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}@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) prevent the distributed computing application from unwarranted access or modification of data and files on the desktop PC, 2) control the distributed computing application's resource usage and behavior as it runs on the desktop PC, and 3) provide protection for the distributed application's program and its data. In this paper we describe the Entropia Virtual Machine, and the solutions it embodies for each of these challenges.

Categories and Subject Descriptors: D.4.7 [Organization and Design]: Distributed Systems, D.4.6 [Security and Protection]: Access Controls

General Terms: Management and Security

Keywords: Grid Computing, Virtual Machine, and Desktop Grids

1. INTRODUCTION

For over five years, the largest computing systems in the world have been based on "distributed computing" through the assembly of a large number of PC's over the Internet. These "grid" systems sustain multiple teraflops continuously by aggregating tens 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 (2.2 Gigahertz machines with multi-gigaop capabilities[20]), 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.

In order for a desktop grid to be accepted for enterprise deployment, the system must adequately protect the desktop machine from malformed or malicious applications. The system must make sure that the application does 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 of the Entropia Virtual Machine (EVM), which meets these requirements. The Entropia Virtual Machine supports the safe and controlled execution of applications expressed as native x86 binaries for Windows NT, 2000 and XP. The EVM uses binary modification technology and device drivers to achieve this mediation.

The paper is organized as follows. Section 2 gives an overview of the Entropia DCGrid Architecture. The requirements for a Desktop Grid Virtual Machine are described in Section 3. The design and capabilities of the Entropia Virtual Machine are explained in Section 4. Section 5 presents data from a performance evaluation of the EVM, using real application programs deployed at commercial sites. Finally, we discuss related work in Section 6, and conclude with a summary of the paper in Section 7.

2. ENTROPIA DCGRID ARCHITECTURE

We first provide a brief overview of the Entropia Desktop Distributed Computing Grid (DCGrid) in order to understand how the Entropia Virtual Machine fits into the overall system. Complete details of the Entropia DCGrid Architecture can be found in [6].

The Entropia DCGrid aggregates the raw desktop resources into a single logical resource. This logical resource is reliable and predictable despite the fact that the underlying raw resources are unreliable (machines may be turned off or rebooted) and unpredictable (machines may be heavily used by the desktop user at any time). This logical resource provides high performance for applications through parallelism, and the Entropia Virtual Machine provides protection for the desktop PC, unobtrusive behavior for the user of that machine, and protection for the application's data.

2.1 Layered Architecture

The Entropia server-side system architecture is composed of three separate layers as shown in Figure 1. At the bottom is the Physical Node Management layer that provides basic

VEE'05, June 11-12, 2005, Chicago, Illinois, USA.

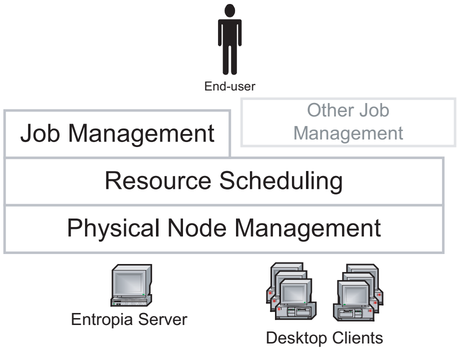

Figure 1: Architecture of the Entropia Desktop Distributed Computing Grid.

communication to and from the client, the naming (unique identification) of client machines, security, and node resource management. On top of this layer is the Resource Scheduling layer that provides resource matching, scheduling of work to client machines, 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. We next give a brief overview of each of these layers and how the Entropia Virtual Machine fits into this system.

Physical Node (Client) Management (PNM) - The desktop distributed computing environment presents unique challenges for providing a reliable computing resource. Individual client machines are under the control of the desktop user or IT manager. They can be shut down, 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 PNM layer of the Entropia DCGrid manages these and other low-level reliability issues to allow the running of applications on top of the Entropia Virtual Machine on the client machines. The PNM layer specifically provides resource and application management. The resource management services capture a wealth of static and dynamic information about each physical client (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. This information is used during Resource Scheduling. The application management provides basic facilities to allow processes to be run on the clients. This includes staging files from the server and clients, and error reporting.

Job Management (JM) - 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 Resource Scheduling layer for execution. A **Subjob** is the unit of schedulable work that is to be scheduled and run on a desktop machine. The Job Management layer of the Entropia DCGrid 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 (e.g., 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.

Resource Scheduling (RS) - The Entropia DCGrid consists of client resources with a wide variety of configurations and capabilities. The Resource Scheduling layer takes the subjobs and matches them to appropriate client resources and schedules them for execution. Since the clients provided by the PNM layer may be unreliable, the RS layer must adapt to changes in the resource status and client availability, and to failure rates that are considerably higher than in traditional cluster environments. Failures can also come from unreliable applications, and the RS layer has logic to identify these subjobs.

Entropia Virtual Machine (EVM) - A separate Entropia Virtual Machine runs on each Desktop Client shown in Figure 1. The EVM is responsible for starting the execution of the subjob, monitoring and enforcing the unobtrusive execution of the subjob, and mediating the subjob's interaction with the operating system to provide security for the desktop client and the subjob being run. The EVM communicates with the Resource Scheduler to get new work (subjobs), and communicates subjob files and their results via the Physical Node Management layer.

The steps below show the typical life of a job as it flows through the Entropia DCGrid, specifying which layer is involved at each step.

1. (JM) Authenticate DCGrid user to system.
2. (JM) User submits a job to the Entropia system.
3. (JM) The job is broken into multiple subjobs.
4. (JM) The binaries to run the job are automatically wrapped using binary modification technology 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. (JM-RS) The subjobs are submitted from the Job Management layer to the Resource Scheduling layer specifying input files (including executables, libraries and output files), resource requirements (including minimum memory, disk, processor speed, run time, priority, etc.), and a script to start executing the subjob on the client.
6. (RS-PNM) The Resource Scheduler schedules the subjob assigning it to a client that meets the resource constraints as specified by the Physical Node Management layer.
7. (PNM-EVM) The subjob and its files are transfered to the client machine to run.

8. (EVM) The subjob is then run under the EVM on the desktop machine.

9. (JM) User optionally checks subjob status.

10. (EVM-PNM-JM) The result files are sent back when the subjob is completed.

This paper focuses on how the Entropia Virtual Machine runs a subjob on a client machine, which corresponds to step 8 above. In addition, we will describe how binaries are modified during job submission (shown in step 4 above) so that they are allowed to run under the EVM.

3. DESKTOP GRID VIRTUAL MACHINE REQUIREMENTS

In order for a desktop grid to be adopted in an enterprise deployment, we found that our customers required a virtual machine to provide the following features:

Desktop Security — The distributed computing system must protect the integrity of the computing resources that it is aggregating. The subjob must be prevented from accessing or modifying the desktop machine's data.

Clean Execution Environment — The state of the client machine (e.g., file system, registry, etc) must be in the exact same identical state after executing a subjob as it was before executing it.

Unobtrusiveness — The subjob running on the desktop machine needs to stay out of the way of the user. The desktop client shares resources (computing, storage, and network resources) with other applications and the user of the machine. Therefore, the use of these resources by the grid application must be unobtrusive, and non-aggressive where there is competition.

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.

4. ENTROPIA VIRTUAL MACHINE

The Entropia Virtual Machine (EVM) achieves the above desktop grid requirements through the use of a Sandbox Execution Layer and a monitor called the Desktop Controller, as seen in Figure 2.

- **Desktop Controller -** monitors and controls the subjob running on the client machine in terms of disk, memory, CPU, and I/O usage as well as other machine resources like the number of processes and threads invoked. It therefore provides many of the unobtrusive requirements. The Desktop Controller gets assigned a subjob from the Resource Scheduling layer, and is responsible for launching the application processes to run the subjob. It is also responsible for monitoring the subjob's processes, which is described in detail in Section 4.2.

- **Sandbox Execution Layer -** provides the virtualization and control of all of the binary's interactions with the operating system, as well as an interface to the desktop controller. It provides desktop security, clean execution environment, application security, and is part of the solution for providing unobtrusive features. To contain the subjob inside the sandbox, we use both device driver and application wrapping technology to insert a mediation layer between the subjob and the desktop system on which it will execute.

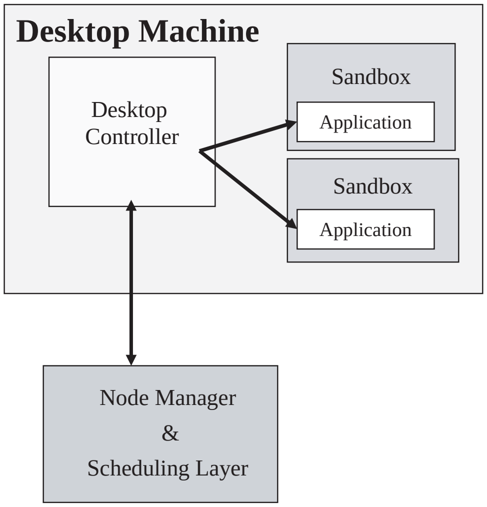

Figure 2: Entropia Desktop Components

The EVM comprises of both the Desktop Controller and the Sandboxed Execution Layer, and is installed and run on each client machine. Only one EVM is run on a client machine, which would be used to control the running of one subjob (for a uniprocessor machine) to multiple subjobs (for a multi-processor). Note that a single subjob can consist of multiple processes and threads, and the Desktop Controller is responsible for all of the Entropia processes running on the client machine as well as all of the subjob processes.

In Section 4.1, we first explain how an application is integrated and validated to run on the EVM. We then describe in Section 4.2 how the Desktop Controller makes sure that the subjob runs unobtrusively on the client. In Section 4.3, we describe the security and virtualization provided by the EVM sandbox. Finally, Section 4.4 describes how the EVM provides protection for the subjob running on the client.

4.1 Creating and Identifying a Valid Sandboxed Application

To support the execution of a large number of applications, and to support the execution in a secure manner, Entropia provides binary sandboxing with no modification to the source code. Our approach allows end-users of the Entropia DCGrid to use their existing Win32 applications and deploy them on the Entropia DCGrid in a matter of minutes.

4.1.1 Wrapping an Application in the EVM

Ease of application integration is key for the applicability and usability of a desktop PC grid computing solution. Entropia's approach to application integration is a process known as "sandboxing". This mediation layer intercepts the system calls made by the application and provides control over the application's interaction with the operating system and the desktop resources.

When an application (subjob) is submitted to the Entropia DCGrid it is automatically wrapped inside the Entropia Virtual Machine using binary modification technology (Step 4 in Section 2.1). The patched binary is then sent into the Resource Scheduler to be run on the Entropia DCGrid. 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 also wrapped interpreters like `cmd.exe`, `perl`, and the `Java Virtual Machine` inside the Entropia Virtual Machine, and provide 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 the desktop VM requirements listed above.

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. 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 ensuring that the Entropia dll is the first dll in the list, we are guaranteed that the Entropia dll will be loaded first. This means that the *dll_main* inside our `vm.dll` will be executed before any non-system code for the program's execution. When our *dll_main* executes it dynamically modifies the loaded binary and any dynamic libraries used to intercept system calls. This is described in Section 4.3.

4.1.2 Validating Binaries for Execution

To provide Desktop Security and Application Security it is important that we do not run executables with the Entropia Virtual Machine that have not been sandboxed. When an application is submitted to run on the Entropia DCGrid, all of the application's non-OS binaries and dll files must be provided. The import table of these files are then patched as described above. Then a cryptographic checksum of each file is created. The checksum is used to verify that the file being invoked (a) has been patched to be contained within the EVM sandbox, and (b) that when we want to execute the file it has not changed since we patched it. This is used to provide Application Security, which we describe in more detail in Section 4.4.

When the application is sent to the client, a configuration file containing the list of sandboxed files and their checksums is also sent to the client to provide file validation. The configuration file is encrypted during subjob submission. After being transfered to the client machine, it is stored on disk in encrypted form, and the key is securely communicated to the EVM on the client to invoke and run the subjob.

Before any application binary file can be invoked by the Desktop Controller, the checksum is first validated. In addition, when a dynamic library is loaded the checksum of the library is validated before execution is allowed to proceed. A sandboxed application is allowed to launch another application using `CreateProcess` only if (a) it is registered with the EVM in the configuration file and (b) its checksum matches. Since we intercept `CreateProcess` to perform this validation, we prevent the EVM sandboxed applications from launching any unsandboxed processes.

In addition, for Desktop Security, we prevent subjob processes from opening with write permissions any binary listed in the configuration file via the EVM Sandbox Execution Layer.

4.2 Desktop Control

To meet the Unobtrusiveness requirement it is important to make sure that the amount of resources a subjob consumes does not interfere with the usage of the desktop machine. For example, if a subjob uses more memory than 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 EVM automatically monitors and limits subjob usage of a variety of key resources including CPU, memory, disk, I/O, threads, processes, etc. If a subjob attempts to use too many resources, the EVM will pause or terminate all of the subjob's processes. In addition, the EVM guarantees that we have strict control over all processes created when running a subjob.

The responsibility for fulfilling the Unobtrusive requirement is partitioned between the Desktop Controller and the Sandbox Execution Layer. The Sandbox Execution Layer provides local control by limiting resources on a per process and per thread basis. It monitors the rate in which resources are used (e.g., file and network I/O, thread and process creation, etc) per second for each process and thread. If the rates exceed a configurable limit then the resources are throttled to keep the rate within the limit. This is achieved by pausing the system routine using the resource, and is described in more detail in Section 4.3.3. In comparison, the Desktop Controller provides a global view of everything running on the client machine and it is the last line of defense for making sure the subjob runs unobtrusively. When a given resource limit is exceeded its main course of action is to either pause or terminate the subjob.

A single Desktop Controller is run on the client machine, and is used to monitor and control *all* of the resources on the desktop machine used by the EVM and the subjob's processes being run on the client machine. This is because a subjob may be running over several separate processes on a machine, so resource control decisions cannot be made on a per process view. Instead, a global view of all of the Entropia components and the subjob processes provided by the Desktop Controller are needed to make decisions to enforce unobtrusive behavior. The Desktop Controller knows which processes belong to the EVM and which belong to the subjob being run, so it can perform resource control separately for these two groups of processes. This is enabled by the use of the VM Portal, which we describe next.

4.2.1 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 sandboxed application starts running it starts a hidden thread (VM Portal) in the application. This is implemented such that the application is completely un-aware of the VM Portal. The application does not see the thread when traversing over the existing threads, and cannot terminate the thread, since we have virtualized those operating system routines. The thread is used to communicate with the 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 before execution is allowed to leave *dll_main*, the VM Portal registers the newly running process 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 from within (commit hari cari). This is used to provide a safety measure for Client cleanliness in case somehow the Entropia client crashes or is terminated. If that happens, then all of the running sandboxed processes would also automatically be shut down.

Since all the running processes register using the VM Portal with the Desktop Controller, this communication path is used to control the termination, pausing and resuming of all of the processes and threads. Having the ability to pause and resume is important since some of the life science applications we deal with can run for days. When pausing the subjob, the Desktop Controller sends a pause command to all of the subjob processes. The VM Portal in each process then suspends all of the threads running in its process, except itself (the VM Portal thread), which sits idle waiting for the next command from the Desktop Controller.

The Desktop Controller may pause and resume a subjob's processes to keep the client unobtrusive when a user starts using a machine. In addition, the pause and resume functionality was a critical feature for some companies to control when the client executed. For example, the Entropia DCGrid 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 processes in the morning, and its memory will be paged to disk. Then execution would be resumed at night, and the processes would be re-paged in to allow execution to continue.

4.2.2 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 EVM and subjob. 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 Controller acts like a guardian keeping watch over the subjob processes keeping their resource usage in line.

The resources monitored by the Desktop Controller include memory, disk, paging, I/O, process resource usage, and other resources. If pausing the subjob processes does not remedy the situation, termination of the subjob may be necessary. The Desktop Controller provides different levels of unobtrusiveness that can be set at install time or set dynamically by an administrator of the Entropia DCGrid. The highest level of unobtrusiveness monitors mouse movement, keyboard usage, memory usage, disk I/O usage, and CPU usage of non Entropia processes. If there is any usage at all, it suspends the subjob's processes using the VM Portal, and monitors the system to determine when to resume the subjob's execution. 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 EVM. The lowest level of unobtrusiveness supported by the Entropia DCGrid ignores keyboard and mouse usage by the user, and instead uses the process and thread priorities to keep the subjob processes out of the way of the user, while still monitoring the rest of the system. This allows the subjob to use the processor between keystrokes.

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 Resource 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 user is doing a lot of memory intensive tasks or the memory requirements specified for that subjob were not right for the input being used. When memory usage or paging exceeds a given threshold, the application processes are paused or terminated by the Desktop Controller.

Some resource issues seen for a DCGrid deployment are actually issues with the subjob, and not because of user contention for the Desktop machine. One example is that application developers often put tracing code in their program, and they may forget to completely disable the tracing code when submitting it to DCGrid. The Desktop Controller has a resource limit for a subjob's disk usage, and if a subjob exceeds this limit the application is terminated, and the reason is sent back to the user who submitted the subjob. Another example we have encountered is having the limit we have set for the maximum number of processes or threads allowed for a subjob being exceeded. This was due to an error in a subjob, which inadvertently started to recursively spawn threads. Disaster was prevented, since the Desktop Controller and our Sandbox layer limited the number of processes and the threads created, and the subjob was terminated when it hit this limit.

4.2.3 Dealing with Resource Problems

Whenever a subjob is terminated due to a resource issue, this information needs to be tracked by the Resource Scheduler to make better scheduling decisions in the future. In addition, the information is sent back to the administrator of the Entropia DCGrid 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 a job. A vital issue in creating a Desktop Grid is how to correctly classify failures into one of the following three categories.

- Desktop Resource Contention - If the subjob is terminated due to resource usage issues, what the resource problem was is communicated back to the Resource Scheduler and the user. In addition, if a subjob is repeatedly paused due to desktop user activity, the subjob will eventually be terminated. The cause of termination is signaled back to the Resource Scheduler so the subjob can be rescheduled on a different client with more appropriate resources. If the subjob is terminated three different clients due to resource limits it is marked as failed and this fact is communicated back to the Job Manager.

- Client Black Hole - There can be a problem with the client, which does not allow the correct execution of subjobs. For example, if for any reason the Entropia client is misconfigured for application launch, or something exists (e.g., a malformed system driver installed by the user) on the desktop machine that prevents appropriate functionality of the client, subjobs which should run for minutes or hours can stop running in seconds. In this case, a client can consume and run through all of the work (subjobs) in the system in a matter of minutes, since as soon as a subjob finishes on the problem client the EVM will request the next subjob to run. This needs to be detected and prevented. If a client machine has 3 different subjobs quickly finish/terminate in a row, the client is marked as problematic at the Physical Node Manager. The client is allowed to retry running subjobs after a back-off period or after an administrator has examined the problematic client and the subjobs it had a problem with.

- Malformed Subjob - Subjobs that never finish and subjobs that are misconfigured need to also be identified. For example, incorrect parameters to a subjob can cause it to terminate immediately or run forever. One way we addressed this is that a subjob is allowed to specify the 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. At the Job Manager, if all of the subjobs for a Job are coming back as failed from different clients, then there is a high probability the subjobs are misconfigured. In addition, as will be described later, any exceptions a subjob encounters are tracked using the EVM and transmitted back to the user.

4.3 Sandbox Execution Layer

The goal of the sandbox is to control the subjob's interaction with the operating system, and to virtualize some of the operating system components. The sandbox mediates subjob access to all system APIs that could affect the behavior of the system. The list is too large to enumerate, but includes the file system, registry, graphical user interface, networking, keyboard, mouse and I/O devices, etc... Some of the APIs (file system, registry, and network) are mediated others (mouse, graphical user interface) are disabled. This prevents the subjob's processes from disturbing the user of the machine, and ensures that after subjob execution, the machine's state can be restored to the same state as it was before running the subjob.

An application's interaction with the desktop machine must be controlled to prevent it from adversely affecting the desktop user, machine configuration, or network. This control will prevent application misbehavior due to software bugs, inappropriate subjob input parameters, and subjob misconfiguration, and provide protection against malicious subjobs. Therefore, the Entropia sandbox isolates the grid application to prevent it from invoking inappropriate system calls, or inappropriately modifying the desktop disk, registry, and other system resources.

An important point is that unlike general desktop applications, an application running on a desktop PC grid only needs access to a subset of the Windows operating system calls. For desktop grid applications, many of the system calls can be disabled or their functionality restricted. 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, playing music, mouse movement, 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.

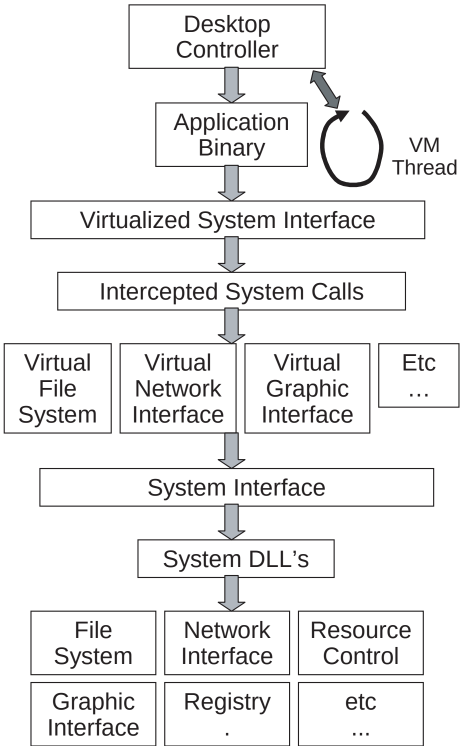

Figure 3: Shows the virtual layer placed between the application binary and dlls and the operating system. In addition, a VM Portal (VM thread) is started in the application to maintain a lifeline to the Desktop Controller.

4.3.1 Creating an OS Interception Layer

The Entropia Virtual Machine uses two levels, with distinct purposes, to provide a virtualized system interface. The first level is implemented via a device driver installed on the client. The device driver intercepts specific Windows API calls that allow access to hardware resources (e.g., file system and registry) that if used can permanently change the state of the machine. The second level is implemented using binary modification to intercept and virtualize parts of the Windows interfaces. In Figure 3, the device driver and binary modification interception *combined* provide the virtualize system interface where the system calls are intercepted and then virtualized. This Sandbox layer is basically a virtual-machine monitor [18] to provide the requirements listed in Section 3.

Using binary modification by itself to provide system call interception has security holes in that an attacker can try to search the virtual address space to find the moved routine and the entry point to an intercepted function. Additional issues are discussed in [13, 15]. This is why we use a device driver approach to provide access security to important system resources, and then a more light weight binary interception approach to virtualize the manipulation of those resources.

4.3.1.1 Desktop Security through Device Driver Mediation.

Desktop security against malicious applications is provided by creating a mediating Windows device driver. The device driver is installed by an administrator when installing the EVM. Our Windows device driver runs as part of the operating system and *every single call to a mediated routine by all processes/threads* is examined by the driver code. This insures that a subjob process cannot bypass the sandbox.

When a system call occurs in Windows, control is transfered to a software interrupt handler inside the operating system. This handler takes as input a system call number and performs a table lookup with it to find the operating system address for the function to be called. Our mediating device driver replaces entries in this jump table with pointers to our own functions in the driver to intercept specific system calls. The driver only effects the routines that we have replaced in the system jump table.

The Desktop Controller knows the handle of all processes running sandboxed under the EVM as described in Section 4.2.1. This allows the EVM to determine what processes should be intercepted and which ones should not. The device driver is also given this list via communication with the Desktop Controller. For each access to a mediated routine, the device driver checks to see if the process handle is in this sandboxed list of processes. If it is, then the call is intercepted and virtualized. If it is not in the list, then the process may just passes on through with no restrictions, depending upon the routine being virtualized.

The driver check creates a minor amount of overhead on all processes for the routines intercepted. This overhead, at minimum, consists of an extra jump and a check to see how to deal with the process handle. Even though this overhead is small, we do not want to apply this type of driver mediation to every single intercepted function in order to minimize the overhead, especially for routines that are called repeatedly. We therefore only apply it to routines that *grant access* to Windows resources. For example, for File I/O, we mediate and virtualize with the driver the NT routines to open and close files, but we do not intercept the file read and write routines using the driver. The read and write routines are mediated using binary interception described below. This is to reduce the overhead for non-sanboxed applications running on the desktop machine. This insures that all sandboxed subjobs cannot open or create a file outside of the sandbox. In addition, we use this to insure that non-sanboxed applications cannot look inside the Entropia directory. To provide appropriate security and to leave the machine in a clean state, the driver interception approach is applied to grant access to file I/O, registry, creating processes and threads, and a few additional OS routines that have to be mediated for Desktop Security.

4.3.1.2 System Call Interception through Dynamic Binary Modification.

Once access is granted to a machine resource using the device driver interception approach, the OS routines used to manipulate those resources are intercepted through dynamic binary modification. This interception is implemented using a trampoline approach analogous to Detours [16]. The binary modification occurs dynamically when the binary starts executing when the EVM *dll_main* is invoked. For a system call to be intercepted, we move the original routine to a new point in the virtual address space. Then any jump to the original routine will instead perform a direct jump (trampoline) to our own virtualized routine. The virtualized routine will mediate the functions execution, which may or may not involve calling the real routine that was moved. Then execution is returned as if the original call is returning.

4.3.2 Self Modifying Code

For Desktop Security and Unobtrusiveness, the EVM does not allow applications with self modifying code to execute. When a sandboxed 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 the 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 prevents data from being executed as code. Note, not all x86 processors support this mode, but we expect them to do in the future. Therefore, pure x86 binary applications that rely upon self-modifying code will not run on the Entropia DCGrid on clients with this feature enabled.

We found that not allowing self modifying code not to be a major concern, since we are dealing with scientific applications. We only had one application that required this, which was when we ran the Java Virtual Machine. For the JVM, we turned off this restriction to allow it to create and execute JIT code.

4.3.3 Virtualized Components

In this section we briefly summarize some of the interesting features of the major components that were virtualized. The operating system interfaces that did not fall into these categories were either left alone if there was no chance they could do any harm, or the interfaces were completely disabled (e.g., mouse and keyboard I/O) by the sandbox.

4.3.3.1 File Virtualization.

All of the File I/O interfaces are intercepted and virtualized to redirect and restrict a subjob's access to the file system and to maintain a clean machine when the application finishes running. For example, a subjob believes that it is accessing a file in the directory `C:\Program Files\` when in fact it is accessing a sandbox directory deep within the Entropia software installation (e.g., `C:\Entropia\root\C:\Program Files\`).

In this example, the file name the subjob sees is `C:\Program Files\`. The virtualization is performed so the subjob only sees the non-virtualized file names. This includes even system routines that return full directory listings. Before opening a file the EVM translates the file name to the virtual file name by redirecting it to the sandbox directory, and the file name is converted to a full path file name.

Certain existing directories on the desktop machine may need to be accessed by subjobs. These directories (e.g., `C:\WINNT\System\`) are marked as read only, and their file names are not necessarily translated. All sandboxed subjobs are allowed to read files from these read only directories, but not allowed to write. If any of the files need to be written to, we use a copy on write mechanism. The file is copied to the sandbox directory, and the access by the subjob is redirected to the local copy, which it is allowed to update. We correctly keep track of the files that exist in the virtual file system, so that file names are correctly virtualized.

Another 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 subjobs do not perform more than a certain amount of file I/O (data reads and writes) per second. We therefore keep track of the amount of file I/O performed on a time interval (1 second) basis. If the amount of file I/O exceeds a configurable limit, then the next call to read or write is suspended for a specified amount of time to throttle the I/O.

Another interesting feature of our file virtualization is the automated file encryption for Application Security, which is described in Section 4.4.1.

4.3.3.2 Registry Virtualization.

The registry was virtualized in a very similar fashion to the file system. Certain parts of the registry were read only, and the rest marked as not accessible by a subjob running under the EVM. All writes to the registry were redirected to a local EVM sandbox registry path, 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 as described above.

4.3.3.3 GUI Virtualization.

The GUI interface for applications was virtualized to make all windows invisible, and to not allow certain windows to be created.

We found that we wanted to run some commercial applications on the Entropia DCGrid, which had windows. We therefore allowed applications that create windows to execute properly on the EVM by hiding the windows that were created. This allowed us to take off the shelf applications like Discreet's 3D Studio Max, and run them under the EVM sandbox without disturbing the user and without the user seeing what was being rendered.

An important feature of our GUI virtualization is the ability to catch errors for Windows programs to (1) prevent a dialog box from popping up, and (2) report the error back to the user. We provide this functionality with our GUI virtualization by intercepting the Window's exception handler. One important reason for preventing the dialog box from popping up is that functionality would suspend execution waiting for someone to press a button on the box. Therefore, our approach prevents all dialog boxes from popping up, and the error string that would have been displayed in the dialog box is stored and sent back to the user. We found this to be an important feature, since it allows a user submitting jobs with an error dialog box popping up to understand what is going wrong with a subjob.

4.3.3.4 Network Virtualization.

We virtualize the network by restricting what IP addresses an application can connect to. The typical use with the EVM is that the application is not allowed to perform any connection, but some applications may need to connect to a database server, and in this case the network virtualization is configured to allow a subjob to connect to a specific IP address. In addition, we provide network bandwidth throttling for send and receive, similar to File I/O throttling above, in order to make sure the subjob stays unobtrusive in terms of the amount of network bandwidth it is consuming.

4.3.3.5 Thread and Process Control.

The thread and process interfaces are virtualized to control the creation of threads and processes, and to prevent intended or unintended fork bombs. In addition, through virtualizing these interfaces we maintain a low thread priority for these subjobs on Windows.

4.4 Application Security

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

It is common in enterprise environments for desktop users to not have administrative privileges. If application protection is a must for a deployment, then (1) desktop users must not have administrative privileges, and (2) the EVM and the subjob processes are run under a special Entropia user account. This prevents other users from being able to look at the data of the EVM and running subjob processes (e.g, `ReadProcess` memory will fail). In addition, when Entropia is installed on the desktop machine the driver described in Section 4.3.1.1 prevents users from seeing any content of the Entropia directories. They therefore, cannot look, invoke or copy any of the Entropia files, subjob files or data when running under Windows.

Even so, a desktop machine can potentially be compromised to get access to data on the hard drive. For example, a machine can be rebooted using a Linux boot disk to access the hard drive. To address this, the Entropia sandbox keeps all data files encrypted on disk, so that their contents are not accessible if the disk can be compromised.

In addition, the sandbox automatically monitors and checks the data integrity of a grid application's input 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 is decrypted (if it is not already), or encrypted.

To provide file encryption we used 3DES encryption, and whole block encryption was 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 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, then encrypted, and finally the full encrypted block written to the

App	Avg Run Time (sec)	Average Bytes Written (KB)	Avg Bytes Read (KB)	Sum Bytes I/O / Run T	Bytes (I/O) VM / No VM	Run Time VM / No VM
DLPOLY	1917.9	1622.0	1019.0	1.4		
(EVM)	1924.9	2396.0	1801.0	2.2	1.6	1.0
FRED	244.1	838.0	552.0	5.7		
(EVM)	242.6	907.0	603.0	6.2	1.1	1.0
HmmSrch	5683.5	0.1	269357.0	47.5		
(EVM)	5763.4	0.1	269357.0	46.9	1.0	1.0
MOE	839.6	77.0	2335.0	2.9		
(EVM)	889.1	853.0	2409.0	3.7	1.4	1.1

Table 1: Execution and I/O results for Virtual Screening and Sequence Analysis applications running with and without the Entropia Virtual Machine.

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 is accessed, 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 (e.g., if the machine was rebooted 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 files as described in Section 4.1.2. This checksum is checked whenever a binary or dll is loaded, or a data file is opened or sent back to the Entropia Server. A new checksum is created for subjob data file on the client machine when a data file is closed and the file has been modified. For subjob binaries, the checksums are created when the application is sent over from the Resource Scheduler.

5. PERFORMANCE EVALUATION

The Entropia DCGrid has been deployed at over 12 industry sites, and has been used for over 50 applications. We have even taken shrink-wrapped applications, such Discreet's 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 the Entropia DCGrid. These 50+ applications have performance in the range shown for the four benchmarks we examine in this section.

The majority of DCGrid commercial deployments are in pharmaceutical companies for Virtual Screening and Sequence Analysis algorithms. Virtual screening is compute-intensive, and sequence analysis can be either compute or I/O-intensive, depending on the parameter settings. We have measured the performance of the Entropia Virtual Machine on four of these applications (DLPOLY, FRED, HmmerSearch, and MOE) to evaluate the EVM's performance. The results are summarized in Table 1. First, we briefly describe each of these application areas to provide further insights into the applications needs.

Virtual Screening - Virtual screening is the extension of wet laboratory high-throughput drug molecule screening techniques to the computational domain. To avoid the cost of wet-laboratory experiments, virtual screening uses computational techniques to evaluate hundreds of thousands to millions of candidate molecules for efficacy in altering the activity of a target protein.

The computational testing typically involves assessing the binding affinity of the candidate molecule to a specific target on a protein using a technique commonly called docking. Docking codes (e.g., FRED [21], DLPOLY [11], and MOE [11]) 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 hours 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 — Sequence analysis (BLAST [1] and HmmerSearch [9]) is an important bio-informatics technique for understanding possible drug toxicity, potency, and other biological interactions. Typically, a single sequence or set of sequences is compared to another sequence or set of sequences and evaluated for similarity. The size of the sequences being compared vary widely, but each comparison is independent. Genomes or proteomes of billions of symbols (gigabytes) can be partitioned into thousands of slices, yielding massive parallelism. Each compute client receives a set of sequences to compare and a slice 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 may even reduce the memory requirements for each run, since many sequence analysis programs hold all the data in memory.

Table 1 shows the results for these four applications alone (first line per program) and when running inside the EVM (second line). We conducted all experiments in this section on a Pentium 4 1.6GHz PC with 1GB memory running Windows XP Professional. All of the EVM results are with the sandbox file encryption feature enabled. The first column shows the average running time, and demonstrates the low overall overhead imposed on these applications through the virtualization provided by the Entropia Virtual Machine. Run times with the EVM increased by a maximum of 6%. The second and third columns present the average bytes written and read per subjob run, and the fourth column shows the ratio of I/O to execution time. The measurements show that all of the applications measured are all compute-intensive. The fifth column compares the quantity of I/O with and without the EVM. These numbers show that in a few cases, the EVM file interception and encryption increased the relative amount of I/O by up to 60%. As described in Section 4.4.1, this increase comes from the block-based encryption, which is only used when Application Security is required. As a result, a write may require a read to bring the rest of the block in to write out the full encrypted block. The final column compares overall execution times of the subjobs without the EVM with runs using the EVM, showing that the overall EVM overhead is modest.

To examine the range of overhead from using the file encryption, we also conducted a controlled experiment using a benchmark application that measures the throughput of

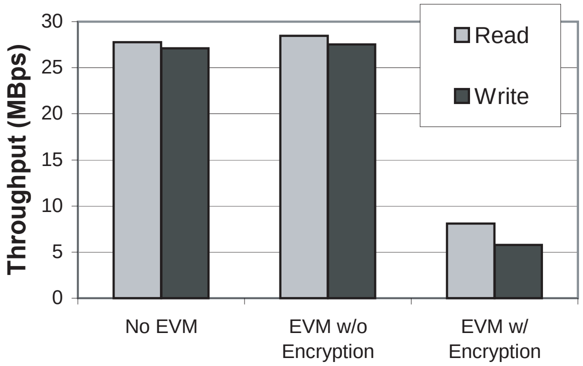

Figure 4: The File I/O achievable when using the Entropia Virtual Machine with and without file encryption in comparison to no virtual machine.

read and write operations. We measured file I/O throughput by repeatedly writing 2GB random data (we also measured 512MB and 4GB, and results are similar) in 8KB blocks into files. We then repeated the same experiment, but this time reading instead of writing. Figure 4 shows the throughput of file I/O achievable for our benchmark application, when using the EVM without file encryption, and with using the EVM with encryption. The results show that when using encryption the file I/O throughput is reduced to 1/4th of that without encryption.

Encryption is only needed if the customer wants the Application Security feature enabled. Even so, we found that the Entropia VM with encryption has very little performance overhead for the benchmarks examined in Table 1. For these applications, we see from 0% to 6% execution time overhead when using file encryption.

6. RELATED WORK

This paper described the Entropia Virtual Machine, which addresses the Desktop Grid Virtual Machine requirements of desktop security, unobtrusiveness, and application security. At the time we provided the Entropia DCGrid (2000-2002), the other desktop distributed systems competing with us required 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 [26, 23, 4]. 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 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.

Another class of virtual machine systems is exemplified by VMware [27, 28] and Connectix [7]. They allow multiple operating systems to run concurrently on the same hardware resource, providing each OS an isolated virtual machine. A goal for VMware and Connectix is to enable applications written for different platforms to seamlessly share the same physical resources. 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 a complex virtual environment with hardware emulation. When we investigated using VMware's virtual machine, some of the enterprise IT departments had issues with installing such an intrusive system and second operating system image on each desktop machine.

Another potential solution is to restrict your grid to only running virtual machine specific languages, such as Java [24, 19] and .NET/MSIL [22]. These solutions require you to compile your code to the corresponding intermediate format to provide all of the VM's security features, but this is not always possible, since often times applications use 3rd party dlls and libraries. In addition, these virtual machines do not provide complete unobtrusive requirements necessary for a Desktop Grid deployment. 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.

Since we provided our solution for Entropia, a lot of research has been performed investigating virtual machines for desktop grids. This includes Virtuoso [8], In-VIGO [12], Terra [14], Denali [29] and NGSCB (formerly Palladium) [10].

Dinda et. al. [8, 12] discussed why virtual machines are the right solution to address many key problems in grid computing, such as security, isolation, and resource control. They studied performance overhead of a prototype system based on VMware and demonstrated the feasibility of the VM approaches for grid computing from a performance perspective. In this work they did not address some of the key requirements for desktop grids, such as unobtrusiveness. Terra [14] and NGSCB [10] also focus on providing a trusted computing platform that can provide desktop security and application security. Denali [29] and Xen [3] explores a different aspect of the problem. They studied how to build lightweight virtual machine monitors to coordinate the execution of many virtual machines.

Another related project, is the Purdue University Network Computing Hubs (PUNCH) [17, 5], which addresses the resource control problem in grid computing systems. They create a logical user for each grid application and a virtual file system (based on NFS and a set of trusted proxies) to separate the resource (storage in this case) between grid applications and local user applications [17]. Moreover, they addressed the client security problem by providing a restricted shell, within which only a restricted set of system calls are allowed. This solution is a Unix/Linux solution for providing application and data security, whereas we had to solve different challenges to provide a desktop virtual machine for Windows.

The above systems focus on providing tamper-resistant isolation of an application or multiple VMs running on the same physical resource, which is typically a dedicated server without interactive users. In comparison, our solution focuses on isolating desktop grid applications from the hosting desktop computers which have interactive users to create an unobtrusive execution environment that will be left in a clean state after execution. In addition, our approach was designed for ease of deployment, since it is very light weight requiring only a single OS driver being installed and a thin binary modification interception layer. Some of the above systems require heavier weight installations, which will prohibit their use by some commercial IT departments we talked to.

KaffeOS [2] and Aroma VM [25] study the resource control inside virtual machines, i.e. resource isolation and scheduling among different active entities inside the same VM. They both focused on the Java VM and they both have the flavor of OS resource containers with rate or quota limiting. The EVM studies a different problem, where the resource contentions are not solely from within one virtual machine, i.e. it is not a closed system where the VM has all the control. In our problem, desktop user and client applications are also involved.

7. SUMMARY

We have described the requirements for Desktop Grid Virtual Machines including ease of application integration, desktop security, application security, unobtrusiveness, and keeping the machine in a clean state after running the application. To meet these requirements, we designed and implemented the Entropia Virtual Machine. The resulting Entropia DCGrid solution has been deployed widely in large enterprise IT environments and has been used for over 50 applications from a variety of application domains. The EVM implementation consists of a Desktop Controller to manage the running of applications, resource control, and unobtrusiveness of the client. The other major component is a light weight Sandbox Execution Layer to mediate and virtual system calls. This was implemented using a combination of a device driver and dynamic binary interception to provide desktop security, unobtrusiveness, and application security.

8. ACKNOWLEDGMENTS

We gratefully acknowledge the contributions of the talented engineers and architects at Entropia to the design and implementation of the Entropia DCGrid. 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. We also thank Mike Smith and the anonymous reviewers for providing useful feedback on this paper.

9. REFERENCES

[1] S.F. Altschul, T.L. Madden, A.A. Schffer, J. Zhang, Z. Zhang, W. Miller, and D.J. Lipman. Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucleic Acids Research*, 25:3389–3402, 1997.

[2] G. Back, W. Hsieh, and J. Lepreau. Processes in kaffeos: Isolation, resource management, and sharing in java. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation*, October 2000.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *19th ACM Symposium on Operating Systems Principles*, October 2003.

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

[5] A. Butt, S. Adabala, N. Kapadia, R. Figueiredo, and J. Fortes. Fine-grain access control for securing shared resources in computational grids. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, April 2002.

[6] A. Chien, B. Calder, S. Elbert, and K. Bhatia. Entropia: Architecture and performance of an enterprise desktop grid system. *Journal of Parallel Distributed Computing*, 63(5):597–610, 2003.

[7] connectix. Virtual pc for windows. http://www.connectix.com/.

[8] P. Dinda. Virtuoso: Distributed computing using virtual machines, 2003. http://www.cs.northwestern.edu/ plab/Virtuoso/.

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

[10] P. Englund, B. Lampson, J. Manferdelli, M. Peinado, and B. Willman. A trusted open platform. *IEEE Spectrum*, pages 55–62, 2003.

[11] 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.

[12] R. Figueiredo, P. Dinda, and J. Fortes. A case for grid computing on virtual machines. In *Proceedings of the 23rd International Conference on Distributed Computing*, May 2003.

[13] T. Garfinkel. Traps and pitfalls: Practical problems in system call interposition based security tools. In *Internet Society's 2003 Symposium on Network and Distributed System Security*, 2003.

[14] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A virtual machine-based platform for trusted computing. In *SOSP*, Bolton Landing, New York, 2003. ACM.

[15] T. Garfinkel, B. Pfaff, and M. Rosenblum. Ostia: A delegating architecture for secure system call interposition. In *Internet Society's 2003 Symposium on Network and Distributed System Security*, 2004.

[16] G. Hunt and D. Brubacher. Detours: Binary interception of win32 functions. In *USENIX Windows NT Symposium*, July 1999.

[17] N. H. Kapadia, R. J. Figueiredo, and J. A. B. Fortes. Enhancing the scalability and usability of computational grids via logical user accounts and virtual file systems. In *Proceedings of the 15th International Parallel and Distributed Processing Symposium*, 2001.

[18] S. King, G. Dunlap, and P. Chen. Operating system support for virtual machines. In *Proceedings of the Annual USENIX Technical Conference*, June 2003.

[19] T. Lindholm and F. Yellin. *The Java Virtual Machine Specification*. Addison-Wesley, 1999.

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

[21] M. McGann. FRED: Fast rigid exhaustive docking, 2001. OpenEye.

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

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

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

[25] N. Suri, J. M. Bradshaw, M. R. Breedy, K. M. Ford, P. T. Groth, G. A. Hill, and R. Saavedra. State capture and resource control for java: The design and implementation of the aroma virtual machine. In *Java Virtual Machine Research and Technology Symposium*, April 2001.

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

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

[28] C. Waldspurger. Memory resource management in vmware esx server. In *Operating Systems Design and Implementation*, Boston, 2002. USENIX.

[29] A. Whitaker, M. Shaw, and S. Gribble. Denali: Lightweight virtual machines for distributed and networked applications. In *Proceedings of the USENIX Technical Conference*, June 2002.