# Dynamically switch/provision clusters on Academic Cloud (Nov 29th)

# Goal

1. Use provided FutureGrid Dynamic Provisioning infrastructure to provision/switch between Bare Metal and Virtual Machine environments.
2. Modify and use messaging-based monitoring software developed in project #2, to monitor the CPU/Memory usage of MPI PageRank Program and overall performance on the dynamic provisioned cluster (with running 2 physical/virtual nodes), if you haven’t done it correctly.
3. Run Project #1 MPI/MPJ PageRank with 1M+ data points in the bare-metal and virtual clusters.
4. Automate the process of setting up the monitoring system and running of MPI Pagerank using Torque PBS job scripts.

# Deliverable (Due Nov 29th)

You are required to turn in following items in a zip file **groupID\_Project4.zip** in this assignment via Oncourse, where **groupID** (e.g. 01) is the two digital number of your group.

1. Source codes of Torque job scripts and executable or source code related to these scripts. Mainly, you are providing a “one-click” solution for each mode. Where your scripts must do the following
   1. Be able to obtain Bare Metal and Virtual Machines (2 independent scripts)
   2. Start Monitoring system and report the performance to the broker
   3. Run MPI/MPJ PageRank with a large enough data (1M+ data points) for 1 min
   4. Terminate MPI/MPJ PageRank and the messaging publisher, exit the BM nodes / Shutdown VMs at the end of the job
2. Source code of modified resource monitoring system, which you will need to change the code to show performance of “Average CPU Usage of MPI PageRank Processes on running nodes”, “Average Memory Usage of MPI PageRank Processes on running nodes”, “Overall average CPU usage on running nodes” and “Overall average Memory usage on running nodes” on your GUI if you haven’t done it for project #2.
3. A README.txt with detailed instructions on how to configure and run this system. Keep the steps simple by using independent script for building and running the programs of Bare Metal and Virtual Machines mode.
4. A report (**groupID\_p4report.docx**) describes the technologies, system architecture, dataflow charts and any interesting findings, and snapshots of your monitor GUI.
5. Demo in class

Points will be reduced (maximum 0.5 point) if the filename or directory structure are different from instructed above.

# Evaluation

The total points of project #4 is 10, where the distribution is as following

* 1. Completeness of your job script (6 point)
  2. Correctness and Completeness of written report (3 point)
  3. Readability and clarity of README.txt (1 point)

# Introduction

Dynamic Provisioning provides the ability and possibility to use on-demand resources in a shared academic Cloud environment; user of this system can simply send a request with specifying their needs to the resource manager [1] to obtain different kinds of computing resources, where the requested computing resources are deployed or instantiated on-the-fly. For example, users of FutureGrid can easily obtain a set of Bare Metal machines from Torque resource manager or boot up a set of Virtual Machines with using India-Eucalyptus.

With the support of FutureGrid System Admin team, we have built a Dynamic Provisioning system which can switch between Bare Metal and Virtual Machine compute cluster environments utilizing XCat, Moab and Torque job scheduler. For this project, you need to understand the system architecture, and implement job scripts to run your MPI PageRank and Monitoring system using the provided dynamic infrastructure.

# System Architecture

Based on the information received from the monitoring infrastructure, users will programmatically switch/re-provision their nodes to another environment (eg: from Linux to Linux VM’s). Figure 1 shows the interactions between components within this system.

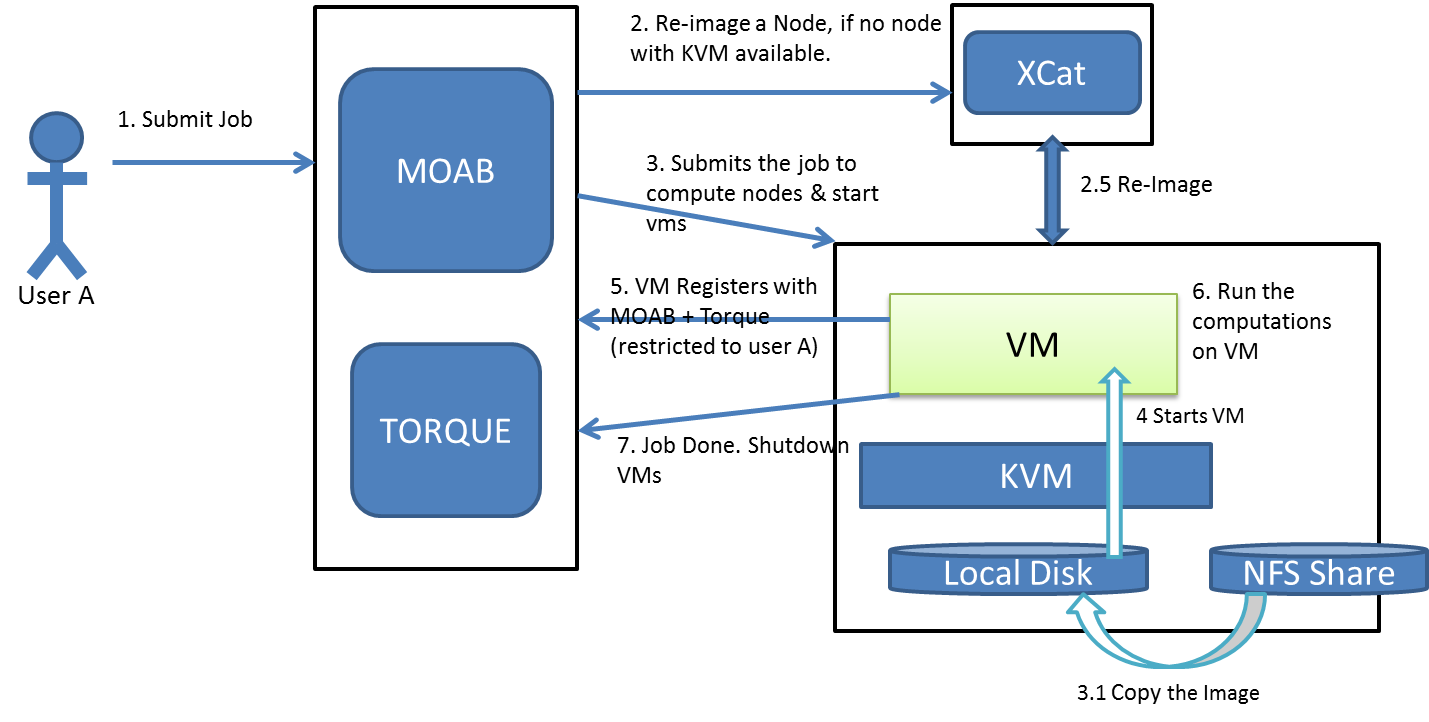


Figure 1 User interactions with Dynamic provisioning system

We have a set of nodes with the above infrastructure reserved for this call under a special job queue (b534) in india.futuregrid.org. Since the number of nodes is limited and the user jobs will be going through the queue, it’s advised that you start working on this assignment soon, to avoid any contention of resources towards the assignment deadline.

Below sections show two main types of jobs.

# 3.1 Batch jobs

With batch jobs, you’ll need to submit a PBS job script, which will wait in the queue and get executed when the resources are available. Refer to [5] for more information on job submissions. Inside the job script, you can obtain a list of compute resources that are assigned to you, using $PBS\_NODEFILE. Refer to section 2.1.6 in [5] to find out the other information you can obtain from PBS. With these information, you can execute your commands accordingly, similar to a shell script. However, you’ll have to add several PBS job directives to your job script to inform your requirements to the PBS job scheduler. Following is an example job script. Lines starting with #PBS are the job directives. “-q” specifies the job queue to be used, “-l nodes” let you specify the type and number of compute resources you require, “-l walltime” specify the estimated upper bound duration for your job.”-o” specify the file to write the output of the job screen printing. There is another error screen printing file generated by the torque service at the directory you submitted the job, which has a filename of ScriptsFileName.e$PBS\_JOBID. These two files will help you to debug your scripts.

We here show a template script specifies the hostname of two nodes, i71 and i72, which is useful if user is assigned to particular resources. You will see an attachment about your node assignment, PLEASE DO NOT USE NODES other than what you have been assigned. Resource assignment is listed here [].

#!/bin/bash

#PBS -q b534   
#PBS -l nodes=i71:ppn=8+i72:ppn=8  
#PBS -l walltime=01:00:00

#PBS -o run\_pagerank\_bm$PBS\_JOBID.out

cat $PBS\_NODEFILE

# (start your MPJ daemons and) run the MPI/MPJ pagerank and monitoring application in Bare Metal Cluster  
# =========   
# your code here  
# =========

echo "Bare Metal job Finished"

**Script 1. Request 2 nodes with specifying their resources**

# 3.2 Interactive jobs

With interactive jobs, you can get an interactive console for the nodes reserved for your job. Interactive jobs will also wait in the queue, similar to batch jobs. You are encouraged to use interactive jobs to get familiar with the environment and to test your programs. After that, however, you will switch to a batch job mode in the cluster. Following is an example of an interactive job submission. Refer to [4] for more information.

eg: qsub -I -l nodes=i71:ppn=8+i72:ppn=8,walltime=01:00:00 -q b534

This will get you a single node with 8 processors per node for 1 hour (wallclock) using the b534 queue. Note that your queue priority and allocation will also depend on the wall clock time you provide. Hence be conservative with the wall clock times.

# 3.3 Provisioning Bare Metal model

You can write a simple job script as in section 3.1 to get bare metal resources. You can also use PBS interactive jobs to obtain them when testing the environment.

# 3.4 Provisioning VM Mode (with Script)

To obtain a VM cluster, you need to write a similar job script that includes the commands to start the virtual machines in the nodes reserved for your job. You can use “/user/local/bin/start\_vms” command to stage and start VM’s in all of your nodes. Then you can use “/usr/local/bin/wait\_for\_vms $VM\_NODEFILE” to wait till the VMs become reachable. After that you can run your tasks in the VMs. Make sure to run “/usr/local/bin/shutdown\_vms” at the end of your job. You’ll be penalized for not shutting down the VM’s. You can also issue the above commands manually, after getting nodes through an interactive job. Always make sure to have the “ppn=8” for VM jobs, so that you’ll receive the whole node and avoid any conflicts.

#!/bin/bash

#PBS -q b534  
#PBS –V  
#PBS -l nodes=i71:ppn=8+i72:ppn=8  
#PBS -l walltime=01:00:00  
#PBS -o run\_pagerank\_vm$PBS\_JOBID.out

# Note that we are actually sourcing (. Script) the following script to make/get the environment variables. otherwise you won’t get the $VM\_NODEFILE  
# environmental variable.

. /usr/local/bin/start\_vms

echo "Waiting for the VM's to be reachable"

. /usr/local/bin/wait\_for\_vms $VM\_NODEFILE

cat $VM\_NODEFILE

# If you want to access the VM\_NODEFILE from a shell script you first have to export it as a different   
# environmental variable.

export NODES=$VM\_NODEFILE  
cat $NODES

# (start your MPJ daemons and) run the MPI/MPJ Pagerank and monitoring application in VM's headnode  
# e.g. ssh $headnode "mpjboot machines"  
# =========   
# your code here  
# =========

echo "shutting down VM's"  
. /usr/local/bin/shutdown\_vms

echo "VM Job finished"

# 3.5 Scripts

As we mentioned above, you are free to use PBS interactive jobs to setup and test your system. However the final deliverable should be 2 job scripts which will perform the following steps in the 2 environments (Bare Metal & VM) respectively.

**Bare Metal Job**

1. Start the monitoring system in all nodes
2. Configure and run the MPI\_Pagerank on all nodes for 1 mins.

**VM Environment job**

1. Start VM’s
2. Wait for VM’s to be reachable
3. Start the monitoring system on all VM’s from their headnode (one of the VM nodes)
4. Configure and run the MPI\_PageRank on VM’s for 1mins.
   1. If you are running with MPI C/C++ PageRank, only for the VM job, please add “--mca btl\_tcp\_if\_exclude lo,eth1” to the mpirun / mpiexec command, if not, mpi will be hanging without any response due to the VM network setting:

mpirun --mca btl\_tcp\_if\_exclude lo,eth1 -np 8 --hostfile nodes mpi\_main …

1. Shutdown the VM’s.

You must be able to monitor the resource usage and the performance metric should be average of CPU and memory usage for all nodes (VM and Bare Metal) that run MPI Pagerank jobs. In addition, You have to use 2 nodes for the final scripts.

# 3.6 Report

In addition to the programming part, you are required to investigate the system architecture of the provide infrastructure with understanding the methodology behind the screen. After that, based upon your observation, you should include thoughtful findings according to the provided system. Here are some ideas for you:

* Look into scripts when obtained nodes in the range of i71 ~ i100 of /usr/local/bin/start\_vms, /usr/local/bin/shutdown\_vms, /usr/local/bin/wait\_for\_vms. You may come up questions like “What are the functions of these scripts?”
* What is the different between this infrastructure and other Virtual Management Infrastructure (e.g. Eucalyptus)?

# Notes for MPJ PageRank

When starting the mpj daemon, for this project, “mpjboot machines” command may be useful other than the way we previously use with “mpjdaemon\_linux\_x86\_64 start”. Also, as far as I tested the VM environment, it is better to sleep at least 30 seconds in between starting MPJ daemons and run the program. This is due to some delay of JVM startup time inside the VM.

# Reference

1. TORQUE Resource Manager <http://www.clusterresources.com/products/torque-resource-manager.php>
2. KVM Hypervisor <http://www.linux-kvm.org/page/Main_Page>
3. libvirt: The virtualization API <http://www.libvirt.org/>
4. Torque Qsub: <http://www.clusterresources.com/torquedocs21/commands/qsub.shtml#I>
5. Torque Job submission: <http://www.clusterresources.com/torquedocs/2.1jobsubmission.shtml>
6. Resource Schedule: <https://docs.google.com/spreadsheet/ccc?key=0AtR8aHmmVF3ydGdKRk9Tb19HMVpQV25HOWctcVRoNXc#gid=0>