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

Future Generation Computer Systems xxx (2004) xxx–xxx

FGCS
FUTURE GENERATION COMPUTER SYSTEMS
www.elsevier.com/locate/future

From virtualized resources to virtual computing grids: the In-VIGO system

Sumalatha Adabala, Vineet Chadha, Puneet Chawla, Renato Figueiredo, José Fortes*, Ivan Krsul, Andrea Matsunaga, Mauricio Tsugawa, Jian Zhang, Ming Zhao, Liping Zhu, Xiaomin Zhu

Advanced Computing and Information Systems (ACIS) Laboratory, University of Florida, Gainesville, FL 32611-6200, USA

Received 25 September 2003; received in revised form 11 November 2003

Abstract

This paper describes the architecture of the first implementation of the In-VIGO grid-computing system. The architecture is designed to support computational tools for engineering and science research In Virtual Information Grid Organizations (as opposed to in vivo or in vitro experimental research). A novel aspect of In-VIGO is the extensive use of virtualization technology, emerging standards for grid-computing and other Internet middleware. In the context of In-VIGO, virtualization denotes the ability of resources to support multiplexing, manifolding and polymorphism (i.e. to simultaneously appear as multiple resources with possibly different functionalities). Virtualization technologies are available or emerging for all the resources needed to construct virtual grids which would ideally inherit the above mentioned properties. In particular, these technologies enable the creation of dynamic pools of virtual resources that can be aggregated on-demand for application-specific user-specific grid-computing. This change in paradigm from building grids out of physical resources to constructing virtual grids has many advantages but also requires new thinking on how to architect, manage and optimize the necessary middleware. This paper reviews the motivation for In-VIGO approach, discusses the technologies used, describes an early architecture for In-VIGO that represents a first step towards the end goal of building virtual information grids, and reports on first experiences with the In-VIGO software under development.

Keywords: Virtual machines; Grid-computing; Middleware; Virtual data; Network computing

1. Introduction

Information "grids" promise to revolutionize scientific computing and information processing by enabling access to unprecedented computing power and functionality of diverse interconnected computers, geographically distributed software applications and other network-accessible resources. Fundamentally, the goal of a grid infrastructure is to provide "flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions, and resources" [1]. Traditional approaches to grid-computing have focused on middleware-driven management of computers, storage, networks and other resources through sharing mechanisms that have evolved from centrally administered domains—multi-user operating systems, user accounts and file systems [2]. The middleware of these approaches interact with physical resources at the same level as local users and applications do.

* Corresponding author.
E-mail address: fortes@acis.ufl.edu (J. Fortes).

0167-739X/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2003.12.021

Therefore, such approaches must deal with the complexity of decoupling local administration policies and configuration idiosyncrasies of distributed resources from the quality of service expected from end users with respect to security, flexibility and performance. This complex decoupling can be simplified, or even eliminated, by fundamentally changing the way grid-computing is performed. The change advocated by this paper and the In-VIGO approach consists of raising the level of abstraction at which the middleware enables resource sharing from physical to virtualized resources.

Virtualization technologies encompass a variety of mechanisms and techniques used to decouple the architecture and user-perceived behavior of hardware and software resources from their physical implementation. The decoupling provided by virtualization enables functions and capabilities that are desired from computer systems, including consolidation of physical resources, security and isolation, flexibility and ease of management. These and other characteristics have motivated the use of virtualization technologies in different areas of computer system design. Examples include:

- Virtual memory [3], which allows multiple virtual address spaces to share a single physical memory, and provides to software the view of a memory larger than its physical implementation.
- Classical virtual machines [4] (e.g. VMware, IBM z800), which enable the sharing of a single physical computer (CPU, memory, I/O devices) by multiple virtual computers, each independently configured with their own operating system and applications.
- Application virtual machines and associated languages (e.g. Java and C#), which allow virtual machines to execute programs in physical machines with different instruction set architectures.
- Virtual private networks [5], which provide secure private networks over a shared public network.

Virtualization is the basis of the In-VIGO architecture. As a distributed computing system that includes processing nodes, storage devices, networks, software applications, and user interfaces, In-VIGO uses virtualization technologies in different forms, which have three fundamental capabilities in common. These capabilities are: polymorphism, manifolding and multiplexing. They allow resources to simultaneously (via multiplexing) appear as multiple resources (via manifolding) with possibly different functionalities (via polymorphism). These properties are illustrated in Fig. 1, where processor virtualization technology is used to enable a single physical computer to simultaneously appear as three separate machines with different operating systems and applications.

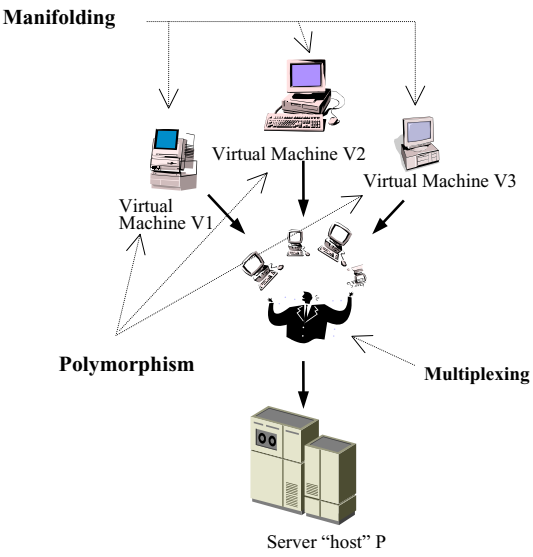


Fig. 1. A virtual machine monitor allows a server P to be shared by three distinct virtual machines.

This paper is organized as follows. Sections 2 and 3 introduce the In-VIGO approach and core techniques to support virtualized grid resources, data, and interfaces. A detailed description of the In-VIGO architecture appears in Section 4. Section 5 discusses the current implementation of In-VIGO. Conclusions from the design, development and initial experiences with In-VIGO appear in Section 6.

2. The In-VIGO concept

In-VIGO provides a distributed environment where multiple application instances can coexist in virtual or physical resources, such that clients are unaware of the complexities inherent to grid-computing. The In-VIGO project leverages experience with early systems and standards for grid-computing [1,6,7], and

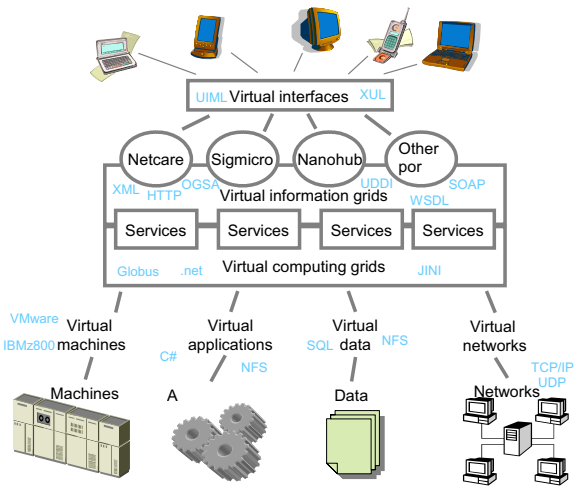


Fig. 2. The In-VIGO approach.

makes extensive use of virtualization technology for the creation of dynamic pools of virtual resources that can be aggregated on-demand.

The In-VIGO approach, as depicted in Fig. 2, adds three layers of virtualization to the traditional grid-computing model. The first virtualization layer creates pools of virtual resources that are the "primitive" components of a virtual computing grid, namely virtual machines, virtual data, virtual applications and virtual networks. This layer decouples the process of allocating applications to resources from that of managing jobs across administrative domains, physical machines and local software configurations. In other words, jobs are mapped to virtual resources (e.g. RedHat Linux 7.0 x86 virtual machines), and only virtual resources get managed across domains and physical environments (e.g. WinXP and Win2000 physical systems at different locations).

In the second layer, grid applications are instantiated as services which can be connected as needed to create virtual information grids. This layer decouples the process of using and composing services from that of managing the execution of the underlying grid applications. Different grid-computing mechanisms (e.g. Globus, Condor-G, .NET and JXTA) can be used to run applications but their implementation details are largely hidden once the grid-enabled applications are encapsulated and composed as services [8] (e.g. using OGSI, OGSI.NET and Jini).

In the third layer, aggregated services (possibly presented to users via portals) export interfaces that are virtualized in order to enable displaying by different access devices. This layer decouples the process of generating interfaces of services (e.g. XML and UIML) from the process of rendering them on specific devices (e.g. HTML for laptop, WHML for a palmtop and WAP WML for a cell phone).

The currently deployed In-VIGO system[1] implements the first layer of virtualization (with exception of virtual networks)—users can develop applications or use one of several grid-enabled tools with interactive as well as batch-oriented interfaces. The authors are currently developing research prototypes for the other virtualization layers. The focus of this paper is on the virtual computing grid layer (i.e. the first virtualization layer discussed above) as it is implemented in the current In-VIGO implementation.

3. Virtualization in In-VIGO

The In-VIGO architecture is built upon components that allow for the virtualization of grid resources and user interfaces. A distributed virtual file system facilitates data transfer across grid resources; virtual machines provide isolation, resource integrity, legacy software support, environment encapsulation and customization; virtual applications allow modification and extension of individual tool behavior as well as the use of a collection of tools as a single unified application; virtual networks allow the configuration of a network that meets the requirements of a specific grid application or user; and virtual interfaces allow applications to conform to changing environments, allowing universal access to a long-lived application session.

3.1. Virtual data and the virtual file system

Virtualization techniques can be applied to facilitate the transfer of data across grid resources. The grid distributed virtual file system forms the basic framework for the transfer of data necessary to In-VIGO applications. It relies on a virtualization layer built on existing Network File System (NFS) components,

[1] Accessible at http://invigo.acis.ufl.edu. Courtesy accounts are available.

and is implemented at the level of Remote Procedure Calls (RPC) by means of middleware-controlled file system proxies [9]. A virtual file system proxy intercepts RPC calls from an NFS client and forwards them to an NFS server, possibly modifying arguments and returning values in the process. Through the use of proxies, the virtual file system currently supports multiplexing, manifolding and polymorphism by (1) sharing of a single file account across multiple users, (2) allowing multiple per-user virtual file system sessions in a single server, and (3) mapping of user and group identities to allow for cross-domain NFS authentication [10]. Proxy-level transformations can also be used to support data virtualization, enabling polymorphism and manifolding of file contents; although this capability is not currently exploited in In-VIGO, it is the subject of ongoing research. Potential applications of data content virtualization include translation of file formats, language translation, summarization and reduction of data, or other data transformations invoked at data-access time (Fig. 3).

3.2. Virtual machines

A "classic" virtual machine (VM) is an efficient, isolated duplicate of the underlying physical machine [4]. Unlike virtual machines associated with a particular programming language (e.g. the Java VM), the layer of indirection in which "classic" VMs operate is that of the instruction set architecture (ISA) of the physical machine. A virtual machine monitor (VMM) is responsible for providing the perception of multiple, isolated machines that share a single physical resource by intercepting and emulating the behavior of privileged ISA instructions. Classic VMs support manifolding (multiple VMs per physical resource), multiplexing (time-sharing of CPU, space-sharing of memory and disk among several VMs), and polymorphism (with respect to virtual hardware configuration, operating systems and applications).

The virtualization properties of classic VMs are attractive in a grid-computing environment for several reasons, including user isolation, resource integrity, legacy software support, environment encapsulation and customization [11]. The classic virtual machine monitors implemented by VMware [12] and IBM zVM are currently used in In-VIGO.

3.3. Virtual applications

A virtual application can appear differently from the actual application or applications being used, and can be replicated so that multiple executions can be run simultaneously over the same physical application. The implementation of virtual applications shares analogies with the approaches used to implement both virtual data and virtual machines. Like virtual data, virtual applications use a virtual file system to support manifolding and multiplexing of application codes. Like virtual machines, virtual applications use a monitor to modify the semantics and capabilities presented to application users, thus supporting polymorphism. In addition, virtual applications can consist of multiple applications, either as multiple

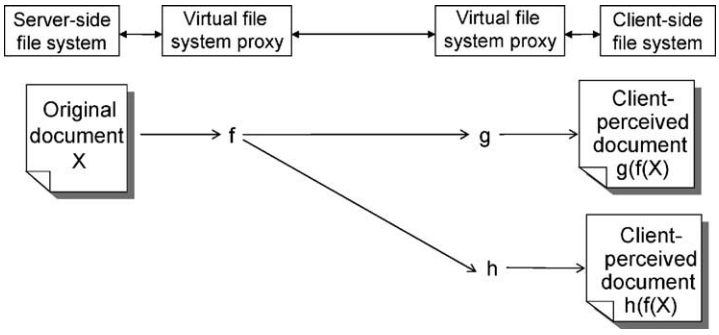


Fig. 3. Proxy transformations can be used to support data virtualization, enabling polymorphism and manifolding of file contents.

instances of the same application or as compositions of distinct applications. Since different applications encapsulated in a virtual application may require distinct hardware and software resources, the virtual application monitor must interact with grid middleware in order to locate, reserve, allocate and aggregate all those resources. In the context of In-VIGO, virtual applications are exposed to the users; the term "tools" is used to refer to the actual applications on which virtual applications are based.

3.4. Virtual networks

In the context of In-VIGO, virtual networks are logical networks that are decoupled from physical (and other virtual) networks, per user and per application. The goal is to enable multiple virtual applications to share a physical network, each having the perception that it uses a private, dedicated network whose functionality and quality of service match the application needs and user access rights. Partial implementations of these virtual networks are already possible in the form of virtual private networks [5] and tunneling [13]. Extensive ongoing research on overlay networks [14–16] is developing technologies that offer promise of enabling not only private and secure communication but also necessary quality of service and customized functionality. Quoting [14], an overlay network is "a configuration within which a base network is used to support some second network, layered upon the underlying infrastructure". According to the same reference, each virtual overlay network has a global unique identifier, some set of access points and a set of strong guarantees. Overlay networks can provide a basis for multiplexing, manifolding and polymorphism as required by virtual networks. However, further research is needed to understand what kinds of functionality to support, and how to efficiently build dynamic instances of such overlay networks. There are other published definitions of virtual networks that are not being considered for implementation in In-VIGO but are worth mentioning. They include networks built upon a physical network and partitioned for use by individuals; logically and/or physically allocated subsets of network resources; collections of logical subsets of physical network resources; and simulations of one or more networks on top of a physical network [17–19].

3.5. Virtual user interfaces

The notion of virtual user interfaces is a particular case of data virtualization. It leverages ongoing efforts to universalize user interfaces [20–22] where applications use an abstract representation of an interface to communicate with a device-independent server that renders the user interface to conform to the abilities of the device. However, unlike HTML, XML or UIML, virtualization of a user interface also allows the appearance of the application to differ according to the context on which an application is deployed (polymorphism) and the access to the application using a variety of devices and device types in a single session (manifolding and multiplexing). An example is an application session where a user starts the application on a workstation, later queries the state of the application using a PDA or a cellular phone and finally retrieves the application results using a laptop or a public Internet machine.

4. The architecture of In-VIGO

Given that the objective of In-VIGO is to support computational tools for engineering and science research, the component that is most evident is the Virtual Application (VAP). The In-VIGO VAP is an aggregation of actual tools into a logical application session that allows users to perform session operations such as setting up execution parameters, importing data files, executing tools and retrieving experiment results. As shown in Fig. 4, users do not interact directly with the VAP, but rather by means of a separate component called the User Interface Manager (UIM). The UIM has two primary objectives: provide the virtual application with mechanisms to interact with the user, and provide the user with a virtual interface that offers a disconnected session that can be accessed using a variety of devices.

Virtual applications, and hence their designers, do not concern themselves with the specifics of resource management, reservation or allocation. A separate resource management component provides the APIs and services needed to allocate and access all the resources needed to execute the desired tools with the desired quality of service. Similarly, virtual applications and the UIM are not concerned with user infor-

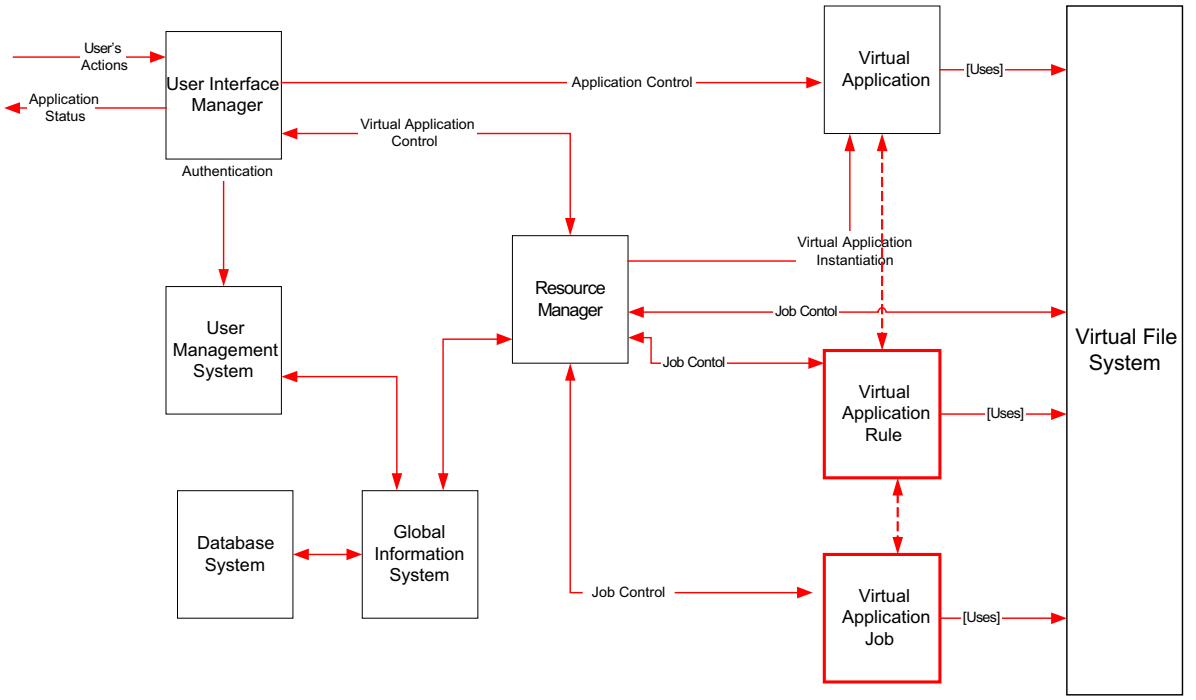


Fig. 4. High-level architecture of the In-VIGO system. The primary components shown in this figure are the virtual application, the virtual file system, the resource manager, the user interface manager, the global Information System, and the user manager.

mation, privileges, authentication or session management. These are managed by the User Management System. Finally, the Resource Manager and User Management System components use the facilities of the Global Information System, which maintains the necessary information in a relational Database System.

Virtual applications access user data through a grid-wide virtual file system that is set up and controlled by the Resource Manager, on behalf of a user. Through this mechanism, applications are presented with the interface of a conventional NFS-based distributed file system—no application-level modifications are required to access user data.

4.1. The virtual application

The virtual application in In-VIGO interacts with the User Interface Manager using a special disconnected secure XML-based protocol called Virtual Application Control Protocol. Users do not interact directly with the Virtual Application, but rather with the User Interface Manager. The Virtual Application is reactive in that it never attempts to contact the UIM (and, hence, the user) unless it is contacted first. It uses a sequence number and a shared password to communicate with the UIM using a network connection that is closed and discarded once the message is processed. Hence, a session in In-VIGO is disconnected, requiring that every action/reply be processed using a different network connection. Disconnected sessions have several desirable properties:

1. Sessions can be long-lived and, so long as the virtual application does not crash, can survive network or server outages. Sessions even survive a crash on the User Interface Manager, since it stores the session required state on permanent storage.
2. Virtual Application sessions are browser-independent and hence it is possible for a user to interface with a session from different devices. For example, a user could initiate a transaction on a desktop computer, monitor the transaction status on a PDA or a cell phone, and finally recover the results using

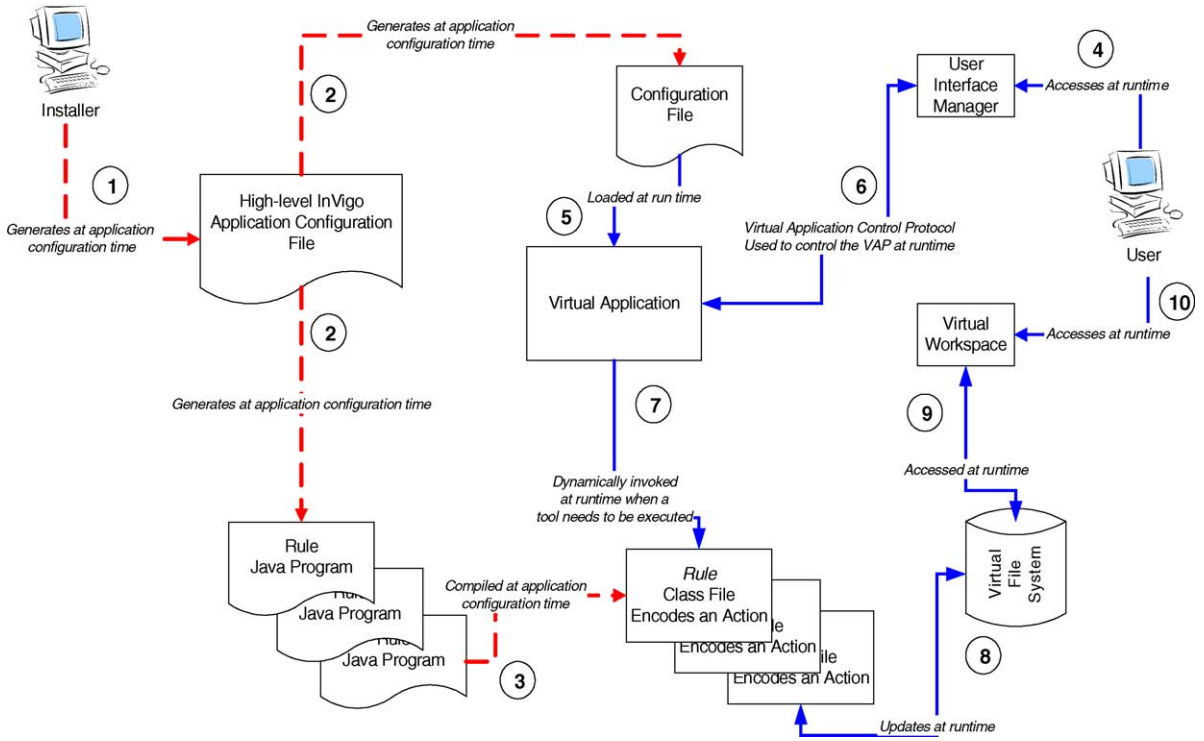


Fig. 5. The architecture of the virtual application. Dotted red arrows represent actions performed at application installation time. Solid arrows indicate actions performed by the system at run-time. The numbers represent the sequence of events from the installation of an application, to the execution of a tool, to the retrieval of execution results.

a laptop or from a public computer of an Internet Cafe.

There is a Virtual Application instance for every tool, and for every tool session. As shown in Fig. 5, the actual behavior of the application is determined at runtime by an application configuration file that describes how the application interacts with the user, and a set of Java classes that implement the actual application behavior (e.g. resource allocation and process execution). Hence, the installation of an application in In-VIGO entails: (1) the installation of a tool in grid-enabled resources, (2) the creation of an XML application configuration file, and (3) the creation of a series of Java classes to control the application behavior. The latter are referred to as "rules" in In-VIGO. The process of creating the configuration file and Java-based rules can be automated, especially for applications with a native graphical user interface (which is presented to the user through a VNC-based virtual display) as well as batch applications that expect command-line arguments.

In-VIGO rules represent the control and resource view of the virtual application in In-VIGO. That is, they specify the resource requirements (operating system, architecture), prerequisites of the execution (directory structure, files) and the execution logic corresponding to the user's input. Using the Resource Manager, rules find the proper resources for a job request and execute it using grid protocols.

The Virtual File System, as discussed later, provides users, virtual applications and rules a unified and grid-wide file system. Rules cannot execute on a file system that can potentially change in the middle of a tool execution, or the execution of a collection of tools that are perceived as a single action by the user, and hence the Resource Manager provides a running rule with a *file fork*, a copy of the directory where the tools will be run, and that is presented to the user once the user-perceived action is finished.

This model has been designed to support the seamless implementation of parameter sweeps in In-VIGO. Virtual application executions that support parametric ranges are referred to as *ranging applications*. Support for ranging relies on the Virtual Application to generate all the values in the range, unfold the values into the parameter and executes the rules as many times as the number of values in the range. Hence, multiple executions are performed over the same data files but with different parameters, and grid-wide resources can be exploited concurrently. Note that the tools themselves are not aware that parameter sweeps are being performed and hence it is possible to range parallel executions for tools that traditionally could only be run sequentially.

4.2. The virtual file system

A key challenge that must be addressed by grid middleware is the provisioning of data to applications, in the absence of a centralized administrative domain. It is desirable that such a data provisioning mechanism be fast, secure, scalable and interoperable with protocols and application programming interfaces (APIs) that are supported by typical grid nodes.

Data management in In-VIGO is provided by a virtualization layer built on top of NFS. The resulting grid virtual file system allows dynamic creation and destruction of file system sessions on a per-user or per-application basis. Such sessions allow on-demand data transfers, and present to users and applications the API of a widely used distributed network file system across nodes of a computational grid. The virtualization layer is implemented via proxies that intercept, modify and forward NFS calls, without requiring modifications to existing clients and servers [9].

The dynamic setup of an In-VIGO virtual file system (VFS) session involves the execution of a sequence of tasks under coordination of the Resource Manager. A virtual file system session is established for a logical user account that is dynamically allocated on behalf of a user, allowing a client-side “shadow” account to access data from a server-side “file” account through an NFS-mounted share [10]. The session setup begins with the execution of server-side proxies for the NFS and MOUNT protocol via a grid-aware job submission mechanism (e.g. SSH or Globus). Once a virtual file system proxy is running, the dynamic session is established by a client-side mount operation, also initiated by the resource manager via a job submission mechanism. Logically, the In-VIGO grid middleware acts as a system administrator to manage dynamic client–server data sessions on a per-user basis.

4.3. The resource manager

Grid resources—CPUs, memory, disks, network, computers, clusters, grids—which may be distributed geographically and across administrative domains, are managed by a wide variety of resource management systems. Examples are: operating systems (e.g. Linux with its process scheduler and file systems) for single-CPU/SMP machines, batch/job/queue management systems (e.g. PBS, Sun Grid Engine, Condor) and distributed file systems (e.g. NFS, AFS, PVFS) for clusters of machines, job management systems for peer-to-peer clusters (e.g. Enfuzion) and grid resource managers (e.g. Globus GRAM, Storage Resource Management for grid data). In-VIGO interacts with these resource managers for two purposes: (a) for running In-VIGO middleware jobs to create virtual resources, and (b) for running In-VIGO user jobs on physical and virtual resources. The core services provided by these resource managers are very similar, however their programming interfaces differ. The resource manager in In-VIGO provides a uniform and simple API to access all these resources.

The resource management functionality in In-VIGO is implemented by providing support for the following phases of running a job:

1. Determine the resource specification for the job(s) based on user/application authorization, application requirements and explicit resource specifications.
2. Select/create/reserve a resource to run the job, based on the current available resources.
3. Run the job, i.e. run any pre-job tasks (e.g. staging in files); submit job; monitor progress of job; find out that the job is done; run any post-job tasks (e.g. staging out files); and release resources.

The resource specification and selection API is based on the Globus RSL language, to specify job requirements, and the Condor ClassAd language, to

describe resources and their constraints. The current implementation of this API supports few resource types and very simple selection criteria (round-robin selection of unused resources). The job management APIs provided by different distributed resource managers (e.g. Globus) are encapsulated to conform to an abstract job submission and control API. This enables uniform management of jobs using simple API calling sequences, independent of the type of resource, and the type of job, i.e. instantiated by the In-VIGO user or the In-VIGO middleware.

4.4. The user interface manager

The front-end for In-VIGO, or User Interface Manager, is a web layer that uses Java Servlets to process requests and construct responses. The User Interface Manager captures the actions of the end user and sends appropriate requests to the Virtual Application Manager using the VACP protocol. The UIM initiates a request in response of a user action and the VAP responds with an appropriate reply in XML form. This reply message is then parsed and converted into different markup representations depending upon the end device. Presently we only support W3C HTML 4.01 compliant devices.

In-VIGO requires the User Interface Manager to easily manage dynamic content. An In-VIGO session requires the web layer to ensure the persistence of the content of the pages to be passed between different components. The MVC [23] (model–view–controller) design pattern is used to provide flexibility for the design. To provide a higher layer of abstraction for managing a large number of concurrent requests the content of the user screen is represented as components: the user screen is logically divided into different parts, which are represented as Java components at the UIM layer. The various components persist the view information for a specific client. This includes the location of the component on the screen, its content and logic to change its view for different requests. All the logical representations are handled by a single dispatcher servlet which acts as the controller. The actual content is generated at the VAP which acts as the model. By dividing the functionality we decouple the application logic, content presentation and user interaction. This model facilitates the dynamic generation of content and provides flexibility in terms of easy transcoding, caching of content and rendering the view.

Fig. 6 shows a sample In-VIGO screen rendered on a Windows Internet Explorer workstation. The screen shows a step of the execution of the Dinero application, where execution parameters are being defined by the user. The labels highlight the screen components, each representing a different aspect of the application execution.

4.5. The global information system

Similar to the Grid Information Service (GIS) [24], the goal of the In-VIGO Information System is to provide a uniform interface (framework) to allow information flow between various In-VIGO components, hiding the complexity of dealing with a variety of information access interfaces. When a user interacts with the In-VIGO web interface (e.g. starts a job), a variety of events take place in the grid through different layers/modules/services of In-VIGO. Each of these components requires access to different sets of information. The Information System in In-VIGO maintains the information about entities that participate in a computational grid: users, access control lists, computational resources (e.g. hosts, shadow accounts, file accounts, virtual machines) and services (e.g. user sessions, VFS sessions, applications).

The current In-VIGO Information System is based on the centralized relational database server MySQL to store and manage the necessary information but it can be extended to support other types of information providers (e.g. Lightweight Directory Access Protocol server, Condor ClassAd Catalog server), as shown in Fig. 7. The relational data model approach is attractive to In-VIGO because the majority of the information is updated on-demand or on-failure, and requires ACID (atomicity, consistency, isolation, durability) properties.

4.6. The user manager

The In-VIGO user management layer deals with user authentication and access control. The information to authenticate a user can be stored in many ways: Unix system files (/etc/passwd etc.), an external LDAP database, an external SQL database or other data storage. The API for each authentication

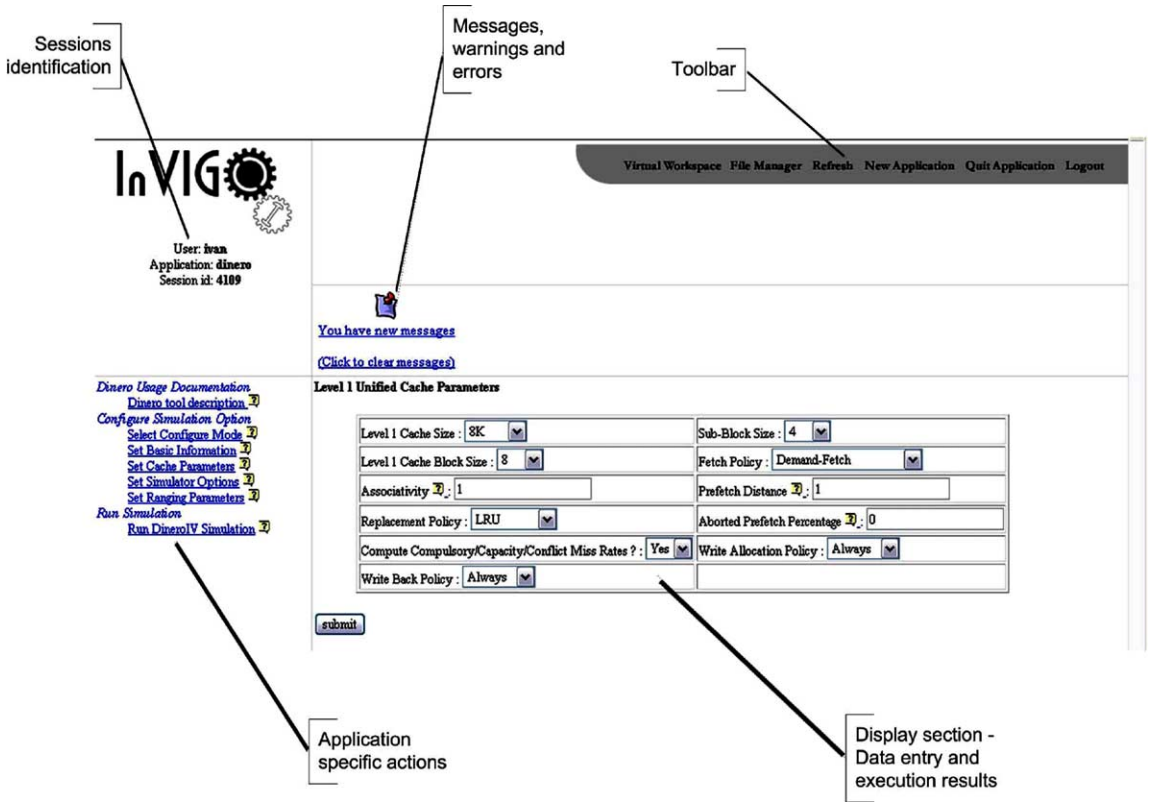


Fig. 6. Sample In-VIGO screen showing the execution of the Dinero simulator and the screen components.

mechanism is different in each case, however the information needed is usually a user/password pair. The User Management layer hides the variety of authentication mechanisms, providing a uniform and simple API to perform authentication for all In-VIGO components. The access control is achieved by using a powerful and flexible mechanism in which applications are able to specify access constraints by assigning action rules to different permission groups, while the user is able to constraint his capability by using different user classes.

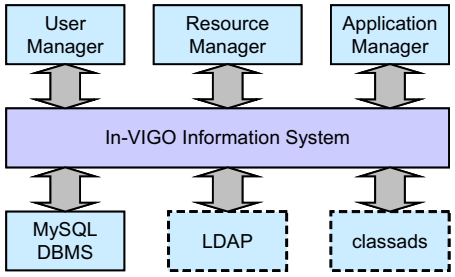


Fig. 7. The In-VIGO information system is based on a centralized MySQL database server but can be extended to support any information provider.

If the services are provided through a stateless protocol such as the HTTP, the layer on the top is required to be stateful to keep track of user actions. The methods for user session tracking commonly used when working with HTTP are: basic authentication, cookies, and URL rewriting. Although the User Manager (UM) provides an interface that supports all of these methods, the last one is used in In-VIGO to avoid cookies or the transmission of the user password in each interaction. Another desired feature of web services provider is the capability of maintaining multiple states for each user. In In-VIGO this is achieved by decoupling the browser identification and the session

identification, which is maintained in the Information System.

5. Implementation

In-VIGO was implemented at the ACIS Laboratory and is at the time of this writing in its first beta version release. It was implemented using Java and Java Servlets for the core of the system, Perl for operating system-specific operations and C for the Virtual File System. In-VIGO leverages on existing standards and open source technologies wherever possible, and in particular uses XML for all configuration files and protocols, Condor ClassAds for resource description, Globus and SSH for resource invocation and SQL servers as data repositories.

In-VIGO has been configured to run a variety of scientific and test applications, including:

- Dinero (a cache simulator for memory reference traces supporting multi-level caches and dissimilar I and D caches).
- CNT-IV (an application that calculates the *C–V* and *I–V* characteristics of ideal, ballistic carbon nanotube field-effect transistors).
- CoWord (a CoWord analysis tool), DLX View (an interactive pipelined computer simulator using the DLX instruction set).
- GAMESS (the General Atomic and Molecular Electronic Structure System—a general ab initio quantum chemistry package).
- LSS (Light Scattering Spectroscopy—used to estimate the diameter, diameter deviation and refractive index of pixels in an image).
- MolCToy (a collection of simple theoretical models of the conduction through individual molecules between two contacts).
- Nanomos (a self-consistent 2D-simulator for thin body, fully depleted, double gated, n-MOSFETs).
- Simple Scalar (a set of fast, flexible and accurate uni-processor simulators based on a close derivative of the MIPS architecture).
- CACTI (an integrated cache access time, cycle time, area, aspect ratio, and power model).
- Huckel-IV (a simple quantitative method which calculates current–voltage characteristics and conductance spectrum of a molecule sandwiched between two metallic contacts one of which could be a scanning probe).
- XSpim (a software simulator that runs programs written for MIPS R2000 and R3000 processors).
- XGobi (a package for plotting scatterplots of multi-dimensional tabular data).
- Fract-O-Rama (a fractal generation program).
- GNU Chess application using the X-Board graphical user interface. Fig. 6 shows a typical screenshot of In-VIGO executing the Dinero Cache Simulator.

All users have a file system that is shared by all applications and that can be accessed by using a file manager or a Unix prompt on a virtual machine that provides users with an isolated and secure virtual operating environment. The Unix virtual machine is called a Virtual Workspace and is accessible through the toolbar on all applications. This Virtual Workspace is a VMware virtual machine with a minimal configuration that mounts the user's file system using the Virtual File System.

In its current deployment version, In-VIGO SSH is used for executing jobs that are latency sensitive and Globus is used for long-running jobs, yet users are unaware of the grid mechanisms used and their jobs can execute on any machine available in the current pool: a 40-node Linux cluster, a Z800 IBM server and 4 dual-processor Linux servers.

6. Conclusions

The In-VIGO grid-computing system described is designed to support computational tools for engineering and science research In Virtual Information Grid Organizations. A novel aspect of this system is the extensive use of virtualization technology, emerging standards for grid-computing and other Internet middleware to enable the creation of dynamic pools of virtual resources that can be aggregated on-demand for application-specific computational grids. Building grids out of physical resources to constructing virtual grids has many advantages but also requires new thinking on how to architect, manage and optimize the necessary middleware. In-VIGO represents a first step towards this goal and our experiences have shown that grid-computing middleware is still

lacking support for robust and fault tolerant distributed computing for interactive applications, and that virtualization can compensate for many of these limitations.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grants No. EIA-9975275, EIA-0224442, ACI-0219925, EEC-0228390 and NSF Middleware Initiative (NMI) collaborative grants ANI-0301108/ANI-0222828, and by the Army Research Office Defense University Research Initiative in Nanotechnology. The authors also acknowledge two SUR grants from IBM and a gift from VMware Corporation. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation, Army Research Office, IBM, or VMware.

References

[1] I. Foster, C. Kesselman, S. Tuecke, The anatomy of the grid: enabling scalable virtual organizations, Int. J. Supercomput. Appl. 15 (3) (2001) 200–222.

[2] K. Krauter, R. Buyya, M. Maheswaran, A taxonomy and survey of grid resource management systems for distributed computing, Softw. Pract. Exp. 32 (2) (2002) 135–164.

[3] J. Fotheringham, Dynamic storage allocation in the Atlas including an automatic use of a backing store, Commun. ACM 4 (10) (1961) 435–436.

[4] R.P. Goldberg, Survey of virtual machine research, IEEE Comput. Mag. 7 (6) (1974) 34–45.

[5] B. Gleeson, A. Lin, J. Heinanen, G. Armitage, A. Malis, A Framework for IP-based Virtual Private Networks, RFC 2764, Internet Engineering Task Force, February 2000.

[6] W.E. Johnston, D. Gannon, B. Nitzberg, L.A. Tanner, B. Thigpen, A. Woo, Computing and data grids for science and engineering, in: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing (CDROM), no. 52, 2000.

[7] N.H. Kapadia, J.A.B. Fortes, PUNCH: an architecture for web-enabled wide-area network-computing special issue on high performance distributed computing of cluster computing, J. Netw. Softw. Tools Appl. 2 (2) (1999) 153–164 (special issue).

[8] I. Foster, C. Kesselman, J.M. Nick, S. Tuecke, The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, Technical Report, Open Grid Service Infrastructure WG, Global Grid Forum, June 2002.

[9] R. Figueiredo, N.H. Kapadia, J.A.B. Fortes, The PUNCH virtual file system: seamless access to decentralized storage in a computational grid, in: Proceedings of the 10th IEEE International Symposium on High Performance Computing, August 2001.

[10] N.H. Kapadia, R. Figueiredo, J.A.B. Fortes, Enhancing the scalability and usability of computational grids via logical user accounts and virtual file systems, in: Proceedings of the Heterogeneous Computing Workshop (HCW) at the International Parallel and Distributed Processing Symposium (IPDPS), April 2001.

[11] R. Figueiredo, P.A. Dinda, J.A.B. Fortes, A case for grid computing on virtual machines, in: Proceedings of International Conference on Distributed Computing Systems (ICDCS), May 2003.

[12] J. Sugerman, G. Venkitachalam, B.-H. Lim, Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor, in: Proceedings of the USENIX Annual Technical Conference, USENIX Association, June 2001, pp. 1–14.

[13] R. Thayer, IP Security Document Roadmap, Internet RFC 2411, November 1998.

[14] K.P. Birman, Technology challenges for virtual overlay networks, IEEE Trans. Syst. Man Cybern. Part A. Syst. Hum. 31 (4) (2001) 319–326.

[15] D. Andersen, H. Balakrishnan, F. Kaashoek, R. Morris, Resilient overlay networks, in: Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP), October 2001, Banff, Canada.

[16] J. Touch, Y. Wang, L. Eggert, G. Finn, Virtual Internet Architecture, Future Developments of Network Architectures (FDNA) at Sigcomm, August 2003. Available as ISI-TR-2003-570.

[17] Z. Dziong, Y. Xiong, L. Mason, Virtual network concept and its applications for resource management in ATM based networks, in: Proceedings of IFIP/IEEE International Conference on Broadband Communications'96, April 1996, pp. 223–234.

[18] G. Woodruff, N. Perinpanathan, F. Chang, P. Appanna, A. Leon-Garcia, ATM Network Resources Management using Layer and Virtual Network Concepts, IM'97, May 1997.

[19] K.H. Randall, Virtual networks: implementation and analysis, Masters Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1994.

[20] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, Extensible Markup Language (XML) 1.0, 2nd ed., W3C Recommendation, 6 October 2000.

[21] D. Raggett, A. Le Hors, I. Jacobs, HTML 4.0 Specification. http://www.w3.org/TR/1998/REC-html40-19980424.

[22] M. Abrams, J. Helms, User Interface Markup Language (UIML) Specification, Language Version 3.0, February 2002. http://www.uiml.org/specs/docs/uiml30-revised-02-12-02.pdf.

[23] Sun Microsystems, Inc., Java BluePrints Model-View-Controller. http://java.sun.com/blueprints/patterns/MVC-detailed.html.

[24] Grid Forum, Grid Forum web site: http://www.gridforum.org.

Sumalatha Adabala is a PhD candidate in Electrical and Computer Engineering at Purdue University. She received her BE from Bangalore University and MSEE from Purdue University. Her research interests include computer architecture, virtual machines, operating systems and distributed systems.

Vineet Chadha is a graduate student in the Advance Computing and Information System Laboratory of the University of Florida. His research interests include distributed file systems, data caching, computer architecture and open source software. He has a Master's degree in computer science from Mississippi State University with software engineering as specialization. He is a member of the honour society of Computer Science Upsilon Pi Epsilon. He loves Linux kernel hacking and playing around with Linux utilities. His hobbies include reading magazines, Indian classical music and an evening walk.

Puneet Chawla is a graduate student in the Advanced Computing and Informations Systems Laboratory at University of Florida. His research interests are in the areas of distributed and enterprise computing. Chawla received his Bachelors of Engineering Degree from Punjab Engineering College, India. He is also a Sun Certified Java Programmer and Developer.

Renato Figueiredo received his PhD degree in Computer Engineering from Purdue University in 2001. He is currently an Assistant Professor in the Department of Electrical and Computer Engineering at the University of Florida. Prior to joining the University of Florida he was an assistant professor at Northwestern University. His research interests include network-centric grid computing systems, virtual machines, distributed file systems and high-performance computer architectures.

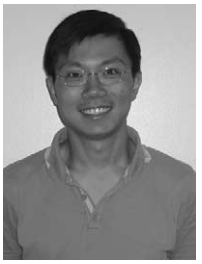
José Fortes received his BS degree in Electrical Engineering from the Universidade de Angola in 1978, his MS degree in Electrical Engineering from the Colorado State University, Fort Collins in 1981 and his PhD degree in Electrical Engineering from the University of Southern California, Los Angeles in 1984. From 1984 to 2001 he was in the faculty of the School of Electrical Engineering of Purdue University at West Lafayette, Indiana. In 2001, he joined both the Department of Electrical and Computer Engineering and the Department of Computer and Information Science and Engineering of the University of Florida as Professor and BellSouth Eminent Scholar. His research interests are in the areas of parallel processing, computer architecture, network-computing and fault-tolerant computing. José Fortes is a Fellow of the Institute of Electrical and Electronics Engineers (IEEE) professional society. He was a Distinguished Visitor of the IEEE Computer Society from 1991 to 1995.

Ivan Krsul was born in Bolivia, South America. He received his Bachelor of Science in Computer Engineering degree from the Catholic University of America in May 1989 and his MS and PhD degrees in Computer Science from Purdue University in May 1994 and May 1998, respectively. He is currently a Visiting Assistant Professor at the Advanced Computing and Information Systems (ACIS) Laboratory, University of Florida department of Electrical & Computer Engineering, having returned to academia from a 4-year incursion into the implementation of digital government solutions for the Government of Bolivia. Dr. Krsul is interested in system security, distributed systems and software engineering solutions for dynamic and complex organizations.

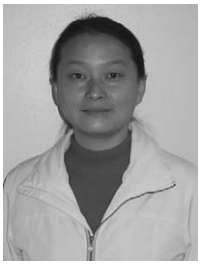
Andrea Matsunaga received her BS degree in Electrical Engineering from the Polytechnic School of the University of Sao Paulo (Brazil), in 1998, her MS degree in Electrical Engineering from the Electronic Systems Department (PSI) of the Polytechnic School of the University of São Paulo (Brazil), in 2001; and is currently a graduate student at the Department of Electrical and Computer Engineering of the University of Florida, working with Dr. Fortes. Her research interests are in the areas of distributed computing, information management, computer architecture and dsm systems.

Mauricio Tsugawa received his BS degree in Electrical Engineering from the Polytechnic School of the University of Sao Paulo (Brazil), in 1998, his MS degree in Electrical Engineering from the Electronic Systems Department (PSI) of the Polytechnic School of the University of São Paulo (Brazil), in 2001; and is currently a graduate student at the Department of Electrical and Computer Engineering of the University of Florida, working with Dr. Jose Fortes. His research interests are in the areas of distributed computing, parallel processing, computer architecture and configurable logic design.

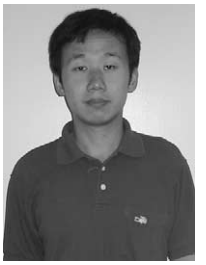
Jian Zhang is currently a PhD student in the ACIS laboratory at the University of Florida. She received her MS degree in Electrical and Computer Engineering from the University of Florida in 2001.



Ming Zhao is a PhD Research Assistant at the ACIS Laboratory, University of Florida. He has received both BS and MS degrees from Tsinghua University, China, in 1999 and 2001, respectively.



Liping Zhu received her BS degree in 1993 from Hubei Automotive Industrial Institute, and her MS degree in 1998 from Beijing University of Posts and Telecommunications, China. She has previously worked in Beijing Telecom R&D Center and is currently a PhD student in the Department of Electrical and Computer Engineering, University of Florida. Her research interests include resource management and virtual applications in grid computing.



Xiaomin Zhu was born in China in 1975. He received his BS degree from Department of Telecommunications at Beijing University of Posts and Telecommunications (China) in July 1997. He was a system engineer in Nortel Networks (China) Ltd. Company and Ericsson (China) Ltd. Company. His working area was 3G wireless communication and Telecommunication Network Management System. Currently, he is a graduate student of Department of Electrical and Computer Engineering at University of Florida and working as a research assistant in the Advanced Computing and Information Systems Laboratory. He is one of the researchers of In-VIGO system and participates in the development of Virtual Application subsystem of In-VIGO. His research interests are in the areas of Grid Computing, Virtualization and Networking.