V-BOINC: The Virtualization of BOINC

Gary A. McGilvary
Edinburgh Data-Intensive Research Group
School of Informatics
The University of Edinburgh
gary.mcgilvary@ed.ac.uk
(student author)

Adam Barker
School of Computer Science
University of St Andrews
adam.barker@st-andrews.ac.uk

Malcolm P. Atkinson
Edinburgh Data-Intensive Research Group
School of Informatics
The University of Edinburgh
mpa@staffmail.ed.ac.uk

Ashley D. Lloyd
Business School
The University of Edinburgh
ashley@edinburgh.ac.uk

1. INTRODUCTION

Volunteer computing, made popular by BOINC and SETI@Home gives members of the general public the opportunity to offer their computational resources to distributed scientific research projects. Despite its popularity, BOINC still has many drawbacks, most of which relate to BOINC applications running in the user space of the volunteer machine.

Project developers are required to port their application to every target machine architecture and must also provide application-level checkpointing to ensure job progress is not lost upon host termination or failures. Furthermore, project developers are limited to creating applications that have no dependencies and the users of BOINC must trust that project servers they attach to, will not distribute malicious or untrustworthy applications. These drawbacks can result in project developers taking additional time to implement measures to solve such problems.

With virtualization, many of these issues are solved. One only needs to port an application to a single virtual machine architecture, host security in which the host is protected from third party applications is inherently addressed by the sandbox environment and system-level checkpointing is available. Applications with dependencies can also easily run where dependencies can be attached to a virtual machine enabling application developers to deploy more complex applications to obtain results of more value. In this abstract we present virtual BOINC, or V-BOINC that introduces virtualization into the BOINC framework.

2. V-BOINC ARCHITECTURE

The foundation of our approach relies on sending lightweight virtual machine images from a modified BOINC server (V-BOINC Server), to volunteer clients allowing BOINC applications to run within the virtual machine itself rather than in the user space of the host. This is implemented by installing a BOINC client within the virtual machine image to fetch applications for a user specified project. This is in addition to the modified BOINC client (V-BOINC Client) installed on the user's host to download the virtual machine image. These components are depicted in Figure 1.

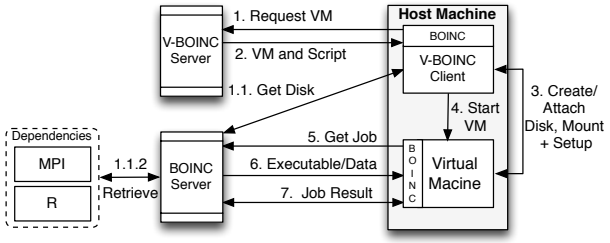

Figure 1: V-BOINC Architecture

Upon a volunteer user submitting the details of the BOINC scientific project they wish to attach to via the V-BOINC Client (e.g the project server URL and their BOINC project weak account key), the host BOINC client is instructed to request a virtual machine image (1). Concurrently, the V-BOINC Client probes the BOINC server to determine if any dependencies exist for the specified project (1.1). If so, a VDI (or *.vdi*) file containing the dependencies is transferred to the V-BOINC Client via *curl*; we assume that developers of BOINC projects who wish to deploy applications with dependencies are prepared to create a virtual disk image containing the dependencies (DepDisk) and make this publicly available on the BOINC Server to allow the V-BOINC Client to determine whether a DepDisk needs to be downloaded.

Concurrently while a DepDisk is downloading, the virtual machine image and an executable script are downloaded to the host BOINC client (2). The V-BOINC Client either attaches the DepDisk, if the application is found to have dependencies, or alternatively creates an empty disk and mounts this to the virtual machine image (3). In either case, a disk must be provided to

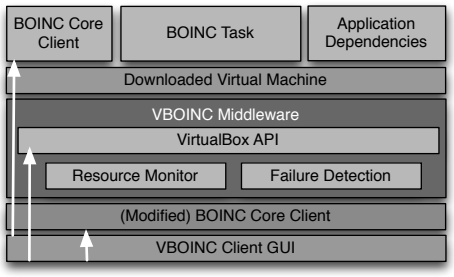

Figure 2: V-BOINC Client Components

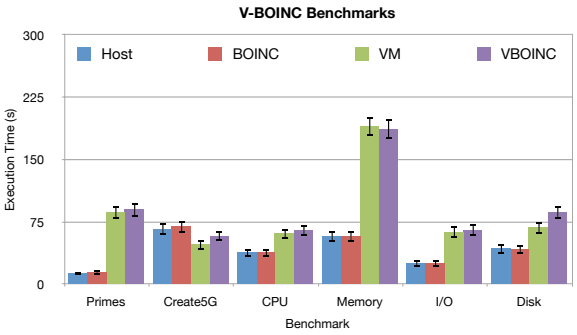

Figure 3: V-BOINC Benchmark Times

give the virtual machine storage space as the V-BOINC virtual machine has been stripped of all unnecessary components such as Linux swap space and unneeded packages to reduce the bandwidth used when sending to the volunteer user. The virtual machine image is then started (4) allowing it to request (5) and receive (6) BOINC jobs and return job results (7).

2.1 Taking Control

After the virtual machine image has been transferred to the volunteer machine via the host, the instantiation script (2) decompresses the virtual machine image tar file and signals the V-BOINC Client to take control of the instantiation process. Afterwards, the V-BOINC Client registers the virtual machine with VirtualBox, performs the steps (3) and (4) above and takes periodic checkpoints when the virtual machine is running.

In this case, further complexities are introduced as a second BOINC client located on the virtual machine needs to be controlled as well as the virtual machine. The interactions the V-BOINC Client makes with the second BOINC client and virtual machine are shown in Figure 2. We also provide components to monitor host resources and virtual machine failure to inform the user at run time the current state of V-BOINC.

3. EVALUATION

Now we outline the experiments to show the achievable performance of V-BOINC when compared to regular BOINC and the storage requirements of our checkpointing mechanism. We performed these by executing a number of benchmark applications: **Primes** calculates the first 300 prime numbers (CPU intensive), **Create5GB** creates a file of 5 GB using the function *dd* (memory and I/O intensive) and **CPU, Memory, I/O** and **Disk** are modified versions of the Stress workload generator.

3.1 V-BOINC vs BOINC

To compare the performance between the two software packages, each benchmark is then run on the host with and without BOINC to show the overhead or BOINC. Similarly, the same benchmarks are run on the V-BOINC virtual machine without the use of V-BOINC and then with V-BOINC to firstly determine the implementation overhead and secondly to determine the performance differences between BOINC and V-BOINC. Our results, shown in Figure 3, show that the implementation of V-BOINC introduces little overhead where the difference is introduced by virtualization alone as seen by the difference in execution times between BOINC and V-BOINC. Similar results are obtained when we use V-BOINC to execute an application with dependencies.

3.2 Checkpointing Storage Requirements

The storage space BOINC is permitted to use can be limited by the volunteer hence making storage valuable. To determine the likely storage space consumed by our checkpointing approach, we executed the same benchmarks and recorded the average disk consumption. Our results show that with the exception of disk-intensive applications, the amount of storage needed per checkpoint is extremely low. This is reassuring as BOINC typically executes CPU intensive applications. The results will be displayed within the poster.

4. CONCLUSIONS

The basic concept and implementation of V-BOINC as well as some experimental results have been presented; more information and results will be displayed within the poster. Many users within the volunteer community have taken advantage of V-BOINC and information on how to do so, as well as further details on V-BOINC can be found at [1]. We will not require a demo to be setup for the exhibition; all available information will be available within the poster.

5. REFERENCES

[1] Gary McGilvary, Adam Barker, Ashley Lloyd, and Malcolm Atkinson. V-boinc: The virtualization of boinc. In *CCGrid 2013*, Delft, The Netherlands, May 2013.