ACADEMIC PRESS

Available at www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

J. Parallel Distrib. Comput. 63 (2003) 597–610

Journal of Parallel and Distributed Computing

http://www.elsevier.com/locate/jpdc

Entropia: architecture and performance of an enterprise desktop grid system

Andrew Chien,*[,1] Brad Calder,[1] Stephen Elbert, and Karan Bhatia

Entropia, Inc., 10145 Pacific Heights, Suite 800, San Diego, CA 92121, USA

Received 8 April 2001; revised 9 November 2001

Abstract

The exploitation of idle cycles on pervasive desktop PC systems offers the opportunity to increase the available computing power by orders of magnitude (10×–1000×). However, for desktop PC distributed computing to be widely accepted within the enterprise, the systems must achieve high levels of efficiency, robustness, security, scalability, manageability, unobtrusiveness, and openness/ease of application integration.

We describe the Entropia distributed computing system as a case study, detailing its internal architecture and philosophy in attacking these key problems. Key aspects of the Entropia system include the use of: (1) binary sandboxing technology for security and unobtrusiveness, (2) a layered architecture for efficiency, robustness, scalability and manageability, and (3) an open integration model to allow applications from many sources to be incorporated.

Typical applications for the Entropia System includes molecular docking, sequence analysis, chemical structure modeling, ;and risk management. The applications come from a diverse set of domains including virtual screening for drug discovery, genomics for drug targeting, material property prediction, and portfolio management. In all cases, these applications scale to many thousands of nodes and have no dependences between tasks. We present representative performance results from several applications that illustrate the high performance, linear scaling, and overall capability presented by the Entropia system.

Keywords: Grid computing; Distributed computing; High-throughput computing; Scientific computing

1. Introduction

For over 4 years, the largest computing systems in the world have been based on "distributed computing" the assembly of large numbers of PCs over the Internet. These "grid" systems sustain multiple teraflops continuously by aggregating hundreds of thousands to millions of machines and demonstrate the utility of such resources for solving a surprisingly wide range of large-scale computational problems in data mining, molecular interaction, financial modeling, etc. These systems have come to be called "distributed computing" systems and leverage the unused capacity of high-performance desktop PCs (up to 2.2 GHz machines with multi-gigaop capabilities [31]), high-speed local-area networks (100 Mbps to 1 Gbps switched), large main memories (256 MB to 1 GB configurations), and large disks (60–100 GB disks). Such distributed computing systems leverage the installed hardware capability (and work well even with much lower performance PCs) and thus can achieve a cost per unit computing (or Return-On-Investment) superior to the cheapest hardware alternatives by as much as a factor of 5 or 10. As a result, distributed computing systems are now gaining increased attention and adoption within enterprises to solve their largest computing problems and attack new problems of unprecedented scale. In this paper, we focus on enterprise distributed computing. We use the terms distributed computing, high throughput computing, and desktop grids synonymously to refer to systems that tap vast pools of desktop resources to solve large computing problems.

*Corresponding author. 9500 Gilman Drive, Dept 0114, La Jolla, CA 92093, USA.

E-mail addresses: achien@entropia.com (A. Chien), bcalder@entropia.com (B. Calder), selbert@entropia.com (S. Elbert), kbhatia@entropia.com (K. Bhatia).

[1] Also affiliated with the Department of Computer Science and Engineering at the University of California, San Diego (UCSD).

0743-7315/03/$ - see front matter
doi:10.1016/S0743-7315(03)00006-6

For a number of years, a significant element of the research and now commercial computing community has been working on technologies for "grid computing" [3,18,19,24,44]. These systems typically involve servers and desktops, and their fundamental defining feature is to share resources in new ways. In our view, the Entropia system is a desktop grid which can provide massive quantities of resources and will naturally be integrated with server resources into an enterprise grid [14,20].

While the tremendous computing resources available through distributed computing present new opportunities, harnessing them in the enterprise is quite challenging. The system must operate in environments of extreme heterogeneity in machine hardware and software configuration, network structure, and individual/network management practice. And because the primary function of the resources is desktop use (e.g. desktop word processing, web information access, spreadsheets, etc.), the resources must be exploited without effecting the desktop user's productivity.

To achieve a high degree of utility, distributed computing systems must capture a large number of valuable applications. Therefore the effort required to deploy an application on the system, in a way that secures the application and its data as it executes, must be minimal. Of course, the systems must also support large numbers of resources, thousands to millions of computers, to achieve the promise of tremendous power and do so without requiring armies of IT administrators.

The Entropia system provides solutions to the above desktop distributed computing challenges. The key advantages of the Entropia system are the ease of application integration and a new model for providing security and unobtrusiveness for the application and client machine. To provide rapid application integration, Entropia uses binary modification technology that obviates access to the applications source code while providing strong security guarantees and ensuring unobtrusive application execution. Other systems require developers to modify their source code to use custom APIs or simply rely on the application to be "well behaved" and provide weaker security and protection [6,36,48]. It is not always possible to get access to the application source code (especially for commercially available applications) and, regardless, maintaining multiple versions of the source code can require a significant ongoing development effort. As to relying on the good intentions of the application programmers, we have found that even commonly used applications in use for quite some time can at times exhibit anomalous behavior. Entropia's approach ensures both a large base of potential applications and a high level of control over the application's execution.

The remainder of the paper includes the history of distributed computing which has led to large-scale distributed computing systems in Section 2; the key technical requirements for a distributed computing platform in Section 3; the Entropia system architecture and implementation, including its key elements and how it addresses the technical requirements in Section 4; and a description of typical applications and their performance on the Entropia system in Section 5. We conclude and briefly present a perspective on the future potential of distributed computing in Section 6.

2. Background

The idea of distributed computing has been described and pursued as long as there have been computers connected by networks. Early justifications of the ARPANET [25] described the sharing of computational resources over the national network as a motivation for building the system. In the mid-1970s, the Ethernet was invented at Xerox PARC, providing high bandwidth local-area networking. This invention combined with the Alto Workstation presented another opportunity for distributed computing and the PARC Worm [38] was the result. In the 1980s and early 1990s several academic projects developed distributed computing systems that supported one or several Unix systems [6,29,30,41,50]. Of these, the Condor Project is perhaps the best known and most widely used. These early distributed computing systems focused on developing efficient algorithms for scheduling, load balancing and fairness. However, these systems provided no special support for security and unobtrusiveness, particularly in the case of misbehaving applications. Furthermore, they did not manage dynamic desktop environments, limited what was allowed in application execution, and incurred a significant management effort for each machine.

At approximately the same time, the scientific computing community began to move from large expensive supercomputers to relatively inexpensive Unix workstations [45] and, in the late 1990s, to low-cost PC hardware [7,42]. During this time, clusters of inexpensive PCs connected with high-speed interconnects were demonstrated to rival supercomputers for some applications. These systems provided clear evidence that PCs could deliver serious computing power.

The growth of the World Wide Web (WWW) [23] and the exploding popularity of the Internet created a new much larger scale opportunity for distributed computing. For the first time, millions of desktop PCs were connected to wide-area networks both in the enterprise and in the home. The number of machines potentially accessible to an Internet-based distributed computing system grew into the tens of millions of systems for the first time. The scale of the resources (millions), the types

of systems (windows PCs, laptops), and the typical ownership (individuals, enterprises) and management (intermittent connection, operation) gave rise to a new explosion of interest in and a new set of technical challenges for distributed computing.

In 1996, Scott Kurowski partnered with George Woltman to begin a search for large prime numbers, a task considered synonymous with the largest supercomputers. This effort, the "Great Internet Mersenne Prime Search" or GIMPS [13,51], has been running continuously for over 5 years with over 200 000 machines participating and has discovered the 35th, 36th, 37th, 38th, and 39th Mersenne primes—the largest known prime numbers. The most recent was discovered in November 2001 and is over 4 million digits in length.

The GIMPS project was the first project taken on by Entropia, Inc., a startup commercializing distributed computing. Another group, distributed.net [10], pursued a number of cryptography related distributed computing projects in this period as well. In 1997, the best-known Internet distributed computing project, SETI@home [43], began and rapidly grew to several million machines (typically about 0.5 million active at any one time). These early Internet distributed computing systems showed that aggregating very-large-scale resources was possible and that the resulting system dwarfed the resources of any single supercomputer, at least for a certain class of applications. But these projects were single-application systems, difficult to program and deploy, and very sensitive to the communication-to-computation characteristic of the application. A simple programming error could cause network links to be saturated and servers to be overloaded.

The current generation of distributed computing systems, a number of which are commercial ventures, provide the capability to run multiple applications on a collection of desktop and server computing resources [8,15,36,48]. These systems are evolving towards a general-use compute platform. As such, providing tools for application integration and robust execution are the focus of these systems.

Grid technologies developed in the research community [19,24] have focused on issues of security, interoperation, scheduling, communication, and storage. In all cases, these efforts have been focused on Unix servers. For example, the vast majority, if not all, of the Globus and Legion activity has been done on Unix servers. Such systems differ significantly from the Entropia system as they do not address issues that arise in a desktop environment including dynamic naming, intermittent connection, untrusted users, etc. Further, they do not address a range of challenges unique to the Windows environment, whose five major variants are the predominant desktop operating system.

3. Requirements for distributed computing

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

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

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

Security—The system must protect the integrity of the distributed computation. Tampering with or disclosure of the application data and program must be prevented. The Distributed Computing system must also protect the integrity of the computing resources that it is aggregating. Distributed computing applications must be prevented from accessing or modifying data on the computing resources.

Scalability—The system must scale to the use of large numbers of computing resources. Because large numbers of PCs are deployed in many enterprises, scaling to 1000s, 10,000s, and even 100,000s are relevant capabilities. However, systems must scale both upward and downward performing well with reasonable effort at a variety of system scales.

Manageability—Any system involving thousands to hundreds of thousands of entities must provide management and administration tools. Typical rules of thumb, such as requiring even one administrator for every 200 systems, would be unacceptable. We believe distributed computing systems must achieve manageability that requires no incremental human effort as clients are added to the system. A crucial part of manageability is client cleanliness: it is crucial that the computing resources state is identical after running an application as it was before running the application.

Unobtrusiveness—The system typically shares resources (both computing, storage, and network resources) with other systems in the corporate IT environment. As a result, the use of these resources should be unobtrusive, and where there is competition, non-aggressive. The distributed computing system must

manage its use of resources so as not to interfere with the primary use of the desktops and networks for other activities. This includes both the use due to system activities as well as use driven by the distributed computing application.

Openness/ease of application integration—Fundamentally, the distributed computing system is a platform on which to run applications. The number, variety, and utility of the applications supported by the system directly affects its utility. Distributed computing systems must support applications developed with all kinds of models, with many distinct needs and with minimal effort.

Together, we believe these seven criteria represent the key requirements for distributed computing systems.

4. Entropia system architecture

The Entropia system addresses the seven key requirements described above by aggregating the raw desktop resources into a single logical resource. This logical resource is reliable, secure and predictable despite the fact that the underlying raw resources are unreliable (machines may be turned off or rebooted), insecure (untrusted users may have electronic and physical access to machines) and unpredictable (machines may be heavily used by the desktop user at any time). This logical resource provides high performance for applications through parallelism while always respecting the desktop user and his or her use of the desktop machine. Furthermore, this logical resource can be managed from a single administrative console. Addition or removal of desktop machines from the Entropia system is easily achieved, providing a simple mechanism to scale the system as the organization grows or as the need for computational cycles grows.

To support the execution of a large number of applications, and to support the execution in a secure manner, Entropia employs proprietary binary sandboxing techniques that enable any Win32 application to be deployed in the Entropia system with no modifications and no special system support. End-users of the Entropia system can use their existing Win32 applications and deploy them on the Entropia system in a matter of minutes. This is significantly different than the early large-scale distributed computing systems like SETI@home and other competing systems that require rewriting and recompiling of the application source code to ensure safety and robustness.

4.1. Enabling applications

The openness/ease of application integration requirement (detailed in Section 3) in some ways naturally conflicts with the security and unobtrusiveness requirements. The former requirement aims to broaden the set of applications that can run on the distributed computing system and allow those applications to be integrated with little burden. This may include applications that were not originally designed to run in a distributed computing setting or applications that may be fragile or in development. Running these applications as-is on thousands or tens of thousands of desktop clients may violate the latter requirements regarding security and unobtrusiveness. These latter requirements necessarily restrict the actions of the application or impose constraints on how they operate, thereby reducing the set of applications that are suitable for distributed computing.

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

As an example of the power of this technique, consider file system access by the application: an application uses standard Windows APIs for opening closing, reading and writing to a file. The Entropia mediation layer enables the distributed computing system to automatically (and invisibly to the application) encrypt the contents of all data files and provide data integrity checks, ensuring that the data is not tampered with by the desktop user. In addition, to protect the desktop user from the application, the sandbox automatically maps the file and directory structure for the application to ensure that all files read or written remain within known bounds. Therefore, an application may believe that it is reading or writing to the directory `C: \Program Files\` when in fact it is writing to a directory deep within the Entropia software installation.

This technique allows the Entropia system to support arbitrary applications written in any programming language that can be compiled down to Windows executables (C, C++, C#, Java, FORTRAN, etc.) and even third-party shrinkwrapped software and common scripting languages. Since no source code is

required, the use of binary sandboxing supports the broadest range of applications possible with little effort. At the same time, because the sandbox mediates the application execution at the system call level, it provides fine-grain control of the system and desktop resources used by the applications.

4.2. Layered architecture

The Entropia system architecture is composed of three separate layers (see Fig. 1). At the bottom is the Physical Node Management layer that provides basic communication and naming, security, resource management, and application control. On top of this layer is the Resource Scheduling layer that provides resource matching, scheduling, and fault tolerance. Users can interact directly with the Resource Scheduling layer through the available APIs or alternatively, users can access the system through the Job Management layer that provides management facilities for handling large numbers of computations and files. Entropia provides an implementation of the Job Management layer, but other implementations can be developed as needed on top of the Resource Scheduling layer.

We briefly describe these three layers, and then describe the advantages of this approach in achieving a robust, scalable distributed computing resource.

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

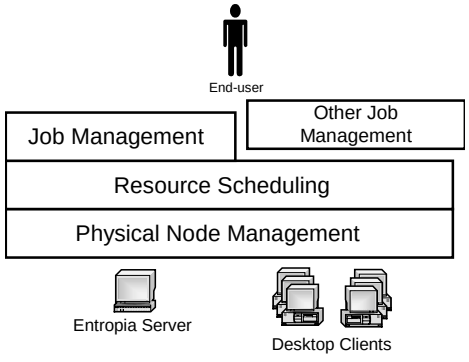

Fig. 1. Architecture of the Entropia Distributed Computing System. The Physical Node Management layer and Resource Scheduling layer span the servers and client machines. The Job Management layer runs only on the servers. Other (non-Entropia) Job Management systems can be used with the system.

In addition to communication and naming, the Physical Node Management layer provides resource management, application control, and security. The resource management services capture a wealth of static and dynamic information about each physical node (e.g. physical memory, CPU speed, disk size, available space, client version, data cached, etc.), reporting it to the centralized node manager and system console. The application control services provide basic facilities for process management including file staging, application initiation and termination, and error reporting. It also ensures that nodes can be recovered from runaway applications, detects and terminates misbehaving applications, and detects and reports any damage to the software client installation. The security services employ a range of encryption and binary sandboxing technologies to protect both distributed computing applications and the underlying physical node. Application communications and data are protected with high-quality cryptographic techniques. A binary sandbox controls the operations and resources that are visible to distributed applications on the physical nodes in order to protect the software and hardware of the underlying machine. This same sandbox also controls the usage of resources (memory, disk, network) by the distributed computing application.

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

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

Job Management: A distributed computing application often involves large amounts of computation (thousands to millions of CPU hours) submitted as a single large job. This job is then broken down into a large number of individual subjobs each of which is submitted into the Entropia system for execution. The Job Management layer of the Entropia system is responsible for decomposing the single job into the many subjobs, managing the overall progress of the job, providing access to the status of each of the generated

subjobs, and aggregating the results of the subjobs. This layer allows users to submit a single logical job (for example, a Monte Carlo simulation, a parameter sweep application, or a database search algorithm) and receive as output a single logical output. The details of the decomposition, execution and aggregation are handled automatically.

This three-layer approach provides a wealth of benefits in system capability, ease of use by end-users and IT administrators, and internal implementation. The Physical Node Management layer manages many of the complexities of the communication, security, and naming, allowing the layers above to operate with simpler abstractions. The Resource Scheduling layer deals 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 provides job decomposition and aggregation and basic data management facilities in a convenient and scalable web interface.

4.3. Implementation

In this section we provide implementation details for the Entropia system's three architecture layers, using an end-to-end scenario to illuminate the function of each part. Fig. 2 shows a high-level view of the Entropia system, annotated with the interactions of the components. We now describe each of these components and their interaction.

In step 1, the user interacts with the Entropia system by submitting a job. A job represents an application, a collection of inputs to be run for that application, and the data to be used. We call each independent set of inputs a *subjob*, which is the unit of work assigned to client machines. When using the Job Manager, each application is registered in advance and consists of an application binary, a pre-processor, and a post-processor. Datasets may also be registered in advance. There is no registration requirement when sending subjobs directly to the Resource Scheduling Layer.

Job Manager: The Job Manager is responsible for pre- and post-processing of a job, submitting the subjobs created to the subjob scheduler, and making sure that all of the subjobs corresponding to the job complete in a timely fashion. At the job submission interface, users specify the application and data files, the priority, and attributes of the clients needed to run the job.

The application the user submits to the Entropia system in step 1 consists of Win32 binaries and dll's, and Visual Basic scripts. The binaries submitted for an application are automatically sandboxed to provide a secure and safe environment for the execution of the subjob on the client.

An application-dependent pre-processor is used in step 2 to break the job up into subjobs. Pre-processing

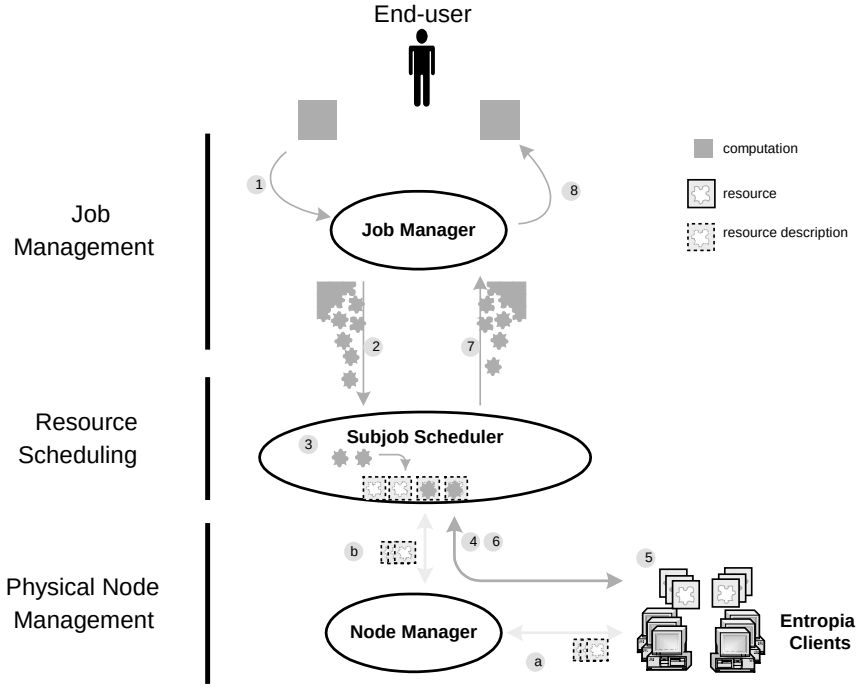

Fig. 2. Application execution on the Entropia system. An end-user submits a computation to the Job Manager (1). The Job Manager breaks up the computation into many independent "subjobs" (2) and submits the subjobs to the Subjob Scheduler. In the mean time, the available resources of a client are periodically reported to the Node Manager (a) that informs the Subjob Scheduler (b) using the resource descriptions. The Subjob Scheduler matches the computation needs with the available resources (3) and schedules the computation to be executed by clients (4, 5, 6). Results of the subjobs are sent to the Job Manager (7), aggregated together, and handed back to the end-user (8).

may be as simple as enumerating through a list of parameters for the application creating a subjob for each set of parameters or as complex as a running a program to split up a database into a number of slices—one for each subjob. Each subjob is submitted to the Subjob Scheduler.

Subjob Scheduler: The Subjob Scheduler is responsible for scheduling subjobs to clients and their robust execution. The subjob scheduler maintains a list of clients on which it is allowed to run subjobs. This list is decorated with attributes such as a client's connectivity (whether they are currently connected or not), a client's status and availability to run a subjob, and static client attributes (e.g., amount of memory on client, OS type, etc.). All of this information is used to schedule subjobs effectively. In Fig. 2, the boxes with a hole in the center, inside the subjob scheduler, represent client machines and their attributes assigned to that subjob scheduler. Step 3 illustrates the scheduler assigning subjobs to clients, ensuring that client's attributes satisfy the subjob requirements. For example, if the user specifies that a job needs a minimum of 128 megabytes of memory, then that job's subjobs will only be assigned to clients with at least that amount of memory.

The subjob scheduler maintains several priority queues of subjobs to be run on the Entropia Grid. As subjobs are submitted to the scheduler, they are placed into the queue of appropriate priority (determined by the priority associated with the job). A subjob scheduler can schedule subjobs for thousands of clients. To provide scalability to larger number of clients and to provide subjob scheduler fault tolerance, additional schedulers can be added to the configuration to serve a job manager.

The subjob scheduler is also responsible for providing fault tolerance, insuring progress towards completion, and identifying faulty subjobs. This is provided by the following set of policies. The priority for a subjob is increased if it is not assigned to a client in a reasonable amount of time. A subjob is rescheduled to run on a different client if the client becomes disconnected for ;too long or fails to return a result within the expected amount of time. If the subjob fails to complete after a given number of tries the subjob is marked as faulty and returned to the job manager. The job manager can then choose to re-submit the subjob or notify the user.

After a subjob has been assigned to a client, the next step is to send the subjob description and security information for the subjob to the client (step 4). When running the subjob (step 5), the files are transferred to the client from a file server or accessed via the network. It is advantageous to cache executable or data files for an application if the same job is run many times on the Entropia system. Caching of files provides scalability and manageability by reducing network traffic.

The next step of subjob execution is to invoke a user-provided script that can be used to orchestrate the starting of the subjob's executables. As the subjob runs, its access to the machine's resources is controlled and monitored by Entropia's binary sandbox to ensure that the subjob is unobtrusive, well behaved, and stays within the resources the client machine has available. In most other systems [49], resource scheduling assumes dedicated resources. This is not the case in a desktop distributed computing environment.

When the subjob completes execution, the client notifies the subjob scheduler (step 6). If there is a problem, the subjob may be resubmitted or deemed to be a failed subjob. Regardless, the subjob scheduler notifies the job manager (step 7) of the success or failure for running the subjob. The job manager provides an interface to monitor the progress of a job and all of its subjobs. When all or a specified fraction of the subjobs are complete, the results are post-processed. For example, a post-processing phase might filter through the results to find the top result. Finally, in step 8 the user retrieves the results of the overall job, or if desired each subjob.

Deployment and Management: The Entropia Grid of clients is managed via the Node Manager as shown in Fig. 2. When the Entropia client is installed on a machine it registers itself with a specified node manager (step a). This registration includes providing a list of all of the client's attributes. These attributes are passed to the subjob scheduler when the node manager assigns to the subjob manager its list of available clients (step b).

The node manager provides a centralized interface to manage all of the clients on the Entropia grid, which is accessible from anywhere on the enterprise network. The interface allows an administrator to easily monitor, add, remove, stop, and restart the clients. The node manager tracks the status of all of the machines. This includes the connectivity of the client, how much work has been performed by the client, and if there are any issues or problems for a specific client. The interface is designed to provide scalable management to vast numbers of clients, requiring minimal effort per client added to the grid.

Desktop Client: The goal of the Desktop Client is to harvest unused computing resources by running subjobs unobtrusively on the machine. First, subjobs are run at a low process and thread priority. The sandbox enforces this low priority on all processes and threads created. Second, the Desktop Client monitors desktop usage of the machine and resources used by the Entropia system. If desktop usage is high, the client will pause the subjob's execution, avoiding possible resource contention. The resources monitored by the client include memory and disk usage, and process and thread allocation for the subjob. If pausing the subjob does not remedy the situation, termination of the subjob may

be necessary. Third, the Desktop Client provides security for the client machine by mediating subjob access to the file system, registry, and graphical user interface. This prevents the subjob's processes from doing harm to the machine, and ensures that after subjob execution, the machine's state is the same as it was before running the subjob.

4.4. Sandboxed desktop execution

To provide this sandboxed environment, we use binary modification to intercept all important Windows API calls. This allows us to have complete control over the application and its interaction with the desktop machine.

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

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

We sandbox the application's access (Fig. 3) and usage of system resources, such as the registry and file system. This ensures that any modification of these resources is redirected to the Entropia sandbox. The sandbox prevents applications from maliciously or accidentally causing harm to the desktop resources, since they cannot modify the desktop's registry or file system. Without this protection, an application can modify system and user files, or even erase or reformat the desktop machine's hard drive.

(1) "Sandbox" the application: a single command that takes the application's executables and associated dll's and creates new "sandboxed" versions.
(2) Authenicate DCGrid user to system.
(3) Submit the subjob specifiing input files (including executables, libraries and output files), resource requirements (including minimum memory, disk, processor speed, run time, priority, etc.), and a script to be executed on the client.
(4) Optionally check subjob status.
(5) Process output files when subjob is complete.

Fig. 3. Minimal steps for submitting an application to the Entropia system.

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

A top requirement of IT departments is that the state of the desktop machine remains unchanged after executing an application. Without this requirement, most IT departments will not allow the deployment of a desktop PC grid computing system inside their enterprise. Entropia's control of an application via our sandbox guarantees that this requirement is met.

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

The Entropia sandbox keeps all data files encrypted on disk, so that their contents are not accessible by non-Entropia applications.

The sandbox automatically monitors and checks data integrity of grid applications and their data and result files. This ensures that accidental or intentional tampering with or removal of grid application files by desktop users will be detected, resulting in the rescheduling of the subjob on another client.

5. Application performance

In this section we describe the applications and the performance of these applications running on the Entropia system.

5.1. Application characteristics

Early adoption of distributed computing technology is focused on applications that are easily adapted and whose high demands cannot be met by traditional approaches whether for cost or technology reasons. For these applications, sometimes called "high throughput" applications, very large capacity provides a new kind of capability.

The applications exhibit large degrees of parallelism (thousands to even hundreds of millions) with little or no coupling, in stark contrast to traditional parallel applications, which are more tightly coupled. These high throughput computing applications are the only ones capable of not being limited by Amdahl's law.

We believe the widespread availability of distributed computing will encourage reevaluation of many existing algorithms to find novel uncoupled approaches, ultimately increasing the number of applications suitable for distributed computing. For example, Monte Carlo or other stochastic methods that are too inefficient using conventional computing approaches may prove attractive when considering time to solution.

We describe four application types successfully using distributed computing: virtual screening; sequence analysis; molecular properties and structure; and financial risk analysis. We discuss the basic algorithmic structure, from a computational and concurrency perspective, the typical use and run sizes and the computation/communication ratio. A common characteristic of all these applications is the independent evaluation of, at most, a few megabytes of data requiring several minutes or more of CPU time.

5.1.1. Virtual screening

One of the most successful early applications is virtual screening, the testing of hundreds of thousands (to millions) of candidate drug molecules to see if they alter the activity of a target protein that results in unhealthy affects. At present, all commercial drugs address only 122 targets, with the top 100 selling drugs addressing only 45 [46]. Current estimates place the number of "druggable" genes at 5–10,000, with each gene encoding for around 10 proteins, with 2–3% considered high value targets. Testing typically involves assessing the binding affinity of the test molecule to a specific place on a protein in a procedure commonly called docking. Docking codes [5,9,12,16,17,26–28,33,34] are well-matched for distributed computing as each candidate molecule can be evaluated independently. The amount of data required for each molecular evaluation is small—basically the atomic coordinates of the molecules—and the essential results are even smaller, a binding score. The computation per molecule ranges from seconds to tens of minutes or more on an average PC. The coordination overhead can be further reduced by bundling sets of molecules or increasing the rigor of the evaluation. Low thresholds can be set for an initial scan to quickly eliminate clearly unsuitable candidates and the remaining molecules can be evaluated more rigorously. Entropia now has experience with numerous docking codes and can generally deploy new codes in less than an hour with the Job Manager if a Win32 executable is available.

5.1.2. Sequence analysis

Another application area well-suited to distributed computing involves DNA or protein sequence analysis applications, including the BLAST programs [1,2,22,32,52], HMMER [11], CLUSTAL W [47], FASTA [35], Wise2 [4] and Smith–Waterman [40]. In these cases one sequence or set of sequences is compared to another sequence or set of sequences and evaluated for similarity. The sequence sizes vary, but each comparison is independent. Sets of millions of sequences (gigabytes) can be partitioned into thousands of slices, yielding massive parallelism. Each compute client receives a set of sequences to compare and the size of the database, enabling it to calculate expectation values properly for the final composite result. This simple model allows the distributed computing version to return results equivalent to serial job execution. Distributing the data in this manner not only achieves massive input/output concurrency, but actually reduces the memory requirements for each run, since many of sequence analysis programs hold all the data in memory. Entropia has experience with a number of the most commonly used sequence analysis applications, including BLAST, HMMER, and versions of Smith–Waterman. Each of these uses a different comparison algorithm with BLAST usually the quickest and Smith–Waterman the slowest. BLAST, which was developed to solve genomic related problems is now also used in solving problems in proteomics, an area with exponentially growing processing requirements.

5.1.3. Molecular properties and structure

The demand for structure and property information for combinatorial chemistry libraries means a heavy demand for molecular modeling programs such as Gaussian 98 [21], GAMESS [37], and Jaguar [39]. These programs are often deployed in a data parallel mode similar to docking, where, again, the data-parallelism arises from independent molecule evaluations. In this case, the evaluation time per molecule may be hours or even days. The algorithms are based on first principles by solving physical laws such as the Schrödinger Equation. The memory (often 256 MB or more) and disk (gigabytes or more) requirements of these programs require their deployment on machines with sufficient resources. Fortunately, many desktops now meet these requirements. Entropia has successfully demonstrated

the ability to use these codes to evaluate large libraries of molecules. The results can be fed back into large corporate databases, which may contain millions of compounds.

5.1.4. Financial risk management

Risk management is an essential element of every financial institution. Monte Carlo methods are used to evaluate a wide range of possible outcomes but the number of samples evaluated has a direct impact on the reliability of results that can be achieved. Several widely used commercial packages typically evaluate 300–1000 samples, falling well short of the number needed to achieve seed independence (10,000), i.e., a result that is independent of the starting point. Further increases in the number of samples as well as increased model complexity can increase the accuracy of results. In this application, each sample simulation is independent, and can be executed on a distinct processor to achieve massive parallelism. Entropia distributed computing is employed with a range of risk analysis applications to significantly increase the accuracy of their Monte Carlo modeling results, while still completing analysis results in a timely fashion.

5.1.5. Demand and scalability

One of the most interesting aspects of distributed computing with these applications is the scalability. The computational demands of these applications scales with rapidly increasing data sets and algorithm complexity and is unlikely to be met anytime soon. As a result, the scalability offered by distributed computing is the only viable solution. For the applications we have discussed there are problems that scale well to hundreds of thousands of processors, with typical cases scaling well for thousands of processors.

Table 1 shows the latent demand of these applications. Each of these problems can easily require several hundred thousand processors, where 24-h turnaround is required. For example, many pharmaceutical libraries have a million potential drug molecules that can be tested against sets of thousands of proteins. Large-scale sequence analysis problems often involve all against all comparisons, in this case 700,000 sequences (2×10^8 amino acids). The molecular structure and property problem is a more thorough evaluation of a combinatorial library of 10^6 molecules (there are many such sets). The risk management problem is a rigorous daily analysis of a million credit card accounts. The data required in each case is less than a few megabytes.

Table 1
Application requirements

Area	Problem	Subjobs	Minutes/subjob	Nodes/24 h
Virtual screening	10^6 mol. vs. 10^4 proteins	10^8	10	200,000
Sequence analysis	2×10^8 AA vs. 2×10^8	10^7	60	400,000
Properties and structure	Evaluate 10^6 mol.	10^6	600	500,000
Risk management	10^4 sim. for 10^6 credit cards	10^7	60	400,000

Interestingly, executing these large, decoupled applications avoid many of the common bottlenecks of traditional parallel computing. For example, load balancing is a problem only for small problems, not large ones. With large problems the longest work unit is still only a small fraction of the time the total job requires, which leads to efficient tiling regardless of the heterogeneity in the system.

We should note that distributed computing is powerful even with as few as 100 processors. Even systems with a few hundred processors can solve many problems a 100 times faster than a single machine. Such speedups can transform a weeklong runtime into less than 1 h. For many compute intensive problems, we believe that the benefits for distributed computing are so dramatic that they will transform how modeling, design, and many other computations are used to drive business and research.

5.2. Experimental results

Managing a heterogeneous volatile grid is a difficult challenge. Fig. 4 is a Gantt chart that illustrates typical behavior encountered in production enterprise environments. The figure illustrates a number of the challenges addressed by the physical node management layer. The Gantt chart depicts activity of nodes in an Entropia distributed computing system where each horizontal line corresponds to a client machine and is marked black when the node is executing a subjob and white when it is not.

The chart depicts the execution of a docking program that is evaluating 50,000 molecules partitioned into 10,000 slices of five molecules each. Each subjob works on one slice and processes five molecules. The start of a subjob is indicated by a dot. Most of the nodes start another subjob as soon as one is completed and it looks like a continuous line with dots. Where blanks occur no subjobs are being run. This may be because the node is no longer available for a variety of reasons (turned off, offline, network problems, etc.).

At the 900-min mark the system administrator added more nodes to the grid. The effect, however, would be similar if another job had finished and its nodes became available to this job.

Triangles indicate when a node was deliberately stopped. The two vertical bands of node stoppages are

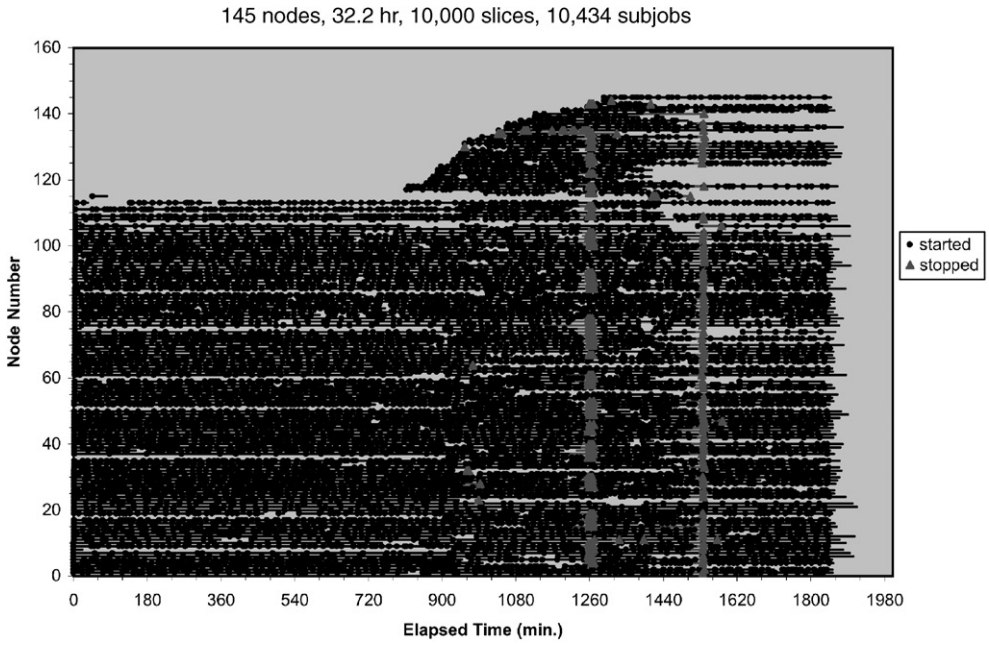

Fig. 4. Gantt chart depicting activity of nodes in enterprise environment.

the result of centrally administered system upgrades that required a reboot. Note that most of the application progress is preserved in each case. Other gaps scattered throughout the plot are due to network outages, machines turned off without proper shutdown, and machines suspected of misbehaving (requesting work without returning results). The latter can be caused by a wide variety of conditions ranging from hardware to software (OS) failure. Each subjob that failed was rerun on another node to successfully complete the overall job.

Ideally this job would have required only 10,000 subjobs, to complete, but in this case, 10,434 were required.

While most of the additional subjobs were caused by the reboots, some were the result of variation in subjobs execution time. In order to ensure that jobs are completed in timely fashion, the system will initiate redundant subjobs when a subjob has failed to return within the anticipated period (determined by the user). This technique trades plentiful resources for improvement in average job completion time. For this job the average subjob ran for 20 min but the range varied widely, from 8 s to 118 min. Of the 10,000 subjobs, 204 of them ran more than the anticipated limit of 60 min.

The distribution of actual subjob execution times (reflecting application work variability, node speed heterogeneity, and available cycles variability) is shown in Fig. 5 as the mixed grid data points. The average subjob execution time for the mixed grid was 20 min with a standard deviation of 13.4 min and a variance 181. Also shown is the distribution for each subjob when running on an 866-MHz Pentium III processor. The average subjob completion time for the 866 MHz environment was 15.3 min with a standard deviation of 7.8 min and a variance of only 60.6.

The distribution is typical of many applications, that is most subjobs complete near the average time but there are significant numbers of subjobs that complete either quickly or slowly. The smoothness is an artifact of 10,000 values sorted by size (one second resolution) and displayed on this scale. The slow subjobs present a challenge to scheduling algorithms to discern between subjobs that are simply slow and those that have failed.

The processors used in this example were Pentium II and III machines with clock speeds ranging from 266 MHz to 1 GHz and running two different versions (service packs) of Windows NT 4.0 and three versions of Windows 2000. The application code was a mixture of C (mostly) and FORTRAN developed for an SGI IRIX environment and required no changes except those related to porting to a Win32 environment. Some additional code was written to do pre- and post-processing of the data.

5.3. Application performance

To assess delivered application performance for a range of system sizes, a variety of calculations docking 50,000 molecules were made in the environment described above using up to 600 processors. The graph in Fig. 6 shows the resulting application throughput (molecules dockings completed per minute) over time for the case of 393 and 600 processors.

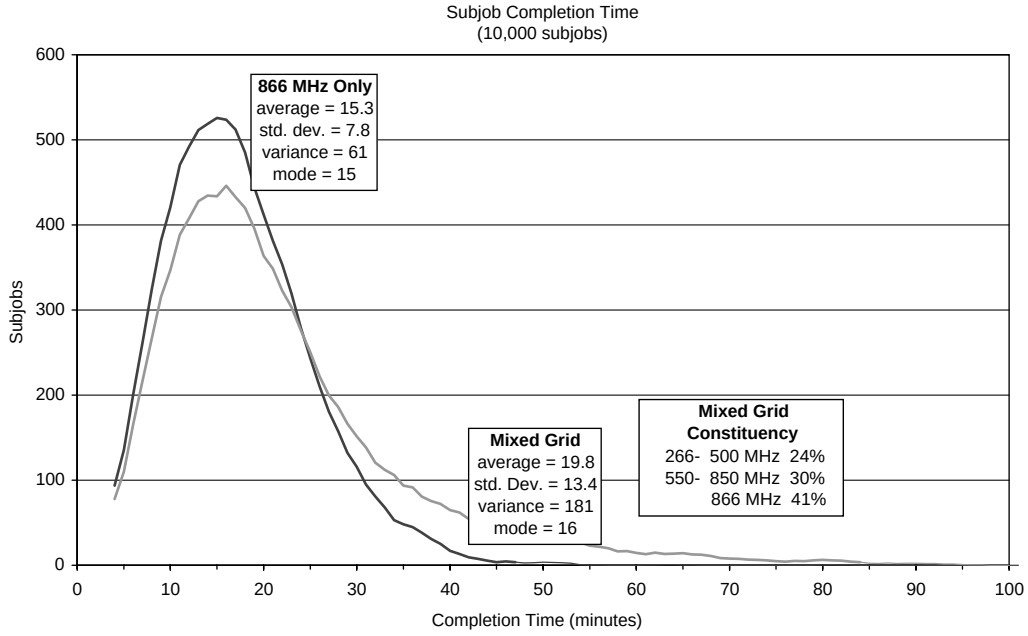

Fig. 5. Distribution of subjob completion times for docking code.

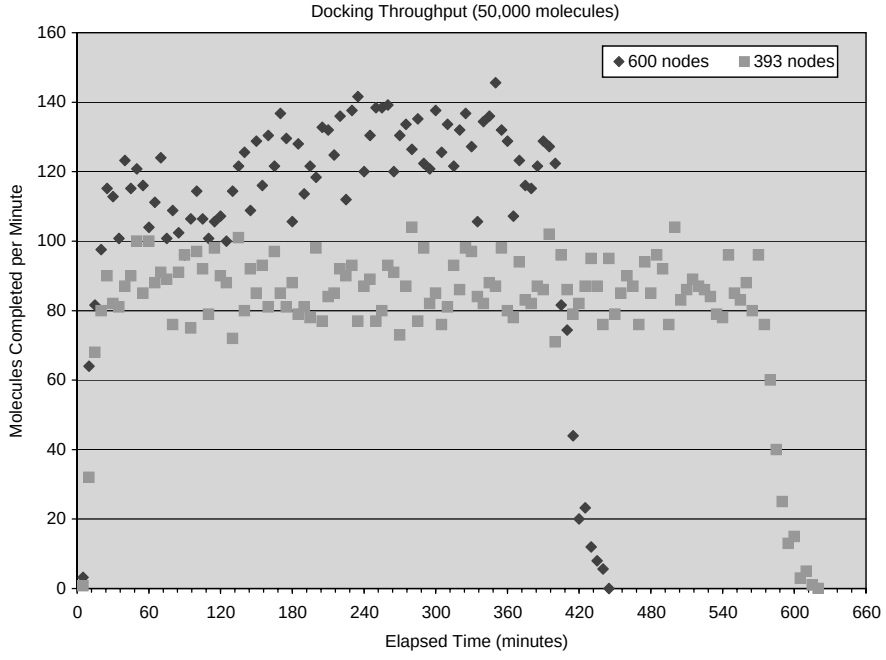

Fig. 6. Molecule dockings completed vs. time, runs on grid of size 393 and 600 nodes.

It takes approximately 20 min (the length of the average subjob) before a steady state of producing results is reached and then it is quite variable due to the variations in the size of individual subjobs. The 600-node job started out at 450 nodes and only reached 580 nodes after 130 min and 600 at the very end. The average throughput for the 393-node case was 87 molecules per minute and 122 molecules per minute for the 600-node case. The startup and finishing phases are discounted in calculating throughput. Plotting the throughput for nine runs of 50,000 molecules, varying the number of nodes from 145 to 600 produced the graph shown in Fig. 7. The figure demonstrates clearly the linear scaling we have observed for all of the applications discussed in the paper. Points representing the throughput for a 20-processor SGI system and a 66-processor Linux cluster are included for calibration, and demonstrate that high levels of absolute performance are being achieved.

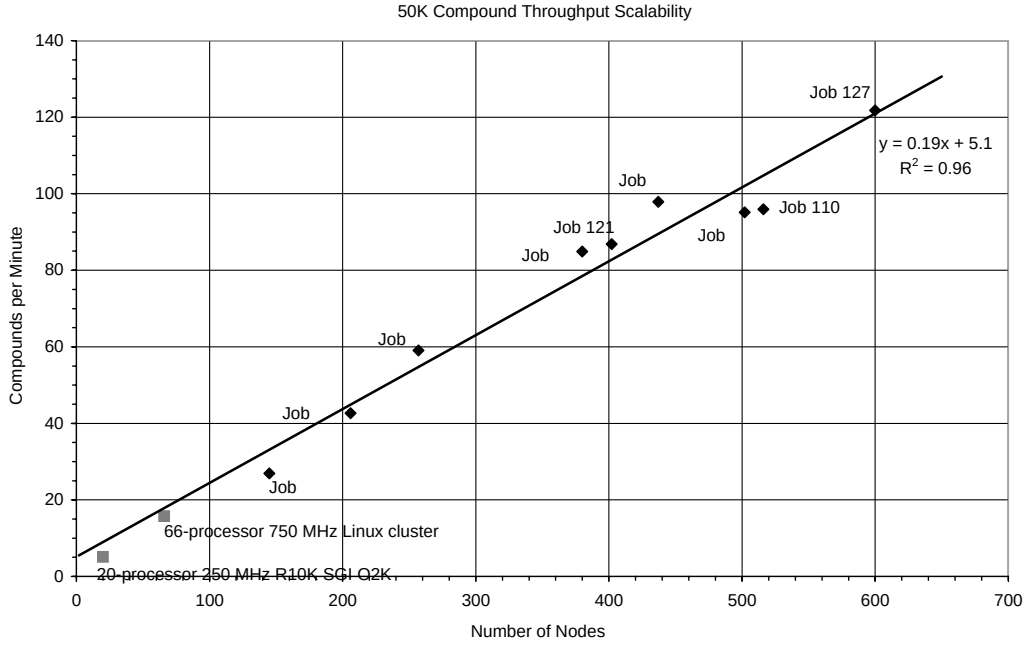

Fig. 7. Scaling of the Entropia system throughput on a virtual screening application.

6. Summary and futures

Distributed computing has the potential to revolutionize how much of large-scale computing is achieved. If easy-to-use distributed computing can be seamlessly available and accessed, applications will have access to dramatically more computational power to fuel increased functionality and capability. The key challenges to acceptance of distributed computing include robustness, security, scalability, manageability, unobtrusiveness, and openness/ease of application integration.

Entropia's system architecture consists of three layers: a physical node management layer, resource scheduling, and job scheduling. This architecture provides a modularity that allows each layer to focus on a smaller number of concerns, enhancing overall system capability and usability. This system architecture provides a solid foundation to meet the technical challenges as the use of distributed computing matures—supporting the broadening the problems supportable by increasing the breadth of computational structure, resource usage, and ease of application integration.

We have described the implementation of the Entropia system, and its use in a number of applications. The implementation includes innovative solutions in many areas, but particularly in the areas of security, unobtrusiveness, and application integration. The system is applicable to a large number of applications, and we have discussed virtual screening, sequence analysis, molecular modeling, and risk analysis in this paper. For all of these application domains, excellent linear scaling has been demonstrated for large distributed computing systems. We expect to extend these results to a number of other domains in the near future.

Despite the significant progress documented here, we believe we are only beginning to see the mass use of distributed computing. With robust commercial systems such as Entropia only recently available, widespread industry adoption of the technology is only beginning. At this writing, we are confident that within a few years, distributed computing will be deployed and in use in production within a majority of large corporations and research sites.

Acknowledgments

We gratefully acknowledge the contributions of the talented engineers and architects at Entropia to the design and implementation of the Entropia system. We specifically acknowledge the contributions of Kenjiro Taura, Scott Kurowski, Shawn Marlin, Wayne Schroeder, Jon Anderson, and Ed Anady to the definition and development of the system architecture. We also acknowledge Dean Goddette, Wilson Fong, and Mike May for their contributions to applications benchmarking and understanding performance data.

References

[1] S. Altschul, W. Gish, W. Miller, E. Myers, D. Lipman, Basic local alignment search tool, J. Mol. Biol. 215 (1990) 403–410.

[2] S. Altschul, T. Madden, A. Schffer, J. Zhang, Z. Zhang, W. Miller, D. Lipman, Gapped BLAST and PSI-BLAST: a new

generation of protein database search programs, Nucleic Acids Res. 25 (1997) 3389–3402.
[3] D. Barkai, Peer-To-Peer Computing: Technologies For Sharing and Collaborating on the Net, Intel Press, Santa-Clara, CA, 2001.
[4] E. Birney, Wise2: intelligent algorithms for DNA searches, http://www.sanger.ac.uk/Software/Wise2/ (2001).
[5] H. Bohm, Towards the automatic design of synthetically accessible protein ligands: Peptides, amides and peptidomimetics, J. Computer-Aided Mol. Design 10 (1996) 265–272.
[6] A. Bricker, M. Litzkow, M. Livny, Condor technical summary, Technical Report 1069, Department of Computer Science, University of Wisconsin, Madison, WI, January 1992.
[7] A. Chien, M. Lauria, R. Pennington, M. Showerman, G. Iannello, M. Buchanan, K. Connelly, L. Giannini, G. Koenig, S. Krishnamurthy, Q. Liu, S. Pakin, G. Sampemane, Design and evaluation of HPVM-based windows supercomputer, Internat. J. High Performance Comput. Appl. 13 (3) (1999) 201–219.
[8] DataSynapse Inc., http://www.datasynapse.com.
[9] K. Davies, THINK, Department of Chemistry, Oxford University, Oxford, 2001.
[10] Distributed.net, The fastest computer on earth, http://www.distributed.net.
[11] S. Eddy, HMMER: profile hidden markov models for biological sequence analysis, http://hmmer.wustl.edu/, 2001.
[12] M. Eldridge, C. Murray, T. Auton, G. Paolini, R. Mee, Empirical scoring functions: I. The development of a fast empirical scoring function to estimate the binding affinity of ligands in receptor complexes, J. Computer-Aided Mol. Design 11 (1997) 425–445.
[13] Entropia, Researchers discover largest multi-million-digit prime using entropia distributed computing grid, Press release, Entropia, Inc., December 2001.
[14] Entropia, Entropia announces support for open grid services architecture, Press release, Entropia, Inc., February 2002.
[15] Entropia, Inc., http://www.entropia.com.
[16] T. Ewing, I. Kuntz, Critical evaluation of search algorithms for automated molecular docking and database screening, J. Comput. Chem. 9 (18) (1997) 1175–1189.
[17] L.F.T. Eyck, J. Mandell, V.A. Roberts, M.E. Pique, Surveying molecular interactions with DOT, in: Proceedings of the ACM Conference on Supercomputing, San Diego, 1995.
[18] I. Foster, The Grid: Blueprint For a New Computing Infrastructure, Morgan Kaufmann, Los Altos, CA, 1998.
[19] I. Foster, C. Kesselman, The globus project: a status report, in: IPPS/SPDP '98 Heterogeneous Computing Workshop, Orlando, FL, 1998.
[20] I. Foster, C. Kesselman, S. Tuecke, The anatomy of the grid: Enabling scalable virtual organizations, Internat. J. Supercomput. Appl. 15 (3) (2001) 200–222.
[21] M.J. Frisch, G.W. Trucks, H.B. Schlegel, et al., Gaussian 98, Gaussian, Inc., Carnegie, PA, 2001.
[22] W. Gish, D. States, Identification of protein coding regions by database similarity search, Nat. Genet. 3 (1993) 266–272.
[23] M. Gray, Internet Growth Summary, MIT Press, Cambridge, MA, 1996.
[24] A. Grimshaw, W. Wulf, The legion vision of a worldwide virtual computer, Comm. ACM 40 (1) (1997) 39–45.
[25] F. Heart, A. McKenzie, J. McQuillian, D. Walden, ARPANET completion report, Technical Report 4799, BBN, January 1978.
[26] G. Jones, P. Willett, R.C. Glen, Molecular recognition of receptor sites using a genetic algorithm with a description of desolvation, J. Mol. Biol. 245 (1995) 43–53.
[27] E. Katchalski-Katzir, I. Shariv, M. Eisenstein, A. Friesem, C. Aflalo, I. Vakser, Molecular surface recognition: determination of geometric fit between proteins and their ligands by correlation techniques, Proc. Natl. Acad. Sci. USA 89 (1992) 2195–2199.
[28] B. Kramer, M. Rarey, T. Lengauer, Evaluation of the flexX incremental construction algorithm for protein-ligand docking, Proteins Struct. Funct. Genet. 37 (1999) 228–241.
[29] M.J. Litzkow, Remote Unix turning idle workstations into cycle servers, in: Proceedings of the Summer 1987 USENIX Conference, USENIX Assoc., Phoenix, AZ, USA, 1987, pp. 381–384.
[30] M.J. Litzkow, M. Livny, M.W. Mutka, Condor—a hunter of idle workstations, in: eighth International Conference on Distributed Computing Systems, San Jose, CA, USA, 1988, pp. 104–111.
[31] J. Lyman, Intel debuts 2.2ghz pentium 4 chip, The News Factor, January 7 2002.
[32] T. Madden, R. Tatusov, J. Zhang, Applications of network BLAST server, Methods Enzymol. 266 (1996) 131–141.
[33] M. McGann, FRED: Fast rigid exhaustive docking, openEye, 2001.
[34] G.M. Morris, D.S. Goodsell, R. Halliday, R. Huey, W.E. Hart, R.K. Belew, A.J. Olson, Automated docking using a lamarckian genetic algorithm and empirical binding free energy function, J. Comput. Chem. 19 (1998) 1639–1662.
[35] W.R. Pearson, D.J. Lipman, Improved tools for biological sequence comparison, Proc. Nat. Acad. Sci. USA 85 (1988) 2444–2448.
[36] Platform Computing, The load sharing facility, http://www.platform.com.
[37] M.W. Schmidt, K.K. Baldridge, J.A. Boatz, S.T. Elbert, M.S. Gordon, J.H. Jensen, S. Koseki, N. Matsunaga, K.A. Nguyen, S. Su, T.L. Windus, M. Dupuis, J.A. Montgomery, General atomic and molecular electronic structure system, J. Comput. Chem. 14 (1993) 1347–1363.
[38] J.F. Schoch, J.A. Hupp, The 'worm' programs–early experience with a distributed computation, Comm. ACM 25 (3) (1982) 172–180.
[39] Schrodinger, Jaguar, http://www.schrodinger.com/Products/jaguar.html, 2001.
[40] T. Smith, M. Waterman, Comparison of biosequences, Adv. Appl. Math. 2 (1981) 482–489.
[41] Z. Songnian, Z. Xiaohu, W. Jingwen, P. Delisle, Utopia: a load sharing facility for large, heterogeneous distributed computer systems, Software Practice Experience 23 (12) (1993) 1305–1336.
[42] T. Sterling, Beowulf Cluster Computing with Linux, The MIT Press, Cambridge, MA, 2001.
[43] W. Sullivan, A new major SETI project based on project serendip data and 100,000 personal computers, Astronomical and Biochemical Origins and the Search for Life in the Universe.
[44] Sun Microsystems, Jxta, http://www.jxta.org.
[45] V. Sunderam, PVM: a framework for parallel distributed computing, Concurrency, Practice Experience 2 (4) (1990) 315–339.
[46] A. Thayer, Genomics moves on, Chemical and Engineering News, October 14, 2002.
[47] J.D. Thompson, D. Higgins, T. Gibson, CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, positions-specific gap penalties and weight matrix choice, Nucleic Acids Res. 22 (1994) 4673–4680.
[48] United Devices, the MetaProcessor platform, http://www.ud.com.
[49] Veridian Systems, Portable Batch System, http://www.openpbs.org.
[50] C.A. Waldsburger, T. Hogg, B.A. Huberman, J.O. Kephart, W.S. Stornetta, Spawn: a distributed computational economy, IEEE Trans. Software Eng. 18 (2) (1992) 103–117.
[51] G. Woltman, The great internet mersenne prime search, http://www.mersenne.org.prime.htm.
[52] J. Zhang, T. Madden, PowerBLAST: a new network BLAST application for interactive or automated sequence analysis and annotation, Genome Res. 7 (1997) 649–656.