On the Use of Virtualization and Service Technologies to Enable Grid-Computing

Andréa Matsunaga, Maurício Tsugawa, Ming Zhao, Liping Zhu, Vivekananthan Sanjeepan, Sumalatha Adabala, Renato Figueiredo, Herman Lam, and José A. B. Fortes

Advanced Computing and Information Systems Laboratory (ACIS)
Dep. of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611
Email: fortes@ufl.edu

Abstract. The In-VIGO approach to Grid-computing relies on the dynamic establishment of virtual grids on which application services are instantiated. In-VIGO was conceived to enable computational science to take place In Virtual Information Grid Organizations. Having its first version deployed on July of 2003, In-VIGO middleware is currently used by scientists from various disciplines, a noteworthy example being the computational nanoelectronics research community (http://www.nanohub.org). All components of an In-VIGO-generated virtual grid – machines, networks, applications and data – are themselves virtual and services are provided for their dynamic creation. This article reviews the In-VIGO approach to Grid-computing and overviews the associated middleware techniques and architectures for virtualizing Grid components, using services for creation of virtual grids and automatically Grid-enabling unmodified applications. The In-VIGO approach to the implementation of virtual networks and virtual application services are discussed as examples of Grid-motivated approaches to resource virtualization and Web-service creation.

1 Introduction

The future envisioned by the concept of Grid-computing is one where users will be able to securely and dependably access, use, "publish" and compose applications as services anywhere and anytime. Transparently to users, Grids will have to aggregate resources, possibly across different institutions, to provide application services. In addition, Grid middleware will have to create in the aggregated resources the execution environments where services and users can securely run or create applications of interest and access needed data. Unless properly designed, individual solutions for each of these requirements can conflict with each other, as shared resources cannot be easily reconfigured to simultaneously provide multiple execution environments securely and on-demand for different users and applications. This article argues that resource virtualization and service technologies provide ideal mechanisms to address these and other key requirements of Grid-computing, and

describes components of In-VIGO, an evolving deployed system that successfully uses this approach [1], [2].

The remainder of this paper is organized as follows. The In-VIGO approach is briefly reviewed in Section 2. Virtual machines and the corresponding services for their creation and management are reviewed in Section 3. Virtual file systems and associated services are overviewed in Section 4. Virtual networking techniques are presented in Section 5. Virtual applications and virtual application services are discussed in Section 6. Section 7 describes how the different In-VIGO components are securely integrated. Conclusions and the current status of In-VIGO middleware and research are presented in Section 8.

2 The In-VIGO Approach

In-VIGO is unique in that it decouples user environments from physical resources by using technologies that virtualize all resources needed for Grid-computing, including

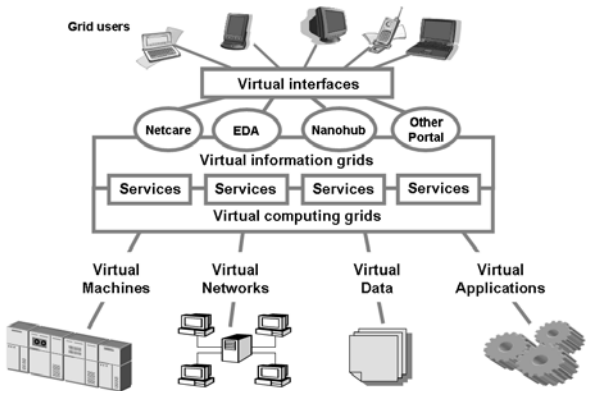

Fig. 1. High-level view of the In-VIGO approach

machines, networks, applications and data (see Figure 1). Users will typically interact with In-VIGO through a portal where they can invoke applications of interest. In-VIGO delivers these applications through Web-enabled user interfaces that interact with virtual application (VA) services. VA services interact with other application services as well as other Grid-computing middleware services. VA services decouple application interfaces from application implementations thus hiding the kinds of codes and machines used to provide services. Transparently to users, VA services engage with virtualization services to create the virtual machines, file systems, networks and possibly other applications needed to generate a virtual grid with the necessary execution environment for the application delivered by the VA service.

Virtualization services decouple users and execution environments needed by applications from the physical machines that provide them, thus allowing different instances of an application service to transparently run on different physical hardware.

Ultimately, Grids will be useful only if they can provide application services for users. In-VIGO provides each user with a persistent private virtual workspace that enables him/her to both launch and develop applications, use and manage private data, and carry out conventional operating system tasks through, for example, a Unix-like shell. It is also very important that, in addition to the use of services, the process of deploying applications as services be as simple as possible. Service creation should not require application developers to know details of how Grid middleware works, and should not require the involvement of administrators. In-VIGO provides automated procedures to create application services that only require developers to provide a description of how a tool works. This description is comparable in nature and complexity to the "man pages" of an operating system command. It includes the command-line grammar and some additional information on software dependencies and other requirements of the application's execution environment.

3 Virtual Machines and Virtual Machine Services

In-VIGO supports dynamic allocation of execution environments per user and per distinct application by using virtual machine technologies (including language-based Java VMs, as well as O/S-based VMs, such as VMware, User-mode Linux) and/or "shadow" accounts. For efficiency and scalability purposes, mechanisms are provided for multiplexing virtual machines and accounts among users and applications without compromising security and customizability. Virtual machines can either be created and destroyed for every In-VIGO session or be made persistent across sessions. Virtual machines used to run applications can also be shared across several applications by using "shadow accounts" [3], which are pre-created accounts on machines that In-VIGO can use on behalf of arbitrary users.

In-VIGO manages virtual machines through a set of Grid service-based middleware components – VMShop and VMPlant [4]. The key differentiators of this approach from related work reflect the design decisions of: (1) supporting different VM technologies, such as VMware, User-Mode Linux; (2) allowing flexible, application-centric VM environment configurations using direct acyclic graph (DAG) representations; and (3) supporting dynamic "cloning" of previously-built VM images. Virtual machines managed using this middleware are highly customizable by the client, on a per-application basis. In contrast, dynamic virtual environments [5] enable the creation of VMs from a master disk (e.g. a Linux distribution with pre-installed Globus software) but do not provide mechanisms for the client to specify the desired machine's configuration.

"Classic" VMs present the image of a dedicated operating system while enabling multiple O/S configurations – completely isolated from each other – to share a single machine. This is an effective mechanism for resource consolidation, and a key reason for the renewed interest and popularity of VMs. They also provide a flexible, powerful execution environment for Grid computing, offering isolation and security

mechanisms complementary to operating systems, customization and encapsulation of entire application environments, and support for legacy applications [6], addressing a fundamental goal of Grid computing – flexible resource sharing.

VMShop provides a single logical point of contact for clients to request three core services: create a VM instance, query information about an active VM instance, and destroy (collect) an active VM instance. Requests for virtual machine creation received by VMShop contain specifications of hardware, network and software configurations. VMShop is then is responsible for selecting a VMPlant for the creation of a virtual machine. This process is implemented through a communication API and a binding protocol that allows VMShop to request and collect bids containing estimated VM creation costs from VMPlants.

The VMPlant implements the process of VM instantiation, using the VM's DAG specification provided by a client through VMShop as its input. In addition to supporting flexibility of VM configuration, the DAG aids the implementation of an efficient VM creation process by supporting partial matches of cached VM images to find a suitable match – a "golden" machine. Once a golden machine has been found, VMPlant clones the machine, and then parses the DAG to perform a series of configuration actions on the new machine. Once a machine is cloned, the configuration process returns a descriptor of the machine, which can be used by the client to make future references to the VM instance when issuing requests to VMShop.

4 Virtual File Systems and Virtual File System Services

In-VIGO uses a Grid Virtual File System (GVFS [7]) to support efficient and transparent Grid-wide data provisioning [1], [8]. GVFS presents a generic file system interface to applications by building a virtualization layer upon the de-facto NFS [9] distributed file system, and does so without changing the existing O/S clients/servers. It achieves on-demand cross-domain data transfers via the use of middleware-managed interchangeable logical user accounts [3] and file system proxy-based data access authentication, forwarding and user-identity mapping [10]. The design supports deployment of one or more proxies between a native NFS client and server. A multi-proxy setup is important to implement extensions to GVFS, provide additional functionality and improve performance.

A unique aspect of In-VIGO is how it integrates virtual machine and file system techniques to provide flexible execution environments and on-demand, transparent data access for unmodified applications. Data management has a key role in realizing the benefits of VM-based Grid computing [6] because a VM computing session typically involves data distributed across three different logical entities: the "state server", which stores VM state; the "compute server", which provides the capability of instantiating VMs; and the "data server", which stores user data. Without a virtual file system, instantiating a VM requires the explicit movement of state files to a compute server, and the explicit movement of user data to the VM once it is instantiated. In contrast, through GVFS, In-VIGO middleware creates dynamic GVFS

sessions between the state and compute servers to support access of VM states for VM instantiation, and between the VM and data servers to support access to user data for application execution within the VM [7].

GVFS supports secure Grid-wide data provisioning for both VM states and user files by way of two mechanisms: private file system channels and session-key based inter-proxy authentication. Privacy and integrity are guaranteed by the SSH connection, and user authentication is independently carried out by each private file system channel. Through the use of the virtualization layer, the session key handling is completely transparent to kernel clients and servers, and it only applies to inter-proxy authentication between tunnel end-points.

Caching is especially important to exploit data locality and hide network latency in Grid environments. In each GVFS session, the client-side proxy can dynamically establish and manage a file system disk cache to complement the kernel memory buffer with much greater capacity. The cache operates at the granularity of NFS RPC calls and satisfies requests with cached file attributes and data blocks. For write requests, it can employ write-back to hide write latencies and avoid transfers of temporary data. Furthermore, GVFS caches can be customized in many aspects (including size, associativity, write policy and consistency semantics) and thus be tailored to the needs of different applications. GVFS' inherent on-demand block-based data access manner allows for partial transfer of files and can benefit many applications, especially VM monitors, which typically access only a very small part of often Gigabyte-size VM disk state. As an application, the middleware can schedule GVFS sessions with VMM-specific coherence to allow for high-performance VM instantiations. For example, a VM with non-persistent state can be read-only shared among multiple users while each user has a "clone" of the VM and independent redo logs, so that aggressive read caching for state files and write-back caching for redo logs can be employed [7].

The data management middleware mentioned above has been implemented as WSRF-compliant services to provide interoperable service interfaces and flexible state management [11]. These services include: 1) file system service, which runs on every server and controls the local file system proxies to establish and configure specific GVFS sessions; 2) data scheduler service, which provides central scheduling and customization of GVFS sessions and interacts with individual file system services to start the sessions; 3) data replication service, which creates and manages data replicas for the purpose of fault tolerance and load balancing. To initiate a VM-based computing session in In-VIGO, the VMPlant service requests the data scheduler service to prepare a GVFS session between the VM state server and the VM host to instantiate a compute VM. Afterwards, the VAS service can request the scheduling of another session between the VM and the data server, so the application can be started inside the VM and access the user files via GVFS.

5 Virtual Networking

Network connectivity is an obvious necessity in Grid-computing, as it makes remote job execution/submission possible and also allows communication between processes

for parallel and/or distributed applications. However, due to firewalls and NAT devices, symmetric connectivity is often absent when resources are distributed across wide-area networks and different administrative domains.

Hosts behind firewalls or NATs can only initiate communication, i.e., they cannot receive communication initiation requests. This limits the hosts' ability to receive remote job execution requests and participate in distributed computations. Existing solutions to the asymmetric connectivity problem still face one or more of the following issues: (1) changes in firewalls or NAT configuration are required (e.g., to allow traffic in some ports or to forward ports), possibly violating security policies; (2) knowledge of network usage (e.g., transport port number) is necessary; (3) high administration overheads are implicit, since actions are required every time a new resource is added or removed from the Grid; and (4) application-transparency is not preserved. Solutions based on address/port translation require either the applications to be aware of resource discovery protocols (e.g., SOCKS [12], DPF and GCB [13]) or changes to be done in OS kernel network stack and/or in the Internet infrastructure (e.g., IPNL [14] and AVES [15]). When networking complexity is abstracted and a new API is exposed, application-transparency is lost (e.g., peer-to-peer networks and the Ibis programming environment [16]). Tunneling-based approaches have difficulties with firewalls and high administrative overhead (e.g., VPN, VNET [17], VIOLIN [18] and X-Bone [19]).

ViNe, the In-VIGO component responsible for network virtualization, has been designed to address all the above issues. It also has additional features such as support for on-demand creation, deployment and removal of isolated virtual networks that specifically connect the necessary machines for execution of a Grid application. The architecture of ViNe is based on IP-overlay on top of the Internet and resembles a site-to-site VPN setup. In each participating network, a ViNe router (VR) is placed in order to handle all ViNe traffic. VRs are responsible for intercepting IP packets destined to ViNe private address space, encapsulate them with ViNe header and forward them to the VR that can deliver the original IP packet. VRs make routing decisions (i.e., to where a packet needs to be forwarded) based on a set of routing tables, which can be updated by secure VR-to-VR communication. The secure update of the tables is the key for the on-demand definition of new virtual networks.

When a VR is placed in a network environment behind a firewall or NAT device, it is called a *limited VR*. Limited VRs cannot receive communication initiated by peer VRs, so a VR without limitations needs to be allocated as an intermediate node, which is called *queue VR*. Routing tables of all VRs are updated to forward to the queue VR the packets that are destined to the limited VR subnet. Since a limited VR can initiate communication, it is its responsibility to contact the queue VR and retrieve packets.

ViNe uses the private IP address space which is not routable in the Internet. Since ViNe nodes cannot be reached directly from the Internet, network security can be discussed with respect to external traffic and internal traffic. External traffic includes VR-to-VR communication, including encapsulated IP packets and control messages. Internal traffic includes the actual communication between hosts in ViNe space. VR-to-VR communication is secured by cryptographically authenticating all messages,

and also by encrypting critical information exchange such as control messages. Internal traffic security is achieved by either implementing all security policies of an organization in the VR or delegating that function to the firewall that may be already present in the site. The latter is possible because ViNe does not modify IP packets, and the firewall can still inspect and filter ViNe internal traffic following original rules.

The first prototype of VR has been implemented in Java, with low level networking handled by C code. Hosts do not need the installation of additional software in order to join ViNe, requiring only the operating systems be able to bind additional IP addresses to a network interface and to define static routes. Those features are present in most modern operating systems, making ViNe platform independent. Experiments showed that ViNe can offer performance that is close to the physical network, both in round-trip latency and throughput.

ViNe enables machines, even if they are connected to private networks, to easily join the Grid, and also can minimize the reluctance of system administrators to share resources by not requiring changes in security policies in the existing networks (a minimal change may be necessary, i.e. allowing ViNe traffic through the shared resources; however, the ViNe traffic will undergo the same packet inspection/filtering as the regular network traffic).

6 Virtual Application Services

The In-VIGO Virtual Application (VA) framework enables developers to automatically and transparently enable unmodified legacy applications "for the Grid" and users to transparently access deployed applications using virtualized resources "on the Grid". This requires the creation of VA services capable of orchestrating the use of previously discussed virtualization and other core Grid-middleware components. GridLab's Grid Application Toolkit (GAT) [20], Application Web Services (AWS) [21] and GridPort [22] are examples of other frameworks that aggregate core Grid-middleware to facilitate execution of applications and construction of Web-portals, but that do not consider exposing each application as a Web/Grid-service.

A *virtual application* consists of a physical application (unmodified application binaries and necessary execution environment) and additional software that (1) customizes the interface of the physical application to appear as multiple different applications with different capabilities for different users, and (2) interacts with other middleware in order to enable multiple simultaneous non-conflicting application instances on Grid resources. In particular, the virtual application makes use of the resource virtualization techniques and services described in the previous sections (virtual machines, virtual file system and virtual networks) to create the execution environment required by the application.

A *virtual application service* is a virtual application whose interfaces comply with WSRF specifications. Grid-enabling is the process of turning command-line applications interfaces into services that can be integrated into Grid-portals and delivered through Web-based interfaces. Unless automated, Grid-enabling demands

considerable time and programmer effort, especially for legacy applications which do not use programming technologies and practices that are well suited for Grid-computing and are not interoperable with other applications. To overcome this issue, the VA approach provides automatic Grid-enabling of legacy applications for which the following information needs to be provided: command-line syntax, description of the command line in natural language, application resource requirements, and execution environment settings.

Generated virtual application services are: (1) Consumable: the VAS can be discovered by, and made available to, other organizations in a technology-neutral manner that hides heterogeneity and allows interoperability and composition; (2) Isolated: simultaneous conflict-free execution of multiple unmodified applications is possible; (3) Customizable: VAS functionality can be customized to be the same as the original application, or it can be restricted, augmented, or composed with other applications, per user or per user-group; (4) Scalable to create and deploy: application virtualization is a one-time automated process that greatly reduces the overhead of creation and deployment of multiple application services; (5) Dynamically enabled: VAS deployment can be done in a "plug-and-play" fashion without having to bring down any part of the Grid infrastructure.

A distinct contribution of the VA approach is the VA language that allows the description of command-line applications interface with potentially complex set of parameters. The specifiable information about the command-line format includes the following: parameter types, default values, number of occurrences of a parameter, groupings of parameters, dependencies among parameters, multiple group choices, and parameter sweeping information. This language allows an application *enabler*, a special user who has knowledge of the application, but not necessarily of the underlying Grid infrastructure, to describe the application in a more comprehensive manner than solutions proposed by SoapLab [23] and Generic Application Factory Service (GAFS) [24]; thus, allowing strong parameter-type validation.

The VA architecture supports three processes: virtual application enabling, virtual application service customization and generation, and virtual application service utilization. It is divided into three tiers: the Web-portal tier which automatically generates web interfaces of the Grid-services, the virtual application tier discussed in this section and the virtual-Grid tier composed of virtualization services described in the previous sections (Fig. 2). Two solutions for the virtual application service customization and generation process were implemented in In-VIGO: (1) Generic Application Service (GAP) [25] in which a generic Grid-service dynamically configures itself according to the application information, making the interface of the specific application available to the service client using a description language developed in the In-VIGO project, and (2) Virtual Application Service (VAS) which generates one specific Grid-service for each application so that the application interface is fully described using the standard Web Services Description Language (WSDL). The VAS framework transforms the application information into XMLSchemas fully using the expressiveness of it, including it as part of the service description (WSDL), and then it generates, compiles and deploys the service implementation. The solution makes use of third party tools like XMLBeans to

generate complex binding types expressed in XMLSchemas, a modified WSDL2Java to generate the service implementation, AdminClient to deploy the service, Apache Ant to coordinate this automated process, and Apache Axis and Tomcat as containers of the generated services.

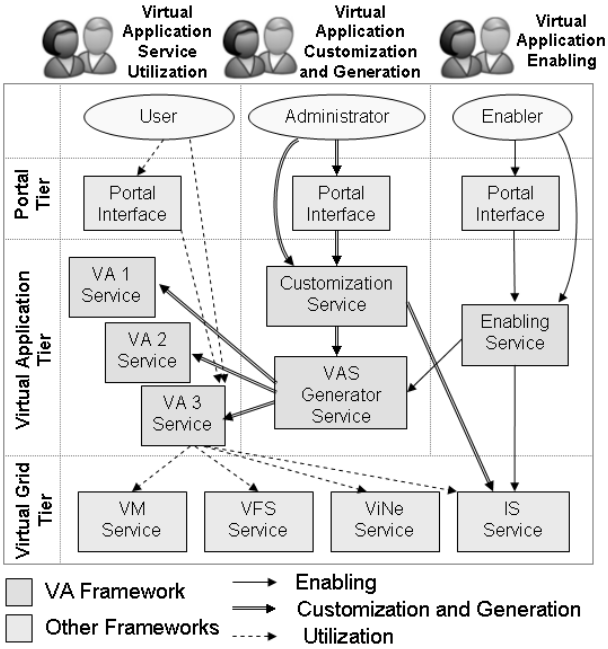

Fig. 2. VA Architecture. Components are separated into portal, virtual application and virtual Grid tiers. From right to left, the diagram depicts paths for: (1) enabling an application by an enabler, (2) customizing and generating the virtual application services VA 1, VA 2 and VA 3 by an administrator and (3) utilizing the virtual Grid (virtual machines, virtual file system, and virtual networks) to deliver the VA 3 service to a user

7 Building Virtual Grids: In-VIGO at Work

In order to enable sharing of geographically distributed computational and data resources with different usage policies, In-VIGO middleware shares with other Grid middleware, the requirement of interfacing with heterogeneous resource access and authentication schemes. Using resources managed by cluster or other Grid middleware, such as Globus or Condor-G, entails delegation of jobs to these middleware components using the appropriate job management syntax, and authentication and authorization scheme. This section describes the approaches used in current In-VIGO deployments to interface with multi-institutional resources for managing tasks associated with In-VIGO middleware and users.

In-VIGO users do not have direct access to, and are completely decoupled from, user accounts on Grid resources where jobs are effectively run. In-VIGO middleware has full control of all resources and is responsible for starting jobs as well as maintaining them, with complete freedom on how to dynamically map Grid users to local users, and possibly recycle local user accounts among Grid users. The approach brings advantages for both Grid users and resource providers: Grid users are freed from the need to manage several credentials; resource administrators are freed from the task of reconfiguring the access control of resources every time a user joins or leaves the Grid.

In-VIGO users authenticate themselves by presenting their username and password to the Grid portal. After login, user actions resulting in access to a Grid resource are handled by the In-VIGO middleware through the use of Role-Based Access Control (RBAC) mechanisms, offering Single Sign-On (SSO) for users. Users are grouped into roles (e.g., regular, Matlab licensed, administrator), while resources are configured by their providers with a set of permission groups which define operations (e.g., a simulator in demo, full and configuration modes). Appropriate mappings between user roles and permission groups are defined, and In-VIGO middleware enforces the mappings when accessing resources on behalf of users. For example, only users in the "Matlab licensed" role would be able to run a Matlab-based simulator in its full operation mode.

Resources, especially local user accounts, need to be isolated from each other because they are recycled among Grid users. To address this need, local accounts are either pre-created by resource providers, or created on-demand for a particular user in VMs where In-VIGO middleware has administrative privileges. In the first case, In-VIGO middleware makes sure that, at any point in time, only one user is mapped to a given local user account, and also that the account is cleaned when the job finishes. In the latter case, accounts are created and destroyed for one Grid user, without the need for recycling. Since a local account does not run processes for two different users simultaneously, user isolation at process level is guaranteed. However, local user accounts also need to have their data access privileges limited to the current assigned Grid user, as isolation is compromised if the local accounts have access to data of all Grid users. GVFS provides the necessary data isolation between Grid users. GVFS controls access at the granularity of directories so that In-VIGO middleware is able to limit the shadow account's access of data to the home directory of the Grid user allocated to it. Further data isolation, among jobs running for the same Grid user, can be achieved by limiting the access of the local user accounts running the jobs to the job working directory, which are subdirectories under the Grid user's home directory.

As the In-VIGO middleware has all the necessary credentials to access accounts (i.e., to remotely submit a job, independently of the mechanism – Condor, GSI, PBS, SSH, etc) to run jobs on behalf of the user, providing SSO access to Grid resources is trivial. More complex SSO solutions are however required when providing users access to interactive applications that require application level authentication from the user. Examples of such applications currently supported by In-VIGO include VNC sessions and a web-based file-manager. In the case of VNC, In-VIGO remotely starts its server process with a random password in a shadow account. When the user

requests access to the VNC desktop, In-VIGO embeds the necessary credential into the VNC client applet and transmits it securely (through SSL) to the user. When the VNC client is run by the user, it authenticates automatically (on behalf of the user) to the server. Adding RBAC to the above process, enables In-VIGO to allow sharing of workspaces among users, i.e., it enables a group of users (belonging to a single user role) to access a given VNC session without the need for users to share credentials and/or passwords.

In-VIGO selects resources for running In-VIGO user or middleware related tasks based on the job requirements specified by the In-VIGO application enabler, and resource availability and usage policies. This resource matching is performed by In-VIGO in the case of resources directly managed by it, or may be delegated to the cluster or Grid software, such as Condor-G or PBS, managing the resources. In the latter case, In-VIGO job requirements need to be mapped to job requirements in the specification syntax of the cluster or Grid software. Allowing for direct specification of job requirements based on the specification syntax of specific cluster/Grid software requires that the application enabler be aware of the types of resources that the application can use. This problem is typically overcome by introducing a uniform specification syntax that subsumes the specification syntax of the varied cluster/Grid software. Since existing cluster/Grid specification syntax used to describe resource/request properties are based on symmetric flat attributes [26], the uniform specification syntax inherits their shortcoming, namely the need for tight coordination between resource providers and consumers to agree upon attribute names and values. To allow for a flexible and extensible approach to resource matching in In-VIGO semantic matching of resource descriptions is used [27]. The In-VIGO job specifications and resource descriptions and usage policies are described using RDF [28] based ontologies, along with semantic entailments for matchmaking. Handlers specific to the type Cluster/Grid software are then used to map job specification and job management information to the software-specific syntax. The asymmetric description of resource and request enables VA descriptions that are decoupled from the supported resources and implementation of resource matching in In-VIGO.

Conclusions and In-VIGO Status

Many challenges faced in early versions of Grid middleware were due to the need to support different applications and distinct users on heterogeneous resources under separate administrative control. The use of virtualization effectively minimizes the impact of hardware and system software dependencies on Grid middleware by generating on-demand the execution environments needed for each application and user. The use of services enables customization of applications for each user while hiding implementation details, thus removing the need for multiple variants of Grid middleware. This “dual rail” decoupling greatly facilitates the management of Grid resources without interfering with other users, and the creation and provision of services without conflicts with other service implementations.

The In-VIGO research reported in this paper confirms the potential benefits of virtualization and services by devising and deploying efficient services for the

creation of virtual resources and virtual grids, and providing techniques for the automatic Grid-enabling of applications as services and their on-demand instantiation. The first version of In-VIGO has been online since July of 2003; this and newer versions of In-VIGO have been the subject of research and development since August of 2001. The concepts discussed in this paper have been implemented in at least one of these versions. Extensive prototyping and experimental evaluation of these concepts have demonstrated that the overheads of using virtualization and services are either minimal or acceptable for most Grid-computing applications. In-VIGO middleware is currently being used to deliver Grid-based computational services to users in several domains of science and engineering, which include computational nanoelectronics, coastal and ocean modeling, materials science, computer architecture and parallel processing.

Acknowledgements

The In-VIGO project is supported in part by the National Science Foundation under Grants No. EIA-9975275, EIA-0224442, ACI-0219925, EEC-0228390; NSF Middleware Initiative (NMI) collaborative grants ANI-0301108/ANI-0222828, SCI-0438246; 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. Adabala, S., Chadha, V., Chawla, P., Figueiredo, R.J., Fortes, J.A.B., Krsul, I., Matsunaga, A., Tsugawa, M., Zhang, J., Zhao, M., Zhu, L., Zhu, X.: From Virtualized Resources to Virtual Computing Grids: The In-VIGO System. Future Generation Computing Systems, special issue on Complex Problem-Solving Environments for Grid Computing, Vol 21/6, 2005, 896–909.
2. Fortes, J.A.B., Figueiredo, R.J., Lundstrom, M.S.: Virtual Computing Infrastructures for Nanoelectronics Simulation. IEEE Proceedings: Special Issue on Blue Sky Technologies (in press), 2005.
3. Kapadia, N., Figueiredo, R.J., Fortes, J.A.B.: Enhancing the Scalability and Usability of Computational Grids via Logical User Accounts and Virtual File Systems. In Proceedings of Heterogeneous Computing Workshop at the International Parallel and Distributed Processing Symposium, April 2001.
4. Krsul, I., Ganguly, A., Zhang, J., Fortes, J., Figueiredo, R.: VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing. In Proceedings of Supercomputing 2004.

5. Keahey, K., Doering, K., Foster, I.: From Sandbox to Playground: Dynamic Virtual Environments in the Grid. In Proceedings of Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04).
6. Figueiredo, R.J., Dinda, P.A., Fortes, J.A.B.: A Case for Grid Computing on Virtual Machines. In Proceedings of International Conference on Distributed Computing Systems, May 2003.
7. Zhao, M., Figueiredo, R.J.: Distributed File System Support for Virtual Machines in Grid Computing. In Proceedings of 13th IEEE International Symposium on High Performance Distributed Computing, June 2004.
8. Figueiredo, R.J., Kapadia, N., Fortes, J.A.B.: The PUNCH Virtual File System: Seamless Access to Decentralized Storage Services in a Computational Grid. In Proceedings of IEEE International Symposium on High Performance Distributed Computing, August 2001.
9. Pawlowski, B., Juszczak, C., Staubach, P., Smith, C., Lebel, D., Hitz, D.: NFS Version 3 Design and Implementation. In Proceedings of USENIX Summer Technical Conference, 1994.
10. Figueiredo, R.J., Kapadia, N., Fortes, J.A.B.: Seamless Access to Decentralized Storage Services in Computational Grids via a Virtual File System. In Cluster Computing, 2004.
11. Zhao, M., Chadha, V., Figueiredo, R.J.: Supporting Application-Tailored Grid File System Sessions with WSRF-Based Services. In Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing, July 2005, 202–211.
12. Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., Jones, L.: SOCKS protocol version 5. RFC1928, March 1996.
13. Son, S., Livny, M.: Recovering Internet Symmetry in Distributed Computing. In Proceedings of the 3rd International Symposium on Cluster Computing and the Grid, May 2003.
14. Francis, P., Gummadi, R.: IPNL: A NAT-Extended Internet Architecture. In Proceedings of the ACM SIGCOMM 2001, August 2001.
15. Eugene Ng, T.S., Stroica, I., Zhang, H.: A Waypoint Service Approach to Connect Heterogeneous Internet Address Spaces. In Proceedings of USENIX 2001, June 2001, 319–332.
16. Denis, A., Aumage, O., Hofman, R., Verstoep, K., Kielmann, T., Bal, H.: Wide-Area Communication for Grids: An Integrated Solution to Connectivity, Performance and Security Problems. In Proceedings of 13th IEEE International Symposium on High Performance Distributed Computing, June 2004.
17. Sundararaj, A., Dinda, P.: Towards Virtual Networks for Virtual Machine Grid Computing. In Proceedings of the 3rd USENIX Virtual Machine Research and Technology Symposium, May 2004.
18. Jiang, X., Xu, D.: VIOLIN: Virtual Internetworking on Overlay Infrastructure. In Proceedings of Parallel and Distributed Processing and Applications: Second International Symposium, ISPA 2004, Hong Kong, China, December 13-15, 2004.
19. Touch, J., Hotz, S.: The X-Bone. Proc. of Global Internet Mini-Conference at Globecom, November 1998.
20. Allen, G., Davis, K., Goodale, T., Hutanu, A., Kaiser, H., Kielmann, T., Merzky, A., van Nieuwpoort, R., Reinefeld, A., Schintke, F., Schott, T., Seidel, E., Ullmer, B.: The grid application toolkit: toward generic and easy application programming interfaces for the grid. In Proceedings of the IEEE, Vol.93, Iss.3, March 2005, 534–550.
21. Pierce, M., Fox, G., Youn, C., Mock, S., Mueller, K., Balsoy, O.: Interoperable Web services for computational portals. In Proceedings of the 2002 ACM/IEEE conference on Supercomputing (Baltimore, MD, 2002), IEEE Computer Society Press, 2002, 1–12.
22. Thomas, M., Boisseau, J.: Building Grid Computing Portals: The NPACI Grid Portal Toolkit. Grid Computing: Making the Global Infrastructure a Reality, Ch 28. F. Berman, G. Fox and T. Hey, eds. John Wiley and Sons, Ltd, Chichester (2003).

23. Senger, M., Rice, P., Oinn, T.: Soaplab - a unified Sesame door to analysis tools. In Proceedings of UK e-Science All Hands Meeting September 2003, 509–513.
24. Gannon, D., Alameda, J., Chipara, O., Christie, M., Dukle, V., Fang, L., Farrellee, M., Kandaswamy, G., Kodeboyina, D., Krishnan, S., Moad, C., Pierce, M., Plale, B., Rossi, A., Simmhan, Y., Sarangi, A., Slominski, A., Shirasuna, S., Thomas, T.: Building grid portal applications from a web service component architecture. In Proceedings of the IEEE, Vol.93, Iss.3, March 2005, 551–563.
25. Sanjeepan, V., Matsunaga, A., Zhu, L., Lam, H., Fortes, J.A.B.: A Service-Oriented, Scalable Approach to Grid-Enabling of Legacy Scientific Applications. In Proceeding of International Conference on Web Services (ICWS), Industry Track, July 2005.
26. Solomon, M., Raman, R. and Livny, M.: Matchmaking distributed resource management for high throughput computing. In Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, Chicago, IL, July 1998.
27. Tangmunarunkit, H., Decker, S. and Kesselman, C.: Ontology-Based Resource Matching in the Grid - The Grid Meets the Semantic Web. The Semantic Web - ISWC 2003, Second International Semantic Web Conference, Sanibel Island, FL, USA, October 20-23, 2003, Proceedings. Lecture Notes in Computer Science 2870 Springer 2003, ISBN 3-540-20362-1
28. Lassila, O., and Swick, R.R.: Resource description framework (rdf) model and syntax specification. In W3C Recommendation, World Wide Web Consortium. February 1999. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222.