BULETINUL Universităţii Petrol – Gaze din Ploieşti	Vol. LX No. 1/2008	39 - 48	Seria Matematică - Informatică - Fizică

Overview of Desktop Grid Systems

Zoran Constantinescu[*], Monica Vlădoiu[**], Cătălina Negoiţă[**]

[*]ZealSoft SRL, str. Tg. Neamţ, nr. 11, Bucureşti
e-mail: zoran@unde.ro

[**] Universitatea Petrol-Gaze din Ploieşti, Bd. Bucureşti 39, Ploieşti, Catedra de Informatică
e-mail: mvladoiu@upg-ploiesti.ro

Abstract

We present here our overview of the most well-known and used desktop grid systems that can be used to scavenge cycles from idle desktop computers either from a local network or from Internet: SETI@home-BOINC, distributed.net, PVM, Entropia, and Condor. This overview could be used also as a guide to choose the most appropriate desktop grid platform to solve a particular problem.

Key words: *grid computing, distributed computing, desktop grid systems, overview of desktop grids*

Desktop Grids

Grid computing is a model of Distributed Computing (DC) that uses geographically and administratively disparate resources that are found on the network. These resources may include processing power, storage capacity, specific data, and other hardware such as input and output devices. In grid computing, individual users can access computers and data transparently, without having to consider location, operating system, account administration, and other details. Moreover, the details are abstracted, and the resources are virtualized. Grid computing seeks to achieve the secured, controlled and flexible sharing of resources (for example, multiple computers, software and data) among various dynamically created virtual organizations [2, 4], which are generally setup for collaborative problem solving and access to grid resources are limited to those who are part of the project. The creation of an application that can benefit from Grid computing (faster execution speed, linking of geographically separated resources, interoperation of software, etc.) typically requires the installation of complex supporting software and an in-depth knowledge of how this complex supporting software works.

Grid computing systems can be classified into two broad types. The first type are heavy-weight, feature-rich systems that tend to concern themselves primarily with providing access to large-scale, intra- and inter-institutional resources such as clusters or multiprocessors. The second general class of Grid computing systems is the Desktop Grids, in which cycles are scavenged from idle desktop computers. The typical and most appropriate application for desktop grid comprises independent tasks (no communication exists amongst tasks) with a high computation to communication ratio [1].

In a desktop grid system, the execution of an application is orchestrated by a central scheduler node, which distributes the tasks amongst the worker nodes and awaits workers' results. It is important to note that an application only finishes when all tasks have been completed. The main difference in the usage of institutional desktop grids relatively to public ones lies in the dimension of the application that can be tackled. In fact, while public projects usually embrace large applications made up of a huge number of tasks, institutional desktop grids, which are much more limited in resources, are more suited for modestly-sized applications. So, whereas in public volunteer projects importance is on the number of tasks carried out per time unit (throughput), users of institutional desktop grids are normally more interested in a fast execution of their applications, seeking fast turnaround time.

The attractiveness of exploiting desktop grid systems is further reinforced by the fact that costs are highly distributed: every volunteer supports her resources (hardware, power costs and internet connections) while the benefited entity provides management infrastructures, namely network bandwidth, servers and management services, receiving in exchange a massive and otherwise unaffordable computing power. The usefulness of desktop grid computing is not limited to major high throughput public computing projects. Many institutions, ranging from academics to enterprises, hold vast number of desktop machines and could benefit from exploiting the idle cycles of their local machines. In fact, several studies confirm that CPU idleness in desktop machines averages 95% [3, 5].

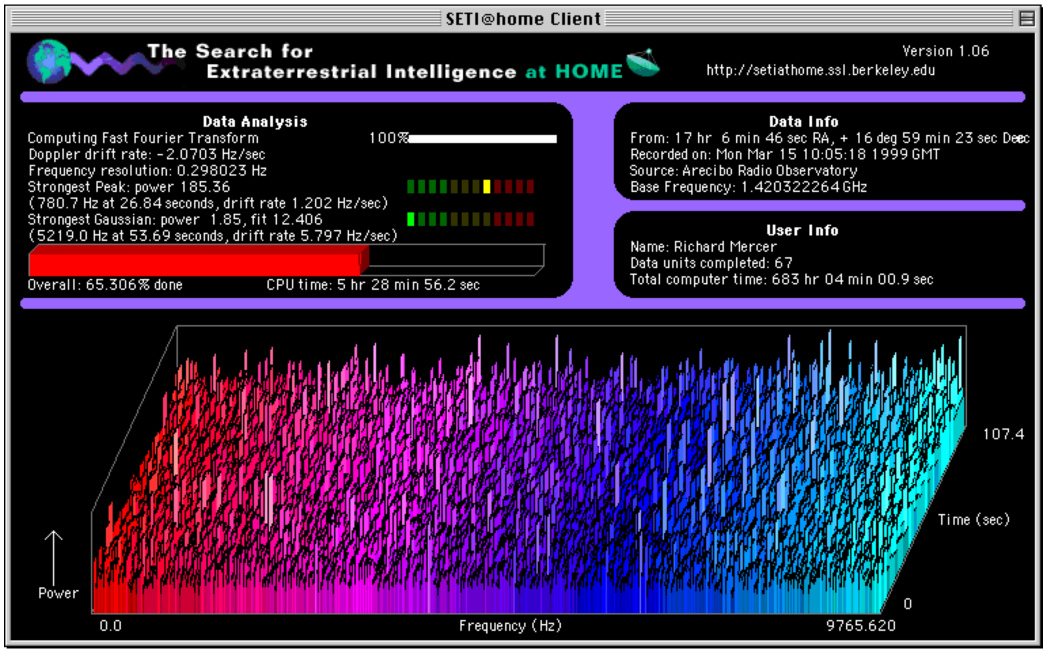
Desktop Grid (DG) has recently received the rapidly growing interest and attraction because of the success of the most popular examples such as SETI@Home and distributed.net. SETI@home is one of the most successful projects that use such a model. One of the reasons for this success is its simplicity in enabling contributors to donate computational resources—when the computer screensaver is activated the application starts by making a request to a remote server to download tasks to be processed. Another reason is its support for Windows operating system, since the majority of the desktop machines around the world run Windows. Based on the same concept, there are other @home projects: FightAIDS@home, Folding@home, evolution@home, etc. All of these projects are primarily targeted for applications that can be expressed as parameter-sweep applications. They have no or lack of support for creating applications consisting of tasks that need to communicate and coordinate their activities by exchanging messages among themselves [9].

SETI@home - BOINC

SETI, or the Search for Extraterrestrial Intelligence, is a scientific effort seeking to determine if there is intelligent life outside Earth. One popular method SETI researchers use is radio SETI, which involves listening for artificial radio signals coming from other stars. Previous radio SETI projects have used special-purpose supercomputers, located at the telescope, to do the bulk of the data analysis. In 1995, a new idea was proposed to do radio SETI using a virtual supercomputer composed of large numbers of Internet-connected computers [12].

SETI@home, developed at the University of California in Berkley, is a radio SETI project that lets anyone with a computer and an Internet connection participate. The method they use to do this is with a screen saver that can go get a chunk of data from a central server over the Internet, analyze that data, and then report the results back. When the computer is needed back, the screen saver instantly gets out of the way and only continues its analysis when the computer is not used anymore. The program that runs on each client computer looks and behaves like a captivating screen saver. It runs only when the machine is idle, and the user can choose from several different colorful and dynamic "visualizations" of the SETI process. Some of these visualizations will look technical, some will look abstract, and some will look decidedly artistic, as it can be seen in the screenshot from Fig. 1.

The data analysis task can be easily broken up into little pieces that can all be worked on separately and in parallel. None of the pieces depends on the other pieces, which makes large deployment of clients and computations very easy over the Internet.



SETI@home Client

The Search for Extraterrestrial Intelligence at HOME

Version 1.06
http://setiathome.ssl.berkeley.edu

Data Analysis
Computing Fast Fourier Transform 100%
Doppler drift rate: -2.0703 Hz/sec
Frequency resolution: 0.298023 Hz
Strongest Peak: power 185.36
(780.7 Hz at 26.84 seconds, drift rate 1.202 Hz/sec)
Strongest Gaussian: power 1.85, fit 12.406
(5219.0 Hz at 53.69 seconds, drift rate 5.797 Hz/sec)
Overall: 65.306% done CPU time: 5 hr 28 min 56.2 sec

Data Info
From: 17 hr 6 min 46 sec RA, + 16 deg 59 min 23 sec Decc
Recorded on: Mon Mar 15 10:05:18 1999 GMT
Source: Arecibo Radio Observatory
Base Frequency: 1.420322264 GHz

User Info
Name: Richard Mercer
Data units completed: 67
Total computer time: 683 hr 04 min 00.9 sec

Fig. 1. *SETI@home screenshot*

SETI@home needs network connection only when transferring data. This occurs only when the screen saver has finished analyzing the work-unit and wants to send back the results. Each work unit is sent multiple times to different users in order to make sure that the data is processed correctly. The system architecture is depicted in Fig. 2.

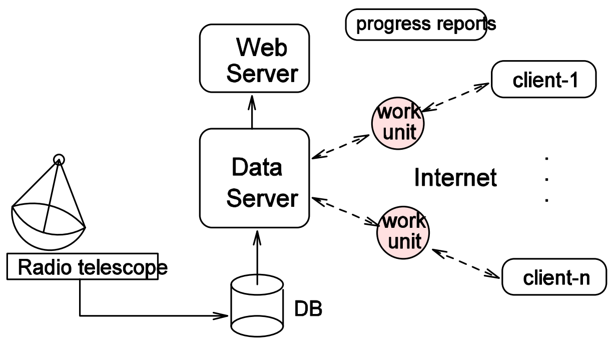


Fig. 2. *SETI@home architecture*

While the "screen saver" is running, the client would be processing the quarter-megabyte data block (work-unit), which would contain 50 seconds within a 20-kilohertz range. The algorithm examines this data for strong signals or "chirps" while taking Doppler shifting into account. False alarms would be prevented by tests for terrestrial interference. Once a block was processed, it would be returned to a centralized SETI@home computer where the results would be stored and organized. This process, when replicated tens or hundreds of thousands of times, has the capacity to analyze the data much more closely than before, perhaps noticing subtle patterns that real-time signal processing missed. The overall results of the search would appear on the SETI@home web site, making the findings immediately available to the public and to the participants. SETI@home is the largest public distributed computing project in terms of computing power: on September 26, 2001 it reached the ZettaFLOP (1021 floating point operations) mark, a new world record, performing calculations at an average of 71 TeraFLOPs/second. For comparison, the fastest individual computer at that time in the world was IBM's ASCI White, which runs at 12.3 TeraFLOPs/second. On June 1, 2002, the project completed over 1 million CPU years of computation.

SETI@home has not been without problems. For all the media attention and public interest, funding has not been forthcoming. Developing new software to run the distributed system and to perform the analysis on the client side is a difficult and expensive process. The SETI@home project has been delayed repeatedly due to lack of corporate sponsorship. "People time", rather than computer power, has proven to be hard to come by, and in the end it seems that expense - the very thing that SETI@home and distributed computing are meant to escape - may be a force as inexorable as gravity. The SETI@home project is for a very specific problem, as described above. There was no general framework for the system, which can be used by other types of applications, and it became SETI@home Classic. Then new funding came for the BOINC project and SETI@home was rewritten for the new framework and it became SETI@home II in 2005. BOINC is open-source software for volunteer computing and desktop grid computing. It includes the following features: project autonomy, volunteer flexibility: flexible application framework, security, server performance and scalability, source code availability, support for large data, multiple participant platforms, open, extensible software architecture, and volunteer community features. BOINC is designed to support applications that have large computation requirements, storage requirements, or both. The main requirement of the application is that it be divisible into a large number (thousands or millions) of jobs that can be done independently. If the project is going to use volunteered resources, there are additional requirements as public appeal and low data/compute ratio [8].

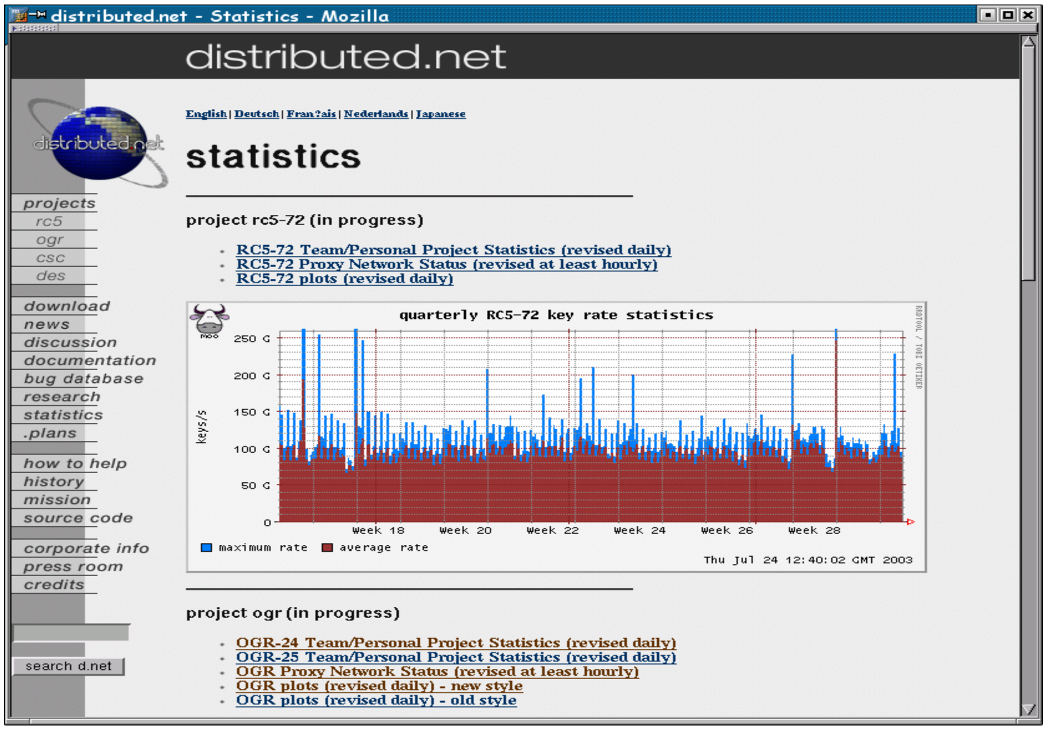
distributed.net

A very similar project is the distributed.net project [10]. It takes up challenges and run projects which require a lot of computing power. Utilizing the combined idle processing cycles of the members' computers solves these. The collective-computing projects that have attracted the most participants have been attempts to decipher encrypted messages. RSA Security (RSA, 2005) a commercial company has posted a number of cryptographic puzzles, with cash prizes for those who solve them. The company's aim is to test the security of their own products and to demonstrate the vulnerability of encryption schemes they consider inadequate. The focus of the distributed.net project is on very few specialized computing challenges. Furthermore, the project only releases the clients' binary code and not the server code, making impossible the adaptation of this to other types of projects.

Typical RSA challenges could either involve factoring, or call for a more direct attack on an encrypted text. In one challenge the message was encoded with DES, the Data Encryption Standard, a cipher developed in the 1970s under U.S. government sponsorship. The key that unlocks a DES message is a binary number of 56 bits (or larger: 64, 72 bits). In general the only way to crack the code is to try all possible keys, of which there are 256, or about 7 * 1016. Another RSA challenge also employed a 56-bit key, but with an encryption algorithm called RC5. Compared with earlier DC projects, the RC5 efforts were not only technically sophisticated but also reached a new level of promotional and motivational slickness.

For example, they kept statistics on the contributions of individuals and teams, adding an element of competition between teams, as it can be seen in Fig. 3. The RSA Challenge numbers are the kind, which are believed to be the hardest to factor; these numbers should be particularly challenging. These are the kind of numbers used in devising secure RSA cryptosystems. The challenges are an effort to learn about the actual difficulty of factoring large numbers of the type used in RSA keys.

Another type of project, which involves a lot of computing power, is the optimal Golomb Ruler (OGL). Essentially, a Golomb Ruler is a mathematical term given to a set of whole numbers where no two pairs of numbers have the same difference. An Optimal Golomb Ruler is just like an everyday ruler, except that the marks are placed so that no two pairs of marks measure the same distance. OGRs have many uses in the real world, including sensor placements for X-ray crystallography and radio astronomy. Golomb rulers can also play a significant role in combinatorics, coding theory and communications. The search for OGRs becomes exponentially more difficult as the number of marks increases ("NP complete" problem).



statistics

project rc5-72 (in progress)

- RC5-72 Team/Personal Project Statistics (revised daily)
- RC5-72 Proxy Network Status (revised at least hourly)
- RC5-72 plots (revised daily)

project ogr (in progress)

- OGR-24 Team/Personal Project Statistics (revised daily)
- OGR-25 Team/Personal Project Statistics (revised daily)
- OGR Proxy Network Status (revised at least hourly)
- OGR plots (revised daily) - new style
- OGR plots (revised daily) - old style

Fig. 3. *distributed.net statistics screen*

Considerations on Parallelism for SETI@home - distributed.net

None of these two systems provide support for parallel application, when communication between programs running on different computers is necessary during the computation. This makes difficult to use such systems for our purpose, where more than one desktop computer are needed to solve a certain problem. Tasks with independent parallelism are suited for this type of computing. In SETI@home, work unit computations are independent, so participant computers never have to wait for or communicate with one another [1].

If a computer fails while processing a work unit, the work unit is eventually sent to another computer. Public-resource computing, with its frequent computer outages and network disconnections, seems ill-suited to parallel applications that require frequent synchronization and communication between nodes. However, scheduling mechanisms that find and exploit groups of LAN-connected machines may eliminate these difficulties [1].

PVM

PVM (Parallel Virtual Machine) is a portable message-passing programming system, designed to link separate host machines to form a virtual machine, which is a single, manageable computing resource. The virtual machine can be composed of hosts of varying types. The general goals of this project are to investigate issues in, and develop solutions for, heterogeneous concurrent computing. PVM is an integrated set of software tools and libraries that emulates a general-purpose, flexible, heterogeneous concurrent computing framework on interconnected computers of varied architecture. The overall objective of the PVM system is to enable such a collection of computers to be used cooperatively for concurrent or parallel computation. Applications can be composed of any number of separate processes and are provided access to PVM through the use of calls to PVM library routines for functions such as process initiation, message transmission and reception, and synchronization via barriers or rendezvous. PVM is effective for heterogeneous applications that exploit specific strengths of individual machines on a network [7].

The PVM system is composed of two parts. The first part is a daemon (called pvmd) that resides on all the computers making up the virtual machine. This is designed in such a way that any user with a valid login can install this daemon on a machine. When a user wishes to run a PVM application, he first creates a virtual machine by starting up PVM. Multiple users can configure overlapping virtual machines, and each user can execute several PVM applications simultaneously. The second part of the system is a library of PVM interface routines. It contains a functionally complete repertoire of primitives that are needed for cooperation between tasks of an application. This library contains user-callable routines for message passing, spawning processes, coordinating tasks, and modifying the virtual machine.

The PVM computing model described in Fig. 4 is based on the notion that an application consists of several tasks. Each task is responsible for a part of the application's computational workload. Sometimes an application is parallelized along its functions; that is, each task performs a different function, for example, input, problem setup, solution, output, and display. This process is often called functional (task) parallelism. A more common method of parallelizing an application is called data parallelism. In this method all the tasks are the same, but each one only knows and solves only a small part of the data. This is also referred to as the SPMD (single-program multiple-data) model of computing. PVM supports either or a mixture of these methods. Depending on their functions, tasks may be executed in parallel and they may need to synchronize or exchange data, although this is not always the case.

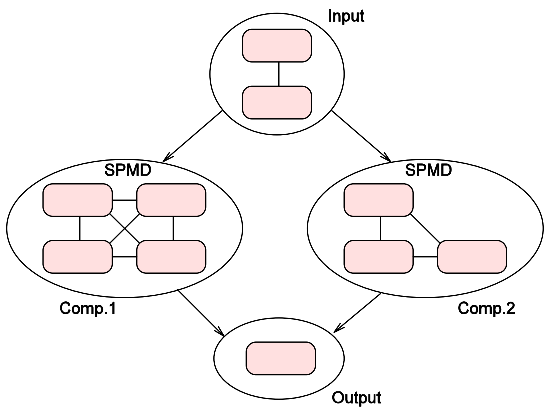


Fig. 4. *PVM Computing Model*

The general paradigm for application programming with PVM is as follows. A user writes one or more sequential programs in C/C++, or Fortran 77 that contain embedded calls to the PVM library. Each program corresponds to a task making up the application. These programs are compiled for each architecture in the host pool, and the resulting object files are placed at a location accessible from machines in the host pool. To execute an application, a user typically starts one copy of one task (usually the "master" or "initiating" task) by hand from a machine within the host pool.

This process subsequently starts other PVM tasks, eventually resulting in a collection of active tasks that then compute locally and exchange messages with each other to solve the problem. Note that while the above is a typical scenario, as many tasks as appropriate may be started manually. Tasks interact through explicit message passing, identifying each other with a system-assigned, opaque task identifier. PVM support both task- and data-parallelism. An advantage of the PVM system is that it is quite popular today and has become a de-facto standard for message passing. There are many algorithms implemented using PVM, and there is a large body of experience in using it. It is well known and accepted in the academic environment, due to its easiness of use and the availability of source code from the public domain. However, it is not very widespread in industry.

Alas, PVM provides only the parallel programming environment and does not offer resource management. This means that the system could not prevent access of different users to the same computing resource. Two or more users could share the same CPU without even knowing that. Such sharing could result in an inefficient use of resources, especially when running a data-parallel application with uniform computational requirements per task. Further job/resource scheduling systems are required to provide exclusive access to CPUs in a PVM environment.

Another disadvantage of the PVM system is that the user needs to have 'login' access to each of the computers involved in a computation. From the user's point of view this is done in a transparent way, by automatically using remote login (usually Rush or ssh) to start the application on each computer. There are certain problems with this, which limits a large-scale deployment of the system in many situations. One is that in a completely heterogeneous environment, consisting of operating systems with different types of user authentication (e.g. Unix, Windows and Mac), allowing users' login access to each computer on the network can be extremely difficult to set up and later maintain it. This could also easily be the cause of a potential security problem. For this reason, in many real-life situations, users are not allow to remotely login to the computers in the network, or if so, to only a very few servers. It has also been found to be difficult to install PVM for recent versions of Windows, making it very hard to deploy and use in a today's typical large-scale corporate network, where desktop machines running different operating systems are usually available. So PVM need 'more' heterogeneity than just Unix systems.

Entropia

DCGrid, developed by the company called Entropia [11] was a PC grid computing platform that provided high performance computing capabilities by aggregating the unused processing cycles of networks of existing Windows-based PCs. The system is no longer used due to the fact that the system was thought in the first place as being commercial. We have chosen to still present it since it was a major desktop grid system, which has had significant contributions to the field.

Existing proprietary and third party applications could be quickly and easily deployed on the DCGrid platform using DCGrid's rapid integration features, which allow enterprises to achieve business objectives faster, with higher throughput, increased precision and more meaningful results in less time than previously possible. DCGrid solutions enabled new and more difficult problems to be solved. Unused PC resources are harvested based on user and organization policies, with settings centrally monitored and managed with a web-based grid management interface. Work is scheduled to PCs based on application resource requirements, and is monitored and rescheduled as necessary if there are system disruptions or resource unavailability. Any native Win32 application could be deployed and executed on the DCGrid platform, and applications are enabled for the platform at the binary code level.

DCGrid contained an isolation technology, which provides full and unobtrusive protection for the grid as well as the underlying resources. DCGrid protected the desktop configuration, programs, and data from corruption by grid application errors as well as the privacy of desktop users from snooping. The grid application could not accidentally or intentionally access or modify the PC configuration or data files. Unlike other error-prone approaches, DCGrid presented a cleanly isolated, corruption-free environment. DCGrid shielded applications, proprietary data, and resources distributed to the desktop PCs by using encryption and tamper detection. Proprietary data and research sent out to hundreds of PCs in an enterprise could be protected from desktop user inspection or malicious corruption. DCGrid automatically monitored and limited grid work so it does not intrude on the PC user. DCGrid remained invisible at all times, never demanding inputs or responses from the desktop user, and never impacting the user's performance.

The approach is to automatically wrap an application in a virtual machine technology (Fig. 5). When an application program is registered or submitted to the Entropia system, it is automatically wrapped inside the virtual machine. This isolation is called sandboxing. The application is contained within a sandbox and is not allowed to modify resources outside the sandbox. The application is fully unaware of being running within a sandbox, since its interaction with the OS is automatically controlled by the virtual machine. The virtual machine intercepts system calls the application makes. This ensures that the virtual machine has complete control over the applications' interaction with the operating system and access to the desktop resources.

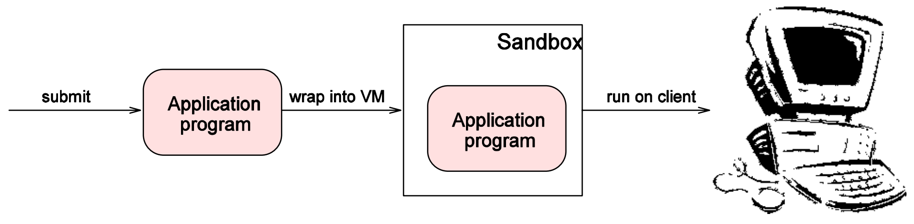


Fig. 5 *Entropia Sandbox Model*

The Entropia system architecture consisted of three layers: physical management, scheduling, and job management. The physical node management layer, provided basic communication and naming, security, resource management, and application control. The second layer was resource scheduling, providing resource matching, scheduling, and fault tolerance. Users could directly interact with the resource-scheduling layer through the available APIs or alternatively through the third layer management, which provides management facilities for handling large numbers of computations and files. Entropia provided a job management system, but existing job management systems can also be used.

The physical node management layer of the Entropia system managed these and other low-level reliability issues. The physical node management layer provided naming, communication, resource management, application control, and security. The resource management services captured a wealth of node information (e.g., physical memory, CPU, disk size and free space, software version, data cached) and collected it in the system manager. This layer also provided basic facilities for process management including file staging, application initiation and termination, and error reporting. In addition, the physical node management layer ensures node recovery, terminating runaway and poorly behaving applications.

The security services employed a range of encryption and binary sandboxing technologies to protect both distributed computing applications and the underlying physical node. Application communications and data were protected with high-quality cryptographic techniques. A binary sandbox controlled the operations and resources visible to distributed applications on the physical nodes in order to protect the software and hardware of the underlying machine. The binary sandbox also regulated the usage of resources by the distributed computing application. This ensured that the application did not interfere with the primary users of the system without requiring a rewrite of the application for good behavior [4].

The resource-scheduling layer of Entropia accepted units of computation from the user or job management system, matched them to appropriate client resources, and scheduled them for execution. The resource-scheduling layer adapted to changes in resource status and availability and to high failure rates. To meet these challenging requirements, the Entropia system supported multiple instances of heterogeneous schedulers. This layer also provided simple abstractions for IT administrators, abstractions that automate the majority of admins' tasks with reasonable defaults but allow detailed control as desired.

Entropia's three-layer architecture provided a wealth of benefits in system capability, ease of use by users and IT administrators, and internal implementation. The physical node layer managed many of the complexities of the communication, security, and management, allowing the layers above to operate with simpler abstractions. The resource-scheduling layer dealt with unique challenges of the breadth and diversity of resources but need not deal with a wide range of lower-level issues. Above the resource-scheduling layer, the job management layer dealt with mostly conventional job management issues. Finally, the higher-level abstractions presented by each layer did simplify application development. One disadvantage of the Entropia system was that it did not support heterogeneous systems. The only platform was Windows that limited the usability of this system in a research environment.

Condor

Condor, developed at the department of Computer Science, University of Wisconsin, Madison, is a High Throughput Computing (HTC) environment that can manage very large collections of distributive owned workstations [6]. This is a computing environment that delivers large amounts of computational power over a long period of time, usually weeks or months. In contrast, High Performance Computing (HPC) environments deliver a tremendous amount of compute power over a short period of time. In a high throughput environment, researchers are more interested in how many jobs they can complete over a long period of time instead of how fast an individual job can complete. HTC is more concerned to efficiently harness the use of all available resources.

The Condor environment is based on a layered architecture that enables it to provide a powerful and flexible suite of resource management services to sequential and parallel applications. Condor is a specialized workload management system for compute-intensive jobs. Like other full-featured batch systems, Condor provides a job queuing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Users submit their serial or parallel jobs to Condor, Condor places them into a queue, chooses when and where to run the jobs based upon a policy, carefully monitors their progress, and ultimately informs the user upon completion.

Condor provides a powerful resource management by match-making resource owners with resource consumers. This is the cornerstone of a successful HTC environment. Other compute cluster resource management systems attach properties to the job queues themselves, resulting in user confusion over which queue to use as well as administrative hassle in constantly adding and editing queue properties to satisfy user demands. Condor implements ClassAds, which simplifies the user's submission of jobs. ClassAds work in a fashion similar to the newspaper classified advertising want-ads. All machines in the Condor pool advertise their resource properties, both static and dynamic, such as available RAM memory, CPU type, CPU speed, virtual memory size, physical location, and current load average, in a resource offer ad. A user specifies a resource request ad when submitting a job. The request defines both the required and a desired set of properties of the resource to run the job. Condor acts as a broker by matching and ranking resource offer ads with resource request ads, making certain that all requirements in both ads are satisfied. During this match-making process, Condor also considers several layers of priority values: the priority the user assigned to the resource request ad, the priority of the user which submitted the ad, and desire of machines in the pool to accept certain types of ads over others.

References

1. Constantinescu Z. - *A Desktop Grid Computing Approach for Scientific Comp. and Visualization*, PhD Thesis, Norwegian Univ. of Science and Technology, Trondheim, Norway, 2008
2. Cummings, M.P. - *Grid Computing*, http://serine.umiacs.umd.edu/research/ grid.php
3. Domingues, P., Marques, P., Silva, L. - Resource usage of Windows computer laboratories, in MARQUES, P., Ed., *Int. Conf. on Parallel Processing Workshops* (ICPP 2005), Leiria, Portugal, 2005
4. Foster, I., Kesselman, C. - *The grid: blueprint for a new computing infrastructure*, Boston, Morgan Kaufmann Publishers, 2004
5. Heap, D.G. - *Taurus - A Taxonomy of Actual Utilization of Real UNIX and Windows Servers*, IBM White Paper, 2003
6. Litzkow, M.L., Mutka, M.W. - Condor - A Hunter of Idle Workstations, *Procedeengs of the 8th International Conference of Distributed Computing Systems* (ICDCS1988), 1988
7. Sunderam, V.S. - PVM: a framework for parallel distributed computing, *Concurrency: Practice and Experience*, 2, pp. 315-339, 1990
8. * * * - *BOINC - open source software for volunteer computing and grid computing*, http://boinc.berkeley.edu/
9. * * * - *distributedcomputing.info*, http://distributedcomputing.info
10. * * * - *distributed.net*, http://distributed.net/
11. * * * - *Entropia*, www.entropia.com
12. * * * - *SETI@home*, http://setiathome.ssl.berkeley.edu/

Trecerea în revistă a sistemelor desktop grid

Rezumat

Aici trecem în revistă cele mai cunoscute şi utilizate sisteme desktop grid, care pot fi folosite pentru a fructifica ciclurile idle ale sistemelor de calcul desktop, fie dintr-o rețea locală, fie din Internet: SETI@home-BOINC, distributed.net, PVM, Entropia, and Condor. Acestă prezentare poate folosi şi ca un ghid pentru a alege cea mai potrivită platformă desktop grid pentru a rezolva o problemă particulară.