

Application Packaging and Checkpointing on the Grid with VMWare

Reid Phillips <rpx01@uark.edu>, Matt Baker <mmb05@uark.edu>

University of Arkansas

Abstract

As the available number grid pools proliferates and subsequently grid computing grows in popularity the ability to coordinate local resources with remote ones becomes even more critical than before. Existing tools such as the Globus Toolkit and Condor perform scheduling and system control for remote resources easing the process of communication and coordination for end users [1,2]. While these tools do facilitate the movement of jobs between sites it is left to the user to ensure that the job will be able to run on the remote system. This can become an arduous task as more often than not each site is unique with regard to what applications are installed and even the paths by which these applications are accessed. To guarantee that a user's job will run, said user must also package all required applications necessary for the job which can become complicated (e.g. packaging a Java virtual machine along with a Java application) [3]. The method proposed in this paper would alleviate much of the work required by a user to submit a job.

Virtual machines have been used for many purposes since their inception. This paper discusses another viable purpose for virtual machines, specifically EMC's VMWare products: VMWare Player and VMWare Server [4]. VMWare products are typically used as sandboxes for development and testing of new or improved software. By providing a cloistered environment VMWare isolates a program within a sandbox so that the local system is unaffected by any detrimental, unintentional as it may be, activity. Also an image

of the VMWare environment can be saved locally, shipped to a remote location, and run by anyone else with the software. This concept is ideally suited for grid computing because now a user need only set up an environment within a local VMWare sandbox and then ship the image to a remote resource and schedule it to run via GT or Condor. All that is required is that both sites have the necessary VMWare packages installed. Also for sites with time restrictions on running jobs the current state of a VMWare environment can be saved when a job is terminated and can immediately be resubmitted and eventually resumed from its previous state rather than started over again. This paper discusses the procedures and tools necessary to enable these features.

Introduction

As grid computing continues to develop, new and improved tools become available to facilitate the process of running applications in a remote environment. Some of these tools previously exist in other arenas but can be adapted to provide functionality in a grid environment as well. The topic discussed in this paper is just such a scenario. EMC's VMWare products are typically used to create a sandbox on a local system for application development and testing [4]. The ability of VMWare products to pause a running system, take a snapshot of the system and save it as an image, and finally resume operation based entirely on the image is well suited for running remote jobs in a grid environment. In fact this capability eases the requirements of a

user who was previously required to perform the error prone process of packaging all of the necessary applications required for their application to run into a single job.

Using the process laid out in this paper, all that is required is that a user set up the environment, save the image, ship the image to the remote resource, and start the image on the remote VM. Now the user is only required to send a single image as a job rather than a bundle of applications. Other features of this technique will also be discussed in further detail.

Configuration

In order for the virtual machines to be easily scheduled by a cluster or grid job scheduler, images need to be executable from the command line. Most free virtualization tools, such as VMWare Player, only provide execution control via a graphical user interface, which complicates automation via a scheduler.

VMWare Server, a free virtualization package, was installed on each node of the cluster in order for VMWare images to be executed on a host machine from the command line. VMWare server installs specialized network drivers that allow virtual machines to communicate over the physical machine's network. The software also includes command line administrative tools for starting, suspending, stopping and checking the status of virtual machines on the computing resource.

After installing the VMWare Server software on the cluster nodes, a virtual machine image was created to serve as a template for future applications. A minimal console-only version of Debian Linux was installed on the image by using the image creation tools included with VMWare Server.

The operating system on the image was then configured to automatically execute a given application at startup, and upon the application's completion, would shut down, terminating the virtual machine's execution. This creates a simple method of job execution, requiring a minimum amount of internal state knowledge of the running virtual machine.

Scheduling and Condor

In order to automate the transfer and execution of VMWare virtual machine images, a shell script was created that starts a virtual machine, monitors its execution, and suspends execution if a special kill signal is trapped, or received, by the script (see Figure 1).

```
#!/bin/sh
/usr/bin/vmware-cmd $1 start

trap "/usr/bin/vmware-cmd $1 suspend hard;
exit 0" SIGUSR1 SIGQUIT SIGTERM

i=0

while [ $i -le 0 ]
do
  j=`/usr/bin/vmrun list | grep $1 | wc
-l`
  if [ $j -eq 0 ]; then
    i=100
  fi
  sleep 1
done
```

Figure 1. Shell script called to start, suspend, and monitor virtual machines.

In order to run a virtual machine named "debian.vmx", for example, users would type the following command:

```
./vmstart.sh /path/to/file/debian.vmx
```

This script is necessary because the VMWare tool "vmware-cmd" is non-blocking, and even though the tool has completed execution, the virtual machine remains in running

mode, without the local job scheduler's knowledge. By executing this script, local job schedulers can monitor the execution of the virtual machine by waiting for the successful termination of the script.

The “trap” command is crucial to the check-pointing feature of the system. When the local scheduler (in this example, Condor) is ready to suspend or kill a job, it will send a kill signal to the running script. The script intercepts the kill signal, inspects it, and if it is a special type of kill signal, such as SIGUSR1, the VMWare suspend command is issued to the running virtual machine. This can be tested by using the “kill -20 jobid” command, where *jobid* is the ID of the *vmstart.sh* script.

Finally, the Condor job submit description file, or ClassAd, is created. This file tells the local Condor job scheduler the names of the executables and files to transfer to the appropriate idle compute node (See Figure 2).

```
Universe = vanilla
Executable = vmstart.sh
Arguments = /home/mmb05/xorgate/other.vmx
KillSig=SIGUSR1
Log = vmware.log
Output = vmware.out
Error = vmware.error
Queue
```

Figure 2. VMWare ClassAd for Condor.

The “KillSig” variable allows a Condor user to specify which type of kill signal to send to the job should Condor need to preempt or kill it.

If it is unsuccessful at killing the job with the user-specified signal for a period of time, Condor will issue the kill 9 signal, forcing the script to terminate immediately without suspending the running virtual machine.

Results

The *vmstart.sh* script successfully starts and suspends running virtual machines when called directly from the command line. By issuing a separate *kill -20* command in another terminal, the script suspended the running virtual machine, saving the internal memory state and terminating itself.

Condor also successfully transfers the disk image and starts the sample virtual machine, using the ClassAd in Figure 2. Condor also sends the correct kill signal to the *vmstart.sh* script. However, Condor did not give adequate time for the script to suspend the virtual machine, terminating the script and leaving the virtual machine in the running state. Further work is needed to determine the necessary configuration changes to allow more time work custom kill commands under Condor.

Future Work

In addition to the Condor reconfiguration mentioned above, more work is needed to adapt the local installation of VMWare to work on the grid. While simple changes to the Condor ClassAd should suffice to allow execution on remote grid resources, custom ClassAd variables maybe necessary to determine whether a particular remote resource has VMWare Server installed (in the above experiment, VMWare Server is assumed to be installed on all nodes in the system).

Also, while the *vmstart.sh* script is monitored for CPU and memory usage, the metrics of the running virtual machine are currently unknown to the job scheduler. If a virtual machine initiated by Condor utilizes 90% of the CPU, Condor will not report this, since it only sees the CPU utilization of the wrapper script

(which runs at an average of under 1%). There are VMWare utility applications that, once installed inside of the virtual machine, can report metrics such as CPU and network utilization back to the host operating system and local job scheduler.

References

[1] "The Globus Alliance," The Globus Alliance, 2006, HYPERLINK "<http://www.globus.org/>"

[2] "Condor Project Homepage," University of Wisconsin, 2006, HYPERLINK "<http://www.cs.wisc.edu/condor/>"

[3] "Java Technology," Sun Microsystems, 2006, HYPERLINK "<http://java.sun.com/>"

[4] "VMware: Virtualization, Virtual Machine & Virtual Server Consolidation," VMware and EMC Company, 2006, HYPERLINK "<http://www.vmware.com/>"

[5] S. Santhanam, P. Elango, A. Arpaci-Dusseau, M. Livny, "Deploying Virtual Machines as Sandboxes for the Grid," University of Wisconsin, 2006