# Harvesting Idle Windows CPU Cycles for Grid Computing

**R. Andersen**
Department of Computer Science
University of Copenhagen
Copenhagen, Denmark

**B. Vinter**
Department of Computer Science
University of Copenhagen
Copenhagen, Denmark

**Abstract -** *In this paper we demonstrate how to efficiently exploit the massive amount of idle CPU cycles from workstations and desktop PCs for Grid computing. The cycle harvesting is achieved by using sandbox technology and a generic guest operating system, specifically designed for the Grid resource. This relieves the resource owner from the complexity of installing and managing not only the resource software but also a large and diverse set of runtime environments that grid jobs may require. In addition, the sandbox provides a secure virtual environment, thus eliminating security issues for users and resource owners. Using this setup, resource owners can install a virtual machine of their own choice to provide a sandbox, and once the screen saver software has been downloaded, the resource is transparently and securely used as a Grid resource when the screen saver is activated.*

**Keywords:** Grid computing, screen saver science, sandboxing.

## 1 Introduction

Cycle scavenging, or Screen Saver Science, is an increasingly popular computing paradigm used within many fields of science that seek to tap the enormous amount of unused processing power from the millions of computers connected to the Internet. The paradigm is best known from the many successful Public Resource Computing projects, such as SETI@Home, where the idle time cycles are used for a dedicated scientific application. However, only a few attempts have been made to combine Screen Saver Science with Grid Computing in order to use idle cycles for any kind of application.

One of the Grid[1] promises is to make it possible to share and effectively use distributed resources on an unprecedented scale. Specifically, this includes harnessing the unused capacity of idle desktop PCs. Much research has been done to Grid-enable idle resources, yet no widely accepted system to effectively scavenge idle cycles, in particular idle Windows cycles, has been found.

The approach we take in this project is to use a virtual machine to provide a sandbox that completely separates the Grid job from the resource host system, so that, on the one hand, a grid job cannot compromise the host system, and on the other hand, the grid job is protected from other users of the system.

This paper addresses the problems that need to be solved in order to scavenge idle desktops for scientific use. First of all, a method to gain access to the CPU cycles on the idle resource must be found. To this aspect, security is a major issue. Ideally, a resource owner should neither install any software nor execute any foreign applications that, intentional or not, could compromise his system. Secondly, the resource, possibly hidden by a Network Address Translation router and a firewall must be attached to the Grid. Thirdly, it must be ensured that a given resource has installed the correct software base that a given Grid job requires. Finally, we introduce extra features to improve the model, for instance, a method to predict the idle time period of a resource in advance. Using this method, a job with a time deadline is submitted to a resource that is predicted to be available in the specified time frame.

The paper is organized as follows: Section 2 presents a generic approach to securely utilize the idle CPU cycles of many types of architectures. The means taken to Grid-enable the proposed model are discussed in Section 3. Section 4 addresses the problems as regards required runtime environments on the resources. A few MiG components that optimize the model are presented in Section 5, an experiment to test the model is carried out in Section 6, before we conclude in Section 7.

### 1.1 Related work

BOINC[2] is a software platform that allows many different distributed computing projects to utilize idle volunteered computer resources. Many Public Resource Computing systems use BOINC and research groups can with little effort create new projects. A project involves a set of applications that will be run in a BOINC client on a user's resource. As such, BOINC could be used for this project by running the proposed virtual machine as the project application.

A few projects have been found to combine Screen Saver Science with Grid computing, for instance the Entropia Virtual Machine[3], which is a commercial product, and

[4] that presents an extensive introduction to the approach of using virtual machines for Grid computing.

## 2 Sandboxing and cycle scavenging

The basic idea is to let resource owners install a so-called sandbox to provide a secure execution environment in which the Grid job is completely isolated from the host machine and vice versa. Such a sandbox may be in the shape of a virtual machine, which is exactly the approach that we have taken in this work. The alternative approach, which will not be investigated in this work, works by intercepting all operating system calls and inspect their validity.

Two techniques can be used to provide a virtual machine: Emulation and Virtualization[5]. Emulation provides the functionality of the target processor completely in software, which makes it a very secure approach. Also, the ability to emulate one processor type on any other processor type makes it ideal for this scenario. However, the method of interpreting the entire guest operating system, rather than running it on the native hardware, results in a significant performance drawback. When emulating a PC architecture on a PC, a compatibility layer that enables the target code to be run directly on the host processor, can reduce the performance penalty.

On the other hand, virtualization partitions hardware in multiple contexts, thus enabling running multiple operating systems on the same hardware resources simultaneously.

Several virtualization approaches exist[6]:

- Bare-metal Architecture
- Para-virtualization
- Full Virtualization, also known as Transparent Virtualization, or Hosted Architecture

The Bare-metal Architecture approach runs the guest operating system in *Ring 0*, the most privileged protection level in x86 architectures. Running multiple operating systems in the same protection level could potentially result in one of the systems compromising the other. Clearly, this approach is not acceptable for resource owners.

The Para-virtualization approach, known from Xen[16] et al., needs to modify the host system by interposing a *hypervisor* between the operating system and the hardware. The *hypervisor* then takes on the *Ring 0* and the operating system must be explicitly ported to run in *Ring 1*. These modifications to the host operating system exclude this approach.

The Full Virtualization approach has performance drawbacks, but is the least intrusive and thus chosen, not only because minimum intrusion is a goal of the MiG project, but also because the least intrusive approach means the highest number of potential participants. As shown in Figure 1, the virtual machine allows a guest operating system to run as an application in the host operating system. The virtual machine emulates the underlying hardware, thus creating a secure sandbox that allows an application written for one operating system, e.g. Linux, to be executed in another, e.g. Windows.

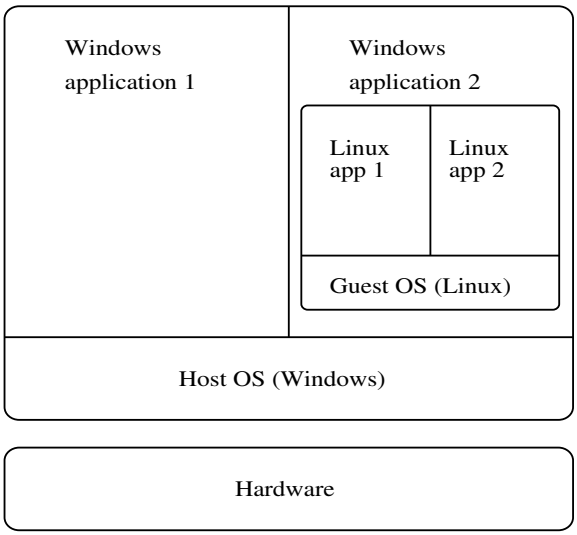


Figure 1: Full virtualization

The performance penalties are mitigated by kernel support that enables it to run most of the target application code directly on the host processor, thus achieving near native speed.

Regarding security, the virtual machine is a user space process that cannot do any harm to the host system as long as the permanent storage is protected properly. If the virtual machine is destroyed by a malicious application, the host system is not affected, and the virtual machine can start afresh.

The authors know of the following such solutions for the Microsoft Windows platform:

- VirtualPC[7]
- VMWare[8]
- Qemu + Qemu Accelerator Module[9]

Having downloaded and installed one of the virtual machines, a resource owner only needs to install a screen saver that starts the virtual machine upon activation and a tailor-made Linux image that is capable of running the Grid resource software automatically. In this manner, when the resource goes into screen saver mode, the virtual machine is activated and the Linux guest operating system is booted. The details of how to Grid-enable the hosted Linux system are explained next.

# 3 Enabling the sandbox for the Grid

The main problem with scavenging CPU-cycles from personal computers is that the vast majority are hidden behind a NAT router, i.e. they do not have global IP address and are therefore not reachable from the Internet.

Hence, to enable the sandbox for Grid Computing, care must be taken to circumvent the missing inbound Internet access. Naturally, this issue is highly dependent on the Grid middleware in question. In this work, the sandbox is enabled for the Minimum intrusion Grid, MiG, which is presented next, before the details of how to tailor the sandbox for MiG are explained.

## 3.1 Minimum intrusion Grid

MiG[10][11] is a stand-alone approach to Grid that does not depend on any existing systems, i.e. it is a completely new platform for Grid computing. The philosophy behind MiG is to provide a Grid infrastructure that imposes as few requirements on users and resources as possible.

The idea is to ensure that users only need a signed X.509 certificate, trusted by Grid, and a web browser that supports HTTP and HTTPS. A resource only needs to create a MiG user on the system and to support inbound ssh and outbound HTTPS. Initially, the resource must register to the MiG system using a certificate.

By keeping the Grid system disjoint from both users and resources, as shown in Figure 2, this model allows the Grid system to appear as a centralized black-box to both users and resources, and all upgrades and trouble shooting can be performed locally within the Grid without intervention from neither users nor resource administrators. Thus, all functionality is placed in a physical Grid system.

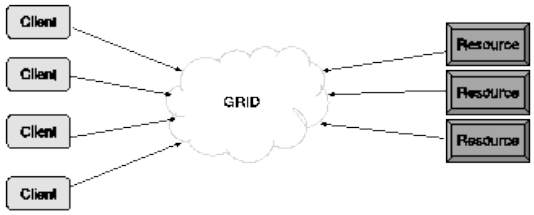


Figure 2: The abstract MiG model

The basic functionality in MiG starts with users submitting jobs to MiG and resources sending requests for jobs. A resource then receives an appropriate job from MiG, executes the job, and sends the result to MiG that can inform the user of the job completion. Thus, MiG provides full anonymity; users and resources interact only with MiG, never with each other.

## 3.2 The MiG Linux Image

As explained above, all that is required for a resource to join MiG, is to create a grid user account and support for incoming SSH and outgoing HTTPS.

So basically, the MiG Linux image can be built using any Linux distribution that runs an x86 system. Since the virtual machine provides a standardized virtualized set of hardware, compatibility amongst the wide range of different hardware setups on the resources will not be an issue.

The main concerns with respect to the distribution is the size and the start-up time. Both issues matter only for practical reasons, the size should be minimized to avoid an excessively large download, and, naturally, the start-up time should be minimized as much as possible.

As a base for the MiG Linux image we have chosen *ttylinux*[12], which is a minimalistic distribution that is easy to customize and, despite its very scarce space usage, provides an environment similar to larger distributions. The only shortcoming is the missing HTTPS support, so this had to installed manually.

In order to circumvent the missing inbound Internet access on resources that use NAT, it must be ensured that all communication is initiated by the resource. In MiG, this was easily integrated by small changes that only apply for sandboxes. In addition, it allows for directing jobs that users point out as Public Resource Computing jobs directly to a free sandbox.

The generic MiG Linux Image consists of a kernel and a Ram-disk that altogether take up less than 3 MB. An online generator modifies the generic image by giving it a unique resource name and a session id needed for requesting a job. Further, the resource owner can choose the size of a hard disk image file to provide as storage for the sandbox. Thus, certified resource owners can have a complete image built with a unique key allowing the sandbox to automatically request and execute Grid jobs.

In the standard MiG model, the identity of a resource requesting a job is verified by keeping the public SSH key of the resource in the MiG system and copying all job files to the resource over SSH. The sandbox model however, is modified to use a pull model on the resource where all files are transferred using HTTPS. Hence, a firewall in front of a resource only needs to be open for HTTP and HTTPS to allow the resource to run Grid jobs.

# 4 Runtime environments

Once the basic sandbox is in place it is possible to execute user applications within the virtual Linux machine. Many applications can be passed as executables from the Grid job and these need no further components to execute. Other, commonly used, applications may benefit from a preinstalled runtime environment, such as they are found on ordinary Grid resources.

Installing runtime-environments in the sandboxed environment could be done as on a conventional resource, which however would require the PC owners to personally maintain the sandboxed Linux distribution and this model is thus not desirable. Alternatively the sandbox image could be distributed with the initial Linux image, but this would greatly increase the size of the distribution image and in addition be a very static model.

The chosen solution allows individual research groups to maintain runtime environments for the sandboxed resources and at the same time allows the individual PC owners to control which runtime environments are downloaded and at which time. The runtime environments are kept in groups in virtual disk-partitions, in the form of a single file.

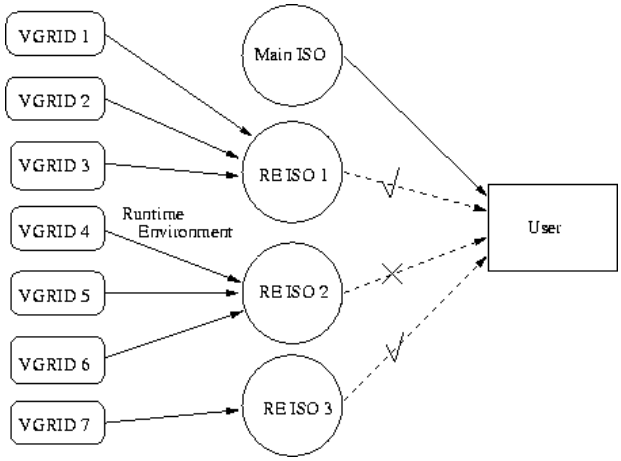


Figure 3: Runtime environments are maintained by the respective VGrids and grouped in virtual disks. Users are obliged to download the MiG Linux image, but can choose among the groups of runtime environments.

As shown in Figure 3, the PC owner can download a group of runtime environments from the VGrids, MiG's notion of a virtual organization, that maintain the runtime environments and when a job that uses a runtime environment is received by the sandboxed resource, the virtual Linux machine will mount the file system that contains the runtime environment. This way each runtime environment is kept isolated from the rest of the system, and can easily be built and maintained by the research groups that need them to be available for their executions.

# 5 MiG features

To improve and simplify the sandbox model further, MiG contains two components that apply: File access and scheduling.

## 5.1 Remote File Access

One difficulty that users report when using Grid is file access, since files that are used by Grid jobs must be explicitly uploaded to a Grid storage element and result files must also be downloaded explicitly. The MiG model introduces home catalogs for all Grid users, and all file references are relative to this home-catalog. This eliminates all naming problems, since MiG provides one simple access entry to a user's home-catalog. Furthermore, using the MiG Remote File Access library [13], a resource can, transparently and without recompiling or relinking applications, access application input and output files remotely, thus only downloading needed data and only uploading modified data.

## 5.2 Scheduling

Contrary to the majority of the existing Grid middlewares, where several levels of scheduling results in jobs being submitted to a resource where another level of scheduling takes place, MiG makes the scheduling for fairness much simpler as the local scheduling comes before the Grid scheduling. Thus, a single job is never left waiting a long time for CPU cycles once it has been submitted to a resource.

Existing Screen Saver Science systems all target problems that have many, usually millions, of independent tasks that often run for tens or hundreds of hours. Once a task has been assigned to a computer it will be processed while the computer is in screen saver mode. Processing is suspended if the screensaver is suspended and similarly resumed again along with the screensaver. An artefact of this model is that one never knows when the result of a given task is ready, and it is very hard to determine if the task has been lost or if it is simply only allowed to proceed very slowly.

For applications such as computational chemistry this model is very poorly suited. The number of tasks is usually in the tens or hundreds and it is often the case that analysis of the results can only start once the result of every task is in.

This is easily addressed by putting an upper time limit on each job, and if the time limit is exceeded, the job is resubmitted to another resource. However, in order to schedule a job with a deadline to a screen saver resource, we need to know how long the resource is availabe, i.e. how much time it takes before the screen saver is deactivated. To predict the available time slot of a screen saver resource, we use exponential average on an hourly

basis, which has proved to converge against the actual resource idle time quite fast.

# 6 Experiment

To verify our model, we have connected 8 sandbox ressources to MiG and submitted jobs that we pointed out as sandbox jobs. The 8 resources are all identical Windows PCs that host the virtual Linux machine that runs the Grid client code and includes the required runtime environment. The jobs are all NAMD[17] jobs, which is a software package for simulation of biomolecular systems.

Since we have 8 machines, we have chosen to submit 25 jobs. This means that there is 3 jobs for each machine and to avoid balanced execution, one last machine needs to execute one additional job before the experiment is completed. Further, using the 'minimization' option in NAMD effectively ensures different running times, thus ensuring unbalanced execution.

Running the jobs sequentially on one of the PCs results in a total running time of 2 hours, 25 minutes, and 22 seconds. When submitted to the Grid, the 25 jobs completed in 31 minutes and 45 seconds. Thus, despite the overhead, the model is deemed successful.

# 7 Conclusion

This work has shown how to eliminate the factors that have previously impeded the fusion of Public Resource Computing and Grid Computing to effectively utilize idle CPU cycles from desktop machines for any kind of Grid job.

The prohibiting factors include NAT-hidden resources, means to utilize Windows desktops, the workload required by a non-expert resource owner to install and manage all resource software, and the security issues involved with installing a large software base on the resource.

Using sandboxing technology and a generic Linux image, the Minimum intrusion Grid has successfully eliminated all of these limitations. Users need only download a bundle consisting of a screen saver, a virtual machine, and a special MiG Linux image in order to share their idle resources, whether they run Linux or Windows. The MiG system has proved flexible enough to easily deal with computers behind network address translators, and mobile processes and automatic resubmission of jobs solve the problem with resources that are cut off the network or leave the screen saver mode. Finally, the sandboxed environment ensures that the host system cannot be compromised.

Using this approach, a desktop computer volunteers as a Grid resource upon screen saver activation, and as soon as the screen saver is deactivated, the executing job either stops or migrates. Thus, the resource owner is completely unaffected by the Grid job.

# 8 References


[1]I. Foster, “The Grid: A New Infrastructure for 21st Century Science”, Physics Today, 55(2):42-47,2002

[2]http://boinc.berkely.edu

[3]B. Calder, A.A. Chien, J. Wang, D. Yang,"The Entropia Virtual Machine for Desktop Grids"

[4]R.J. Figueiredo, P.A. Dinda, J.A.B. Fortes, "A Case for Grid Computing on Virtual Machines"

[5]N. Kiyanclar, "A Survey of Virtualization Techniques Focusing on Secure On-Demand Cluster Computing"

[6]http://www.vmware.com/pdf/virtualization.pdf

[7]http://www.microsoft.com/windows/virtualpc/default.mspx

[8]http://www.vmware.com/products/player/

[9]http://fabrice.bellard.free.fr/qemu/

[10]B. Vinter ``The architecture of the Minimum intrusion Grid: MiG" In Communicating Process Architectures: pp. 189-201 Broenink J, Roebbers H, Sunter J, Welch P, Wood D (eds.) IOS Press, 2005

[11]Henrik Hoey Karlsen, Brian Vinter, "Minimum intrusion Grid - The Simple Model," wetice, pp. 305-310, 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (WETICE'05), 2005

[12]http://www.minimalinux.org/ttylinux/

[13]Rasmus Andersen, Brian Vinter, "Transparent Remote File Access in the Minimum Intrusion Grid," wetice, pp. 311-318, 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (WETICE'05), 2005

[14]W.L. George, J. Scott, "Screen Saver Science: Realizing Distributed Parallel Computing with Jini and JavaSpaces"

[15]M.J Litzkow, M. Livny, M.W. Mutka, "Condor-a hunter of idle workstations", Proc. of the 8th International Conference of Distributed Computing Systems, pp. 104-111, June, 1988

[16]P. Barham et al., "Xen and the Art of Virtualization", In Proc. SOSP 2003, Bolton Landing, New York, U.S.A. Oct 19-22, 2003

[17]James C. Phillips et al., “Scalable molecular dynamics with NAMD”, Journal of Computational Chemistry, 26:1781-1802, 2005