobs. It directs the mother superior of the job to create a distributed radix of size ,int. between sisters.<br>• **Stagein=file_list**<br>• **Stageout=file_list** – Specifies which files are staged (copied) in before job start or staged out after the job completes execution.<br>Format: *local_file@hostname:remote_file[,...]*<br>• **Umask=XXX** – Sets umask used to create stdout and stderr sool files in pbs_mom spool directory. |
| -x | | Use this attribute to run the script (for the job) in interative mode. |
| -X | | Use this attribute to enable X11 forwarding. |
| -z | | Use this attribute to direct the qsub command in not to write the job identifier assigned to the job to the commands standard output. |

**Appending qsub in PHP:**

```
#!/bin/bash

#set -x

echo "command started with \"$*\"" >/tmp/qsub.log

USER_NAME=$1

JOB_SCRIPT=$2

if [ ! "$USER_NAME" ] || [ ! "$JOB_SCRIPT" ]; then

    echo "$0 <UserID> <JOB_SCRIPT>"

    exit 10

fi


echo "Executing \"ssh $USER_NAME@sdo2600 \"qsub $JOB_SCRIPT\"\"" >>/tmp/qsub.log

ssh 1>&2 2>>/tmp/qsub.log -vvv $USER_NAME@sdo2600 "qsub $JOB_SCRIPT"
```

[2] **qstat** - monitoring jobs

Usage:-*qstat [-f[-1]][-W site_specific][job_identifer…| destination…][time]*

*qstat [-a|-i|-r|-e][-n[-1]][-s][-G|-M][-R][-u user_list][job_identifier…|destination]*

*qstat -Q [-f[-1]][-W site_specific][destination..]*

*qstat -q [-G|-M][destination..]*

*qstat -B [-f[-1]][-W site_specific][server_name…]*

TABLE 2: QSTAT ATTRIBUTES

| | |
|---|---|
| -f | full status |
| -a | ALTERNATIVE FORMAT i.e if operand is a jobid, information of job is displayed |
| -e | only jobs in executable queues |
| -i | alternative format regardless of status |
| -r | status of the job |
| -n | node allocated to the job |
| -1 | if the -n and -1 then displays all the nodeson the same line as the jobid |
| -s | comment by scheduler |
| -G | Displays job size in gigabytes |
| -M | Displays job size in megawords |
| -R | disk reservation information |
| -t | expands the output to display array |
| -u | userlist are displayed |
| -Q | queue status are displayed |
| -q | request for queue status in alternative format |
| -B | request names of the batch server request |

**PBS Terminal output on qstat -wans:**

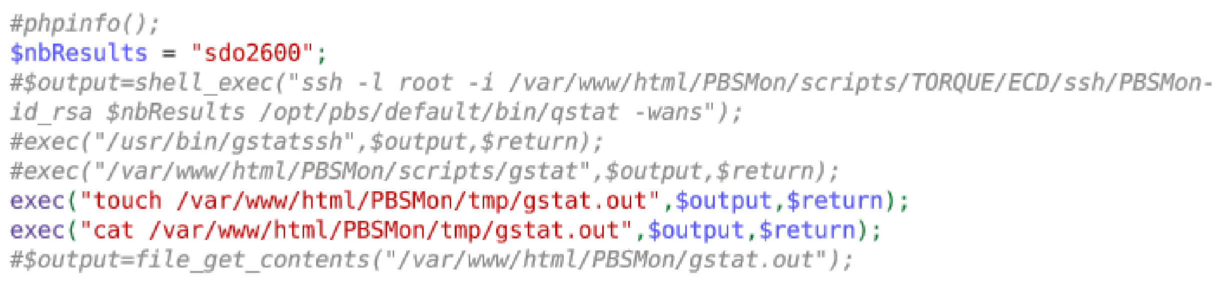
On executing *qstat -wans* command, the output returns the following details. Information regarding the jobid,username,queue,jobname,session id,nds,tsk,required memory,status,elaptime. This information is to be monitored and later utilized as referrence to perform other pbs operations.

```
                                                                                          Req'd  Req'd   Elap
Job ID                    Username      Queue        Jobname        SessID   NDS  TSK     Memory Time  S Time
------------------------------ --------------- --------------- --------------- -------- ---- ----- ---------- ----- - ----------------------
160287.sdo2600          shruti       long         job_script       31899    1     4       16gb   --  R 166:47:26
  node07/2*4
 161463.sdo2600          shruti       long         job_script    29185    1     4       16gb   --  R 21:33:42
  node02/0*4
161636.sdo2600          shruti       long         job_script    14016    1     4       5000mb   --  R 04:14:52
  node05/0*4
161655.sdo2600          shruti       long         job_script    26549    1     4       32gb   --  R 02:42:53
  node06/0*4
```

**Appending qstat in PHP:**

The bash script in following bin path contains PBS Environment variables. Each execution of PBS job uses PBS environment for performing PBS operations. The PBS jobs are identified by the standard PBS directive. Once the script finds the directive then it consider the rest of code in file to be user operations. Below the qstat output is appended into PHP

**/opt/pbs/default/bin/qstat -wans > /var/www/html/PBSMon/tmp/gstat.out**



```
#phpinfo();
$nbResults = "sdo2600";
#$output=shell_exec("ssh -l root -i /var/www/html/PBSMon/scripts/TORQUE/ECD/ssh/PBSMon-
id_rsa $nbResults /opt/pbs/default/bin/qstat -wans");
#exec("/usr/bin/gstatssh",$output,$return);
#exec("/var/www/html/PBSMon/scripts/gstat",$output,$return);
exec("touch /var/www/html/PBSMon/tmp/gstat.out",$output,$return);
exec("cat /var/www/html/PBSMon/tmp/gstat.out",$output,$return);
#$output=file_get_contents("/var/www/html/PBSMon/gstat.out");
```

FIGURE 8:QSTAT->GSTAT.OUT

The output contents of the file returned are retrieved by algorthim and scripted by PHP and made it available for monitoring.The PHP file hosting logically used for web monitoring is as following:

```
$j=0;

$jobs=array();

for($i = 5; $i < count($output); ++$i) {

 $jid=trim(substr($output[$i],0,30));

$jobid[$jid]=trim(substr($output[$i],0,30));

 $username[$jid]=trim(substr($output[$i],30,15));

 $queue[$jid]=trim(substr($output[$i],47,15));

 $jobname[$jid]=trim(substr($output[$i],63,15));

  $sessionid[$jid]=trim(substr($output[$i],79,8));

  $nds[$jid]=trim(substr($output[$i],88,4));

  $tsk[$jid]=trim(substr($output[$i],93,5));

  $reqmem[$jid]=trim(substr($output[$i],99,6));

  $reqtime[$jid]=trim(substr($output[$i],106,5));

   $state[$jid]=trim(substr($output[$i],112,2));

   $elaptime[$jid]=trim(substr($output[$i],114,8));

if(isset($_POST['list_queuestatus']))

{

          $selected= $_POST['list_queuestatus'];
```

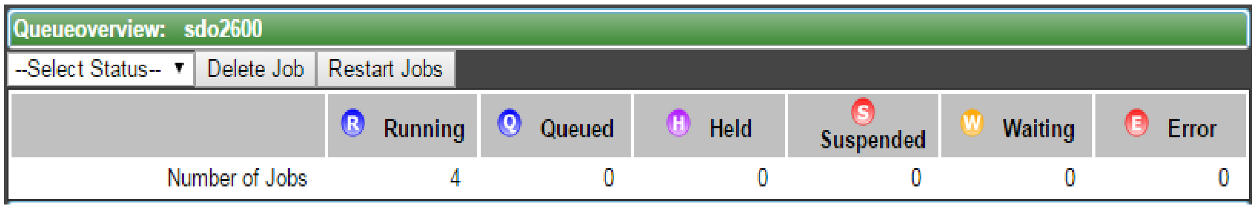
```
switch($selected[$job]){
case 'R':
         if($state[$jid]=="R"){
           $jobs[$j]=$jid;
         }
              break;
  case 'Q':
         if($state[$jid]=="Q"){
              $jobs[$j]=$jid;
         }
              break;
  case 'H':
         if($state[$jid]=="H"){
              $jobs[$j]=$jid;
         }
              break;
  case 'S':
         if($state[$jid]=="S"){
              $jobs[$j]=$jid;
```

```
                    }
                        break;
                case 'W':
                    if($state[$jid]=="W"){
                        $jobs[$j]=$jid;
                    }
                        break;
                case 'E':
                    if($state[$jid]=="E"){
                        $jobs[$j]=$jid;
                    }
                        break;
                case 'A':
                    $jobs[$j]=$jid;
                        }
}
else{
 $jobs[$j]=$jid;
    break;
```

}

}

For instance : Job Monitoring Status of qstat



| | Running | Queued | Held | Suspended | Waiting | Error |
|---|---|---|---|---|---|---|
| Number of Jobs | 4 | 0 | 0 | 0 | 0 | 0 |

FIGURE 6: JOB MONITORING STATUS

[3] **qhold** - helding a job

Usage:- *qhold [{-h <HOLD LIST>|-t <array_range>}] <JOBID>[<JOBID>] ….*

TABLE 3: QHOLD ATTRIBUTES

| | | |
|---|---|---|
| -h | hold_list | holdlist is combination of u,o,s where<br>u=user<br>o=other<br>s=system |
| -t | array_range | Uses a range of ids<br>For instance:1-100 |

**PBS Terminal output on qhold:**

```
                                                          Req'd  Req'd   Elap
Job ID              Username    Queue     Jobname    SessID  NDS  TSK  Memory Time  S Time
----------------------------- --------------- --------------- --------------- -------- ---- ----- ------ ----- - -----------------------
160294.sdo2600          shruti      long       job_script      --    1    4    16gb  --  H  --
160295.sdo2600          shruti      long       job_script      --    1    4    16gb  --  H  --
160296.sdo2600          shruti      long       job_script      --    1    4    16gb  --  H  --
160297.sdo2600          shruti      long       job_script      --    1    4    16gb  --  H  --
```

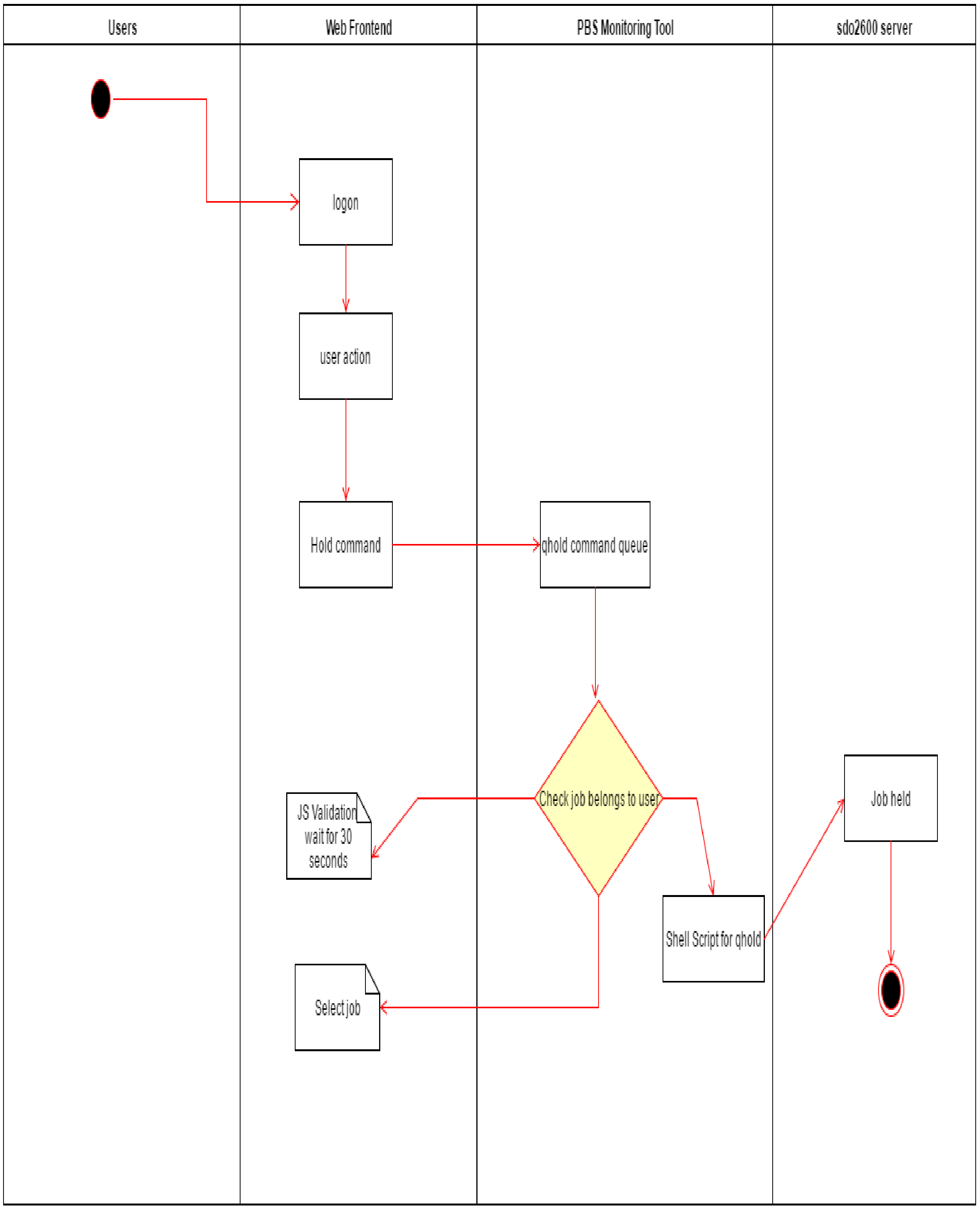


FIGURE 7: QHOLD SEQUENCE DIAGRAM

**Shell Script for qhold:-**

```
#!/bin/bash

#set -x

echo "command started with \"$*\"" >/tmp/qhold.log

USER_NAME=$1

JOBID=$2

if [ ! "$USER_NAME" ] || [ ! "$JOBID" ]; then

    echo "$0 <UserID> <JOBID>"

    exit 10

fi


echo "Executing \"ssh $USER_NAME@sdo2600 \"qhold -h n $JOBID\"\"" >>/tmp/qhold.log

ssh 1>&2 2>>/tmp/qhold.log -vvv $USER_NAME@sdo2600 "qhold $JOBID"
```

[4] **qrls** - releasing the qheld jobs

Usage:- *qrls [{-h <HOLD LIST>|-t <array_range>}] <JOBID>[<JOBID>] ….*

**TABLE 4: QRLS ATTRIBUTES**

| | | |
|---|---|---|
| -h | hold_list | holdlist is combination of u,o,s where<br>u=user<br>o=other<br>s=system |
| -t | array_range | Uses a range of ids<br>For instance:1-100 |

**PBS Terminal output on qrls:**

```
                                                                            Req'd  Req'd   Elap
Job ID               Username     Queue       Jobname      SessID  NDS  TSK   Memory Time  S Time
---------------------------- --------------- --------------- -------------- -------- ---- ----- ------ ----- - -----------------------
160294.sdo2600           shruti      long        job_script        --    1     4      16gb   --  R  --
160295.sdo2600           shruti      long        job_script        --    1     4      16gb   --  R  --
160296.sdo2600           shruti      long        job_script        --    1     4      16gb   --  R  --
160297.sdo2600           shruti      long        job_script        --    1     4      16gb   --  R  --
```

**Shell Script for qrls:-**

```
#!/bin/bash
#set -x
echo "command started with \"$*\"" >/tmp/qrls.log
USER_NAME=$1
JOBID=$2
if [ ! "$USER_NAME" ] || [ ! "$JOBID" ]; then
        echo "$0 <UserID> <JOBID>"
        exit 10
fi

echo "Executing \"ssh $USER_NAME@sdo2600 \"qrls $JOBID\"\"" >>/tmp/qrls.log
ssh 1>&2 2>>/tmp/qrls.log -vvv $USER_NAME@sdo2600 "qrls $JOBID"
~
```

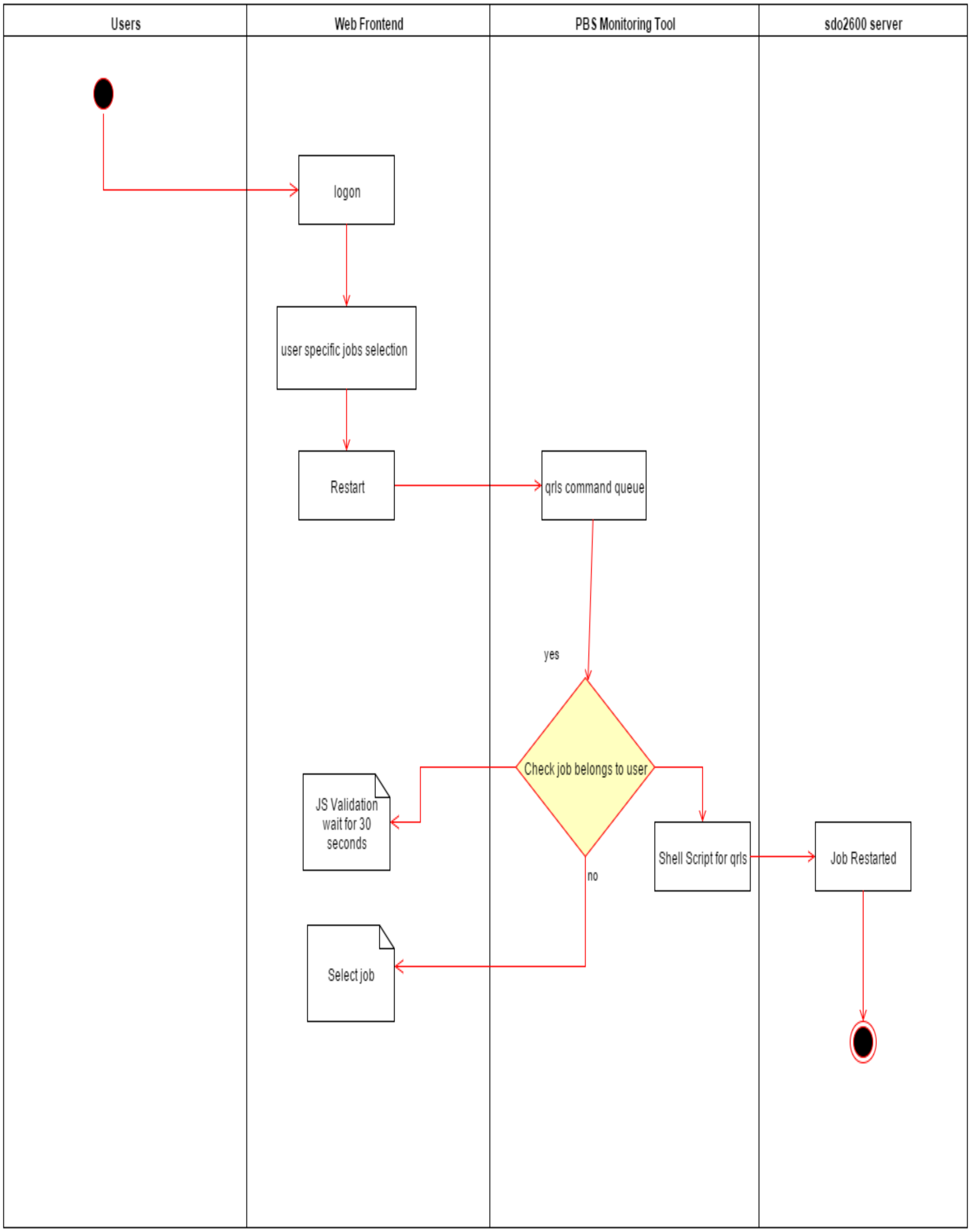


FIGURE 8: RELEASED JOBS SEQUENCE DIAGRAM

FIGURE 9: RELEASED JOBS SEQUENCE DIAGRAM

[6] **qdel** - Deletes an executable script to PBS server to delete a job

Usage: *qdel [{-a <asynchronous delete>|-m <message>|-p|-W <delay>|-t <array_range>}]<JOBID>[<JOBID>]...|'all'|'ALL'*

TABLE 5: QDEL ATTRIBUTES

| | | |
|---|---|---|
| -a | asynchronous delete | User can delete job asynchronously |
| -W | delay | Specify wating time for delete |
| -p | purge | Forcibily deletes a job |
| -m | message | Sends email message with deletion |
| -t | array_range | The command is used for a range of jobs specified in an array. |

**Shell Script for qdel:-**

```
#!/bin/bash

#set -x

echo "command started with \"$*\"" >/tmp/qdel.log

USER_NAME=$1

JOBID=$2

if [ ! "$USER_NAME" ] || [ ! "$JOBID" ]; then

    echo "$0 <UserID> <JOBID>"

    exit 10

fi

echo "Executing \"ssh $USER_NAME@sdo2600 \"qdel $JOBID\"\"" >>/tmp/qdel.log

ssh 1>&2 2>>/tmp/qdel.log -vvv $USER_NAME@sdo2600 "qdel $JOBID"
```

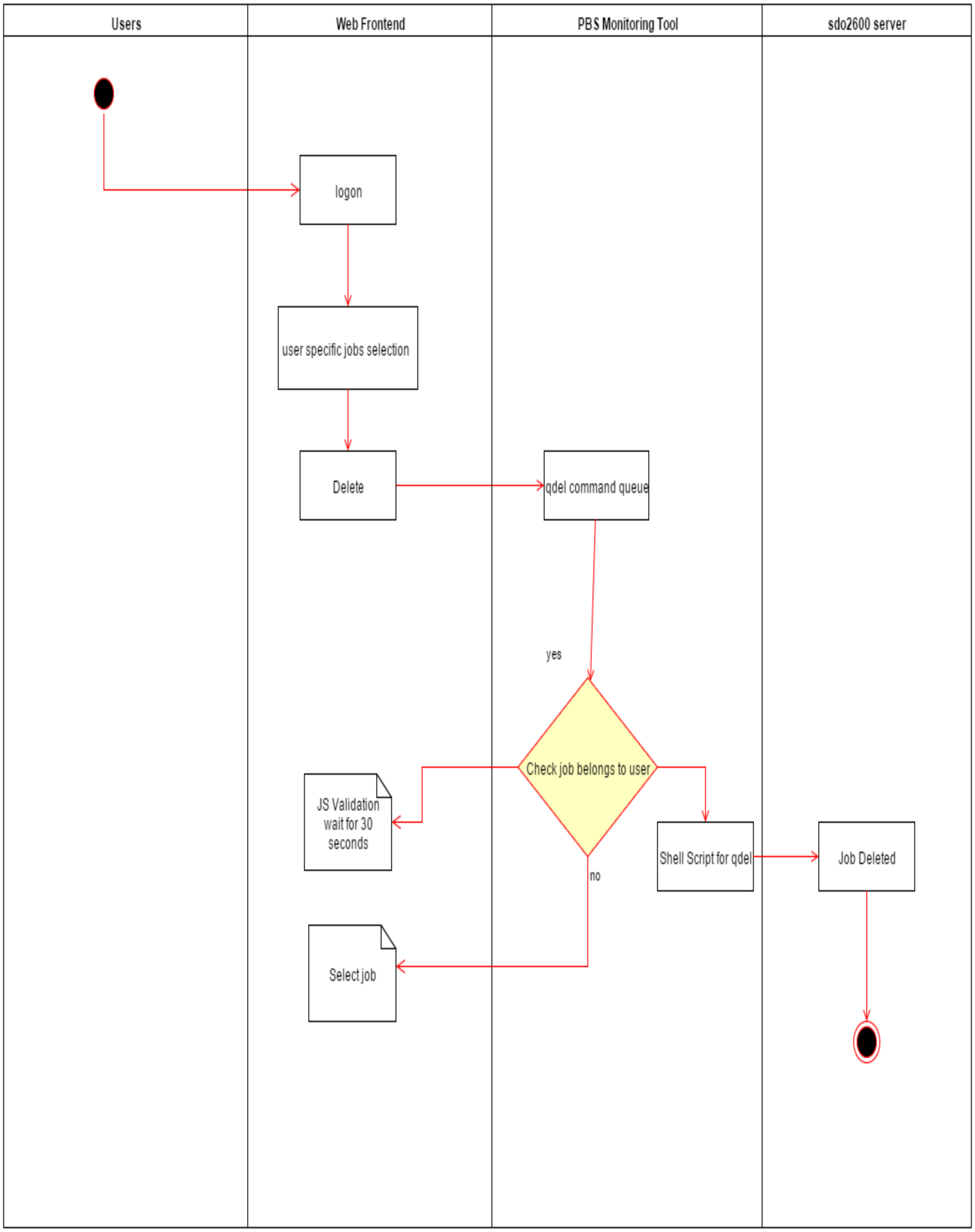


**FIGURE 9: DELETED JOBS SEQUENCE DIAGRAM**

## 2.6 REFERRING QSUB,,QHOLD,QRLS,QDEL RESOURCES FOR WEB BASED MONITORING IN PHP

```
echo '<form action="main.php?page=queuestatus&';

echo '" method="POST">';

echo '<input type=="text" name="job_submission">';

echo '<select name="list_queuestatus" onchange="this.form.submit();">';

echo get_options($selected);

echo '</select>';

echo '<input type=submit name="sub" value="DELETE Jobs"/>';

echo '<input type=submit name="submission" value="SUBMISSION Jobs"/>';

echo '<input type=submit name="held" value="HOLD Jobs"/>';

echo '<input type=submit name="restart" value="RESTART Jobs"/>';

echo '</form>';

$checkbox_id="";

$submission_id="";


//qdel

if(isset($_POST['sub']))

{

    if(!empty($_POST['chkid']))

    {

    foreach($_POST['chkid'] as $checkbox_id){

    $logged_user=$_SERVER['PHP_AUTH_USER'];
```

```
        exec("/usr/bin/qdel $logged_user $checkbox_id");
            }
        }
}

//qsub
if(isset($_POST['submission']))
{
        foreach($_POST['job_submission'] as $submission_id){
        $logged_user=$_SERVER['PHP_AUTH_USER'];
        exec("/usr/bin/qsub $logged_user $submission_id");
        }
}

//qhold
if(isset($_POST['held']))
{
        if(!empty($_POST['chkid']))
        {
        foreach($_POST['chkid'] as $checkbox_id){
        $logged_user=$_SERVER['PHP_AUTH_USER'];
        exec("/usr/bin/qhold $logged_user $checkbox_id");
            }
        }
```

```
}


//qrls
if(isset($_POST['restart']))
{
        if(!empty($_POST['chkid']))
        {
        foreach($_POST['chkid'] as $checkbox_id){
        $logged_user=$_SERVER['PHP_AUTH_USER'];
        exec("/usr/bin/qrls $logged_user $checkbox_id");
            }
        }
}
```

## 2.7 CONFIGURATION CHANGING DURING RUNTIME

Changes takes place in configuration for each user depending on its autorization and authentication. Henceforth, it is known to be user specific authorization which means only user can have the authentication to own part of work. In PHP the small part of code makes it valueable.

```
foreach($stuffs as $value)
{
        if(isset($jobid[$job]))
            {
            $logged_user=$_SERVER['PHP_AUTH_USER'];
            if($username[$job]==$logged_user)              {
                $jobaction='<input type="checkbox" name="chkid[]" value="'.$jobid[$job].'"/>';
            }
        else{
                 $jobaction='';
                          }
            }
}
```

# Chapter 3

## 3 Implementation

The Job Monitoring section is implemented using the LAMP (Linux, Apache, MySQL, PHP).

### 3.1 LINUX APACHE MYSQL PHP FOR JOB MONITORING

The statement on wikipedia explains most of the things about LAMP "*LAMP is an archetypal model of web service solution stacks,named as an acronym of the names of its original four open-source components:the Linux operating system,the Apache HTTP Server, the MySQL relational*"[8]

There are many best things of using LAMP that it comes into software bundle of four software components which is required for web development and are open source i.e easily available. Being opensource makes it inexpensive and also with the feature of best security.The acronym stands for the software components Linux, Apache, MySQL, PHP but for configuration of LAMP there is a typical procedure to follow.

[8] https://en.wikipedia.org/wiki/LAMP_(software_bundle)

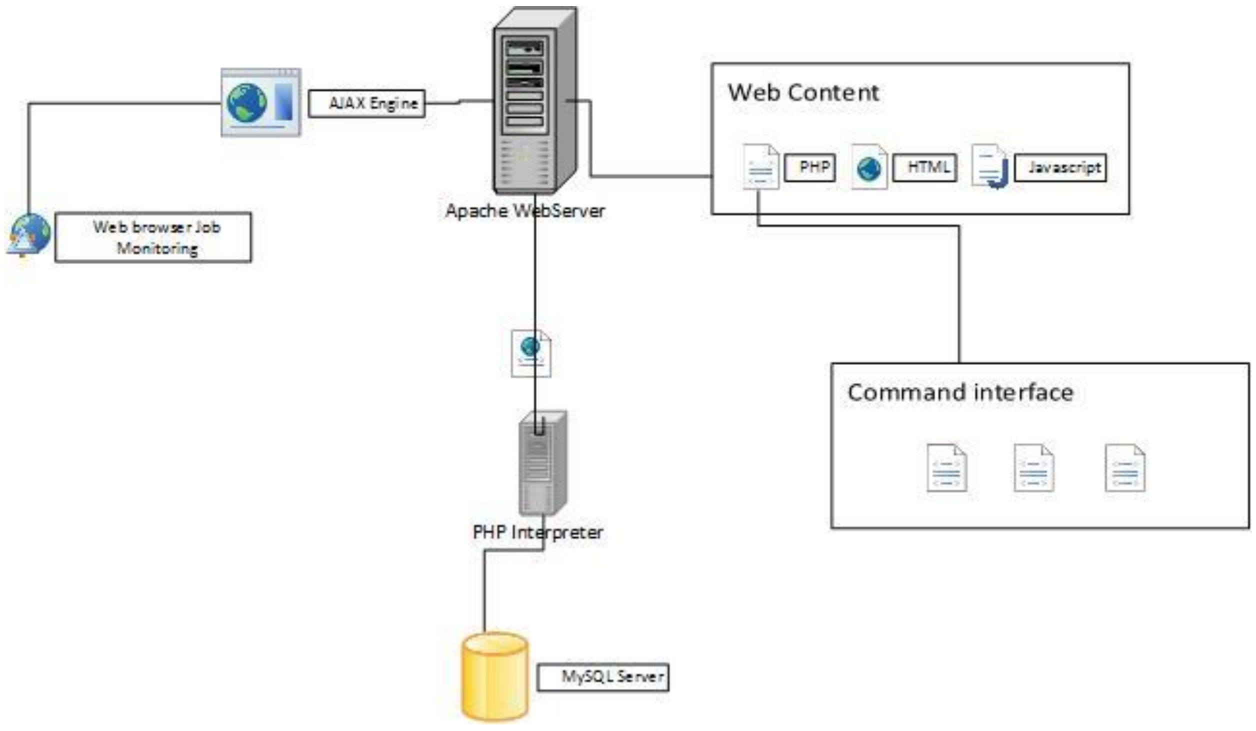


FIGURE 10: ARCHITECTURE DIAGRAM

### 3.1.1 LINUX FOR JOB MONITORING

The best things about Linux operating system is commercially distributed and holds GNU licenses i.e General Public License. It supports utilization as desktop and server. These systems provide excellent support for enterprise devices and capable of handling securely high performance computing applications.The accessibility for code in Linux allows full access to write code and also provides the feasibility to do manipulations or any changes necessary. This makes it easy to make improvements and fix bugs. Applications like SSH and scripting applications with better support of apache webserver then Linux is most considered.

### 3.1.2 APACHE WEBSERVER FOR JOB MONITORING

Apache HTTP Server is opensource and most widely used web servers. Apache HTTP server is most important component for hosting a web application. The Apache webserver using the http protocol works with client and server architecture. However,the web browser is responsible for client requests and the apache webserver responses to client requests as a server. To install apache in Linux Redhat the command mentioned below is used.

To install:- *sudo yum install htttp*

To start apache:- *sudo service httpd start*

The file httpd.conf on the system is the configuration file for Apache web server. The default location for the file is /etc/http/conf/httpd.conf and any changes to the file is done such as port 80 to port 8080 by editing this file.

When a client processes an request the user IP address is sent with the following http credentials such as the port information and also with the connection details. The resource which comes in the form of client request is in the form of URI(Uniform Resource Identifier)known to be the URL(Uniform Resource Locator). The communication for the client with the server is possible until or unless the port is made avaible to the client by the server. There more functionalities which make it simpler for hosting a webpage.

The Apache HTTP webserver GET or POST the client requests and the requests are read by the help of HTTP header.The LAMP provides PHP installation which means as an addon the PHP versioning is present within the HTTP header.The Apache module known to be *mod_rewrite* [9]is used to provide a rule-based rewriting engine. It is used to rewrite the requested URL and also capable of manipulating URLs by appending them using various number of rules. The rule can be also applied the .htaccess data specifically the client user agents that is detected to be a malicious are blocked. Howerver, the IP range is specified for the client user agents are declared from a particular source but never from other sources. The user agents are modified in http request header and granted access control. Most of parsing of request header is carried out by parsing the http request header. To have the .htaccess files access the user should be able to provide server configuration that allows authentication directives to them. Only authorized user would have the

[9] http://httpd.apache.org/docs/current/rewrite/

username and password credentials required to access the server. The web application contains scripting file,those are the php files. The HTTP header request is parsered using the authentication methods in PHP. The PHP parser understands the server-side scripting and makes authentication possible by validating the user logging information from database and the request header.If the server has a user list which is in query string stated in the server configuration file that makes a connection to the database server. For instance, if the user is present in the *Users* table then the user would have the access to the server. The error caused can be considered in the error log file created by Apache webserver. There are some fixed content when errors occur or unauthorized user tries to access the server and these content is fired on errors. However,if the content is successful then provides logon to the user and these response is sent by the server depending on the PHP written script. Everytime for logging is the first priority for the content handler and secondly priority the failure in connection process is to be written within the error log file.

#### 3.1.2.1 Connection dropped

The connection with the client is most important factor performed by the HTTP. To maintain this connection client request should be open until some duration. Therefore, http pipelining is a used which creates multiple HTTP requests are sent on a singlr TCP connection without waiting for server responses. The web application sends all dynamic data which it consumes time, makes slower processing of data over IP address or the network. The HTTP pipelining creates more frequency in loading of web content. HTTP pipelining works efficently with web servers,web browsers and proxies. Multiple request and responses sent over a single TCP connection in other concepts means the connection once created can be used by multipe request and responses i.e reusage of single TCP connection. Such kind of connection is known to be HTTP keep-alive or HTTP persistance connection.[10] The HTTP header are persistance until they are stated to be persistant. Keep-alive in the additional header to the client request connection. The request and response generated once using same connection won't drop until it is decided to end

[10] https://en.wikipedia.org/wiki/HTTP_persistent_connection

the connection by the client or the server. The server will wait for on persistent connection unless [11]the connecetion is dropped.

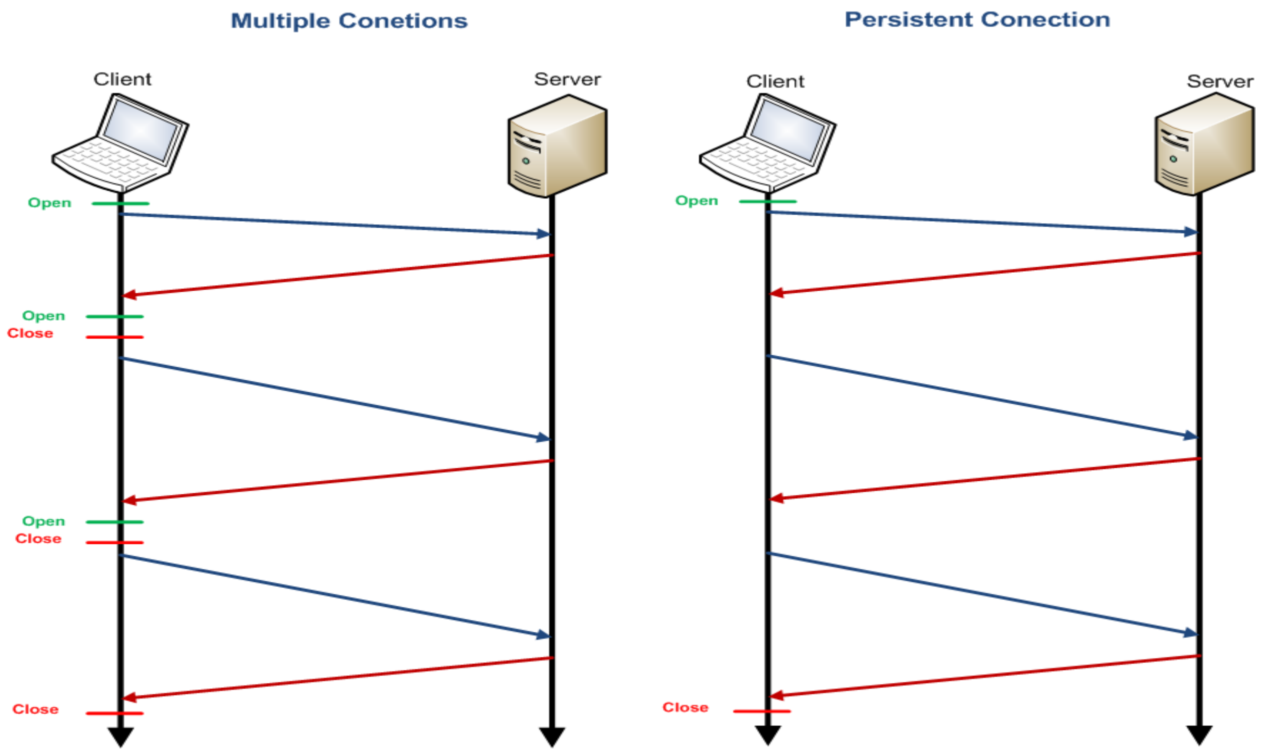


FIGURE 11: WORKING OF KEEP ALIVE IN APACHE SERVER

### 3.1.3 MYSQL PHP FOR JOB MONITORING

The most basic issues are latency, reliability, security, bandwidth and availability. In the web application, the HTTP GET and POST made it simpler for web hosting with high performance. They are used to retrieve data regardless of programming environment, the data is requested and responded to and fro via HTTP GET and POST.

[11] https://upload.wikimedia.org/wikipedia/commons/a/aa/Keep-alive_true_or_false.png

PHP web application is highly interactive which is capable to integrate with databases and supports large number of large protocols like POP3[12], IMAP[13] and LDAP[14]. It manages dynamic web content, databases, session data and can be used in building ecommerce websites. PHP is well known language for web based application and compatible with various databases such as MySQL,Oracle and more. To build up a message oriented software application the exchange of data and data has to retrieved from MySQL database. The configuration file php.ini is normally starts when the web server starts to initalize a PHP or each time httpd is started. This file is used by the PHP parser. In LAMP, the configuration file can be found in server instalation directory /etc/php.ini.

MySQL is central component in LAMP and widely used for web applications. The database has all information stored like projects, users, jobs and applications. These information is stored relational database tables in database. The database view concepts is used for merging different database tables entities as per the retrival of data. The retrieval of user data from the database tables navigates the web based application in it's direction. The web pages lets the access data from the database server over network . It lets to maintain user data for third normal form i.e 3NF where the normal form is utilized in normalizing the database design for reduction of duplicates. In order to understand indetailed architecture of application it is necessary to understand the software design, database schema with it's injection with PHP and web services used. All you require is the information of classes, database, web services and application architecture.

## 3.2 COMMAND INTERFACE

The users usually first access the main php and the main php is the interface of web based application. It represents central part of  the application and differentiated into user access rights and administrator access rights.The focus on user tools because the motive of providing web based

[12] https://en.wikipedia.org/wiki/Post_Office_Protocol

[13] https://en.wikipedia.org/wiki/Internet_Message_Access_Protocol

[14] https://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol

pbs operations is to be executed over these files. TheThe main.php in the application from which the application  traverses the user to other php given below

TABLE 6: PHP WEB FILES

| **User Tools** | | | |
|---|---|---|---|
| in_queueinfo | in_queuestatus | in_hardwareoverwiew | In_hardwarestatus |
| in_licenceoverview | in_licencetool1 | in_licencetool2 | in_usersetting |
| in_licencetool3 | in_licencestatus | in_home | |
| **Admin Tools** | | | |
| in_adminprojects | in_adminusers | in_adminorganization | in_adminqueues |
| in_clusterhistory | in_clusterstatus | in_costmodules | in_announcements |
| in_exitstatusmodification | in_configuration | in_database | in_systemlogfiles |

The main is initialization is allowed to be loaded on validation of configuration modules and exceptions are occurred on failure. The code below provides connection to server by the host and configures connection to the mysql database. On inappropriate configuration an exception is occurred.

```
 echo "INFO:connecting LDAP@sdo2600"

set --$(shh sdo2600 "ldapsearch -w localhost -x -D cn=Manager,dc=de uid=$user"|grep "^cn:"|sed "s/cn:\([A-Za-z]*\),\([A-Za-z]*\)/\1\2/")

if[[-n $2]];then

echo    "INFO:creating User username:$user firstname: $2 lastname:$1"

echo    "insert into Users(username,firstname,lastname)values('$user',$2,$1');">$sql_user_add

mysql--user=$Database_User --password=Database_Password $Database_Name< $sql_user_add
```

```
        echo    "INFO: activating User $user for DEFAULT project"

        echo    "select uid from Users where username='$user';">$sql_user_show

uid=$(mysql--skip-column-names--user=$Database_User--password=Database_Password
$Database_Name< $sql_user_show)

        echo    "insert into Meta_ProjectsUsers(pid,uid)values(1,$uid)">$sql_user_add

mysql--user=$Database_User--password=Database_Password $Database_Name< $sql_user_add

        else

                        echo "ERROR:user $user not available at connection"

        fi

        else

                        echo"INFO:User $user already exists in PBSMon database"

        fi

[-f $sql_user_add]&& /bin/rm $sql_user_add

[-f $sql_user_show]&& /bin/rm $sql_user_show
```

FIGURE 12: PROGRAM FOR CONFIGURING USER

The pbs operations are executed in the section of user tools is accessed while there are many database tables and entities accessed in which the Users table is used. The views are created such as JobDetails contains rows and colums of various real tables such as Users, Jobs.The php file uses the JSON[15] with AJAX functionalities over web development monitoring.When pbs operation is performed the php are called and the directly or indirectly touches all the mentioned files.

TABLE 7: EXECUTION FILES

| | | | | |
|---|---|---|---|---|
| class_database | class_user | class_project | class_queue | class_host |
| class_architecture | JSON | access | duration_out | memory_out |
| /usr/bin/qdel | /usr/bin/qsub | /usr/bin/qstat | /usr/bin/qhold | /usr/bin/qrls |
| /tmp/qdel.log | /tmp/qsub.log | /tmp/qstat.log | /tmp/qhold.log | /tmp/qrls.log |
| gstat.out | images | js files | css files | scripts |
| epilogue | prologue | Configuration files | database.sql | getJobs |
| getUser | getProject | getQueue | tools | HTMLCreator |
| title | closewindow | header | | |

The job is identified by the scripts files prologue and epilogue. These script files if the job sent belongs to a user and other credentials. If the credentials are not successfully completed the script terminates the job which will write the termination cause in log file. The prologue.php file requests for starting the job and epilogue.php sends the request for closure of job. The pbs web operations are also executed via these script files toward the server. Immediately these scripts start a job execution and stores job information. Each job in prologue scripts checks for the following entities in a job which states the job is a pbs job. The entites used by the jobs are mentioned below:

- Job_id
- Start_date
- End_date

[15] http://www.json.org/

- User_id
- Project_id
- Used_cores
- Walltime
- Queue
- Requested walltime

The qsub command in script file scans for the directive with characters '#!' or ':' and reads the job data below the following the commands until the entities values are fullfilled to schedule a job. To execute this qsub command the web frontend sends a request to the command by exec(). The exec() executes an external command which appends the command until end. The applicaction program interface i.e API on client side request to perform a web based pbs operation which executes the exec() in php file and resulting into behavioural change on jobs. The prologue and epilogue script files interacts the API with the batch just prior or later a job. The batchsystem automatically are called through these script files and executed on batch server. The scripts contain ssh to batch server to execute API files executing the script file in php.

## 3.3 DEPLOYMENT

The application server manages the jobs ,queues, nodes to handles the user and projects by providing a view where the users can visualize current and previous results. The user submits a job on server is visualized on client side by accessing the client connection. For user pbs operations the application server will access the command interface to execute particular task allotted. The deployment main task of automation of pbs operation on web inteface rather than using another interface for performing operations on jobs.

The application architecture consists of three layers in deployment perspective which is the frontend client layer, logical application layer and backend data server layer. The object relation mapping which manages the translation of objects into relational databases and visa-versa. PHP utilizes the concept of objects and MySQL utilizes the concept of relational. However, objects data correspond to rows in table where the tables are related by foreign keys. The data is returned in simple arrays or through pojo's. The sql CRUD operations can be performed on database server by the application server. The application one class per table is the basic logic which make it easy to deploy and command interface to perform pbs operation. The command interface is a part of application server but not as a whole. The mapping used for getting number of jobs and job information whereas for complex operations mapping JSON seems better.

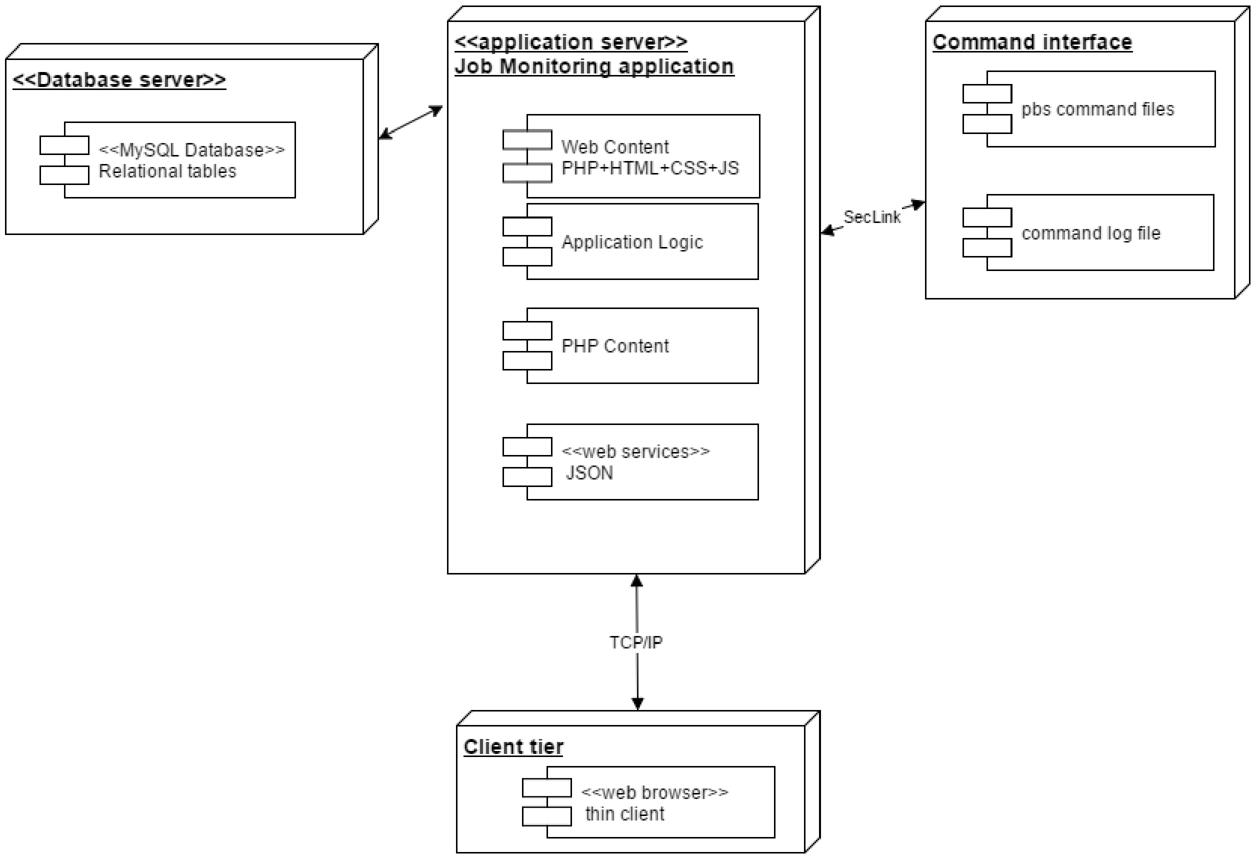


FIGURE 12: DEPLOYMENT DIAGRAM

# Chapter 4

## 4 PHP, Ajax, JSON, RESTful Web Services

Communication between the client, server and database is via network protocol. The web browsers on user machines are used to access network resources and the are linked with various other webpages and web based applications.The database server stored data in backend is accessed through TCP/IP protocol and connects with ldap connection which comes with the protocol.

```
echo "INFO:connecting LDAP@sdo2600"

set --$(shh sdo2600 "ldapsearch -w localhost -x -D cn=Manager,dc=de uid=$user"|grep "^cn:"|sed "s/cn:\([A-Za-z]*\),\([A-Za-z]*\)/\1\2/")
```

The frontend utilizes object oriented PHP5,embedded HTML,CSS and JavaScript or AJAX. The user interaction towards the server side needs AJAX methods with JSON data.

### 4.1 OBJECT RELATIONAL MAPPING

When the topic of objects and database come into picture then object relational mapping comes into first approach. Managing data in PHP is implemented using non-scalar entities. The object relation mapping which manages the translation of objects into relational databases and visa-versa. PHP utilizes the concept of objects and MySQL utilizes the concept of relational. However, objects data correspond to rows in table where the tables are related by foreign keys. The data is returned in simple arrays or through pojo's. The sql CRUD operations can be performed on database server by the application server. The application one class per table is the basic logic which make it easy to deploy and command interface to perform pbs operation. The command interface is a part of application server but not as a whole. The mapping used for getting number of jobs and job information whereas for complex operations mapping JSON seems better.

## 4.2 JSON WITH PHP

JSON is language independent and lightweight text based interchange format so it used with PHP and AJAX. The version PHP5 consist of JSON extension bundled and compiled by default. The functions and libraries used in the application interface are as mentioned below:

**TABLE 8: JSON STANDARD FUNCTIONS**

| Function | Libraries |
|---|---|
| json_encode | It returns the JSON represtation of a value |
| json_decode | It decodes a JSON string |
| Json_last_error[16] | It returns the last error occured |

The PHP json_encode() and json_decode() function are used for encoding and decoding JSON in PHP. The json_encode function returns the json encoded value on successful execution or FALSE on failure whereas the json_decode returns the decoded value from json to PHP.

## 4.3 AJAX WITH JSON

AJAX[17] which is used to create asynchronous web applications on client side development which can send and retrieve server data asynchronously. Job monitoring web application needs to show live data of the jobs, queue, etc. Hence, for live results should be updating instantly which is possible by AJAX. If the job information has to be stored on the server such that the webpage can retrieve the required information then AJAX with JSON data is used. The updated data using

[16] http://www.json.org/

[17] https://en.wikipedia.org/wiki/Ajax_(programming)

AJAX can be retrieved by JSON files through javascript such that it can be parsed when necessary and perform operations. JSON format is utilized of seralizing and transmitting structured data between the web application server and web browser over network.

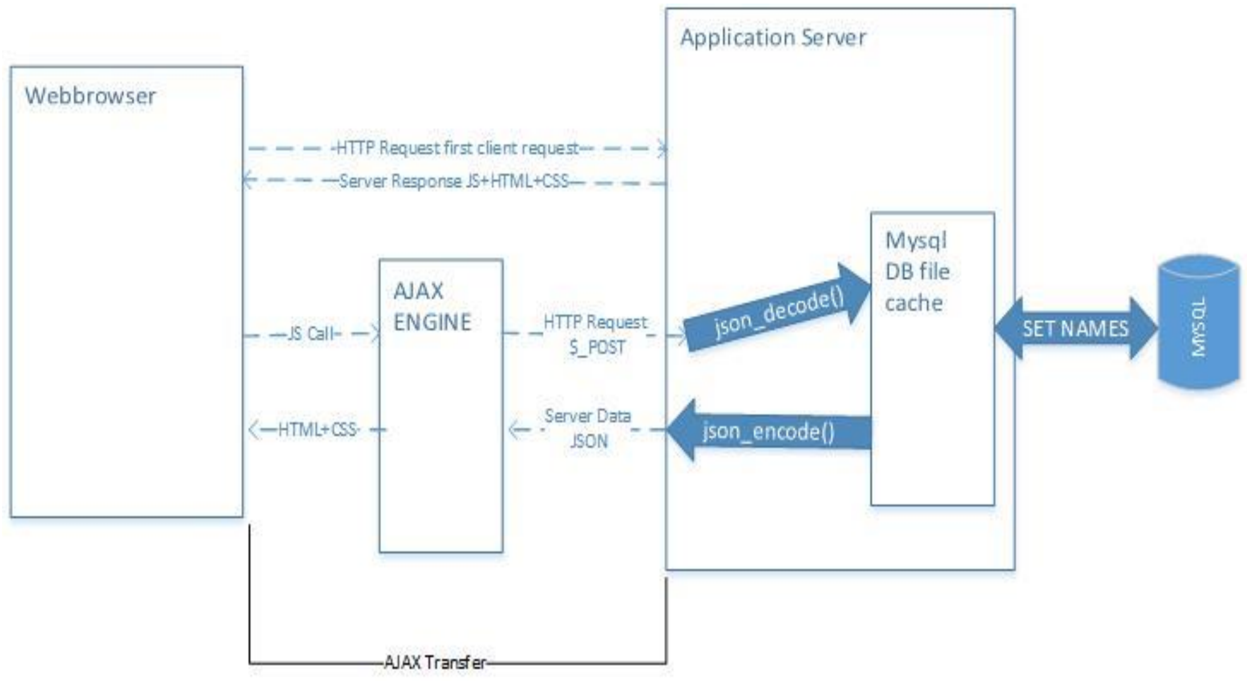


**FIGURE 13: WEB SERVICES**

The web browser wants to transfer data via AJAX to the PHP application server and store information in MySQL database tables. Normally, the client sends request to server and server sends back response but AJAX comes into picture the methodology changes.

Well to understand this methodology the components and interfaces such as web browser, application web server, Ajax engine, PHP/ SQL interfaces, command interface, database and php files are set up UTF-8 encoding which transfers POST data to PHP application server via Ajax and then into the database using PHP/SQL interface. Once a connection is established through HTTP then the keep alive mechanism started by apache web server to maintain the sanity of the connection. Further in same connection the data transfer is processed through Ajax engine which holds jQuery and JSON formatted data.

The json_decode used to decode the POST or GET data strings of Ajax request and transfers the string to database. On the server side once the Ajax data goes correctly into the database through proper decoding on server side. If data is not properly decoded, then there is usually need to set names for php/sql interface by which the database is prepared for the incoming and outgoing data streams. Later the relational database defines the tables, rows and columns.

To transfer POST data over Ajax request to the server the GET or POST mechanism is used. For Ajax data transfers there has to be an ajax environment and also java script environment to take JS calls. The content type used by the JS Call is defined is jQuery[18] for character encoding to transfer data.

#header (‘Content-Type:        text/html; charset= UTF-8’[19])

Further $_POST array transfer is decoding by json_decode() where the strings are saved into database table entities. Similarly for the server response, the data entities should be encoded into data strings and json_encode() accepts all input strings from the interface. The json_encode makes sure all data is in array coming from php or from the database.

## 4.4 RESTFUL WEB API8

Ajax applications follow REST design principlesand REST api works with http methods using GET,PUT,POST DELETE. Each http request is considered as REST service request and each JSON response is consider to be REST service reponse.

REST foccuses on interaction with stateful resources instead of messaging and operations where URL are tightly associated with REST service. Normally, the REST architecture uses

[18] http://api.jquery.com/jquery.ajax/

[19] https://en.wikipedia.org/wiki/UTF-8

WSDL which offers support to HTTP request methods.The RESTful web api is a collection of resources with four defined request:

[1] The base URI for the web service

[2] The network media type of the data supported by the web services which is JSON in the application

[3] The set of supported operations by the HTTP methods like POST is used in here.

[4] The application interface must be hypertext driven.

Basically we are using JSON REST API via PHP and the PHP apiclient will take a URL string and output a PHP array of POST objects. The key difference is that POST is necessary when the server is in control of storing information where the server is responsible for storing database information and assign unique values. The server process request from the client and reponses are returned. They are built around the transfer of representations of resources which can be addressed. A representation of a resource is typically a document that captures the current state of a response. The client begins sending request when it is ready to make the transtion for a new state whereas for more requests the client is considered in transition. The represtation of application state consist of links which can be reused the client chooses to initiate a new state transition.

**Stateful**

The request from the client consist information to service the request and session state is held in the client. On the server side state the URL is considered as resource. This makes the server stateful and makes it visible for monitoring which is also reliable for network failures and scalability. On stateful server can be identified by

-Session

-logging

-dynamic content

-acknowledge segments

**Caching**

Client can cache responses on word wide web. Hence, the responses must be define themselves as cacheable or not that is a task to prevent clients reusing inappropriate data in response to upcoming requests. Well managed caching removes some interactions of client serverby improving scalability and improment in performance.

**Layered System**

The client server is cannot be stated wether they are connected directly or indirectly. The intermediate servers may improve performance and scalability by load balancing and handling caches rather they are enforced by security policies. Each time the user performs pbs operation the server will ask for user password. However it is clear to use a ssh passwordless concept. To use Linux and OpenSSH to automate the passwordless task which will automate login from client host to server host. To call shh from within a shell script some pair of authentication keys are a necessity.

<u>Output of authentication files for establishment of connection:</u>

• debug1: Connection established.

• debug1: identity file /var/www/.ssh/identity type -1

• debug1: identity file /var/www/.ssh/identity-cert type1

• debug1: identity file /var/www/.ssh/id_rsa type -1

• debug1: identity file /var/www/.ssh/id_rsa-cert type -1

• debug1: identity file /var/www/.ssh/id_dsa type -1

• debug1: identity file /var/www/.ssh/id_dsa-cert type -1

• debug1: identity file /var/www/.ssh/id_ecdsa type -1

• debug1: identity file /var/www/.ssh/id_ecdsa-cert type1

• debug1: Remote protocol version 2.0, remote software version OpenSSH_4.3

**Demanding source code**

Servers are able to customize or extend features of a client by transfer of executable code. Example clientside scripts like Javascript call scripts

**Uniform interface**

The uniform interface between client and server makes the architecture simpler and decodes which enables each part to evolve independently.

# Chapter 5

## 5 Load balancer,Pooling,queueing

### 5.1 POLLING

Polling which is utilized synonymously with busy-wait polling and when the input or output operation is required, the computer then check status.of device until it is ready for acessing. However due to this it will repeatedly check for state of ready. For single processing system it is an perfect process whereas multiple processing or mutiple processors it means too many devices that leads to consumption of time.

#### 5.1.1 HTTP LONG-POLL[20] IN AJAX:

The web applications are developed around the client -server architecture where the client first request to server and the server on client request the data is pushed. There no such way for the server to push data without making a request. Over such issue the HTTP long polling plays a role. The HTTP long polling is where the client polls the server requesting data and the server holds the data until a new request is requested. On available, the server responds and send another set of data and repeates. The web application for pbs operations via PHP uses Java script. The

[20] http://www.abrandao.com/2013/05/php-http-long-poll-server-push/

process is carred out for single and multiple clients. The HTTP long poll process utilizes the functionality of HTTP keep alive methodolgy as dicussed in [chapter 3]

### 5.1.2 CLIENT SIDE AJAX POOLING

On the client side, the web service is supportive to all data base operations where the job resources data change over and over. To have track of the changes the Restful servcies can go normal technique or the jQuery technique.

In the normal technique, the web service might manage the list of connected users and auto send to the users the changes where the user will periodically pool the service with HTTP GET or POST request and therefore pulling the current representation of the resource. If jQuery is disabled then the PHP will follow these steps

[1] Fetch HTTP GET or POST request from the webpage

[2] Set or check a cookie

[3] The request consist of unique IP address

[4] Store it in database

[5] Return the output via html file

In jQuery technique, the client are continuously been pulled by HTTP GET until JSON response. If jQuery is enabled then the PHP will follow these steps

[1] Fetch HTTP GET or POST request from the JS call

[2] The request consist of unique IP address

[4] Store it in database

[5] Return the output via JSON file

The HTML embeds a jQuery that continously pools the service. The pooling results through JSON file done by json_encode.

<u>Seletal function of Ajax for json format result file:</u>

```
$.ajax({

url:            url,

method:         'POST',

dataType:       'json',

contentType:    "application/json;      charset=utf-8",

cache:          false,

success:        function(resource){displayUser(resource.userlist);},

error:          function(resource){console.log(resource);},

complete:       function(){

                                setTimeout(function(){fetch()},2000);}

});
```

### 5.2 REST IN CONNECTION POOLING

To understand why REST in connection pool make a important role. Connection pools are built to avoid the expenses of creating expensive resources hence they are created and stored. They are used again and again by returning  them back to pool. This technique is utilized by database connections. Well over HTTP connection most of HTTP methods use threads which highligths the thread pooling concept. Object pooling and caching  are similar to this thread pooling where the objects are pooled and stored with the help of queuing. The objects are retrieved when required and as soon as the work is completed, they are returned back to the pool. REST is stateless instead objects are unique and hence it is possible to cache the objects based on their unique id. This makes REST in connection pool.

## 5.3 QUEUING AJAX REQUEST

A scenario for Ajax requests in web applications is queuing which plays important role when the objects are pooled. This means not eveytime the interface will represent a database state. While Ajax has a asynchronous interface  which mannerly updates the user interface before sending requests.

Suppose for instance, the user creates a record and as mentioned above it updates with sending the POST or request. Suddenly the user deletes then the delete request is sent to the server.The delete request is reponded before the POST request to create. Server throws an error. Appling race condition to queuing Ajax requests using POST with a queue flag

*jQuery.ajax({type:’POST’, queue:true});*

## 5.4 LOAD BALANCER

The web application is deployed using LAMP architecture where PHP and Linux support the roundrobin algorithm. It roundrobin does the iteration over the list of configured servers. In PHP the roundrobin filter is utilized to pick a server for statement execution. The plugin reaches the end of the list then it wraps to the start and picks configured server. With the configures server it also matters the priority to a server. Apache as webserver has HAProxy as the load balancer.

# Chapter 6

## 6 Testing

### 6.1 TESTING

The PuTTY interface and application server interface have to be tested. The reason for tesing on two different interface. The results might have success or failure for different commands on both the interfaces.

For example, the pbs command qhold shows different result on PuTTY interface on accesing the server and the application server interface show a different perspective on testing. The performance of the executing it on two different interface adds to time consumption.

### 6.2 AVOIDING SCRIPTED SQL ATTACKS

The database queries fired from class files have counter values and fires back on invalid access. For scripted sql injection attacks then the special characters are converted into HTML tags which class would inspect and verify on user specific input for the injections.

### 6.3 TESTING METHODS

- ✓ Integration Testing

  Testing to verify functional, performance and reliability requirements on software modules integration of putty interface using pbs command flows and application server interface making it available for hosting on job monitoring.

- ✓ Performance Testing

  Testing to determine the responsiveness, throughput, reliability, scalability of a application system under a given workload of single user job and multiple user jobs.

## 6.4 TEST SCENARIO: INTEGRATION TESTING FOR QSUB COMMAND

Test Case Description:

The job file sent by the user needs to be submitted by the qsub command. To test when the scenario of submitting a single job on the application server. It does not need an another interface like PuTTY to submit a job on the application server. It is userfriendly on montoring tool and the application is user specific. Suppose to create a job script file name *my_script* is to be submitted.

### 6.4.1 TEST CASE 1

Test data:

1. To create a *my_script* file with PBS standards

2. Using the qsub command the user needs to submit a single job is submitted on the application server and the user should submit own job i.e user specific submission.

   *qsub my_script*

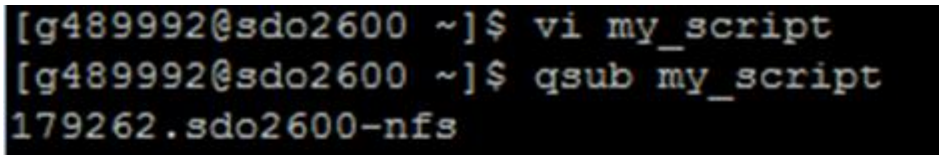
3. The user need to write the script name in the textbox and press submit button

Expected result:

The job must be submitted successfully and the output should be displayed on job monitoring tool

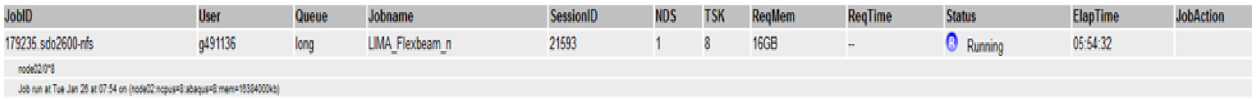
Actual result:

The user's job is submitted sucessfully and the output is displayed on job monitoring tool



```
[g489992@sdo2600 ~]$ vi my_script
[g489992@sdo2600 ~]$ qsub my_script
179262.sdo2600-nfs
```

FIGURE 14 PUTTY OUTPUT FOR QSUB



| JobID | User | Queue | Jobname | SessionID | NDS | TSK | ReqMem | ReqTime | Status | ElapTime | JobAction |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 179235.sdo2600-nfs | g491136 | long | LIMA_Flexbeam_n | 21593 | 1 | 8 | 16GB | -- | R Running | 05:54:32 | |
| node02/0*8 | | | | | | | | | | | |
| Job run at Tue Jan 26 at 07:54 on (node02:ncpus=8:abaqus=8:mem=15384000kb) | | | | | | | | | | | |

FIGURE 15: SERVER OUTPUT FOR SINGLE QSUB JOB

Conclusion:

The pbs operation of submitting a job is successful from the web application and does not need an another interface like PuTTY to submit a job on the application server. It is userfriendly on montoring tool and the application is user specific. The user has no more necessity to educate regarding usage of PuTTY and following with PBS commands.

### 6.4.2 TEST CASE 2

Test data:

1. To create multiple a *my_script* file with PBS standards

2. Using the qsub command the user needs to submit multiple job is submitted on the application server and the user should submit own job i.e user specific submission.

*qsub my_script*

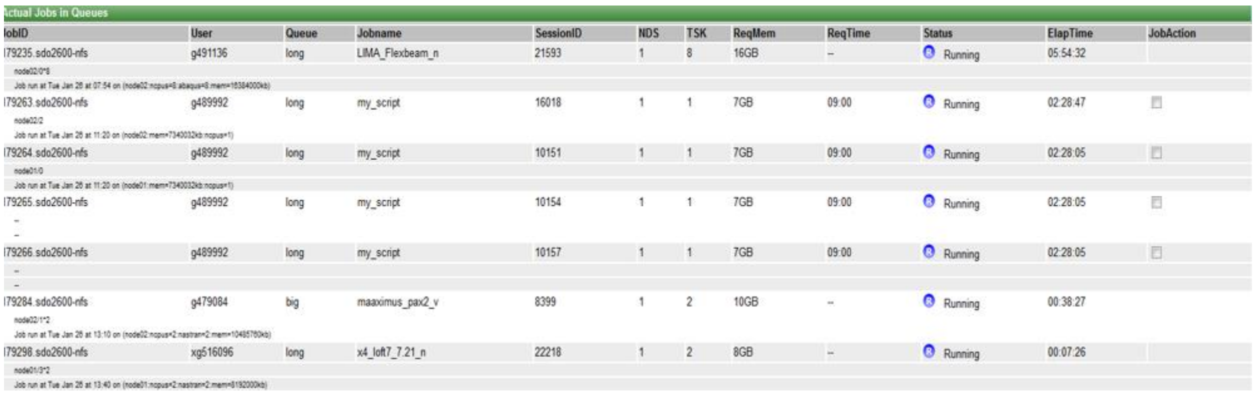
3.The user need to write the script name in the textbox and press submit button

Expected result:

The job must be submitted successfully and the output should be displayed on job monitoring tool

Actual result:

The user's multiple job is submitted sucessfully and the output is displayed on job monitoring tool but takes few seconds of time.



Actual Jobs in Queues

| JobID | User | Queue | Jobname | SessionID | NDS | TSK | ReqMem | ReqTime | Status | ElapTime | JobAction |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 179235.sdo2600-nfs | g491136 | long | LIMA_Flexbeam_n | 21593 | 1 | 8 | 16GB | -- | Running | 05:54:32 | |
| node02/0*8 | | | | | | | | | | | |
| Job run at Tue Jan 26 at 07:54 on (node02:ncpus=8:abaqus=8:mem=16384000kb) | | | | | | | | | | | |
| 179263.sdo2600-nfs | g489992 | long | my_script | 16018 | 1 | 1 | 7GB | 09:00 | Running | 02:28:47 | ☐ |
| node02/2 | | | | | | | | | | | |
| Job run at Tue Jan 26 at 11:20 on (node02:mem=7340032kb:ncpus=1) | | | | | | | | | | | |
| 179264.sdo2600-nfs | g489992 | long | my_script | 10151 | 1 | 1 | 7GB | 09:00 | Running | 02:28:05 | ☐ |
| node01/0 | | | | | | | | | | | |
| Job run at Tue Jan 26 at 11:20 on (node01:mem=7340032kb:ncpus=1) | | | | | | | | | | | |
| 179265.sdo2600-nfs | g489992 | long | my_script | 10154 | 1 | 1 | 7GB | 09:00 | Running | 02:28:05 | ☐ |
| -- | | | | | | | | | | | |
| -- | | | | | | | | | | | |
| 179266.sdo2600-nfs | g489992 | long | my_script | 10157 | 1 | 1 | 7GB | 09:00 | Running | 02:28:05 | ☐ |
| -- | | | | | | | | | | | |
| -- | | | | | | | | | | | |
| 179284.sdo2600-nfs | g479084 | big | maaximus_pax2_v | 8399 | 1 | 2 | 10GB | -- | Running | 00:38:27 | |
| node02/1*2 | | | | | | | | | | | |
| Job run at Tue Jan 26 at 13:10 on (node02:ncpus=2:nastran=2:mem=10485760kb) | | | | | | | | | | | |
| 179298.sdo2600-nfs | xg516096 | long | x4_loft7_7.21_n | 22218 | 1 | 2 | 8GB | -- | Running | 00:07:26 | |
| node01/3*2 | | | | | | | | | | | |
| Job run at Tue Jan 26 at 13:40 on (node01:ncpus=2:nastran=2:mem=8192000kb) | | | | | | | | | | | |

FIGURE 16: SERVER OUTPUT FOR MULTIPLE QSUB JOBS

Conclusion:

User can submit multiple jobs from the frontend without any interface required for submission of job. It is userfriendly on monitoring tool and the application is user specific.

### 6.4.3 TEST CASE 3

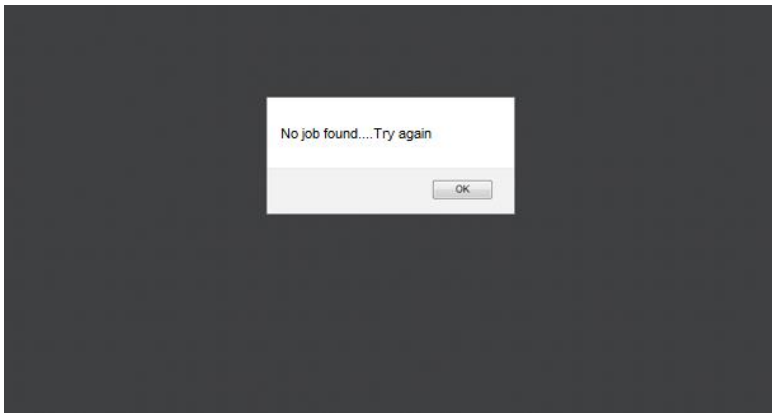
Test data:

1. To create no script file
2. No submission of jobs or submission of non existing jobs.

Expected result:

To display no data found for creation of jobs without validation control on actual job displayed because the page displays live data. Just a validation for user to understand there was no such file.

Actual result:


No job found....Try again

Conclusion:

On searching a non existing job the notification of "no job found….try again" is displayed and in the section on actual jobs displayed you won't view any result because there isn't one.

## 6.5 TEST CASE SCENARIO: INTEGRATION TESTING FOR QSTAT COMMAND

Test Case Description:

The job files sent by the user needs to be submitted by the qsub command and view it . To test when the scenario of displaying a single job or multiple jobs on the application server. It does not need an another interface like PuTTY to monitor a job on the application server. It is userfriendly on montoring tool and the application is user specific with an option of sorting as per status. Suppose to display a job script file name *my_script* is to be submitted.

### 6.5.1 TEST CASE 1

Test data:

1. To test monitoring a single job in the section of actuals jobs displayed
2. After successfully submission of job it should be brought to the frontend throught the application server response.

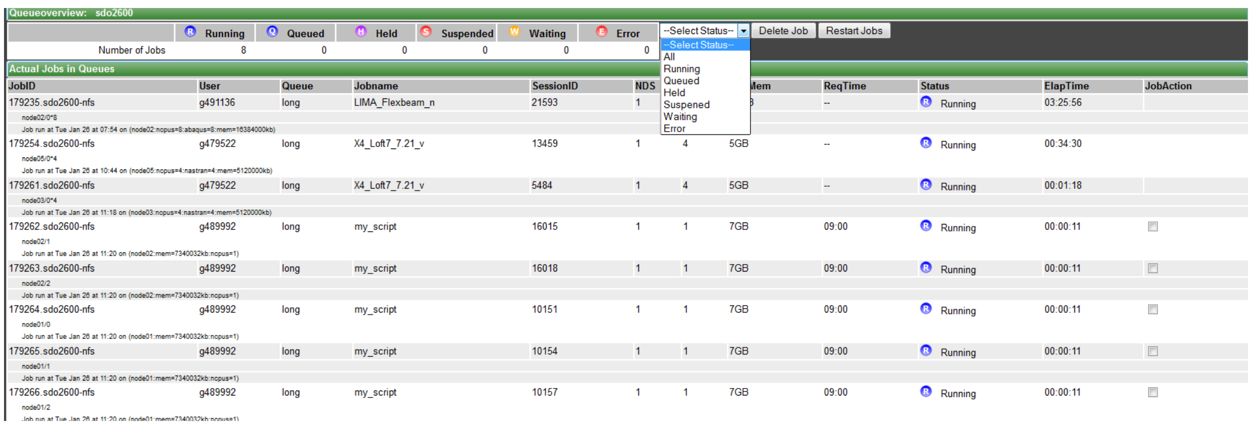
   *qstat -wans*

   In Putty the command is used which gives indetailed information about job, it's status.
3. Sorting according to the user selected status like running, held, queued, suspended, waiting, error.

Expected result:

To monitor a job as per it's status

Actual result:



Queueoverview: sdo2600

| | Running | Queued | Held | Suspended | Waiting | Error |
|---|---|---|---|---|---|---|
| Number of Jobs | 8 | 0 | 0 | 0 | 0 | 0 |

Actual Jobs in Queues

| JobID | User | Queue | Jobname | SessionID | NDS | | Mem | ReqTime | Status | ElapTime | JobAction |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 179235.sdo2600-nfs | g491136 | long | LIMA_Flexbeam_n | 21593 | 1 | [illegible] | [illegible] | -- | Running | 03:25:56 | |
| node02/0*8 | | | | | | | | | | | |
| Job run at Tue Jan 26 at 07:54 on (node02:ncpus=8:abaqus=8:mem=16384000kb) | | | | | | | | | | | |
| 179254.sdo2600-nfs | g479522 | long | X4_Loft7_7.21_v | 13459 | 1 | 4 | 5GB | -- | Running | 00:34:30 | |
| node05/0*4 | | | | | | | | | | | |
| Job run at Tue Jan 26 at 10:44 on (node05:ncpus=4:nastran=4:mem=5120000kb) | | | | | | | | | | | |
| 179261.sdo2600-nfs | g479522 | long | X4_Loft7_7.21_v | 5484 | 1 | 4 | 5GB | -- | Running | 00:01:18 | |
| node03/0*4 | | | | | | | | | | | |
| Job run at Tue Jan 26 at 11:18 on (node03:ncpus=4:nastran=4:mem=5120000kb) | | | | | | | | | | | |
| 179262.sdo2600-nfs | g489992 | long | my_script | 16015 | 1 | 1 | 7GB | 09:00 | Running | 00:00:11 | ☐ |
| node02/1 | | | | | | | | | | | |
| Job run at Tue Jan 26 at 11:20 on (node02:mem=7340032kb:ncpus=1) | | | | | | | | | | | |
| 179263.sdo2600-nfs | g489992 | long | my_script | 16018 | 1 | 1 | 7GB | 09:00 | Running | 00:00:11 | ☐ |
| node02/2 | | | | | | | | | | | |
| Job run at Tue Jan 26 at 11:20 on (node02:mem=7340032kb:ncpus=1) | | | | | | | | | | | |
| 179264.sdo2600-nfs | g489992 | long | my_script | 10151 | 1 | 1 | 7GB | 09:00 | Running | 00:00:11 | ☐ |
| node01/0 | | | | | | | | | | | |
| Job run at Tue Jan 26 at 11:20 on (node01:mem=7340032kb:ncpus=1) | | | | | | | | | | | |
| 179265.sdo2600-nfs | g489992 | long | my_script | 10154 | 1 | 1 | 7GB | 09:00 | Running | 00:00:11 | ☐ |
| node01/1 | | | | | | | | | | | |
| Job run at Tue Jan 26 at 11:20 on (node01:mem=7340032kb:ncpus=1) | | | | | | | | | | | |
| 179266.sdo2600-nfs | g489992 | long | my_script | 10157 | 1 | 1 | 7GB | 09:00 | Running | 00:00:11 | ☐ |
| node01/2 | | | | | | | | | | | |
| Job run at Tue Jan 26 at 11:20 on (node01:mem=7340032kb:ncpus=1) | | | | | | | | | | | |

FIGURE 17: SERVER OUTPUT FOR QSTAT

Conclusion:

On the monitoring tool, the qstat command is utilized to monitor a job and also sorted as per status.

### 6.5.2 TEST CASE 2

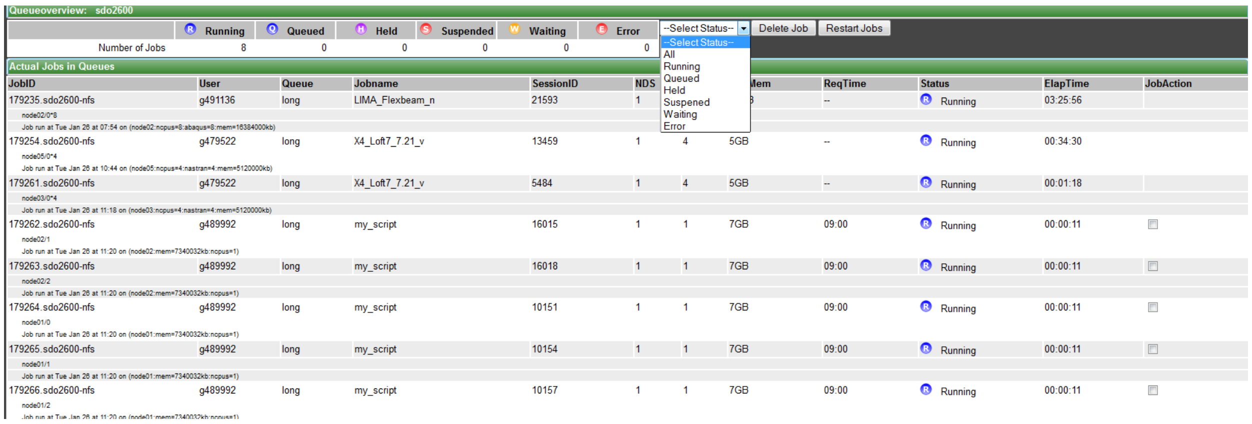
Test data:

1. To monitor multiple jobs in the section of actual jobs displayed.
2. After successful submission of multiple jobs, it should be visible on frontend.
3. Sorting according to status.

Expected result:

To monitor multiple jobs as per status.

Actual result:



Queueoverview: sdo2600

| | Running | Queued | Held | Suspended | Waiting | Error |
|---|---|---|---|---|---|---|
| Number of Jobs | 8 | 0 | 0 | 0 | 0 | 0 |

Actual Jobs in Queues

| JobID | User | Queue | Jobname | SessionID | NDS | | Mem | ReqTime | Status | ElapTime | JobAction |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 179235.sdo2600-nfs | g491136 | long | LIMA_Flexbeam_n | 21593 | 1 | [illegible] | B | -- | Running | 03:25:56 | |
| node02/0*8 | | | | | | | | | | | |
| Job run at Tue Jan 26 at 07:54 on (node02:ncpus=8:abaqus=8:mem=16384000kb) | | | | | | | | | | | |
| 179254.sdo2600-nfs | g479522 | long | X4_Loft7_7.21_v | 13459 | 1 | 4 | 5GB | -- | Running | 00:34:30 | |
| node05/0*4 | | | | | | | | | | | |
| Job run at Tue Jan 26 at 10:44 on (node05:ncpus=4:nastran=4:mem=5120000kb) | | | | | | | | | | | |
| 179261.sdo2600-nfs | g479522 | long | X4_Loft7_7.21_v | 5484 | 1 | 4 | 5GB | -- | Running | 00:01:18 | |
| node03/0*4 | | | | | | | | | | | |
| Job run at Tue Jan 26 at 11:18 on (node03:ncpus=4:nastran=4:mem=5120000kb) | | | | | | | | | | | |
| 179262.sdo2600-nfs | g489992 | long | my_script | 16015 | 1 | 1 | 7GB | 09:00 | Running | 00:00:11 | ☐ |
| node02/1 | | | | | | | | | | | |
| Job run at Tue Jan 26 at 11:20 on (node02:mem=7340032kb:ncpus=1) | | | | | | | | | | | |
| 179263.sdo2600-nfs | g489992 | long | my_script | 16018 | 1 | 1 | 7GB | 09:00 | Running | 00:00:11 | ☐ |
| node02/2 | | | | | | | | | | | |
| Job run at Tue Jan 26 at 11:20 on (node02:mem=7340032kb:ncpus=1) | | | | | | | | | | | |
| 179264.sdo2600-nfs | g489992 | long | my_script | 10151 | 1 | 1 | 7GB | 09:00 | Running | 00:00:11 | ☐ |
| node01/0 | | | | | | | | | | | |
| Job run at Tue Jan 26 at 11:20 on (node01:mem=7340032kb:ncpus=1) | | | | | | | | | | | |
| 179265.sdo2600-nfs | g489992 | long | my_script | 10154 | 1 | 1 | 7GB | 09:00 | Running | 00:00:11 | ☐ |
| node01/1 | | | | | | | | | | | |
| Job run at Tue Jan 26 at 11:20 on (node01:mem=7340032kb:ncpus=1) | | | | | | | | | | | |
| 179266.sdo2600-nfs | g489992 | long | my_script | 10157 | 1 | 1 | 7GB | 09:00 | Running | 00:00:11 | ☐ |
| node01/2 | | | | | | | | | | | |
| Job run at Tue Jan 26 at 11:20 on (node01:mem=7340032kb:ncpus=1) | | | | | | | | | | | |

FIGURE 18: SERVER OUTPUT FOR QSTAT MULTIPLE JOBS

Conclusion:

The qstat command display the results for multiple jobs. The clause for user logged on will display job action control. In case the job does not belong to user then the job action control is not displayed. In the backend, a warning is generated.

[Fri Nov 20 12:34:00 2015] [error] [client 53.144.71.16] PHP Notice: Undefined index: check_id in /var/www/html/PBSMon/includes/in_queuestatus.php on line 281

```
[Fri Nov 20 12:34:00 2015] [error] [client 53.144.71.16] PHP Notice:  Undefined index: check_id in /var/www/html/PBSMon/includes/in_queuestatus.php on line 281, referer: http://sdo2600/PBSMon/main.php?page=queuestatus&

[Fri Nov 20 12:34:00 2015] [error] [client 53.144.71.16] PHP Warning:  Invalid argument supplied for foreach() in /var/www/html/PBSMon/includes/in_queuestatus.php on line 281, referer: http://sdo2600/PBSMon/main.php?page=queuestatus&
```

### 6.5.3 TEST CASE 3

Test data:

1. Removal of all jobs

Expected result:

No jobs should be displayed.

Actual result:

No visible jobs in actual jobs section

Conclusion:

On no jobs or removal of all jobs then the actual status displays no jobs without any validation .Well this is because live data is monitored.

## 6.6 TEST CASE SCENARIO: INTEGRATION FOR QHOLD COMMAND

Test Description:

After the user submitted jobs and successfully hosted for monitoring, the job needs to be held. The scenario is to check the jobs are held by the qhold command and reviewing the result on helding single job, multiple jobs and held a non existing job. On using the qhold command these running jobs to check the change in status.

### 6.6.1 TEST CASE 1

Test data:

1.To held a single running job using the qhold command.

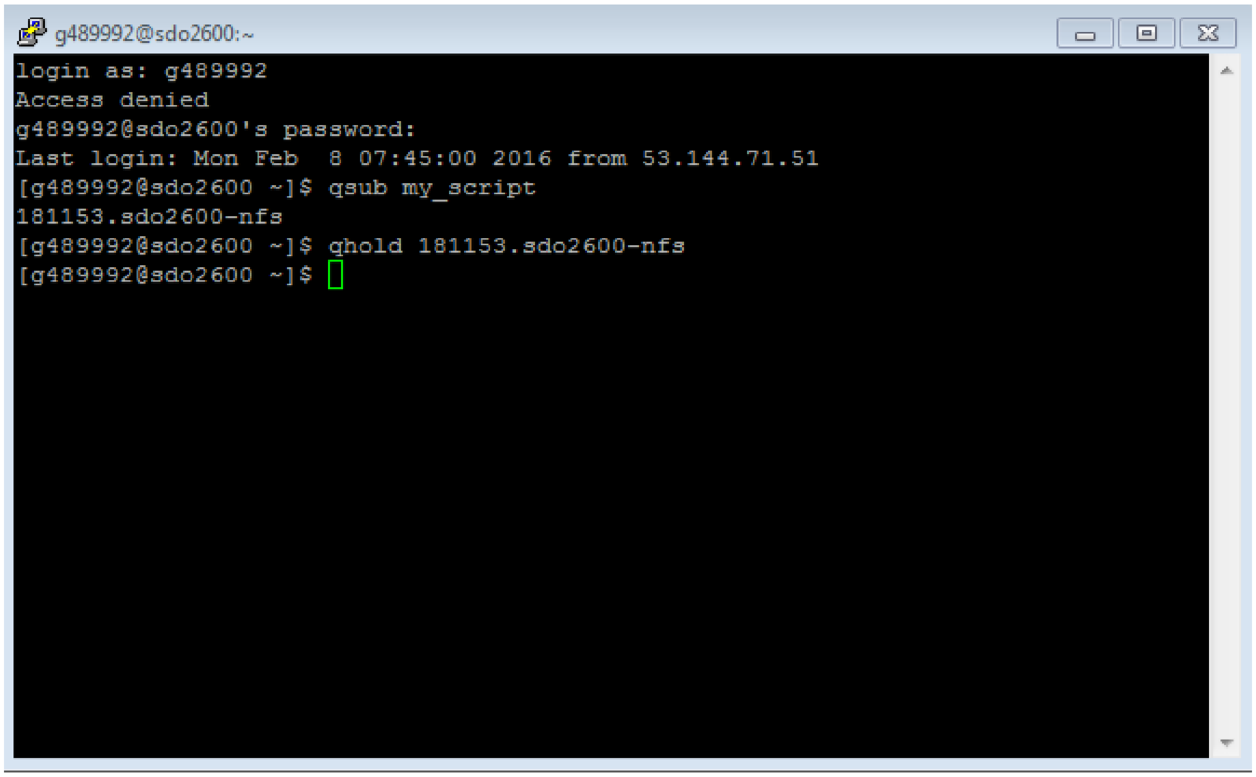
2. Using the qhold command user should be able to hold a running job on the application server and the user should be able hold own job .

3. The job should be in held state with the running status.

Expected results:

The job should be in held state with the running status for a single running job.

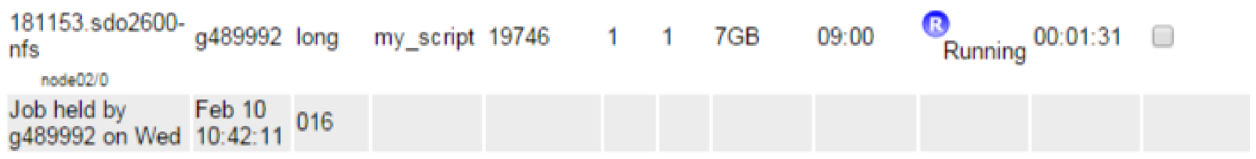
Actual result:



```
login as: g489992
Access denied
g489992@sdo2600's password:
Last login: Mon Feb  8 07:45:00 2016 from 53.144.71.51
[g489992@sdo2600 ~]$ qsub my_script
181153.sdo2600-nfs
[g489992@sdo2600 ~]$ qhold 181153.sdo2600-nfs
[g489992@sdo2600 ~]$
```

FIGURE 19: PUTTY INTERFACE ON QHOLD

Same job on application server when held displays status running but the job is held. The user has held his own job.



| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 181153.sdo2600-nfs<br>node02/0 | g489992 | long | my_script | 19746 | 1 | 1 | 7GB | 09:00 | Running | 00:01:31 | ☐ |
| Job held by g489992 on Wed | Feb 10 10:42:11 | 016 | | | | | | | | | |

**FIGURE 20: SERVER OUTPUT FOR QHOLD 1**

Conclusion:

To achieve successful execution the testing had two failurewhich are  until ssh was made passwordless to execute command and extra attribute with qhold command.

Failure 1:

Httpd connect se linux  setsebool -P httpd_can_network_connect ->on

```
[root@sdo2600 bin]# cat /tmp/qhold.log

command started with "g489992 180305.sdo2600-nfs"

Executing "ssh g489992@sdo2600 "qhold 180305.sdo2600-nfs""

OpenSSH_5.3p1, OpenSSL 1.0.1e-fips 11 Feb 2013

debug1: Reading configuration data /etc/ssh/ssh_config

debug1: Applying options for *

debug2: ssh_connect: needpriv 0

debug1: Connecting to sdo2600 [53.144.34.45] port 22.

debug1: connect to address 53.144.34.45 port 22: Permission denied

ssh: connect to host sdo2600 port 22: Permission denied
```

- Failure 2:

Test case failure as it command did not accept accept extra arguments

qhold [-h hold_list] job_identifier exception handled:

### 6.6.2 TEST CASE 2

Test data:

1.To held a multiple running jobs using the qhold command.

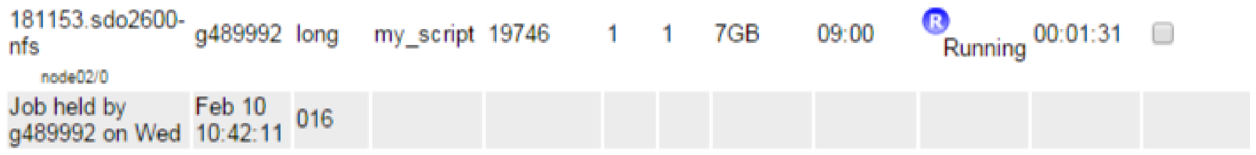
2. Using the qhold command user should be able to hold a running jobs on the application server and the user should be able hold own jobs .

3. The jobs should be in held state with the running status.

Expected results:

The job should be in held state with the running status for a multiple running jobs.

Actual result:



| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 181153.sdo2600-nfs<br>node02/0 | g489992 | long | my_script | 19746 | 1 | 1 | 7GB | 09:00 | R Running | 00:01:31 | ☐ |
| Job held by g489992 on Wed | Feb 10 10:42:11 | 016 | | | | | | | | | |

FIGURE 21: SERVER OUTPUT FOR QHOLD 2

Conclusion:

User can held multiple jobs from the frontend without any interface required for holding of jobs. It is userfriendly on monitoring tool and the application is user specific where the user can hold own jobs. The status is running for a job which are held.

### 6.6.3 TEST CASE 3

Test data:

1. To hold non-existing job
2. No submission of jobs or submission of non existing jobs and trying to held that job which does not exists

Expected result:

To display no data found for helding of jobs without validation control on actual job displayed because the page displays live data. Just a validation for user to understand there was no such file.

Actual result:

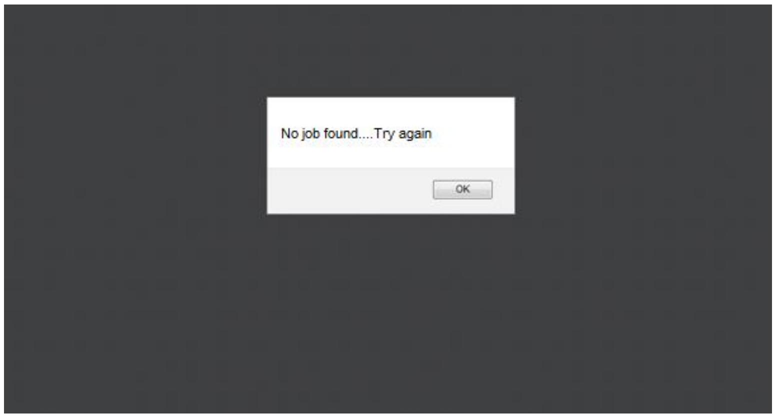

No job found....Try again

FIGURE 22: SERVER VALIDATION OUTPUT FOR QHOLD NON EXISTING JOB

Conclusion:

On searching a non existing job the notification of "no job found….try again" is displayed and in the section on actual jobs displayed you won't view any result because there isn't one for holding a job.

## 6.7 TEST SCENARIO: INTEGRATION OF QRLS COMMAND

Test Description:

After the held is applied on a running job and needs to release the job from the held status with the qrls command. The scenario is to check the job gets released on qrls command and checking the result for single job, multiple jobs and non-existing job. Checking the status and on released job the job should restart the same job giving it a fresh start.

### 6.7.1 TEST CASE 1

Test data:

1.To release a single running job using the qrls command.

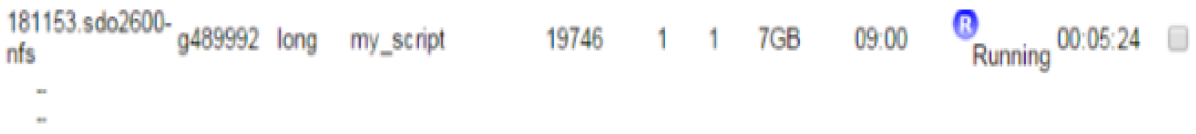
2. Using the qrls command user should be able to release a running job on the application server and the user should be able release his own job .

3. The job should be in release state with the running status.

Expected results:

The job should be in held state with the running status for a single running job.

Actual result:



```
181153.sdo2600- g489992 long my_script 19746 1 1 7GB 09:00 R Running 00:05:24
nfs
```

FIGURE 23:SERVER OUTPUT FOR QRLS 1

Conclusion:

User can release single job from the frontend without any interface required for releasing of jobs. It is userfriendly on monitoring tool and the application is user specific where the user can release own jobs. The status is running for a job which are released. The released job display no state because they are resubmitted.

### 6.7.2 TEST CASE 2

Test data:

1.To release a multiple running jobs using the qrls command.

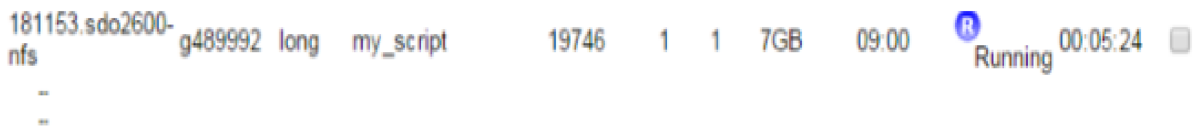
2. Using the qrls command user should be able to release a running jobs on the application server and the user should be able release own jobs .

3. The jobs should be in release state with the running status.

Expected results:

The job should be in release state with the running status for a multiple running jobs.

Actual result:



```
181153.sdo2600- g489992  long   my_script      19746   1   1   7GB    09:00   R Running 00:05:24
nfs
```

FIGURE 24:SERVER OUTPUT FOR QRLS2

Conclusion:

User can release multiple jobs from the frontend without any interface required for releasing of jobs. It is userfriendly on monitoring tool and the application is user specific where the user can release own jobs. The status is running for a job which are released. The released jobs display no state because they are resubmitted.

### 6.7.3 TEST CASE 3

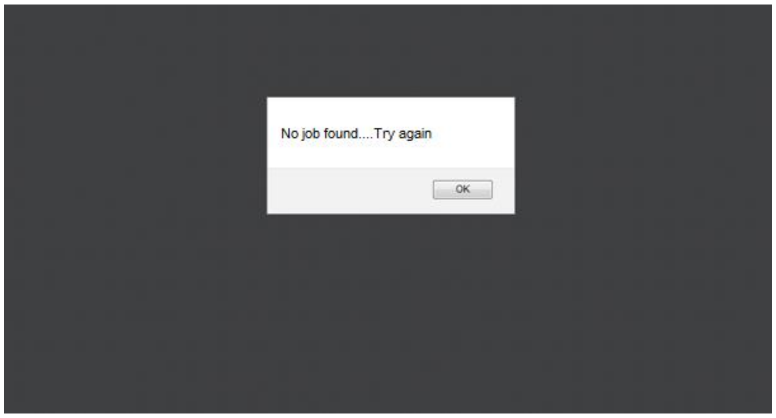
Test data:

1. To releasing non-existing job
2. No submission of jobs or submission of non existing jobs and trying to releasing that job which does not exists by qrls command.

Expected result:

To display no data found for releasing of jobs without validation control on actual job displayed because the page displays live data. Just a validation for user to understand there was no such file.

Actual result:



No job found....Try again

FIGURE 25: SERVER VALIDATION OUTPUT FOR QRLS NON EXISTING JOB

Conclusion:

On searching a non existing job the notification of "no job found….try again" is displayed and in the section on actual jobs displayed you won't view any result because there isn't one for releasin a job.

## 6.8 TEST SCENARIO: INTEGRATION OF QDEL COMMAND

Test Description:

Though a job is running or the job is held or the job is released, while the deletion command is applied on the job in which ever state the job is in it has to delete the job. On deletion the job should not be present in the frontend.

### 6.8.1 TEST CASE 1

Test data:

1. On newly created job or on held job, use job id with qdel command.

2. Using the qdel command the user needs to delete a single job which is submitted on the application server and the user should delete own job i.e user specific deletion of job

*qdel job_id*

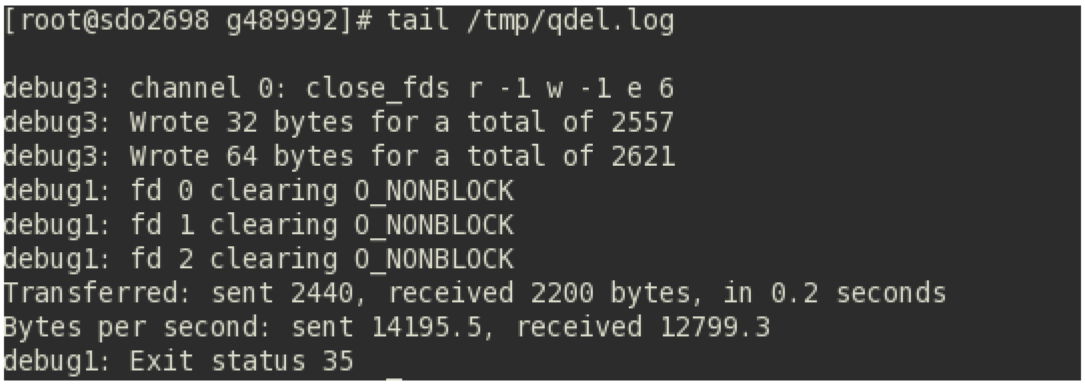
3.The user need to check the job_id name in the jobaction and press submit button

Expected result:

2. The job must be deleted successfully and the output should be displayed on job monitoring tool by being invisible anymore.

Actual result:

Successfully executed without no blocks



```
[root@sdo2698 g489992]# tail /tmp/qdel.log

debug3: channel 0: close_fds r -1 w -1 e 6
debug3: Wrote 32 bytes for a total of 2557
debug3: Wrote 64 bytes for a total of 2621
debug1: fd 0 clearing O_NONBLOCK
debug1: fd 1 clearing O_NONBLOCK
debug1: fd 2 clearing O_NONBLOCK
Transferred: sent 2440, received 2200 bytes, in 0.2 seconds
Bytes per second: sent 14195.5, received 12799.3
debug1: Exit status 35
```

FIGURE 26: SERVER DELETE LOG FILE OUTPUT

Conclusion:

On Failure because of SSH was bloking and resulted in qdel.log file and sucessful results on making it passwordless

```
[root@sdo2699 bin]# cat /tmp/qdel.log

command started with "g489992 180305.sdo2600-nfs"

Executing "ssh g489992@sdo2600 "qdel 180305.sdo2600-nfs""

OpenSSH_5.3p1, OpenSSL 1.0.1e-fips 11 Feb 2013

debug1: Reading configuration data /etc/ssh/ssh_config

debug1: Applying options for *

debug2: ssh_connect: needpriv 0

debug1: Connecting to sdo2600 [53.144.34.45] port 22.

debug1: connect to address 53.144.34.45 port 22: Permission denied

ssh: connect to host sdo2600 port 22: Permission denied
```

### 6.8.2 TEST CASE 2

Test data:

1. On newly created jobs or on held jobs, use job id with qdel command.

2. Using the qdel command the user needs to delete a multiple job which is submitted on the application server and the user should delete own jobs i.e user specific deletion of job

   *qdel job_id*

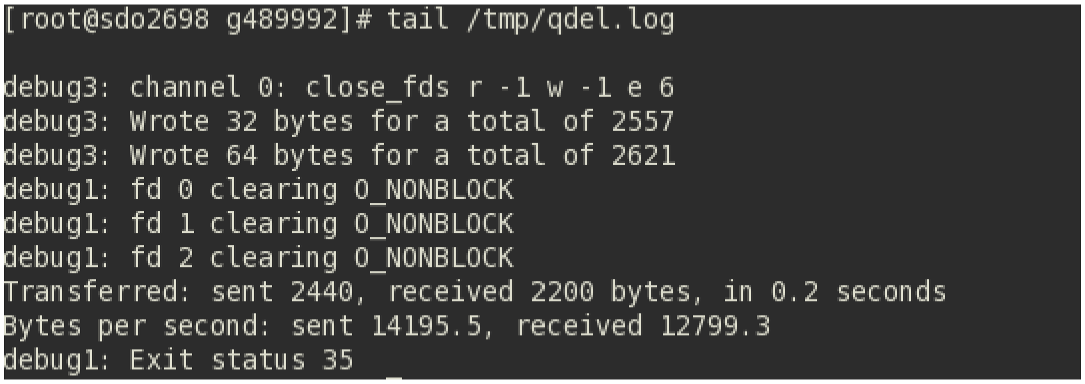
3.The user need to select the job_id in the checkbox of job action and press submit button.

Expected result:

2. The jobs must be deleted successfully and the output should be displayed on job monitoring tool by being invisible anymore.

Actual result:

Successfully executed without no blocks



```
[root@sdo2698 g489992]# tail /tmp/qdel.log

debug3: channel 0: close_fds r -1 w -1 e 6
debug3: Wrote 32 bytes for a total of 2557
debug3: Wrote 64 bytes for a total of 2621
debug1: fd 0 clearing O_NONBLOCK
debug1: fd 1 clearing O_NONBLOCK
debug1: fd 2 clearing O_NONBLOCK
Transferred: sent 2440, received 2200 bytes, in 0.2 seconds
Bytes per second: sent 14195.5, received 12799.3
debug1: Exit status 35 _
```

FIGURE 27: SERVER OUTPUT FOR QDEL.LOG

Conclusion:

User can delete multiple jobs from the frontend without any interface required for submission of job. It is userfriendly on monitoring tool and the application is user specific regardless of the status of the jobs.

### 6.8.3 TEST CASE 3

Test data:

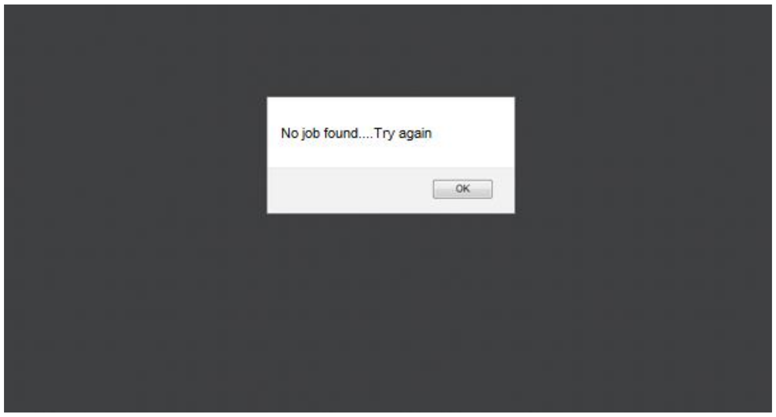
1. To deleting non-existing job
2. No submission of jobs or submission of non existing jobs and trying to delete that job which does not exists

Expected result:

To display no data found for deleting of jobs without validation control on actual job displayed because the page displays live data. Just a validation for user to understand there was no such file.

Actual result:

Result for non -existing jobs when tried to delete



No job found....Try again

FIGURE 28:SERVER VALIDATION OUTPUT FOR QDEL NON EXISTING JOB

Conclusion:

On searching a non existing job the notification of "no job found….try again" is displayed and in the section on actual jobs displayed you won't view any result because there isn't one for deleting a job.

# Chapter 7

## 7 Conclusion

In recent years, high performance computing has emerged as a solution to many enterprise issues that can be used to meet the continously growing processing requirements of today’s applications. The entire study has focused on the making PBS commands user friendly on monitoring platform such that if the user is new to the system then the user has no more the necessity to educate regarding usage of remote login and learn the PBS commands for performing operations on the their jobs.

The study regarding the implementation of PBS command with entire discussion how to make pbs commands web based from the monitoring frontend. The creation of command interface in PHP is focused by scripts which makes the commands work without obstacles. The concept of web based monitoring via PHP is new in the entire thesis by using the functionality of different PBS commands and web based application interface in PHP on LAMP achitecture using RESTful services.

1. High Performance LAMP-Lasituationen grosser Websites planen und meistern by Mirko Giese [Book]
2. [Internet] https://www.nagios.org/ - Infrastructure IT solutions
3. The RED HAT ENTERPRISE LINUX ADVANTAGE [Ebook] https://www.redhat.com/f/pdf/rhel/RHEL6_Advantage_WP.pdf
4. PuTTY- https://en.wikipedia.org/wiki/PuTTY [Internet]
5. PuTTY- http://www.chiark.greenend.org.uk/~sgtatham/putty/ [Internet]
6. [pdf] http://www.pbsworks.com/documentation/support/PBSProUserGuide11.2.pdf
7. [Internet] http://www.pbsworks.com/SupportGT.aspx?d=PBS-Professional,-Documentation
8. [Internet] PHP- https://en.wikipedia.org/wiki/PHP
9. [Internet] http://httpd.apache.org/docs/current/mod/mod_rewrite.html
10. [Internet]http://linux-blog.anracom.com/2014/10/13/character-sets-ajax-php-json-decode-encode-strings-properly/
11. [Internet] https://blog.alexmaccaw.com/queuing-ajax-requests
12. [Internet] Ajax (programming) - Wikipedia, the free encyclopedia_files
13. [Internet] Altair_ Using Simulation Technology to Synthesize and Optimize Designs, Processes and Decisions_files
14. [Internet] Apache mod_rewrite - Apache HTTP Server Version 2.4_files
15. [Internet] HTTP long poll example in php ajax ← ABrandao.com_files
16. [Internet] HTTP persistent connection - Wikipedia, the free encyclopedia_files
17. [Internet] Internet Message Access Protocol - Wikipedia, the free encyclopedia_files
18. [Internet] jQuery.ajax() _ jQuery API Documentation_files
19. [Internet] JSON_files
20. [Internet] LAMP (software bundle) - Wikipedia, the free encyclopedia_files
21. [Internet] Lightweight Directory Access Protocol - Wikipedia, the free encyclopedia_files
22. [Internet] Nagios - The Industry Standard In IT Infrastructure Monitoring_files

23. [Internet] PHP - Wikipedia, the free encyclopedia_files
24. [Internet] PHP_ exec - Manual_files
25. [Internet] Post Office Protocol - Wikipedia, the free encyclopedia_files
26. [Internet] PuTTY - Wikipedia, the free encyclopedia_files
27. [Internet] PuTTY_ a free SSH and Telnet client_files
28. [Internet] UTF-8 - Wikipedia, the free encyclopedia_files
29. [Ebook] Hack Proofing Linux
30. [pdf] 115 <myJAM/> –AccountingundMonitoringauf Rechenclustern
31. [ebook] GNU-Linux-Tools-Summary
32. [ebook] PBSProUserGuide11.2
33. [Ebook] Redhat Linux Rhce Cramsession

## The Affidavit

I certify that this thesis  does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any university: and that to the best of my knowledge and belief it does not contain any material previously published or written by another person where due reference is not made in the text.

Hof, 22.06.2016

-----------------------------

Shruti Mahesh Kulkarni