2011 IEEE International Parallel & Distributed Processing Symposium

libboincexec: A generic virtualization approach for the BOINC middleware

Diogo Ferreira, Filipe Araujo
CISUC
Dept. of Informatics Engineering
University of Coimbra, Portugal
{*defer,filipius*}*@dei.uc.pt*

Patricio Domingues
Research Center for Informatics and Communiations
School of Technology and Management
Polytechnic Institute of Leiria, Leiria, Portugal
patricio@estg.ipleiria.pt

***Abstract*—BOINC is a client-server desktop grid middleware that has grown to power very large computational projects. BOINC clients request computing jobs to a central server and run them alongside other regular applications. Unfortunately, this kind of execution causes two kinds of problems. Firstly, developers must port their application to every single operating system target, which usually means maintaining several different versions of the same application. Secondly, any application running natively on desktop grid hardware is a potential security threat to the volunteer client.**

During the course of this research we sought an efficient and generic method for alternative execution of jobs in BOINC clients. Our approach is strongly guided by the principles of non-intrusiveness and contains two main components. The first is a library, libboincexec, which is able to control several virtual machines monitors. The second is a modified BOINC wrapper that provides the glue between libboincexec and the middleware.

Through the use of this solution we are able to effectively use virtual machines to perform computation on desktop grids. This computation is inherently safe because virtual machines provide sandboxing. Additionally, by targeting the same virtual operating system, the problem of maintaining different versions of an application does not exist, thereby solving the heterogeneity problem of desktop grid nodes.

***Keywords*-boinc; desktop grids; virtualization; sandboxing;**

I. Introduction

The BOINC middleware [1] has been around since the early 2000's, to support volunteer computing projects. These projects consist of many independent jobs that run on worker nodes that volunteer to compute them, usually for the sake of scientific topics related to astronomy, medicine or mathematics. These worker nodes install the BOINC client software, which contacts a centralized infrastructure, the server, in order to download computational jobs. Upon finishing these jobs, workers upload results back to the server. This platform grew to become very popular, with over 5 million hosts (500,000 active), and approximately 3.6 PetaFLOPs of computing power, more than any supercomuter as of November 2010[1].

Unfortunately, running software on hardware donated by volunteers involves a certain amount of risk. Although volunteers trust that the project will cause no harm to their hardware or data, developers may intentionally or not produce misbehaving applications. Desktop grid appliances such as BOINC [1] implement security to a certain degree, based on user permissions. This approach assumes that the client system is properly configured. Another big issue is the need to deal with many different architectures and heterogeneous setups.

Virtualization solves these issues by providing hardware isolation on virtualized guests by definition [2]. Virtual machines may also simplify application development by allowing developers to target one operating system which runs as a guest.

In this paper we will present libboincexec, a library which extends BOINC with virtualization support. libboincexec contributes with a generic plugin-based virtualization approach which improves the two important BOINC limitations: client security and ortability for application developers. The work described in this paper also contributes execution plugins for VirtualBox and VMWare.

This paper is structured as follows: In section II we discuss related work regarding BOINC virtualization and execution environments in general; In section III we describe the libboincexec's design principles and architecture; Section IV talks about the several execution plugins we support; Section V describes how to use libboincexec; Section VI shows our testing of libboincexec in a simulated environment; Finally, section VII discusses the contributions and presents future directions.

II. Related Work

This section discusses other implementations of different execution environments in grid middlewares as well as virtualization solutions specific to BOINC.

One of the earliest implementations of provisioning execution environments is SoftEnv[2]. This system configures

[1] http://boincstats.com/stats/project_graph.php?pr=bo.

[2] http://www.mcs.anl.gov/hs/software/systems/

1530-2075/11 $26.00 © 2011 IEEE
DOI 10.1109/IPDPS.2011.349

a specific environment according to a set of rules so it is able to perform execution of a target application. The EGEE middleware also contains similar technologies named *pilot jobs* which prepare the machine for the execution of real jobs [3]. EGEE also contains provisioning for virtualized worker nodes which lay a custom execution environment based on virtualization [4].

Several other security-related appliances provide sandboxed or limited environments where safe execution can take place. Vx32 [5] and Google Native Client [6] are examples of such environments.

On the BOINC front, Gonzales et al. [7] contributed a method which is able to run virtualized Matlab workunits using VMWare instances. As a derivative of this work, Ben Segal[3] provided a BOINC wrapper capable of interfacing with VMWare for application execution. Atilla Marosi provided a BOINC client modification which is able to delegate work to a QEMU virtual machine [8].

The solutions discussed above are tied to a specific virtual machine monitor. Further attempts at generalization can be found in David Quintas' work[4]. This solution provides an abstraction layer and an architecture based on message passing and XML-RPC calls. While aiming for generality, the solution implements only a VirtualBox-specific interface and requires a set of heavy client-side dependencies such as a Python interpreter and a message-passing middleware.

III. Design and Architecture of *libboincexec*

libboincexec is guided by two main design principles: make no assumptions about the execution environment and avoid changing the client in away that makes it backwards incompatible.

libboincexec is a system composed of three major components: a Plugin Manager, a number of Execution Plugins and a Glue Application. Execution Plugins are specific to each virtualization middleware and can handle the required behaviours for execution inside a virtual machine. The Plugin Manager handles the detection of Execution Plugins and is responsible for exposing an interface that abstracts the underlying virtual machine to applications which want to use libboincexec. A Glue Application calls the interface provided by the Plugin Manager and exposes its functionality to another system such as BOINC. The first two are described in detail in the following subsections while the Glue Application for BOINC is discussed in Section III-C.

A. Plugin Manager

The Plugin Manager plays a key role in libboincexec's functionality. It is the component that allows applications such as the BOINC client to use the exposed virtualization functionality.

Since this component must be distributed to clients along with applications, either as a separate library or as a statically linked component, great care was taken to make it portable across many different operating systems and platforms. Ideally, this component was to conform to ISO C++ as a means to attain cross-platform compatibility. We are, however, dealing with loading software modules in runtime and operating system specific calls cannot be avoided. Thus, we brought in a dependency on *libltdl*[5], a lightweight library which is able to load software modules in various operating systems. This dependency can be avoided in the future by having several implementations of this routine and pick the one we need at compile time.

The Plugin Manager acts as a broker between glue applications and execution plugins. Its core functionality involves scanning for available plugins and loading or unloading them on demand. The functionality exported by the plugin can then be used by applications that request it. Communication with glue applications follows the plugin interface described in the following section.

B. Execution Plugins

Execution Plugins are loadable modules which are able to manipulate the execution of a BOINC application inside a virtual machine. Plugins must implement a common abstract interface that is known both to glue applications and to the plugin manager. This interface is designed to hide the virtual-machine specific parts behind an abstract interface so that it can be treated as a blackbox.

The plugin interface is composed of five functions:

getMetaData: Returns a metadata object filled with plugin information – name, description and configuration options.

prepareForWork: Prepares the target virtual machine to accept running executables – starts the virtual machine and waits for its ready state.

copyFile: Provides features to copy files back and forth between host and guest.

runCommand: Runs a command inside the virtual machine.

performCheckpoint: Checkpoints the virtual machine.

The above interface is implemented using a C++ class with pure virtual methods. In order for the unit to be compiled, all pure virtuals must be implemented, assuring that the module can be loaded by the Plugin Manager. Plugins are also written in C++, with a possibility of expansion to other languages being work in progress.

Using this interface we have implemented three fully functional execution plugins: Local, VMWare and VirtualBox. These plugins are described in detail in Section IV.

[3]http://boinc.berkeley.edu/track/wiki/VmApps

[4]http://boinc.berkeley.edu/track/wiki/VirtualBox

[5]http://www.gnu.org/software/libtool/manual/html_node/Using-libltdl.html

C. Glue Applications

As mentioned above, a Glue Application sits between libboincexec and a final application that uses it, such as the BOINC client, and calls it as needed in order to perform execution.

For BOINC integration are two available options. The first is replacing the fork/exec approach used by BOINC by the corresponding libboincexec functionality. The second is using a BOINC wrapper. A wrapper is an application that facilitates the execution of applications that do not conform to the BOINC API. While initially used for backwards compatibility with legacy applications, the wrapper has found new uses in state of the art virtualization mechanisms for BOINC. The wrapper is a thin application that implements the BOINC API but delegates computation to a third party application. In the context of virtualization, the wrapper can be used to delegate computation to libboincexec.

Changing the BOINC client is the cleanest and most straightforward alternative. If this were to happen, developers would be able to write third-party plugins for the BOINC client which suited their needs. While this change would not bring fundamental changes to the high-level BOINC architecture, it requires several changes to the core client and execution workflow. As stated above, BOINC is massively deployed and these changes would be likely to break compatibility with older clients.

By using a wrapper, changing the client can be totally avoided since all necessary functionality can be implemented as a third party application. As such, we developed a libboincexec wrapper which delegates BOINC execution calls into libboincexec, allowing applications to be executed using any Execution Plugin thereby supporting as many virtual machine targets as libboincexec does.

IV. Implemented Plugins

A. Local

The purpose of the local plugin is to run an application directly in the host machine. This plugin serves two purposes: show that libboincexec can be used for any execution environment, not only virtualization, and provide an initial test subject for the remaining components.

This straightforward plugin implements the functions described above by copying files back and forth to a folder in the host system and by performing execution using operating-system dependant calls (such as CreateProcess on Windows or fork/exec on POSIX-compliant systems).

In the future this plugin might be used as a fallback if all others fail which can be useful when no virtual machine hypervisors are present in the client system.

B. VMWare

VMWare was the first virtualization plugin we implemented. This is due to the fact that it possesses a feature-rich controller library named VIX[6]. This API provides several features, the most interesting being: start/stop/pause the virtual machine, copy files back and forth, execute a command. These functions map almost directly to our abstract plugin interface. Due to the completeness of this API, we managed to quickly create a plugin that exposes VMWare functionality and enables execution in the BOINC middleware while using a cross-compatible, portable API provided by the virtual machine vendor.

This plugin is able to control several VMWare appliances such as Player, Server and Workstation.

C. VirtualBox

While VMWare provides solid functionality and API support, we decided to work on a second plugin because the usage of VMWare brings several licensing issues when distributing or automatically installing their software or API packages. Another technical aspect is that most of the virtualization monitors, such as VirtualBox, do not provision for file copying or execution inside their guest virtual machines.

In order to implement the interface we defined above we started by adding support to prepare the virtual machine environment for execution. This task was performed by writing a wrapping a set of functions around *VBoxManage*, a command line tool that controls this hypervisor.

For file copying and execution, however, a more elaborate solution was needed. Other BOINC virtualization implementations use custom-written backdoor daemons inside the virtual machine which provide an interface for file copying and execution.

After considering the alternatives, we decided to implement these functions over SSH. SSH, the Secure-Shell Protocol describes a way of accessing remote shells in a safe, encrypted way. It also supports file copying via the Secure-Copy extension. The major advantage is that only a standard SSH server such as OpenSSH or Dropbear is needed inside the guest virtual machine and a small SSH client library (which can be distributed through BOINC) in the client. Using SSH to copy files over might cause a slight overhead due to its encrypted channel. In the future, these file copying operations can be modified to use simpler protocols such as FTP. We implemented SSH support independently from this plugin so it can be reused by other plugins if there are no standard copying and execution primitives available.

V. Using libboincexec

In order to use libboincexec with BOINC for virtualization purposes we need three things: An application compiled for the target operating system in the virtual machine; A supported virtual machine in the users' computers; A job configuration file.

Since libboincexec and the modified wrapper implement the BOINC API, applications running using this system do

[6]http://www.vmware.com/support/developer/vix-api/

```
<job_desc>
 <task>
   <application>worker</application>
   <innerjob>pairs</innerjob>
   <vm>ubuntu-vbx</vm>
   <user>vmuser</user>
   <password>vmpassword</password>
   <datadir>/home/vmuser</datadir>
   <data>f.in</data>
   <output>f.out</output>
 </task>
</job_desc>
```

Figure 1. Sample job description file for the modified wrapper

not need to make BOINC-specific calls. Any application that takes input files, executes and outputs to a set of files works unmodified. At the moment libboincexec does not deal with virtual machine deployment and as such projects must coordinate with users in order to fulfill this requirement.

The final requirement is a simple job configuration file which describes computation, it contains the same entries as a regular BOINC job configuration file with some additions: *innerjob*: Specifies the binary that contains the actual application to be run inside the virtual machine; *vm*: The virtual machine image that should be started; *user/password*: Credentials for authentication in the guest operating system; *data*: Input files and libraries that need to be copied to the guest; *output*: One or several expected output files. An example of this configuration file is shown in Figure 1.

The developer can then add our modified BOINC wrapper as a BOINC application and setup the job description, innerjob as input files, in addition to the regular input files. This will make BOINC download all the required data for execution. When the client requests a unit of work, it will receive all the associated files and start the wrapper, which will trigger the virtual environment and produce output files which can then be sent back to the server.

VI. Tests

In order to measure how the usage of libboincexec affects computation of BOINC tasks, we prepared a simulated testing environment and performed a set of tests on all the execution plugins we implemented. The following subsections detail the experiment, we will start by describing the infrastructure and application, then client configurations, the application, the methodology and then present results and analysis.

Our goal with these tests is to measure the effects of libboincexec on computation. This comparison focus more on libboincexec than on the performance internals of each virtualization middleware. There are, however, other studies that focus on the performance of virtual machine monitors [9].

A. Infrastructure and Application

The BOINC middleware was installed in a server feature a Core 2 Quad Processor running at 2.66GHz and 4GiB of RAM. The operating system of choice was a Linux distribution, Debian 4.0. We configured a standard BOINC deployment with the Apache HTTP server and MySQL.

We decided to use one of the BOINC samples as a candidate for testing since they are simple, tested applications that make use of BOINC's functionality. The application we picked was *uppercase*. This application performs the trivial task of uppercasing each character in the input file. In order to mimic long execution, this application does each computation several times and adds delays so that each execution lasts an average of 13 seconds on our test client.

The application was compiled as a native BOINC executable. Another version, stripped of BOINC's API calls, was prepared in order to test each execution plugin in libboincexec. We added both versions of the application to the project and added several sample workunits for initial functionality testing.

B. Client

For this testing environment, our client was a Core 2 Duo processor running at 2.4GHz and featuring 2GiB of RAM. This client also ran a Linux distribution and a standard BOINC client.

The BOINC client was configured to use both cores at all times, thereby discarding the default opportunistic behaviour. This way it will never stop even if there is some user interaction or CPU spike caused by other applications.

C. Methodology

We defined a test as a pack of 100 workunits for the *uppercase* application. This pack takes long enough for evaluation of metrics like error rates and differences between vanilla BOINC execution and execution via libboincexec. The testing procedure is divided in three phases:

- Collect a timestamp and start the client.
- Wait for the client to finish computation.
- Collect a time stamp and error rates from the server.

Each test was conducted five times in the following scenarions:

1) Default BOINC execution.
2) BOINC with libboincexec using the Local plugin.
3) BOINC with libboincexec using the VMWare plugin.
4) BOINC with libboincexec using the VirtualBox plugin.

Test	R1(s)	R2(s)	R3(s)	R4(s)	R5(s)	AVG(s)	STDEV(s)	ERR(%)
1	1723	1691	1630	1725	1658	1685.4	33.7	0
2	1758	1650	1687	1972	1913	1704	34.08	0
3	1901	1863	1896	1972	1913	1882	37.64	0
4	1928	1901	1930	1977	1987	1914.5	38.29	0

Table I
TEST RESULTS FOR THE DIFFERENT SCENARIOS

D. Results and Analysis

Table I shows the results for each of the five runs of batch tests, averaged values and accumulated error rates.

When using the Local plugin, BOINC shows a run time which is very close to execution without libboincexec. Since the Local plugin uses the same fork/exec approach as the default BOINC execution, these results show that libboincexec's plugin manager and wrapper do not introduce noticeable overhead.

When dealing with the virtual machine plugins, the overhead becomes noticeable as expected, plugins perform more work and have to wait longer times in order to transport files back and forth and start and stop the virtual machines. This additional work introduces some overhead. Apart from startup procedures, the fact that computation is being virtualized also increases run time. BOINC with the VMWare plugin shows a computation time 11% bigger than standard BOINC, in average.Interestingly, this overhead is similar to the one measured in other contributions [9].

BOINC with the VirtualBox plugin also shows similar overhead in computation time, but slightly bigger. As before, other studies confirm that VirtualBox is slightly slower than VMWare for CPU-intensive operations.

An interesting observation is that no errors occurred during computation. Usually, BOINC projects will have several errors as they grow since they cannot predict the availability of volunteer machines. In our test the client is always available in a dedicated fashion. The fact that they share the same local network also reduces the probability of network failure stalling the client and causing errors.

These tests are very close to what we expected and show that for this workload, libboincexec by itself does not introduce a significant overhead in the total computation time. Its plugins, however, might introduce additional expected overhead in virtual machines due to their inherent implementation.

VII. CONCLUSIONS AND FUTURE WORK

This paper discusses a library capable of controlling and running BOINC applications inside virtual machines. This library adds security by sandboxing and may also simplify deployment for project managers and developers who would otherwise have to deal with numerous target operating systems. Through careful study of the design and architecture of BOINC we managed to integrate our library using a Wrapper, thus leaving the BOINC infrastructure unmodified. By making no changes to the client we guarantee backwards-compatibility.

We discussed other implementations of virtualization above. libboincexec is different in the sense that no assumption is made on the virtual machine monitor. One of the key points is also being lightweight, written in the same language as BOINC and dragging no outstanding dependencies to the BOINC environment. Our implementation of an SSH connector also contributes an easy way to interface with API-lacking virtual machine hypervisors. Finally, we contributed execution of plugins for VMWare and VirtualBox. This plugin architecture allows transparent execution using any plugin and BOINC applications are able to run using any available plugin.

There are, however, some challenges and issues that may be tackled in the future, namely the deployment of virtual machine monitors throughout clients. As far as we know, this is impossible without user intervention, a possible solution would be to install this middleware during the BOINC client installation. The plugin collection can also be enriched with other virtualization middlewares.

ACKNOWLEDGMENT

This work has been partially supported by the project PTDC/EIA-EIA/102212/2008, High-Performance Computing over the Large-Scale Internet.

REFERENCES

[1] D. P. Anderson, "Boinc: A system for public-resource computing and storage," in *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 4–10.

[2] G. J. Popek and R. P. Goldberg, "Formal requirements for virtualizable third generation architectures," *Commun. ACM*, vol. 17, no. 7, pp. 412–421, 1974.

[3] "Egee middleware architecture," pp. 1–95, 2005. [Online]. Available: http://www.nsc.liu.se/ngssc-grid-05/glite-nixon.pdf

[4] S. Childs, B. Coghlan, and J. McCandless, "Dynamic virtual worker nodes in a production grid," in *Frontiers of High Performance Computing and Networking ISPA 2006 Workshops*, ser. Lecture Notes in Computer Science, G. Min, B. Di Martino, L. Yang, M. Guo, and G. Ruenger, Eds. Springer Berlin / Heidelberg, 2006, vol. 4331, pp. 417–426.

[5] B. Ford and R. Cox, "Vx32: Lightweight userlevel sandboxing on the x86," in *In Proceedings of the USENIX Annual Technical Conference*, 2008.

[6] B. Yee, D. Sehr, G. Dardyk, J. B. Chen, R. Muth, T. Ormandy, S. Okasaka, N. Narula, and N. Fullagar, "Native client: A sandbox for portable, untrusted x86 native code," August 2009, pp. 79–93. [Online]. Available: http://dx.doi.org/10.1109/SP.2009.25

[7] D. L. na González, F. F. de Vega, L. Trujillo, G. Olague, and B. Segal, "Customizable execution environments with virtual desktop grid computing," in *Parallel and Distributed Computing and Systems, PDCS*, Porto, Portugal, 2007.

[8] A. Csaba Marosi, P. Kacsuk, G. Fedak, and O. Lodygensky, "Using virtual machines in desktop grid clients for application sandboxing," Institute on Architectural Issues: Scalability, Dependability, Adaptability, CoreGRID - Network of Excellence, Tech. Rep. TR-0140, August 2008. [Online]. Available: http://www.coregrid.net/mambo/images/stories/TechnicalReports/tr-0140.pdf

[9] P. Domingues, F. Araujo, and L. Silva, "Evaluating the performance and intrusiveness of virtual machines for desktop grid computing," in *IPDPS '09: Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–8.