Reducing Heterogeneity in Volunteer Computing using Virtual Machines

Kevin Reed
University of Illinois Graduate Student
reed21@illinois.edu

Abstract: Volunteer computing derives its processing power from many computers volunteered by the public. Obtaining processing power in this fashion means that there will be large variations in the operating system, operating system version, installed software and library versions on the different contributing computers. This heterogeneity creates many problems for the administrators of a volunteer computing project. This paper looks at one particular problem this heterogeneity causes and reviews a prototype that solved this problem using virtual machines.

1. Introduction

Volunteer computing is a type of grid computing where the computers that process work for the grid are volunteered by members of the public. As the computers are not controlled by the person or organization conducting processing on the grid, the volunteered computers cannot be trusted. In order to produce reliable results from untrusted computers, volunteer computing systems such as BOINC provide mechanisms to support redundant computing.[1] Redundant computing is a technique which sends each unit of work to at least two computers. The results from those computers are returned to the server and compared to see if the results agree. If they do not, then additional copies are sent to different computers until a consensus can be reached. Results can be compared to see if they are either identical or within an acceptable range of variance.

Volunteered computers are extremely diverse in terms of the operating system that is run, operating system version, available memory, processor manufacturer and model.[2] In addition, some research applications are sensitive to minor variations.[3] Small differences can compound and yield large differences in the final result. One of the sources of this problem is due to differences in compilers and optimizations settings chosen.[4] This creates a problem for validating results returned from different computers.

One way to address this problem is to reduce the variations between the machines. Virtual machines are an ideal candidate to help reduce the variation between machines and standardize the execution environment for the grid even on diverse machines.

This paper briefly reviews the problem of differing floating point computations as described by Lopez, Taufer and Teller [3]. It then discusses the requirements that are required for a virtual machine to be useful on a volunteer computing grid. Next, it discusses the prototype that was constructed . Finally, it discusses how this could be integrated into the BOINC software, other work being done in this area and where this work could be extended in the future.

2. Demonstration of problem

The problem described in the Lopez, Taufer and Teller paper is illustrated in the following example. I developed a small application which estimates the value of pi by summing the first 1000 terms of the Leibniz series. I then compiled this application using Microsoft Visual C++ 2008 Express Edition on Windows XP and gcc version 4.3.1 on Open Suse 11. I used 2 different compiler optimization options on each platform. Table 1 shows the value of pi at after 1the first 000 terms of the series have been summed.

Operating System	Optimization	Estimated Pi/ Difference
Windows XP	Disabled (/Od)	3.140592575 0.001000079
Windows XP	Full (/Ox)	3.140592575 0.001000079
Open Suse 11	None Set	3.140592575 0.001000079
Open Suse 11	-O3	3.140592860 0.000999793

Table 1. Results for PI with different compilers and optimization settings

In the fourth case, the value of pi has a small difference from the other cases. Although this is a small difference, certain applications can compound this difference, or other similar differences, to generate large differences in result values.[3]

3. Software Selection

In order for virtual machines to be used widely on volunteer computing projects, the technology needs to meet the following criteria:

1. The virtualization software must be available at no cost to the volunteers
2. The guest operating system must be available at no cost to the volunteers
3. The virtual machine should run unobtrusively on the volunteers machine
4. There must exist an management interface that will allow the BOINC client to insert and remove files, start and monitor applications in the guest, and start and stop the virtual machine
5. It must be straightforward for volunteer to install and maintain the virtual machine monitor software

Based on this criteria, I choose to create a prototype using the VMware Server 2 using Windows as the host and Linux as the Guest. Both VMware Server 2 and Linux are available at no cost to the volunteers. This meets criteria 1 and 2 above. VMware Server 2 allows you to launch a virtual

machine with no visible interface (an interface can be started via the admin console but it is not required) which meets criteria 3. The VIX libraries (http://www.vmware.com/support/developer/vix-api/) and VMware Tools provide an interface that can be used to control the operations of a virtual machine. This meets criteria 4. Unfortunately, there is a some complexity involved in installing and setting up VMware Server so criteria 5 was only partially met.

4. Prototype

In order to demonstrate that the a virtual machine can solve the problem describe earlier, the prototype will need to run the Linux binary within a previously prepared virtual machine and generate a value consistent the Linux binary running on native Linux. Additionally, in order to demonstrate the suitability of running on a volunteer computing project, the prototype needs to demonstrate the following capabilities:

1. Manage the Image: use a provided virtual machine image, but allow for a subsequent task to also use the original unmodified virtual machine image
2. Boot the virtual machine
3. Copy files on the host into the guest
4. Run a program on the guest
5. Pause and unpause the guest
6. Retrieve files from the guest
7. Shutdown the guest

Use of the prototype requires that VMware Server 2.0 or higher is installed on the computer where it is run. The virtual machine image needs to located in a subdirectory of a directory configured as a data store in VMware Server.

4.1 Manage the Image

For the prototype, I created a Linux image using OpenSuse11. It took several iterations in order to get an image built that would meet the requirements, but also be as small as possible. Unfortunately, the image I produced was larger than I would have liked (after compression it is 670MB). This was due to the fact that the VWware Tools needed to be built specifically for OpenSuse11 and thus required a number of development tools to be installed. Further work could be done in the future to create a smaller image that minimizes the tools installed, and remove tools only required to build the VMware Tools. Following the creation of the image, I took a baseline snapshot of the image.

The prototype requires three parameters: a reference to the virtual machine image, the user name and password for the user that VMServer is running under.

Using this information, the prototype begins by connecting to the VMware server (VixHost_Connect) running on the local computer, registers the passed in image with the server (VixHost_RegisterVM) and then opens the image(VixVM_Open). Next, it obtains a reference to the snapshot (VixVM_GetRootSnapshot) that I took after the image was created and reverts the instance to that snapshot(VixVM_RevertToSnapshot). This ensures that the virtual machine is in an unmodified state regardless of any other task that may have been run in the image.

It is worth noting, that my original preference would have been to use the function VixVM_Clone to create a 'linked' clone. A linked clone shares the virtual disks with the parent in much of the same way as a child process shares memory with its parents using the copy on write strategy. Unfortunately, the function is not supported with VMware Server – only VMware Workstation.

4.2 Boot the virtual machine

Booting the virtual machine consists of powering on the virtual machine (VixVM_PowerOn) and then waiting for the startup and boot process to complete. VMware provides a function that lets you know when the VMware tools are available in the guest (VixVM_WaitForToolsInGuest). They suggest using this to determine when the guest has finished booting.

4.3 Copy files on the host into the guest

Coping files from the host into the guest is the next step. In order to execute the next several calls, you must first login to the guest as a user that is known to the guest. This is done using the VixVM_LoginInGuest function.

The next step is create a directory in the guest (VixVM_CreateDirectoryInGuest) and then copy two files from the host into the newly created directory (VixVM_CopyFileFromHostToGuest). The second file was the application I wanted to run. The first file as a shell script. VMware does not guarantee that the current working directory will be set for a program on the guest is executed by a process running on the host. Therefore, the script that was copied will set the current working directory to the directory that was created above and then it will run the program that was copied.

I used the version of the test application that was compiled with the -O3 optimization in order to demonstrate that we could obtain the same result as on a Linux host.

4.4 Run a program on the guest

The files that were copied into the guest did not have execute permissions set on them. In order to the run the program, I first had to set the permissions using chmod. This was done using the VixVM_RunProgramInGuest function. Finally, I was able to run the script (which in turn calls the application). This was also run using the VixVM_RunProgramInGuest function. The script redirects the output into a new file.

4.5 Pause and unpause the guest

The VIX api provides two functions that allow you to

temporarily stop execution of the virtual machine. These are VixVM_Pause and VixVM_Unpause. These are used to demonstrate that they work as documented in the API.

4.7 Retrieve files from the guest

Files can be copied from the guest to host using VixVM_CopyFileFromGuestToHost. In this case we retrieve the file that contains the redirect stdout of the example application. As we had hoped, the file contains an estimated value of pi of 3.140592860. Matching the result computed when run directly on a Linux host.

4.8 Shutdown the guest

Now that the program has finished running, the virtual machine can be powered off (VixVM_PowerOff) and then removed from the servers registry (VixHost_UnregisterVM).

5. Proposed Integration with BOINC

The following discussion provides an example of how the techniques demonstrated in the prototype discussed above could be used to make it convenient to use virtual machines with BOINC.

BOINC has a concept of a platform which is a combination of chip architecture and operating system. The client will inform the server what is its preferred platform and what alternate platforms it will accept. This was initially developed for 64 bit clients so that they could download and use 64 bit applications if they were available, but if they weren't available then they would accept a 32 bit client.

I propose that BOINC extend this concept further so that the client can inform the server that it supports additional platforms. These additional platforms are available to the client based upon the virtualization software and images available on the client.

BOINC stores a number of files and settings in a directory called the data directory. This directory is located by default at C:\Documents and Settings\All Users\Application Data\BOINC. I propose that BOINC add a subdirectory to this folder called 'alt_platforms'. Within this subdirectory, there would reside additional subdirectories, each of which would be named for the platform that it represents. In the case of the prototype build above, this name would be 'i686-pc-linux-gnu'. The name i686-pc-linux-gnu is the standard name that a BOINC Linux client would report to the server during its communications. Within this directory would be an XML definition file that describes the manager application and all files, as well as the manager application and the virtual machine files.

The BOINC client would add alternate platforms that it finds within the alt_platforms directory to the list of platforms it supports when it communicates with the project server. When it is assigned a workunit for the platform, the following steps would performed by the BOINC client:

1. Copy the application binaries and input files into a 'slot' directory as normal. The slot directory is a working space for a in-progress workunit
2. Copy the alt-platform directory into a working directory. I.e. recursively copy C:\Documents and Settings\All users\Application Data\BOINC\alt_platforms\ i686-pc-linux-gnu to C:\Documents and Settings\All Users\Application Data\BOINC\alt_platforms_working\slotX (where X would be the slot number the workunit was assigned.
3. Start the manager application for the alternate platform and send it messages as it would for any research application. These messages would include pause, unpause, exit, etc
4. When the manager application exits, review the return code to determine if the application ran correctly or not. Retrieve the result files from the slot directory and return them to the server.
5. Cleanup the slot directory and the alt_platform_working directory that was used for the workunit

The manager application should be developed as a standard BOINC research application. This means that it should know how to respond to events sent by the BOINC client and handle them appropriately. In particular, the manager application should be able to do the following:

1. Register the virtual machine with VMware server and then open the virtual machine.
2. If a snapshot was distributed with the research application, then revert to the supplied snapshot. This allows a project to provide updates to a previously released virtual machine
3. At initial startup, start the virtual machine and copy the research application and workunit input files into the virtual machine
4. Periodically checkpoint the virtual machine by creating a new snapshot.
5. Pause and unpause execution of the virtual machine as directed by the client
6. Power down the virtual machine when requested by the client
7. If restarted while a work unit is in progress, restore the most recent checkpoint and resume execution from that point
8. Monitor the application status. Once the application finishes running, then power down the virtual machine and exit with an appropriate return code based on the return code of the research application

It would be useful for the the BOINC project to provide a reference image that volunteers could download and install into the alt_platform directory. Other BOINC research projects that want to use virtualization would either provide their own 'alt_platforms' or they would distribute a snapshot with their research applications that could be applied against the reference image. In order for snapshots to be applied, the BOINC client would need to be aware of a requirement that the application version should be run within a given virtual platform.

The volunteers who run the BOINC client on their machines would need to install the VMware server

application and configure a data store to reside at C:\Documents and Settings\All Users\Application Data\BOINC\alt_platform_working. This is required so that the manager application can register a virtual machine with the VMware server.

6. Related Work

There are two groups that have been working on a similar effort to what is describe in this paper.

The first group that has been working on this problem is at CERN. At CERN there is a research application AltFast that, for the purposes of the Athena research, is tied directly to a specific version of a Linux build called Scientific Linux Cern (SLC).[6] In their investigation of using virtualization, they rule out VMware early because they stated that it is not free. At the time they did their work, VMWare player may have been the only free version available. However, the existence of a free version VMWare server should cause them to review this point.

They made the decision to pursue using XEN and run the BOINC client within a virtual machine on XEN. XEN is not an option for most volunteers due to the complexity in configuring XEN. Additionally, most volunteers use Windows XP or Windows Vista and will not be interested in installing XEN to run BOINC. Finally, one of the strengths of BOINC is that it has been developed so that it can minimize the impact of running the research application on the active user of the computer. If the BOINC client is running within a virtual machine, then the client cannot correctly perform certain tasks such as stopping the research application when the end user is active.

Having said this, it is possible that this choice is a good choice within CERN. Inside a single organization, machines can be configured by experts in a consistent fashion. These experts could ensure that the specific image desired is running on each machine. However, for broad use in the volunteer community, this technique will not be sufficient.

The second group that is working on a related problem documented their findings in a CoreGRID technical report.[5] They are specifically looking at providing a secure sandbox for running research applications. They have a similar set of criteria as the one defined in this paper. They put much greater weight on the requirement that it be easy for the user to install – to the extent that the project could distributed the virtualization software. This requirement eliminates the use of a product such as VMware Server. Instead, they opted to use QEMU with the KQEMU accelerator for x86 processors. In their paper they provide an analysis of the loss in performance by shifting to a virtualization technology. They found that running a KQEMU Linux guest at below normal priority on a Windows host resulted in about a 6% longer execution time for workunit compared to the time it took to run on native Linux. Their findings and efforts demonstrate that QEMU/KQEMU are also an excellent choice for using virtual machine technology with volunteer computing.

7. Conclusions and Future Work

This paper has demonstrated that virtualization technology can be used to increase the homogeneity of resources in a volunteer computing grid. A prototype was created and explained that showed that all the required functions are available in VMware server. The possibility of being able develop a research application for only one or two limited platforms instead of many platforms is appealing as it simplifies the work required of project administrators. Although not a key point of this paper it is worth mentioning that virtualization provides certain functions such as 'checkpointing' as part of the infrastructure rather than something that must be implemented in the research application. This certainly lowers the burden on a project administrator.

The next steps in this work is to take a deeper look at the proposed integration with BOINC and develop a more detailed design that will add direct support for virtualization to BOINC so that it can be used easily. This will consist of developing a set of virtual machine managers that can properly handle the popular virtualization software platforms.

Additionally, work needs to be done to create some standard 'alt_platforms' that are both references and starting points for projects to use.

References

1. David P. Anderson. BOINC: A System for Public-Resource Computing and Storage. 5th IEEE/ACM International Workshop on Grid Computing. November 8, 2004, Pittsburgh, USA.
2. David P. Anderson and Kevin Reed. To appear in the Hawaii International Conference on System Sciences (HICSS), January 5-8, 2009.
3. M. Taufer, D. Anderson, P. Cicotti, C. L. Brooks III, "Homogeneous Redundancy: a Technique to Ensure Integrity of Molecular Simulation Results Using Public Computing," ipdps,pp.119a, 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 1, 2005
4. G. Lopez, M. Taufer, and P.J. Teller: Evaluation of IEEE 754 Floating-Point Arithmetic Compliance Across a Wide Range of Heterogeneous Computers. In Proceedings of the 2007 Richard Tapia Celebration of Diversity in Computing Conference, October 2007, Orlando, Florida, USA.
5. A. Marosi, P Kacsuk, G Fedak, O. Lodygensky, "Using Virtual Machines in Desktop Grid Clients for Application Sandboxing," CoreGRID TR-140. August 31, 2008.
6. D. Weir, "BOINC And Paravirtualization" CERN Twiki: https://twiki.cern.ch/twiki/bin/view/LHCAtHome/BOINCAndParavirtualization

Appendix A – Source code used to produce table 1

```
#include <stdio.h>
#include <string.h>
#include <math.h>

#define PI 3.14159265358979323846264338327 9

int main ()
{
   float pi = 0;
   int limit = 1000;
   float temp;
   unsigned int i;
   for(i=0; i<limit; i++) {
      if (i%2==0) temp=4.0f/(2*i+1);
      if (i%2==1) temp=-4.0f/(2*i+1);
      pi=pi+temp;
      printf("%d: diff: %.15f, est pi: %.15f\n",i,PI-pi,pi);
   }
}
```