The Entropia Virtual Machine for Desktop Grids

Brad Calder Andrew A. Chien Ju Wang Don Yang
Department of Computer Science and Engineering
University of California, San Diego
{calder,achien,jwang}@cs.ucsd.edu

Abstract

Desktop distributed computing allows companies to exploit the idle cycles on pervasive desktop PC systems to increase the available computing power by orders of magnitude (10x - 1000x). Applications are submitted, distributed, and run on a grid of desktop PCs. Since the applications may be malformed, or malicious, the key challenges for a desktop grid are how to 1 control the distributed computing application's resource usage and behavior as it runs on the desktop PC, 2) provide protection for the distributed application's program and its data run, and 3) provide a virtualized execution environment, decoupling correct application execution from desktop configuration heterogeneity.

We describe the Entropia Virtual Machine, and the solutions it embodies for each of these challenges. We first outline the distinct requirements for a virtual machine in a desktop grid environment, which differ from web-based and hardware-level virtual machines. We then describe how the Entropia Virtual Machine (EVM) uses binary rewriting to solve each of these problems, guaranteeing the usability and protection of the desktop machine in the face of malicious or malformed application code. We also show that our approach provides security for the application and their data. We also detail how the virtual machine supports a virtualized execution environment. We present performance results for the EVM which show its performance overhead is modest.

1 Introduction

For over four years, the largest computing systems in the world have been based on "distributed computing" the assembly of large numbers of PC's over the Internet. These "grid" systems sustain multiple teraflops continuously by aggregating hundreds of thousands to millions of machines and demonstrate the utility of such resources for solving a surprisingly wide range of large-scale computational problems in data mining, molecular interaction, financial modeling, etc. These systems have come to be called "desktop distributed computing" systems and leverage the unused capacity of high performance desktop PC's (up to 2.2 Gigahertz machines with multi-gigaop capabilities[12]), high-speed local-area networks (100 Mbps to 1Gbps switched), large main memories (256MB to 1GB configurations), and large disks (60 to 150 GB disks).

Deployed in an enterprise, these systems harvest idle cycles from PCs sitting on hundreds to ten thousands of employees desks. Such distributed computing systems [7, 10, 1, 17, 6, 8, 4] leverage the installed hardware capability (and work well even with low performance PC's) and thus can achieve a cost per unit computing (or Return-On-Investment) superior to the cheapest hardware alternatives by as much as a factor of five or ten. As a result, distributed computing systems are now gaining increased attention and adoption within enterprises to solve their largest computing problems and attack new problems of unprecedented scale.

In order for a desktop grid to be accepted for enterprise deployment, the system must adequately protect the desktop machine from a malformed or malicious applications. Specifically, the system must guarantee that the application not interfere with the performance experienced by the desktop user and that no harm comes to the machine or its data. Some deployments also require that the application code and data are also protected from users of the desktop PCs.

In this paper we describe the design and evaluation of the Entropia Virtual Machine (EVM) which meets all of these requirements. The Entropia Virtual Machine supports the safe and controlled execution of applications expressed as native x86 Windows binaries, and EVM uses binary modification technology to achieve full mediation. In contrast most a web-based VM's express applications in a standard (and pointer-safe) intermediate language.

To meet the requirements of desktop grid systems, we designed a binary level virtual machine with three components – Desktop Control, Sandbox, and Application Security. Our approach uses binary modification to create a light weight interface which mediates between the application processes and the operating system. This mediation is use to control the application behavior, and hook in other parts of the security system. Our VM also uses an external process on the desktop machine to monitor and control all of the application processes and threads running on the machine. Our system exploits the mediation to implement a sandbox which manages the execution of the application so that the it can do no harm to the PC and so that it leaves the desktop PC in the same state as before the system was run. The application security is provided through a combination of a device driver, IT support and automated file encryption.

The Entropia Virtual Machine is a commercial product part, and is sold as part of Entropia's DC Grid enterprise solution (www.entropia.com). The EVM has been deployed as an enterprise grid at more then a dozen commercial sites, with over 50 applications deployed on the Grid. Since the majority of desktop machines run Windows x86 machines, Entropia focused purely

on provided a Windows x86 based solution. It is a complete system for Windows x86 machines running across three generations of Windows operating systems NT, 2000, and XP. Since it is important to span many generations of Windows, we do not use any advanced OS features from Windows XP to provide the above features. Instead, we created a solution that is based on the common components across these three generations of Windows.

The following are the main contributions of our submission:

- A detailed description of our overall solution for a Binary Level Virtual Machine for Desktop Grid Computing.
- Virtual Machine features to enforce unobtrusive behavior through a combination of the Desktop Controller and Sandbox. These monitor overall resource usage and provide Resource and Process Control, as well as the ability to terminate, pause and resume all of the application's work.
- A solution to Application Security that is acceptable within enterprise deployments without minimal driver support, and no hardware support.

Creating a system to run native binaries under a virtual machine provides a key advantage, which is the ease of application integration, that is enabling an application to run on the system. Using binary modification technology we are able to provide strong security guarantees and to ensure unobtrusive application execution as described above. Other desktop distributed systems require developers to modify their source code to use custom APIs or simply rely on the application to be "well behaved" and provide weaker security and protection[19, 15, 2]. These solutions are not desirable, since it is not always possible to get access to the application source code (especially for commercially available applications) and, regardless, maintaining multiple versions of the source code can require a significant ongoing development effort. As to relying on the good intentions of the application programmers, we have found that even commonly used applications in use for quite some time can at times exhibit anomalous behavior. Entropia's approach ensures both a large base of potential applications and a high level of control over the application's execution.

Other approaches to meeting some of the desktop grid requirements exist, for example, one could use VMWare's [**?**] virtual machine, and run each application on the desktop inside VMWare. Even so, these environments do not typically provide complete resource control and monitoring of the system at the level required to completely keep the virtual machine out of the way of the user. In addition, these approaches provide an extremely complex virtual environment, due to the limited size of our engineering team, and the complexity of making such technology work we chose a light weight solution which also provides a complete sandbox with the features we desire. A further advantage is that a separate OS image is not required.

Another potential solution is to restrict your grid to only running virtual machine specific languages, such as Java and MISL. These solutions require you to compile your code to the corresponding intermediate format, to provide all of the VM's security features, but often times applications use 3rd party dlls and libraries. To provide ease of application integration and the largest language support we chose to focus on binary level integration. In addition, our solution is sufficiently robust, in that we were able to easily wrap the Java Virtual Machine for windows inside of our Entropia Virtual Machine in order to run Java bytecode long with pure x86 binaries.

The remainder of the paper is organized as follows. In Section 2 describes desktop grid system requirements, and Section 2.1 the Entropia system Architecture. The Entropia Virtual Machine is explained in Section 3. Section 5 presents data from a performance evaluation of the Entropia VM, using real application programs. Finally, we discuss our results and put them in the context of related work in Section 6, concluding with a summary of the paper in 7.

2 Desktop Distributed Computing Overview

Distributed computing systems begin with a collection of computing resources, heterogeneous hardware and software configurations distributed throughout a corporate network, and aggregate them into a single easily managed resource. Distributed computing systems must do this in a fashion that ensures there is little or no detectable impact on the use of the computing resources for other purposes; it must respect the various management and use regimens of the resources as set forth by a company; and it must present the resources as one single robust resource. The following are the features needed for a desktop grid solution:

Efficiency — The system must harvest unused cycles efficiently, collecting virtually all of the resources available. The Entropia system gathers over 95% of the desktop cycles unused by desktop user applications.

Robustness — The system must complete computational jobs with minimal failures, masking underlying resource and network failures. In addition, the system must provide predictable performance to end-users despite the unpredictable nature of the underlying resources.

Scalability — The system must scale to the use of large numbers of computing resources. Because large numbers of PC's are deployed in many enterprises, scaling to 10,000's, of machines is imperative. However, systems must scale both upward and downward performing well with reasonable effort at a variety of system scales.

Manageability — Any system involving ten thousands of entities must provide management and administration tools. Typical rules of thumb, such as requiring even one administrator for every two hundred systems, would be unacceptable. We believe distributed computing systems must achieve manageability that requires no incremental human effort as clients are added to the system.

Ease of Application Integration — Fundamentally, the distributed computing system is a platform on which to run applications. The number, variety, and utility of the applications supported by the system directly affects its utility. Distributed

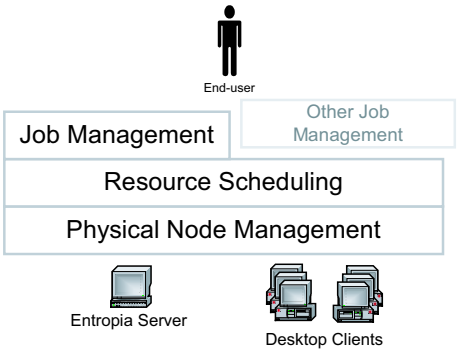

Figure 1: Architecture of the Entropia Distributed Computing System. The Physical Node Management layer and Resource Scheduling layer span the servers and client machines. The Job Management layer runs only on the servers.

computing systems must support applications developed with all kinds of languages and models, while still providing the above features.

2.1 Entropia System Architecture Overview

The Entropia system addresses the requirements described above by aggregating the raw desktop resources into a single logical resource. This logical resource is reliable, secure and predictable despite the fact that the underlying raw resources are unreliable (machines may be turned off or rebooted), insecure (untrusted users may have electronic and physical access to machines) and unpredictable (machines may be heavily used by the desktop user at any time). This logical resource provides high performance for applications through parallelism while always respecting the desktop user and his or her use of the desktop machine.

2.1.1 Layered Architecture

The Entropia system architecture is composed of three separate layers (see Figure 2.1.1). At the bottom is the Physical Node Management layer that provides basic communication and naming, security, resource management, and application control. On top of this layer is the Resource Scheduling layer that provides resource matching, scheduling, and fault tolerance. Users can interact directly with the Resource Scheduling layer through the available APIs or alternatively, users can access the system through the Job Management layer that provides management facilities for handling large numbers of computations and files.

Physical Node Management The distributed computing environment presents numerous unique challenges to providing a reliable computing capability. Individual client machines are under the control of the desktop user or IT manager. They can be shutdown, rebooted, or have their IP address changed. A machine may be a laptop computer that is disconnected for long periods of time, and when connected must pass its traffic through network firewalls. The Physical Node Management layer of the Entropia system manages these and other low-level reliability issues, while running the application on top of the Entropia Virtual Machine.

In addition to communication and naming, the Physical Node Management layer provides resource and application management. The resource management services capture a wealth of static and dynamic information about each physical node (e.g. physical memory, CPU speed, disk size, available space, client version, data cached, etc.), reporting it to the centralized node manager and system console. The application management provides basic facilities for process management including file staging, application initiation and termination, and error reporting.

Resource Scheduling The distributed computing system consists of resources with a wide variety of configurations and capabilities. The Resource Scheduling layer accepts units of computation from the user or job management system, matches them to appropriate client resources, and schedules them for execution. Despite the resource conditioning provided by the Physical Node Management layer, the resources may still be unreliable (indeed the application may be unreliable in its execution). Therefore the Resource Scheduling layer must adapt to changes in resource status and availability and to failure rates that are considerably higher than in traditional cluster environments. To meet these challenging requirements the Entropia system can support multiple instances of heterogeneous schedulers.

This layer also provides simple abstractions for IT administrators, which automate the majority of administration tasks with reasonable defaults, but allow detailed control as desired.

Job Management A distributed computing application often involves large amounts of computation (thousands to millions of CPU hours) submitted as a single large job. This job is then broken down into a large number of individual subjobs each of which is submitted into the Entropia system for execution. The Job Management layer of the Entropia system is responsible for decomposing the single job into the many subjobs, managing the overall progress of the job, providing access to the status of each of the generated subjobs, and aggregating the results of the subjobs. This layer allows users to submit a single logical job (for example, a Monte Carlo simulation, a parameter sweep application, or a database search algorithm) and receive as output a single logical output. The details of the decomposition, execution and aggregation are handled automatically.

The steps below shows the typical life of a job as it flows through our three-layered approach:

1. Authenticate DCGrid user to system.
2. User submits a job the the Entropia system
3. The job is broken into multiple sub-jobs (individual runs)
4. The binaries to run the job are automatically wrapped in the Entropia Virtual Machine through a single command that takes the application's executables and associated dll's and creates new "sandboxed" versions.
5. Submit the subjob to the Resource Scheduler specifying input files (including executables, libraries and output files), re-

source requirements (including minimum memory, disk, processor speed, run time, priority, etc.), and a script to be executed on the client.

6. System schedules and then run the subjob under the EVM on a desktop machine
7. User optionally check subjob status.
8. Process output files when subjob is complete to analyze the results.

3 Desktop Grid Virtual Machine Requirements

In order for a desktop grid to be adopted in an enterprise deployment, it essential that a virtual machine protect the desktop machine from a malformed or malicious applications, guarantee that the application will stay out of the way of the user, and that no harm will come to the machine or the data on the machine. The following are the requirements for a desktop grid virtual machine:

Desktop Security — The Distributed Computing system must protect the integrity of the computing resources that it is aggregating. Distributed computing applications must be prevented from accessing or modifying data on the computing resources.

Application Security — Some enterprise environments have a requirement that the system must protect the integrity of the distributed computation. Tampering with or disclosure of the application data and program must be prevented.

Unobtrusiveness — The system typically shares resources (both computing, storage, and network resources) with other systems in the corporate IT environment. As a result, the use of these resources should be unobtrusive, and where there is competition, non-aggressive. The distributed computing system must manage its use of resources so as not to interfere with the primary use of the desktops and networks for other activities. This includes both the use due to system activities as well as use driven by the distributed computing application.

To enforce unobtrusiveness, killing an application's processes may be necessary, but if possible the VM needs to instead pause the applications and later resume the application when it will no longer interfere with the desktop machine. This is a vital feature, since some applications run for days, and killing the application every time there is resource contention would result in no jobs completing.

Another crucial feature a system must provide before an IT department will deploy it is client cleanliness: it is crucial that the computing resources state is identical after running an application as it was before running the application.

Existing virtual machines focus primarily on providing desktop and application security from Java to VMWare, but they have not focused on the complete list of unobtrusiveness features described above. In addition, our approach focuses on providing x86 binary virtual machine to span Windows NT, 2000, and XP that can quickly installed and uninstalled into an IT environment, leaving a clean system behind.

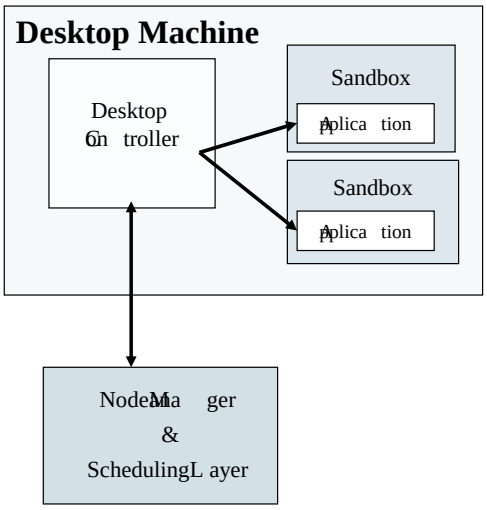

Figure 2: Entropia Desktop Components

4 Entropia Virtual Machine

The Entropia Virtual Machine achieves the above desktop virtual machine requirements, and its functionality can be broken up into the following three components:

- Desktop Controller - Monitors and controls the complete Entropia system and applications running on the desktop machine in terms of disk, memory, CPU, and I/O usage as well as other machine resources like the number of processes.
- Sandboxed Execution - Is the part of the EVM that is wrapped around the binary and lives within the virtualized binaries address space. It provides the virtualization and control of all of the binaries interactions with the operating system, as well as an interface to the desktop control mechanism. It is responsible for ensuring the security of the desktop machine.
- Application Security - Responsible for ensuring the security of the application and its data running on the desktop machine.

Figure 2 shows that the Desktop Controller gets assigned a subjob from the scheduling layer, and is responsible for launching the applications to run the subjob. It is also responsible for monitoring their behavior, which is described in detail in Section 4.3.

To contain the application inside the sandbox, application wrapping is used to insert a mediation layer between the application and the desktop system on which it will execute. We call this process, application virtual machine wrapping. In the following, we first explain how application wrapping is implemented, and the properties it achieves. Subsequently, we discuss sandboxing, desktop control, and application security in turn.

4.1 Application Wrapping in the Entropia Virtual Machine

To support the execution of a large number of applications, and to support the execution in a secure manner, Entropia employs binary sandboxing techniques that enable any Win32 application to be deployed in the Entropia system with no modification to the

source code and no special system support. End-users of the Entropia system can use their existing Win32 applications and deploy them on the Entropia system in a matter of minutes. This is significantly different than the early large-scale distributed computing systems like SETI@home and other competing systems that require rewriting and recompiling of the application source code to ensure safety and robustness.

4.1.1 Enabling Applications

Ease of application integration is key for the applicability and usability of a desktop PC grid computing solution. However, ease of application integration naturally conflicts with security and unobtrusiveness requirements. Many applications that were not designed to run in a distributed computing setting or applications that may be fragile or in development on desktop grids as-is may violate requirements of security and unobtrusiveness. These latter requirements necessarily restrict the actions of the application or impose constraints on how they operate, thereby reducing the set of applications that are suitable for distributed computing.

Entropia's approach to application integration, a process known as "sandboxing", is to automatically wrap an application in our virtual machine technology. When an application program is submitted to the Entropia system for execution, it is automatically sandboxed by the virtual machine. An application on the Entropia system executes within this sandbox and is not allowed to access or modify resources outside of the sandbox. However, the application is completely unaware of the fact that it is restricted within the sandbox since its interaction with the operating system is automatically mediated by the Entropia Virtual Machine. This mediation layer intercepts the system calls made by the application and ensures complete control over the application's interaction with the operating system and the desktop resources. The standard set of resources that are mediated include file system access, network communication, registry access, process control and memory access.

4.1.2 VM Wrapping

When an application is submitted to the Entropia system it is automatically wrapped inside our Virtual Machine using binary modification technology. The patched binary is then sent into the job submission system to be run on the Entropia grid. Since we wrap native binaries, we can support any language that compiles to x86 and 3rd party libraries (e.g. C, C++, C#, Java, FORTRAN, etc. and most important third-party shrink-wrapped software and common scripting languages). No source code is required, supporting the broadest possible range of applications. We use binary modification technology to enable strong security guarantees and ensure unobtrusive application execution.

We also wrapped interpreters like `cmd.exe`, `perl`, and `Java Virtual Machine` inside the Entropia Virtual Machine, and provided these wrapped versions with the Entropia installation, all contained within the EVM sandbox. This allows us to run scripts and java bytecode while still maintaining all of the desktop VM requirements listed above. Integrating these systems this way allows our EVM to provide resource and process control for these environments, which are described in more detail below.

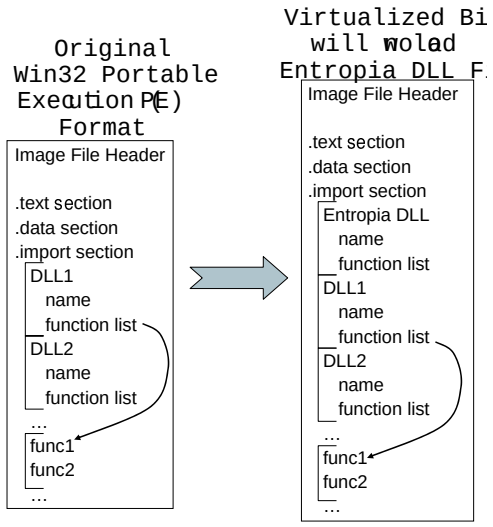

Figure 3: The left figure shows the import table before inserting the Entropia DLL. The right part of the figure shows the import table after the DLL has been inserted.

The wrapping of the binary is achieved by rewriting the import table of the binary, so that the Entropia vm.dll is the first dll in the list. This form of dll insertion is a binary insertion technique known in the Windows community. The before and after picture of the import table is shown in Figure **??**. When a binary is loaded under Windows, all of the dlls are loaded in the order in which they appear in the import table. By inserting the Entropia dll as the first dll in the list we are guaranteed that the Entropia dll will be loaded 1st and more importantly that the dll_main inside our vm.dll is executed before any non-system code for the program's execution. In dll_main we then encase the whole binary and any dynamic libraries it uses inside of the Entropia VM.

4.2 Sandboxed Execution

The goal of the sandbox is to control the application's interaction with the operating system, and to virtualize some of the operating system components. The sandbox mediates subjob access to the file system, registry, and graphical user interface. This prevents the subjob's processes from doing harm to the machine, and ensures that after subjob execution, the machine's state is the same as it was before running the subjob.

An application's interaction with the desktop machine must be strictly controlled to prevent it from adversely affecting the desktop user, machine configuration, or network. This control will prevent any application misbehavior (due to software bugs, inappropriate input parameters, misconfiguration, virus, etc.). Therefore, the Entropia sandbox isolates the grid application and ensures that it cannot invoke inappropriate system calls, nor inappropriately modify the desktop disk, registry, and other system resources.

As an example of the power of this technique, consider file system access by the application: an application uses standard Windows API's for opening, closing, reading and writing to a file. To protect the desktop user from the application, the sand-

box automatically maps the file and directory structure for the application to ensure that all files read or written remain within the Entropia sandbox. Therefore, an application may believe that it is reading or writing to the directory `C:\Program Files\` when in fact it is writing to a sandbox directory deep within the Entropia software installation.

Unlike general desktop applications, an application running on a desktop PC grid only needs access to a subset of the Windows operating system calls. The Windows operating system provides a rich set of API functionality, much of which is focused around an application's interaction with a user or with external devices. For example, there are Windows API calls for displaying graphics and playing music, and even for logging off a user or shutting down the machine. If these functions are invoked by an application, they would definitely disturb the desktop user. The Entropia sandbox prevents the grid application from accessing the parts of the Windows API that can cause these inappropriate interactions with the desktop machine and user.

4.2.1 Creating an OS Interception Layer

We *dynamically* intercept all important operating system calls before any non-system code can execute a program when our dll_main is invoked. In addition, the application binary and any libraries loaded are scanned to ensure that it does not perform any direct operating system interrupts. This leaves us with only having to intercept the system calls in the Windows system dlls. This layer that is inserted between the application binaries and the operating system is shown in Figure 4. We intercept and virtualize the Windows interfaces at the lowest level of the Windows NT dll system calls (mainly NT.dll, USER32.dll, GDI32.dll, and KERNEL32.dll). Therefore, all system calls used by the main binary and any dlls loaded are intercepted as they try to reach the operating system through the Windows NT dlls.

To provide this sandboxed environment, we use binary modification to intercept all important Windows API calls. This allows us to have complete control over the application and its interaction with the desktop machine. By interception, we mean that instead of calling the operating system interrupt with a set of parameters, our own virtualized routine is called instead. This allows us to completely block the call if desired, or to virtualize (change) its behavior before or after the real call to the operating system.

Most of the routines are intercepted using a trampoline approach analogous to [**?**]. In this approach, for a system call to be intercepted, we move the original routine, and any jump to the original routine will instead perform a direct jump (trampoline) to our own virtualized routine. Virtualization is performed, which may or may not involve calling the real routine. Then execution is returned as if the original call is returning.

4.2.2 Disallowing UnSandboxed Code

For our system to be secure, it is vital that we prevent the creation of executable code on disk or memory that is not virtualized or preprocessed under the Entropia Virtual Machine.

When starting a VMized process, we first scan the binary to make sure it is VMized. This ensures that the process will

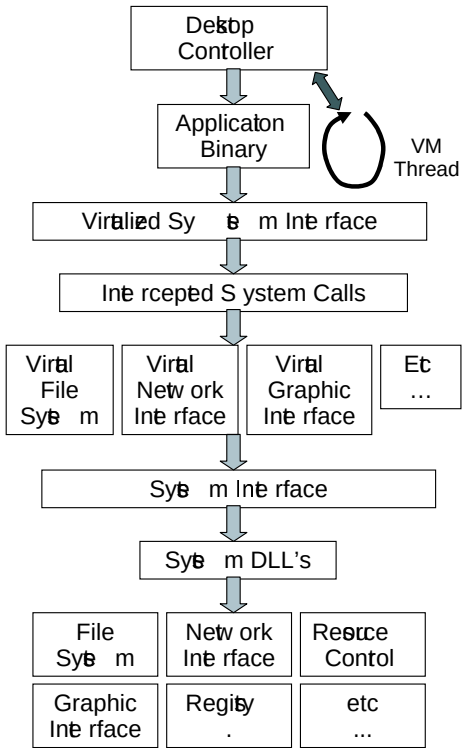

Figure 4: Shows the virtual layer placed between the application binary and dlls and the operating system. In addition, an EVM thread is started in the application to maintain a lifeline to the Desktop Controller.

run under the Entropia sandbox. This prevents an application from writing out a binary file, and then trying to invoke it. If the application being run writes one out, the create process will fail, since that operating system routine is intercepted and it will only allow VMized binaries to be invoked. In addition, we make sure that the binary matches the encrypted checksum sent over with the binary, which we describe in more detail in Section **??**.

The EVM does not allow applications with self modifying code to execute. We enforce this restriction to prevent non-vmized code being generated that can potentially directly call the operating system. When a VMized process starts execution, in dll_main, we lock down the binary's virtual address space specifying that the code portions of the address space are "executable" and "non-writable". Then all rest of the virtual address space has their permissions set as "non-executable". Windows provides the ability to specify these permissions for the virtual address space. By marking all code as executable and non-writable, and the rest of the virtual address space as non-executable, we guarantee that this cannot occur. Therefore, x86 binary applications the rely upon self-modifying code will not run on the Entropia system. We found this limitation not to be an, since we are dealing with scientific applications.

4.2.3 Virtualized Components

In this section we briefly summarize the different components that time was spent vitalizing. The operating system interfaces that did not fall into these categories were either left along if

there was no chance they could do any harm, or the interfaces were completely disabled by the sandbox.n

File Virtualization All of the File I/O interfaces in NT.dll are intercepted and virtualized to restrict applications access to the file system and to maintain a clean machine when the application finishes running.

When installing the Entropia system certain directories are marked as read only for the sandbox (e.g., $C:$
$WINNT$
$System$). All sandboxed applications are allowed to files from these ready only directories, but not allowed to write. If any of the files need to be or are written to, we use a copy on write mechanism. The file is copied to the sandbox directory, and the local copy is allowed to be updated.

Another unique feature of our file virtualization is to provide File I/O throttling. In order to maintain a certain level of unobtrusiveness, we want to ensure that the applications do not perform more than a certain amount of file I/O (reads and writes) data per second. We therefore, keep track of the amount of file I/O performed on a per time interval (1 second), and if the amount of file I/O exceeds this limit then the next call to read or write is suspended for a configurable amount time.

The final unique feature of our file virtualization is the automated file encryption described later in Section **??**.

GUI Virtualization The UI interface for applications was virtualized to make all windows invisible. Applications could create windows, we would ensure that they were hidden. You could therefore take off the shelf applications like 3D Studio Max, and run them under the sandbox without disturbing the user and without the user seeing what was being rendered.

An important feature of our UI virtualization is the ability to catch errors for windows programs and (1) prevent a dialog box from popping up, and (2) reporting the error back to the user. We provide this functionality with the GUI virtualization by intercepting the exception handler, preventing a dialog box from popping up, and then an error string is stored and sent back to the user. This is vital for the user to understand what is going wrong when there are problems with an application in a Desktop Grid system.

Registry Virtualization The registry was virtualized in a very similar fashion as the file system. Certain parts of the registry were read only, and writes were redirected to a local Entropia sandbox directory, while the application thinks it is updating the original location. Similarly, an update of a restricted registry entry will result in a copy on write.

Network Virtualization We virtualize the network by restricting what IP addresses an application can connect to. The typical use is that the application is not allowed to do any connection, but some applications may need to connect to a database server, and the IP address it is allowed to connect to is controlled through network virtualization.

Thread and Process Control The thread and process interfaces are virtualized to control the creation of threads and processes to prevent intended or unintended fork bombs. In addition, through vitalizing these interfaces we restrict subjobs to be run at a low process and thread priority. This is enforced whenever a CreateProcess enforces this low priority on all processes and threads created.

4.3 Desktop Control

In a desktop environment, it is important to make sure that the amount of resources an application consumes does not interfere with the usage of the desktop machine. For example, if an application uses more memory than what is available on a machine, spawns a significant number of threads, or uses up too much disk space, the machine can become unresponsive to user interaction and possibly even crash. To prevent this behavior, the Entropia system automatically monitors and limits application usage of a variety of key resources including CPU, memory, disk, I/O, threads, processes, etc. If an application attempts to use too many resources, the Entropia sandbox will pause or terminate all of the application's processes. The Entropia sandbox guarantees that you have strict control over all processes created when running an application.

A single Desktop Controller is used to monitor and control all of the resources on the desktop machines relative to the Entropia system. This is because an application may be running over several separate processes, so resource control decisions cannot be made on a per process view. Instead, a global view of all of the Entropia components and the application processes provided by the Desktop Controller is needed to make decisions to enforce unobtrusive behavior. Therefore, the controller needs to keep track of all of the running process of the Entropia system and applications on the desktop. This is enable by the use of the EVM Portal.

4.3.1 EVM Portal Thread and Process Control

A key component for the Entropia Virtual Machine is the ability to maintain control over the Entropia application processes running on the desktop machine. This functionality is provided through what we call the VM Portal.

When a VMized application starts running it starts a hidden thread in the application, we call the VM Portal thread. This is implemented such that the application is completely un-aware of the VM portal thread. The application does not see the thread when traversing over the existing threads, and cannot terminate the thread, since we have virtualized all of the operating system routines. The thread is used to communicate with the Entropia Desktop Controller to find out if it should pause or continue running the application, as well as to maintain a lifeline to the Desktop Controller.

As soon as the VM Portal starts up and and before execution is allowed to leave dll_main, the VM portal registers the running application with the Desktop Controller. After initial contact, a heart beat is kept between the VM portal and the Desktop Controller. If at anytime the heartbeat is lost, then the VM Portal thread will terminate the process. This is used to provide a safety measure for Client cleanliness in case somehow the En-

tropia client was terminated. If that happens, then all of the running VMized processes would also automatically shut-down.

Since all the running processes register with the Desktop Controller with the VM Portal, the portal is used to control all of the processes and threads to terminate, pause or resume their execution. This is important since some of the life science applications we deal with can run for days. When pausing the Entropia desktop, the Desktop Controller sends a pause command to all of the applications, and the VM Portal thread suspends all of the threads running in all of the application processes, except the VM Portal thread, which waits for the next command from the Controller.

Note, that the Desktop Controller may need to pause and resume the applications for other reasons besides resource controller. The Entropia DCGrid system allows IT departments to configure the system so that applications are only run during certain times of the day (e.g., at night). In this case, the Desktop Controller will pause the running application, and its memory will be paged to disk and repaged in when execution is resumed.

4.3.2 Enforcing Resource Limits

The goal of the Desktop Controller is to harvest unused computing resources by running subjobs unobtrusively on the machine. To accomplish this, it monitors desktop usage of the machine and resources used by the Entropia system and application. If desktop usage is high, the client will pause the subjob's execution using the VM Portal, avoiding possible resource contention. In this manner the Desktop Client acts like a big brother keeping watch over the subjob processes keeping their resource usage in line.

The resources monitored by the Controller include memory, disk, paging, I/O and process resource usage. If pausing the subjob processes does not remedy the situation, termination of the subjob may be necessary. The Controller provides different levels of unobtrusiveness that can be set at install time or set dynamically by an IT person. The highest level of unobtrusiveness monitors mouse movement, keyboard usage, memory usage, disk I/O usage, and CPU usage of not Entropia processes. If there is any usage at all, it suspends the subjobs using the VM Portal, and monitors the system to determine when to resume the subjobs. In addition, all threads and processes created are guaranteed to run at the lowest priority levels using Windows priorities to stay out of the way of the user. This is enforced by the sandbox. The lowest level of unobtrusiveness ignores keyboard and mouse usage, but uses the process and thread priorities to keep the subjobs out of the way of the user while monitoring the rest of the system.

In our deployments, the Desktop Controller usually has to take control of a subjob when a system starts to do external paging. When a subjob is submitted to the system, either the user specifies the amount of expected memory usage or an administrator associates a typical memory usage footprint with the application. The DCGrid Scheduler knows the amount of memory on every machine, and schedules the subjob appropriately to a machine with potentially enough memory. Memory and paging issues arise when either the users is doing a lot of memory intensive tasks or the memory requirements specified for that application or subjob are not right for the input being used. As stated above, when memory usage or paging exceeds a given threshold the application processes are paused or terminated.

Some resource issues seen for a DCGrid deployment are actually issues with the application, and not the because of user contention for the Desktop machine. One example, is that application developers often put tracing code in their application, and may forget to completely disable the tracing code off when submitting it to the grid. The Desktop client has a limit for a subjobs disk usage, and if a subjob exceeds this limit the application is terminated. Another example we have encountered is the number of processes or threads created for a sub-job is limited by the Desktop client. As described earlier, the Controller and Sandbox limits the number of processes and the threads created prevent a purposeful or inadvertently fork bomb.

4.3.3 Dealing with Resource Problems

Whenever an application is terminated due to a resource issue, this information needs to be tracked by the DCGrid scheduler to make better scheduling decisions in the future. In addition, the information is sent back to the administrator of the Desktop Grid so they can find problematic clients. Aggregate information about the failure of subjobs are also sent back to the Job Manager so the user knows that there is an issue with the subjob. An vital issue in creating a Desktop Grid is how to correctly classify failures into one of the following three categories.

- Client Black Hole - Need to determine if there is a problematic client. STATE SOME REASONS WHY A CLIENT CAN BE PROBLEMATIC. In this case, a client can consume and run through all of the work in the system. This needs to be detected and prevented. If a desktop machine has a job fail 3 times in a row it is marked as problematic at the physical node manager, and it is allowed to retry running jobs after a back-off period.
- Malformed Subjob - Need to deal with subjobs that never finish and and subjobs that have problems (e.g., incorrect parameters). One way to deal with some of this is that a subjob is allowed to specify minimum and max time time to run, and if the subjob violates those bounds, then there is something wrong with either the client or the subjob.
- Desktop User Contention -

4.4 Application Security

The last major component of the Entropia Virtual Machine is the ability to provide application security. Protection of the application and its data is another important aspect of security for grid computing. It is important to make sure that users cannot examine the contents of an application's data files, or tamper with the contents of the files when an application is run on a desktop machine. This application protection is needed to ensure the integrity of the results returned from running an application, and to protect the intellectual property of the data being processed and produced.

It is common in enterprise environments that desktop users do not have admin privileges. If application protection is a must

for a deployment, then the Entropia client and the sub-job processes are run under a special Entropia user account. This prevents other users from being able to look at the data of the running subjob processes (e.g, ReadProcess memory will fail). In addition, when Entropia is installed on the desktop machine a driver is installed that prevents users from seeing any content of the Entropia directories. They therefore, cannot look, invoke or copy any of the Entropia files, application files or data when running under Windows.

Even so, a desktop machine can potentially be compromised for example, by being rebooted using a Linux boot disk. To address this, the Entropia sandbox keeps all data files encrypted on disk, so that their contents are not accessible by non-Entropia applications. In addition, the sandbox automatically monitors and checks data integrity of grid applications and their data and result files. This ensures that accidental or intentional tampering with or removal of grid application files by desktop users will be detected, resulting in the rescheduling of the subjob on another client.

4.4.1 File Encryption

File encryption can be turned on or off during installation based on the application security needs of the customer. The capability is provided through the sandboxed file virtualization of the file system. Whenever a read or write is performed the corresponding information in decrypted (if it is not already), or encrypted.

To provide file encryption we used 3Des encryption. Whole block encryption is used. The size of the block can be set as small as 8 bytes, but we use a 64 byte block size. When reading from a file, a whole block needs to be read in in order to decrypt the block to get access to the requested data. When writing to a file, the block may need to be read in first, decrypted, the part of the block updated, encrypted and then the full block written to the file.

The only complication we ran into for doing encryption is dealing with memory mapped files. This was handled by intercepting the memory mapped file exception handler, so that whenever a memory mapped page that was not in memory, we would manually load it in and decrypt it.

4.4.2 Detecting Application Tampering

To make sure the application and data files have not been tampered with if the machine was rebooted e.g. under Linux so that a user had access to the Entropia file system, a configuration file is kept that maintains an encrypted checksum of the binary and data file. This checksum is checked whenever a binary is loaded or a data file is opened or sent back to the Entropia Server. Then when a data file is closed and the file has been modified the checksum is created.

5 Performance Evaluation

The Entropia DC Grid has been deployed at over 12 industry sites, and have been used for over 50 applications. We have even taken shrinked wrapped applications1 like 3D Studio Max and the Java Virtual Machine, and were able to automatically wrap them inside of the Entropia Virtual Machine and run them on DC Grid.

The majority of deployments of DC Grid have been to pharmaceutical companies running Virtual Screening and Sequence Analysis algorithms. In Figure **??**, we demonstrate the behavior of the Entropia Virtual Machine on four of these applications (DLPOLY, FRED, HmmerSearch, and MOE), but first briefly describe these application areas to provide an example of the applications needs.

Virtual Screening - One of the most successful early applications is virtual screening, the testing of hundreds of thousands (to millions) of candidate drug molecules to see if they alter the activity of a target protein that results in unhealthy affects. At present, all commercial drugs address only 122 targets, with the top 100 selling drugs addressing only 45[18]. Current estimates place the number of "druggable" genes at 5-10,000, with each gene encoding for around ten proteins, with two to three percent considered high value targets. Testing typically involves assessing the binding affinity of the test molecule to a specific place on a protein in a procedure commonly called docking. Docking codes (e.g., FRED [13], DLPOLY [5], and MOE [5]) are well-matched for distributed computing as each candidate molecule can be evaluated independently. The amount of data required for each molecular evaluation is small–basically the atomic coordinates of the molecules–and the essential results are even smaller, a binding score. The computation per molecule ranges from seconds to tens of minutes or more on an average PC. The coordination overhead can be further reduced by bundling sets of molecules or increasing the rigor of the evaluation. Low thresholds can be set for an initial scan to quickly eliminate clearly unsuitable candidates and the remaining molecules can be evaluated more rigorously.

Sequence Analysis — An important application area is DNA or protein sequence analysis applications (e.g., HmmerSearch[3]). In these cases one sequence or set of sequences is compared to another sequence or set of sequences and evaluated for similarity. The sequence sizes vary, but each comparison is independent. Sets of millions of sequences (gigabytes) can be partitioned into thousands of slices, yielding massive parallelism. Each compute client receives a set of sequences to compare and the size of the database, enabling it to calculate expectation values properly for the final composite result. This simple model allows the distributed computing version to return results equivalent to serial job execution. Distributing the data in this manner not only achieves massive input/output concurrency, but actually reduces the memory requirements for each run, since many sequence analysis programs hold all the data in memory.

Figure **??**, shows the results for these four applications without the Entropia Virtual Machine and with the EVM. The first column shows the average running time, the next two columns the average bytes written and read per run, followed by a column showing the ratio of I/O to execution time. The final two columns compares the ratio of I/O to execution time and execution time of the non-virtualized runs and the runs using the Entropia Virtual Machine. All of the EVM results are with the

Application	Average Running Time (seconds)	Average Bytes Written (kb)	Average Bytes Read (kb)	Sum Bytes I/O / Run Time	Sum Bytes (I/O) NonVmized / Vmized	Run Time NonVmized / Vmized
DLPOLY	1917.9	1622.0	1019.0	1.4		
DLPOLY w/ EVM	1924.9	2396.0	1801.0	2.2	1.6	1.0
FRED	244.1	838.0	552.0	5.7		
FRED w/ EVM	242.6	907.0	603.0	6.2	1.1	1.0
HmmSearch	5683.5	0.1	269357.0	47.5		
HmmSearch w/ EVM	5763.4	0.1	269357.0	46.9	1.0	1.0
MOE	839.6	77.0	2335.0	2.9		
MOE w EVM	889.1	853.0	2409.0	3.7	1.4	1.1

Figure 5: Execution and I/O results for Virtual Screening and Sequence Analysis applications running with and without the Entropia Virtual Machine.

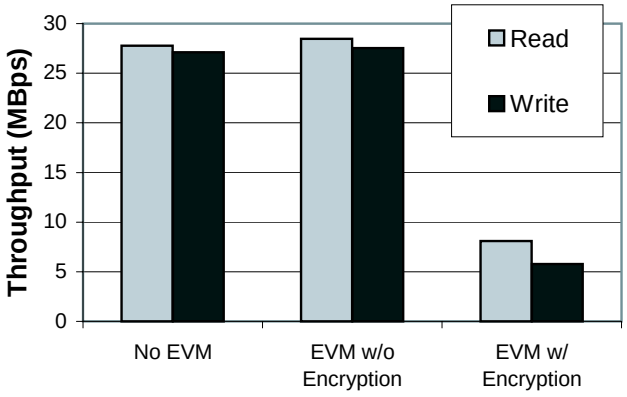

Figure 6: The File I/O achievable when using the Entropia Virtual Machine with and without encryption in comparison to no virtual machine.

sandbox file encryption feature on. The results show that 0% to 60% increase in I/O is created by the file encryption. As described in Section **??**, this is because the encryption is performed on a block basis, and a write may require a read to bring the rest of the block in to write out the full encrypted block.

To range of overhead capable from using the Entropia Virtual Machine we conducted a controlled experiment using an application that read and wrote X bytes per N instructions executed, and did it at a stride of Y. Figure 6 shows the amount of file I/O achievable without any virtual machine for our test application, when using the Entropia Virtual Machine without encryption, and with using EVM with encryption. The results show that when using encryption the file I/O throughput is reduced to be 1/4 of that without encryption.

Overall, we have found that the Entropia VM without encryption has very little overhead. Encryption is only needed if the customer wants the Application Security feature. For the applications we have deployed at these customer sites we see from 0% to 60% overhead when using file encryption.

6 Related Work

This paper introduces Entropia Virtual Machine, which addresses a few key problems in desktop grid systems, such as desktop client security, application data integration and confidentiality, unobtrusiveness, ease of application integration, manageability and so on. These requirements from the desktop grid environment distinct Entropia VM from other virtual machines in the literature.

Web-based general purpose virtual machines, such as Java Virtual Machine (JVM) [16, 11] and .NET [14], have been well studied for many years. They are different to Entropia VM in many ways. First, they focus on user-interactive business applications; while Entropia VM focuses on desktop grid applications, which are mostly computationally bound and/or data intensive. Second, these web-based VMs have a very different security model, which is appropriate for business applications, but not suitable for desktop grid applications. Third, and more importantly, because of the less stringent performance requirement, web-based VMs normally sacrifice performance to meet their different security and portability goals. On the other hand, performance is one of the most important requirement in desktop grid systems.

Another class of virtual machine systems is exemplified by VMware [20, 21] and Terra [9]. They allow multiple operating systems to run concurrently on a same hardware resource, providing each OS an isolated virtual machine. A main goal for VMware [20, 21] is to enable applications written for different platforms to seamlessly share the same physical resources. Entropia VM addresses different problems and they are complementary in the sense that VMware technology can enable a wider range of applications being deployed in the desktop grid environment.

Terra [9] has a different design goal than VMware from the observation that different applications may have entirely different security requirements, which fundamentally post very different requirements on the underlying operating system design. Therefore a common operating system interface cannot most effectively meet all those requirements. Terra addresses this problem by isolating each OS into a virtual machine protected by Terra. Therefore, applications can pick a most appropriate operating system to meet their security needs; while multiple applications can still share the same physical resources. Terra differs from Entropia VM in that it is a general framework to provide tamper-resistant isolation among multiple VMs running on a same physical resource; whereas Entropia VM is more focused on isolating desktop grid applications from the hosting desktop computers, which is more specific than what Terra addresses and consequently Entropia VM is a more light-weight solution. Furthermore, Entropia VM also addresses other problems in desktop grid computing, such as unobtrusiveness and manageability, which are not the focus of Terra.

7 Summary and Futures

Need to write summary...

8 Acknowledgments

We gratefully acknowledge the contributions of the talented engineers and architects at Entropia to the design and implementation of the Entropia system. We specifically acknowledge the contributions of Madhu Bommasamudram, Rajiv Gupta, Steve Kroetsch, Majid Entezam, and Shawn Marlin to the definition and development of the Entropia Virtual Machine.

References

[1] David Barkai. *Peer-To-Peer Computing: Technologies For Sharing and Collaborating on the Net*. Intel Press, 2001.

[2] A. Bricker, M. Litzkow, and M. Livny. Condor technical summary. Technical Report 1069, Department of Computer Science, University of Wisconsin, Madison, WI, January 1992.

[3] S.R. Eddy. HMMER: Profile hidden markov models for biological sequence analysis, 2001. http://hmmer.wustl.edu/.

[4] Entropia. Entropia announces support for open grid services architecture. Press release, Entropia, Inc., Feb 2002.

[5] T.J.A Ewing and I.D. Kuntz. Critical evaluation of search algorithms for automated molecular docking and database screening. *Journal of Computational Chemistry*, 9(18):1175–1189, 1997.

[6] I. Foster. *The Grid: Blueprint For a New Computing Infrastructure*. Morgan Kaufmann, 1998.

[7] I. Foster and C. Kesselman. The globus project: A status report. In *IPPS/SPDP '98 Heterogeneous Computing Workshop*, 1998.

[8] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.

[9] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. Terra: A virtual machine-based platform for trusted computing. In *SOSP*, Bolton Landing, New York, 2003. ACM.

[10] A. Grimshaw and W. Wulf. The legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1), 1997.

[11] Tim Lindholm and Frank Yellin. *The Java Virtual Machine Specification*. Addison-Wesley, 1999.

[12] J. Lyman. Intel debuts 2.2ghz pentium 4 chip, Jan 7 2002. The News Factor.

[13] Mark McGann. FRED: Fast rigid exhaustive docking, 2001. OpenEye.

[14] Microsoft Corporation. http://www.microsoft.com/net/.

[15] Platform Computing. The Load Sharing Facility. http://www.platform.com.

[16] Sun Microsystems. http://java.sun.com.

[17] Sun Microsystems. Jxta. http://www.jxta.org.

[18] A. Thayer. Genomics moves on, Oct 14 2002. Chemical and Engineering News.

[19] United Devices. the MetaProcessor platform. http://www.ud.com.

[20] VMware. Vmware virtual platform technology white paper. Technical report, VMware Inc., February 1999.

[21] Carl A. Waldspurger. Memory resource management in vmware esx server. In *OSDI*, Boston, 2002. USENIX.