Toward a Doctrine of Containment: Grid Hosting with Adaptive Resource Control

Lavanya Ramakrishnan[†] Laura Grit[‡] Adriana Iamnitchi[§]
David Irwin[‡] Aydan Yumerefendi[‡] Jeff Chase[‡]

lavanya@renci.org,{grit,irwin,aydan,chase}@cs.duke.edu,anda@cse.usf.edu

[†]Renaissance Computing Institute [‡]Duke University [§]University of South Florida

Abstract

Grid computing environments need secure resource control and predictable service quality in order to be sustainable. We propose a grid hosting model in which independent, self-contained grid deployments run within isolated containers on shared resource provider sites. Sites and hosted grids interact via an underlying resource control plane to manage a dynamic binding of computational resources to containers. We present a prototype grid hosting system, in which a set of independent Globus grids share a network of cluster sites. Each grid instance runs a coordinator that leases and configures cluster resources for its grid on demand. Experiments demonstrate adaptive provisioning of cluster resources and contrast job-level and container-level resource management in the context of two grid application managers.

1 Introduction

The investments in grid research and technology have yielded large-scale cyberinfrastructure deployments that serve the needs of multiple scientific communities. The TeraGrid and Open Science Grid (OSG) grew out of pioneering efforts to promote sharing of computational resources and datasets within *virtual organizations*—distributed user communities sharing across administrative boundaries.

For public grid systems to be dependable and economically sustainable, they must support *resource control* mechanisms and standards that are sufficiently powerful to balance the needs of resource providers and consumers.

- Resource provider sites should have autonomy to control how much of each resource type they allocate to each consumer at any given time.
- Resource consumers need predictable service quality (performance isolation) even in the presence of competition for shared resources. Service quality is especially crucial for urgent computing applications such as weather prediction and disaster response, and for real-time distributed computing, e.g., teleimmersion.

Secure, integrated resource control is essential for participants to quantify and control what they contribute to a grid and what they obtain from it. A number of projects have addressed resource control and adaptation [9, 10, 15, 16, 23, 29, 31, 33]. Even so, effective resource control continues to be elusive in the practice of grid computing.

This paper [1] proposes to advance resource management goals by integrating resource control functions at two different levels of abstraction: *jobs* and *containers*. Jobs—individual tasks or task workflows—are the basic unit of work for high-throughput computing, so middleware systems for clusters and grids focus on job management as the basis for resource control. Our premise is that the architecture should also incorporate resource control functions at the level of the logical context or "container" within which the jobs and the middleware services run. Advances in virtualization technologies—including but not limited to virtual machines—create new opportunities to strengthen container abstractions as a basis for resource control and for isolation and customization of hosted computing environments, including grid environments [8, 19, 22, 26, 30, 31].

Our goal is to evolve the foundations of the grid to enable flexible policies governing the *physical resources* that are bound to the containers hosting grid services and applications. This paper makes the following contributions:

- We propose an architecture for *grid hosting* that provides container-grained resource management functions in a *resource control plane* operating at a level below the middleware and even below the node operating system. The control plane may be viewed as "underware" rather than middleware.

SC2006 November 2006, Tampa, Florida, USA
0-7695-2700-0/06 $20.00 ©2006 IEEE

[1]This research is supported by the National Science Foundation through ANI-0330658, CNS-0509408, EIA-99-72879, and CNS-0451860, and by IBM, HP Labs, and Network Appliance. Laura Grit is a National Physical Science Consortium Fellow.

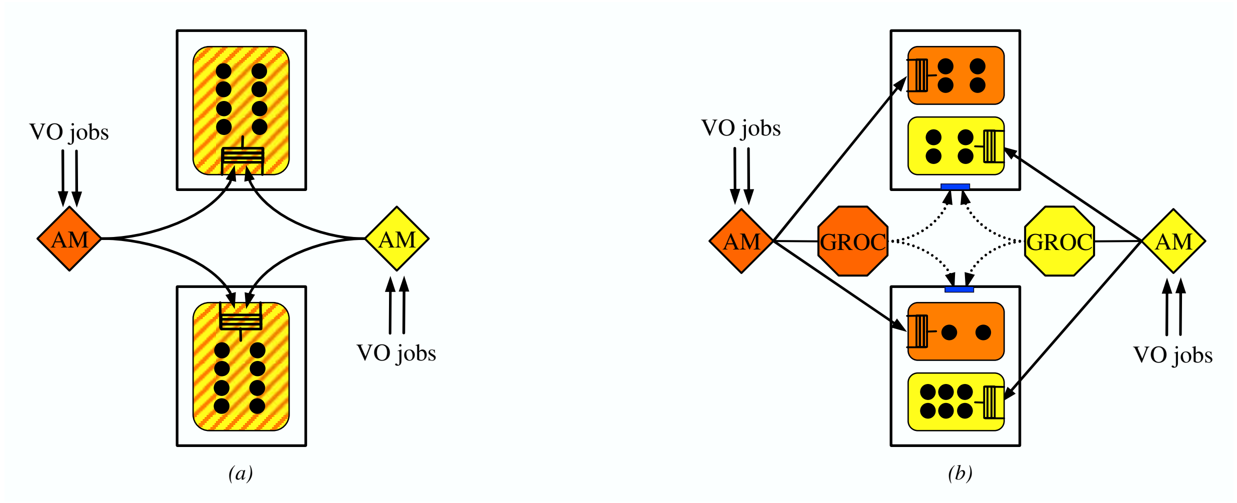


Figure 1: Two architectural alternatives for resource providers serving multiple grid user communities, or VOs. In (a), the VOs' application manager (AM) submit jobs through a common gatekeeper at each site; job scheduling middleware enforces the policies for resource sharing across VOs. In (b), each VO runs a private grid within isolated workspaces at each site. Isolation is enforced by a foundational resource control plane. Each VO grid runs a coordinator (GROC) that controls its middleware and interacts with the control plane to lease resources for its workspaces.

- We show how hosted grids can negotiate with the resource control plane to procure resources across grid sites in response to changing demand. We present the design and implementation of a prototype system based on the Shirako [19] toolkit for secure resource leasing from federated resource provider sites. Cluster sites are managed with Cluster-on-Demand [8] and Xen virtual machines [3]; the hosted grid software is based on the Globus Toolkit (GT4) [13].

- Within this supporting infrastructure, we explore coordinated mechanisms for programmatic, automatic, service-oriented resource adaptation for grid environments.

2 Overview

In grid systems, user communities, or virtual organizations (VOs), generate streams of jobs to execute on shared resource sites, e.g., cluster farms. These cluster sites provide computational resources to VOs. We refer to the entities that generate the jobs as *application managers*. The term denotes a domain-specific entry point to a grid; VO users may submit jobs through a portal framework or gateway, a workflow manager, or a simple script interface. Section 4 presents experiments with application managers for a storm surge prediction service (SCOOP [28]) and a web-based bioinformatics service (Bioportal [5]).

Figure 1(a) depicts an example of a standard Globus grid with two VOs executing on two sites. A VO's application manager submits each task to a "gatekeeper" at one of the sites, which validates it and passes it to a local batch scheduling service for execution. There are four key dimensions to resource control policy in such a system:

- *Resource allocation to VOs.* The sites control their resources and determine how to allocate them to serve the needs of the competing VOs. A site may assign different shares or priorities to contending VOs, and/or may hold resources in reserve for local users.

- *Resource control within VOs.* VOs determine the rights and powers of their users with respect to the resources allocated to the VO.

- *Task routing.* The application managers for each VO determine the routing of tasks to sites for timely and efficient execution.

- *Resource recruitment.* Entities acting on behalf of the VOs negotiate with provider sites for resources to serve the VO's users.

One important feature of current practice is that the sites implement their own *resource allocation policies* as job-level policies within the batch schedulers. A scheduler may give higher priority to jobs from specific user identities or VOs, may export different queues for different job classes, and may support job reservations. *Resource recruitment* is currently based primarily on reciprocal and social agreements requiring human intervention (person-to-person rather than peer-to-peer); a recent example is the notion of *right-of-way tokens* in the SPRUCE [32] gateway extensions for urgent computing. Many current deployments also rely on ad hoc routing of tasks to grid sites, given the current lack of standard components to coordinate task routing.

2.1 Resource Control with Containers

Figure 1(b) depicts the architectural model we propose for hosted grids with container-level resource control. Each site instantiates a logical container for all software associated with its hosting of a given VO. The container encapsulates a complete isolated computing environment or *workspace* [14] for the VO grid's point-of-presence at the site, and should not be confused with the individual JVMs that run Java components at the site. Each VO grid runs a separate batch task service within its workspace. The site implements resource control by binding resources to containers; the containers provide isolation, so each instance of the batch scheduler only has access to the resources bound to its container, and not other resources at the site.

In essence, we propose a "Grid" comprising a set of autonomous resource provider sites hosting a collection of independent "grids":

- Each grid serves one or more communities; we speak as if a grid serves a single VO, but our approach does not constrain how a hosted grid shares its resources among its users.
- Each grid runs a private instance of its selected middleware to coordinate sharing of the data and computing resources available to its user community.
- Each grid runs within a logically distributed container that encapsulates its workspaces and is bound to a dynamic "slice" of the Grid resources.

2.2 GROC

While the sites control how they assign their resources to each hosted grid, the grids control the other three areas of policy internally. We propose that each hosted grid include a coordinating manager, which we will call the GROC—a loose acronym for Grid Resource Oversight Coordinator.[2] The GROC performs two inter-related functions, which are explained in detail in Section 3:

- The GROC is responsible for advising application managers on the routing of tasks to sites. In this service brokering role the GROC might be called a *metascheduler* or *superscheduler*.
- The GROC monitors the load and status of its sites (points of presence), and negotiates with providers to grow or shrink its resource holdings. It may resize the set of batch worker nodes at one or more sites, set up new grid sites on resources leased from new providers, or tear down a site and release its resources.

[2]The novelist Robert Heinlein introduced the verb *grok* meaning roughly "to understand completely". The name GROC emphasizes that each hosted grid has a locus of resource policy that operates with a full understanding of both the resources available to the grid and the grid's demands on its resources.

The GROC thus serves as the interface for a VO application manager to manage and configure its resource pool, and may embody policies specific to its application group. Crucially, our approach requires no changes to the grid middleware itself. Our prototype GROC is a service built atop the Globus toolkit and it is the sole point of interaction with the underlying resource control plane.

2.3 Resource Control Plane

The GROC uses programmatic service interfaces at the container-level resource control plane to acquire resources, monitor their status, and adapt to the dynamics of resource competition or changing demand. The control plane is based on the SHARP [17] leasing abstractions as implemented in the Shirako toolkit [19]. Each lease represents a contract for a specified quantity of typed resources for some time interval (*term*). Each resource provider runs a local resource manager called Cluster-on-Demand (COD [8]), and exports a service to lease *virtual clusters* from a shared server cluster. Each virtual cluster comprises a dynamic set of nodes and associated resources assigned to some guest (e.g., a VO grid) hosted at the site. COD provides basic services for booting and imaging, naming and addressing, and binding storage volumes and user accounts on a per-guest basis.

The GROC interacts with the site to configure its virtual clusters and integrate them into the VO's grid (Section 3.4). When the lease expires, the grid vacates the resource, making it available to other consumers. The site defines local policies to arbitrate requests for resources from multiple hosted grids. In our prototype (Section 3) the leased virtual clusters have an assurance of performance isolation: the nodes are either physical servers or Xen [3] virtual machines with assigned shares of node resources. We use Xen VMs because they boot faster and more reliably than physical servers, but the concept applies equally to physical servers.

2.4 Separation of Concerns

While the hosted VOs and their grid middleware retain their control over job management, the GROC managers interact with the resource control plane to drive the assignment of resources to VOs. The assignment emerges from the interaction of GROC policies for requesting resources and the resource provider policies for arbitrating those resource demands. In effect, the architecture treats the grid nodes and their operating systems as managed entities. Provider sites allocate resources to workspace containers without concern for the details of the middleware, applications, or user identities operating within each workspace isolation boundary.

Grid hosting with container-level management is particularly important as the Grid evolves toward a stronger separation between resource providers and consumers. TeraGrid and Open Science Grid are examples of the growth of large infrastructure providers. They signal a shift from a traditional emphasis on reciprocal peer-to-peer resource sharing

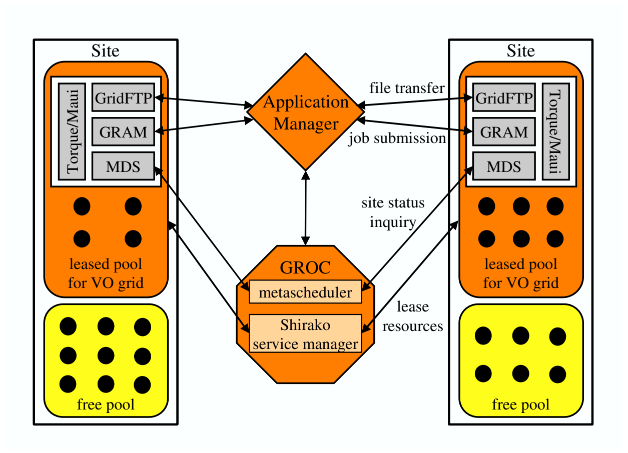


Figure 2: Overview of components for a GROC managing a VO grid hosted on virtual clusters leased from multiple cluster sites. The application manager interacts with Globus services, instantiated and managed by the GROC, for job and data management.

within VOs to a new emergence of resource providers that serve computational resources to multiple competing user communities or VOs.

Containment and container-level management also enable resource providers to serve more diverse needs of their VOs. A resource provider site can host different grid stacks or other operating software environments concurrently. For example, this flexibility may make it possible to unify the hosting infrastructure for the Grid and NSF GENI network testbed initiatives. In the longer term, containment can pave the way for a practical cyberinfrastructure economy: one path to reducing the overhead of economic protocols (e.g., bidding and auctions) is to apply them at the container level, rather than at the granularity of individual jobs.

Our approach assumes that the grid middleware can adapt to a dynamically changing set of worker nodes at the sites. In fact, adaptation is always required in a dynamic world: compute servers may fail or retire, and provider sites deploy new servers in response to bursts of demand or funding. With a grid hosting model, grids may grow dynamically to use additional resources as they become available. One limitation is that batch services often do not have adequate support to checkpoint or reschedule nodes when worker nodes fail or shutdown. Checkpointing and migration continue to be active research topics, and these capabilities are increasingly crucial for long-running jobs in a dynamic world.

3 Design and Implementation

We present the design and implementation of a prototype system that coordinates dynamic resource leasing and task routing, based on the grid hosting architecture outlined above. Our prototype leverages the standard Globus Toolkit (GT4) for resource management within each hosted grid: job management, resource discovery, identity management and authorization, and file transfer. Dynamic resource leasing is based on Shirako, a service-oriented toolkit for constructing SHARP resource managers and COD cluster sites, which is described in detail in [19].

Figure 2 illustrates the interactions among the most important components within a hosted grid, as implemented or used in the prototype.

- The nucleus of the hosted grid is the GROC, which orchestrates task flow and resource leasing. The GROC is the point of contact between the Globus grid and the Shirako resource control plane.
- The application managers (e.g., portals) control the flow of incoming job requests. They consult the GROC for task routing hints (Section 3.2), then submit the tasks to selected sites.
- A Globus Resource Allocation Manager (GRAM) runs on a master node (*head node*) of a virtual cluster at each provider site, acting as a gatekeeper to accept and control tasks submitted for execution at the site.
- The application managers interact with a secure staging service on each head node to stage data as needed for tasks routed to each site, using Reliable File Transfer (RFT) and Grid File Transfer Protocol (GridFTP).
- When a task is validated and ready for execution, GRAM passes it to Torque, an open-source batch task service incorporating the Maui job scheduler.
- The GROC receives a stream of site status metrics as a feedback signal to drive its resource requests (Section 3.1). Each site exposes its status through a Globus Monitoring and Discovery Service (MDS) endpoint.
- The GROC acts as a Shirako *service manager* to lease resources on behalf of the VO; in this way, the GROC controls the population of worker nodes bound to the hosted grid's batch task service pools (Section 3.3). The GROC seamlessly integrates new worker nodes into its grid (Section 3.4) from each site's free pool.

The following subsections discuss the relevant aspects of these components and their interactions in more detail.

3.1 Site Monitoring

In our prototype, the GROC acts as a client of WS-MDS (a web service implementation of MDS in GT4) to obtain the status of the resources at each site, including the number of free nodes and the task queue length for each batch pool. The WS-GRAM publishes Torque scheduler information (number of worker nodes, etc.) through the MDS aggregator framework using the Grid Laboratory Uniform Environment (GLUE) schema. MDS sites may also publish information to upstream MDS aggregators; in this case, the GROC can obtain the status in bulk from the aggregators.

Currently the GROC queries the MDS periodically at a rate defined by the MDS poll interval. The poll interval is a tradeoff between responsiveness and overhead. We use a static poll interval of 600 *ms* for our experiments. The results of the *site poll* are incorporated immediately into the task routing heuristics. A simple extension would use MDS triggers to reduce the polling, but it is not a significant source of overhead at the scale of our experiments.

3.2 Task Routing

A key function of the GROC is to make task routing recommendations to application managers. The GROC factors task routing and other resource management functions out of the application managers: one GROC may provide a common point of coordination for multiple application managers, which may evolve independently. The task routing interface is the only GROC interface used by a grid middleware component; in other respects the GROC is non-intrusive.

To perform its task routing function, the GROC ranks the sites based on the results from its site poll and a pluggable ranking policy. Information available to the policy includes cluster capacity at each site, utilization, and job queue lengths. In addition, the policy module has access to the catalog of resources leased at each site, including attributes of each group of workers (e.g., CPU type, clock rate, CPU count, memory size, interconnect).

The coordinating role of the GROC is particularly important when multiple user communities compete for resources. The GROC maintains leases for the resources held by the VO grid: its task routing choices are guided by its knowledge of the available resources. Since it observes the grid's complete job stream, it can also make informed choices about what resources to request to meet its demand.

Our goal at this stage is to evaluate the grid hosting architecture, rather than to identify the best policies. Our prototype policy considers only queue length and job throughput for homogeneous worker nodes. In particular, we do not consider data staging costs. Job routing in our prototype uses a simple load balancing heuristic. It estimates the aggregate runtime of the tasks enqueued at each site, and the time to process them given the number of workers at each site. It selects the site with the earliest expected start time for the next job.

3.3 Resource Leasing

In the absence of support for resource leasing, the GROC could act as a task routing service for a typical grid configuration, e.g., a set of statically provisioned sites with middleware preinstalled and maintained by administrators at each site. In our system, the GROC can also use the resource control to change the set of server resources that it holds. The GROC invokes Shirako's programmatic resource leasing interface to acquire and release worker nodes, monitor their status, and/or instantiate points of presence at new cluster sites when resources are available and demand exists. This control is dynamic and automatic.

The GROC seeks to use its resources efficiently and release underutilized resources by shrinking renewed leases or permitting them to expire. This good-citizen policy is automated, so it is robust to human failure. An operator for the VO could replace the policy, but we presume that the VO has some external incentive (e.g., cost or goodwill) to prevent abuse. Note that our approach is not inherently less robust than a conventional grid, in which a greedy or malicious VO or user could, for example, submit jobs that overload a site's shared storage servers. In fact, the leased container abstraction can provide stronger isolation given suitable virtualization technology, which is advancing rapidly.

Resource provider sites in SHARP delegate power to allocate their resource offerings—possibly on a temporary basis—by registering them with one or more *brokers*. A SHARP broker may coordinate resource allocation across multiple sites, e.g., to co-schedule resources for a VO across the wide area and/or to arbitrate global resources at a common point. However, we do not experiment with shared brokers in this paper. Instead, each site keeps exclusive control of its resources by maintaining its own broker. We use the term "site" to mean the resource provider (COD server) and its broker together.

The GROC uses pluggable policies to determine its target pool sizes for each site. Section 4 defines the policies used in our experiments. The prototype GROC uses a predefined preference order for sites, which might be based on the site's resources or reputation, peering agreements, and/or other factors such as cost. Similarly, the sites implement a fixed priority to arbitrate resources among competing GROCs.

3.4 Configuring Middleware

Typically, grid middleware is configured manually at each site. One goal of our work is to show how to use Shirako/COD support to configure grid points of presence remotely and automatically. The responsibility—and power—to manage and tune the middleware devolves to the VO and its GROC, within the isolation boundaries established by the site. This factoring reduces the site's administrative overhead and risk to host a grid or contribute underutilized resources, and it gets the site operators out of the critical path, leaving the VOs with the flexibility to control their own environments.

COD does require site operators to administer their clusters using the RFC 2307 standard for an LDAP-based network information service. Standard open-source services exist to administer clusters and networks from an LDAP repository compliant with RFC 2307. The COD site authority configures virtual clusters in part by writing to the site's LDAP repository.

Configuration of a COD node follows an automated series of steps under the control of the Shirako leasing core. When a site approves a lease request for new worker nodes, the GROC passes a list of *configuration properties* interpreted by a resource-specific plugin *setup* handler that executes in the site's domain. The *setup* handler instantiates, images, and boots the nodes, and enables key-based SSH access by installing a public key specified by the GROC. It then returns a lease with *unit properties* for each node, including IP addresses, hostnames, and SSH host keys. The GROC then invokes a plugin *join* handler for each node, which contacts the node directly with key-based root access to perform an automated install of the middleware stack and integrate the node into the VO's grid. Similarly, there is a *teardown* handler that reclaims resources (e.g. machines), and a *leave* handler that cleanly detaches resources from the middleware stack. To represent the wide range of actions that may be needed, the COD resource driver event handlers are scripted using *Ant* [2], an open-source OS-independent XML scripting package. We prepared *join* and *leave* handler scripts to configure the middleware components shown in Figure 2.

To instantiate a point of presence at a new site, the GROC first obtains separate leases for a master node (with a public IP address) that also serves as a scratch storage server for data staging. It instantiates and configures the Globus components, Torque and Maui on the master, and configures the file server to export the scratch NFS volume to a private subnet block assigned to the virtual cluster. When a new worker node joins, the *join* handler installs Torque and registers the worker with the local master node. The *join* handler for the master configuration is about 260 lines of Ant XML, and the worker join handler is about 190 lines.

Our prototype makes several concessions to reality. It assumes that all worker nodes are reachable from the GROC; in the future, we plan to proxy the worker *join* operations through the public head node for each virtual cluster so that workers may use private IP addresses. The *setup* attaches a shared NFS file volume containing the Globus distribution to each virtual cluster node, rather than fetching it from a remote repository. For the sake of simplicity, all the hosted grids use a common certificate authority (CA) that is configured using Globus's SimpleCA, although there is nothing in our architecture or prototype that prevents the hosted grids from each using a private CA. Interaction with the CA is not yet automated; instead, the GROC has preconfigured host certificates for the DNS names that its master nodes will receive for each potential site that it might use. The Shirako mechanisms for the GROC to install user identities for the virtual cluster are not yet complete, so a set of common user identities are preconfigured at the sites. Finally, for this paper, we prestage all applications and data required by the VO's users when we instantiate the site. We leave dynamic data staging to future work.

Currently, we use the default First Come First Served (FCFS) scheduling policies for Torque/Maui, but the GROC is empowered to set policies at its points of presence as desired. Thus, the application manager is able to rely on the VO's GROC to implement policies and preferences on how its available resources might be used by different members of the community, and to adapt these policies as the resource pool size changes.

3.5 Robustness

The GROC is stateless and relies on recovery mechanisms in Shirako, which maintains all lease state in a local LDAP repository. If a GROC fails, it will recover its knowledge of its sites and resource holdings, but it will lose its history of task submissions and the MDS feedback stream from the sites. Once recovered, the GROC maintains its existing leases and monitors grid operation for a configurable interval before adjusting its lease holdings. Reliable job submission and staging are handled using existing Globus mechanisms that do not involve the GROC.

As noted in Section 2.4, robust grid services must be capable of restarting jobs when nodes fail or leave the service. In our approach, nodes may depart due to resource competition, as governed by the site policies and the GROC interactions with the dynamic resource control plane. Although the GROC has advance warning of node departures, the Torque batch service in our current prototype is not able to suspend or migrate tasks running on those nodes; thus some tasks may be interrupted. We believe that support for virtual machine *checkpoint/migrate* is a promising path to a general solution; our Xen-based prototype supports VM migration, but we do not explore its use for robust adaptation in this paper.

3.6 Security

The important new security requirement of our architecture is that each GROC must have a secure binding to each of its candidate hosting sites. Each SHARP actor has a keypair and digitally signs its control actions. To set up the trust binding, there must be some secure means for each site and GROC to exchange their public keys. Related systems to delegate policy control to a VO—or a server (such as a GROC) acting on behalf of a VO—also make this assumption. Examples include the VO Membership Service (VOMS) [1] and Community Authorization Service (CAS) [25].

One solution is to designate a common point of trust to endorse the keys, such as a shared certificate authority (CA). Although each grid selects its own CA to issue the certificates that endorse public keys within the grid, the provider site authorities exist logically outside of the VO grids in our architecture. Thus reliance on a common CA would presume in essence that the public key certificate hierarchy (PKI) extends upwards to include a common CA trusted by all resource provider sites and all hosted grids. An alternative is to rely on pairwise key exchange among the sites and VO operators. In our prototype the public keys for the brokers and GROC s are installed through a manual operator interface.

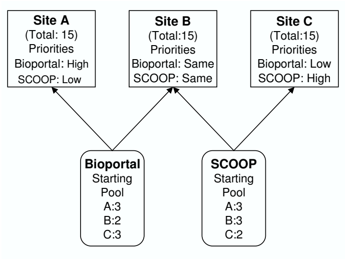


Figure 3: The testbed has three cluster sites with a maximum capacity of 15 virtual machines each. There are two hosted grids (the Bioportal and SCOOP applications). Each site assigns a priority for local resources to each grid, according to its local policies.

To instantiate a new site point of presence, the GROC passes the gateway host certificate and private key in an encrypted connection during *join*. Note, however, that the GROC cannot hide the site private keys used by its middleware from the hosting resource provider, since the resource provider knows the private SSH key of each leased node. There are many ways that a malicious resource provider can subvert or spy on its guests.

4 Evaluation

We conducted an experimental evaluation of the prototype to illustrate how hosted grids configure and adapt their resources to serve streams of arriving jobs. The experiments demonstrate on-demand server instantiation for hosted grids, dynamic adaptation driven by GROC policies, and the interaction of policies at the sites and grids

Application workloads. We consider here two specific grid application services: Bioportal [5], a web-based interface that allows VO users to submit bioinformatics jobs, and SCOOP [28], a system that predicts storm surge and local winds for hurricane events. Bioportal uses a simple policy to route user jobs to a local cluster and the TeraGrid. In its original incarnation it has no mechanism to ensure predictable service quality for its users. We selected four commonly used Bioportal applications (*blast*, *pdbsearch*, *glimmer*, *clustalw*) from the Bioportal tool suite to represent the workload.

The North Carolina SCOOP Storm Modeling system is an event-based system that triggers a series of Advanced Circulation (ADCIRC) runs on arrival of wind data. Executions are triggered periodically during the hurricane season based on warnings issued by the NOAA National Hurricane Center (NHC). One interesting aspect of SCOOP is its ability to forecast its demand since the hurricane warnings are issued every six hours during storm events. In the original version, a simple resource selection interface schedules the runs when each warning arrives; although SCOOP knows when runs will be issued, it cannot ensure that sufficient resources will be available to complete the models in a timely manner.

The experiments use GROC policies appropriate for each workload. Bioportal uses an *on-demand* policy that maintains a target upper bound on waiting time. The total number of nodes to request at each decision point is given by:

$$BioportalRequest_t = \max\left\{\frac{(WaitingJobs_t - FreeCPUs_t)}{WaitingFactor * Resources_t}, 0\right\}$$

Our experiments use $WaitingFactor = 2$.

SCOOP's GROC uses a *look-ahead* policy to reserve resources in advance of expected demand. It considers the current backlog and expected arrivals over a sliding time window. The total number of new nodes to request is given by:

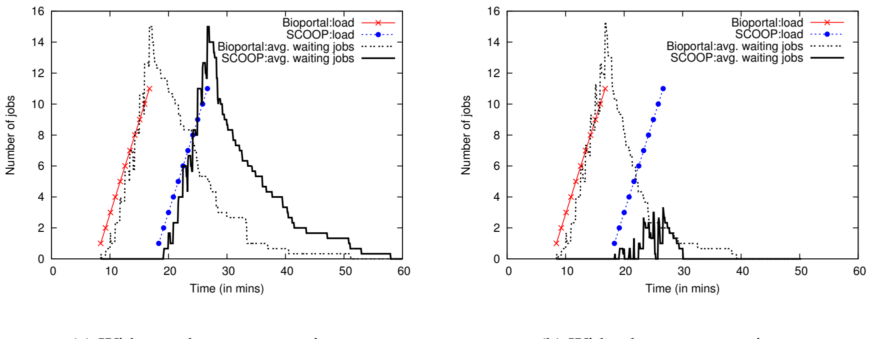
$$SCOOPRequest_t = \max\left\{\left((WaitingJobs_t - FreeCPUs_t) + \sum_{i=t}^{t+\Delta t} ExpectedJobs_i\right), 0\right\}$$

Experimental setup. All experiments run on a testbed of IBM x335 rackmount servers, each with a single 2.8Ghz Intel Xeon processor and 1GB of memory. Some servers run Xen's virtual machine monitor version 3.0.2-2 to create virtual machines. All experiments run using Sun's Java Virtual Machine (JVM) version 1.5. COD uses OpenLDAP version 2.2.23-8, ISC's DHCP version 3.0.1rc11, and TFTP version 0.40-4.1 to drive network boots.

We partition the cluster into three sites (Figure 3). Each site consists of a COD server that configures and monitors allocated machines, a broker server that implements the site's policy for allocating its resources to competing consumers, and five physical machines. The sites divide the resources of each physical machine across three virtual machines, giving a total resource pool of 45 machines for our experiment. Previous work [19] has shown that the leasing and configuration mechanisms scale to much larger clusters. The sites in our experiments use a simple priority-based arbitration policy with priorities as shown in Figure 3. All leases have a fixed preconfigured lease term.

Reservations and priority. This experiment illustrates how GROCs procure resources to serve growing load, and illustrates the mechanisms and their behavior. We consider two synthetic load signals that have a linearly increasing number of jobs arriving over a short interval. The duration of the load is 50 minutes and worker node lease term is 4 minutes.

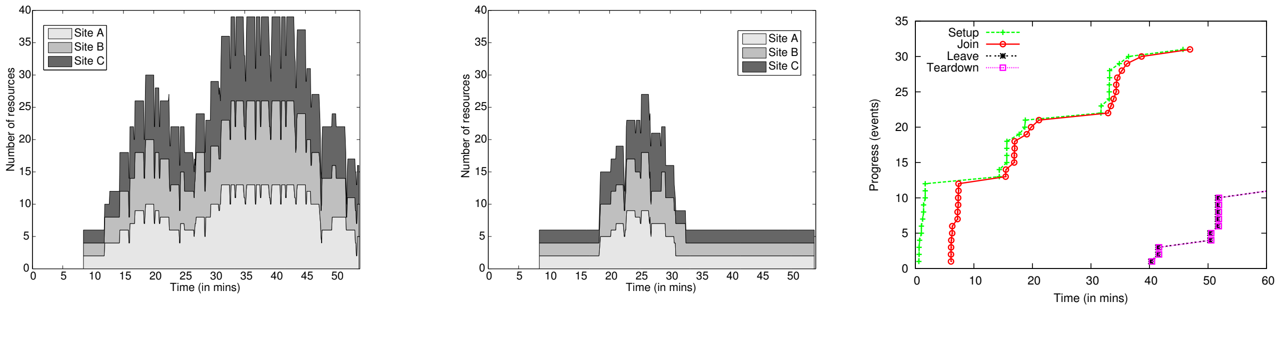
Figure 4 shows the average number of waiting jobs across the three sites (a) without and (b) with advance reservations. In both cases, the sites use priorities from Figure 3, and Bioportal uses its simple *on-demand* resource request policy. In Figure 4 (a), SCOOP's look-ahead horizon is zero, so it effectively uses an on-demand request policy as well. In Figure 4 (b), SCOOP reserves resources in advance of its anticipated need, significantly reducing its job delays and queue lengths.



(a) Without advance reservations

(b) With advance reservations

Figure 4: Average number of waiting jobs. In (b), the SCOOP grid reserves servers in advance to satisfy its predicted demand.



(a) Bioportal resource holding at each site

(b) SCOOP resource holding at each site

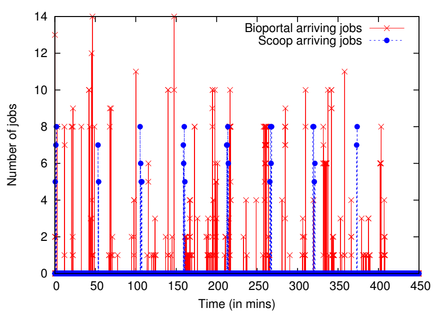
(c) Progress of server configuration events.

Figure 5: Site resources are allocated to competing GROCs according to their configured priorities. (a) shows the decrease in resources available to Bioportal as more machines are reserved to SCOOP, as shown in (b). Bioportal reacquires the machines as SCOOP releases them. (c) shows the progress of resource configuration events on sites and GROCs.

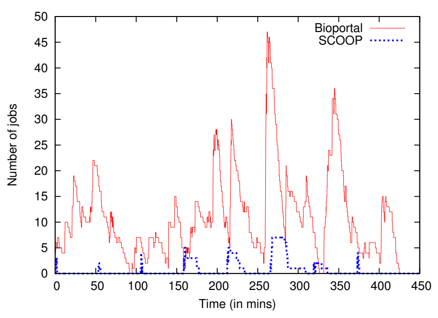
Figures 5 (a) and (b) show the distribution of resources among the two GROCs, illustrating the impact of site policy. This experiment is slightly different in that the Bioportal load submits jobs at a constant rate after it reaches its peak, producing a backlog in its queues. As more computation is allocated to serve the SCOOP burst, Bioportal's worker pool shrinks. The impact is greatest on Site C where Bioportal has lower priority. As SCOOP's load decreases, Bioportal procures more resources eventually reduces its backlog.

The GROCs adapt to changing demand by adding and removing worker nodes as the experiment progresses, using the mechanisms described in Section 3.4. Figure 5 (c) shows the completion times of configuration events across all three sites for an experiment similar to Figure 5. At the start of the experiment, each GROC leases and configures a master node at each of the three sites. These six nodes boot (*setup*) rapidly, but it takes about 336 seconds for the master *join* handler to copy the Globus distribution from a network server, and untar, build, install, and initialize it. As jobs arrive, the GROC also leases a group of six worker nodes. Once the master nodes are up, the workers *join* rapidly and begin executing jobs; as load continues to build, both GROCs issue more lease requests to grow their capacity. After each worker boots, it takes the GROC's worker *join* handler about 70 seconds to initialize the node with a private copy of Torque, and register it with its Torque master at the site. The GROCs permit some leases to expire as the queues clear; the *leave* (deregister) and *teardown* handlers complete rapidly. In this experiment, the Bioportal takes a while to clear its queued jobs, so the remainder of the *leaves* and *teardowns* occur later in the experiment.

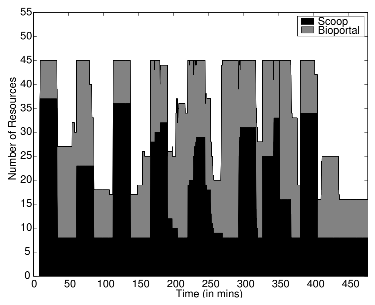
Adaptive provisioning with varying load. This experiment demonstrates adaptive resource provisioning by competing grids under a more realistic load signal. The Bioportal workload consists of a steady flow of jobs, with occasional spikes in job arrivals. The job arrival times were obtained from traces of a production compute cluster at Duke University. We scaled the load signals to a common basis



(a) Load signal



(b) Waiting jobs at Site A



(c) Resource holdings

Figure 6: Adaptive provisioning under varying load. The load signal (a) gives job arrivals. (b) shows the waiting jobs queue at Site A, while (c) shows a stacked plot of the resource holdings of each grid across the three sites.

that is appropriate for the size of our resource pools. The SCOOP workload runs a small set of ADCIRC jobs periodically according to a regular schedule. In practice, the resource demand for the runs in each period may vary according to weather conditions or post-processing results. For this experiment we use a synthetic load generator to create load spikes lasting a small time period (approximately 1 minute), at intervals of approximately 50 minutes. The duration of this experiment is 420 minutes and the lease length of each worker node is set to 25 minutes.

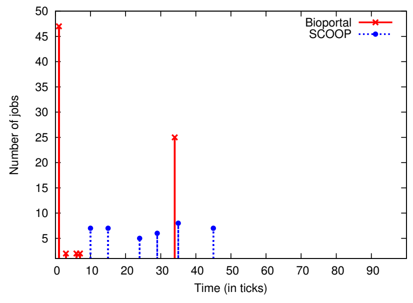
Figure 6 shows the load signal, the waiting jobs queued at Site A, and the resources that each GROC holds across the three sites. We see that each GROC is able to procure resources according to its varying load. SCOOP periodically demands resources to complete its runs, temporarily reducing Bioportal's resource holdings. However, Bioportal successfully retrieves resources between SCOOP's periods of activity. For simplicity, we omit the distribution of waiting jobs at Site B and Site C, which are similar to Site A.

Resource efficiency and lease length. The last experiment compares container-level control with job-level control with respect to efficiency and fairness of resource assignments to two competing VO grids. The power and generality of container-level resource control comes at a cost: it schedules resources at a coarser grain, and may yield schedules that are less efficient and/or less fair. In particular, a container holds any resources assigned to it even if they are idle—in our case, for the duration of its lease. Another container with work to do may be forced to wait for its competitor's leases to expire. Our purpose is to demonstrate and quantify this effect for illustrative scenarios.

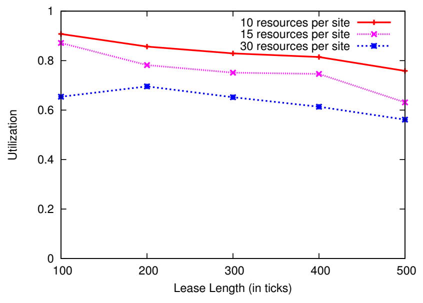
In this experiment, the job-level control is a standard First Come First Served (FCFS) shared batch scheduler at each site. The container-level policy is Dynamic Fair Share assignment of nodes to containers: the GROCs request resources on demand and have equal priority at all sites. Node configuration and job execution are emulated for speed and flexibility. We implement a grid emulator as a web service that emulates the Globus GRAM and MDS interfaces (job submission and status query) and also exports an interface to instantiate grid sites and add or remove worker nodes from a site. An external virtual clock drives the emulation. The site emulation incorporates a Maui scheduler with a modified resource manager module to emulate the job execution on worker nodes. Note that the core components (GROC, Shirako/COD, Maui) are identical to a real deployment. One difference is that the emulation preempts and requeues any job running on an expired worker node, although the batch scheduler configured in our prototype (Torque) does not support preemption.

Figure 7 (b) shows the *utilization* of container-level control with different lease lengths using a bursty load signal derived from a real workload trace (Figure 7 (a)) across different cluster sizes. We measure utilization as how effectively GROCs use their allocated resources: one minus the percentage of unused computational cycles. As lease length increases, container-level utilization decreases because the system is less agile and it takes longer for resources to switch GROCs. The decline is not necessarily monotonic: if the job and lease lengths are such that jobs complete just before the lease expires, then the Dynamic Fair Sharing container policy will redeploy the servers, maintaining high utilization. However, an advantage of longer leases is that they can reducing "thrashing" of resources among containers; in this emulation we treat the context switch cost as negligible, although it may be significant in practice due to initialization costs. Also, at smaller cluster sizes, resources become constrained, causing utilization to increase.

To compare job-level and container-level control, we also measure the efficiency of the resource pools. We define *efficiency* as one minus the percentage of usable resources that are wasted. A server is "wasted" when it sits idle while there is a job at the same site which could run on it. By this measure, the efficiency of a site-wide batch scheduler using

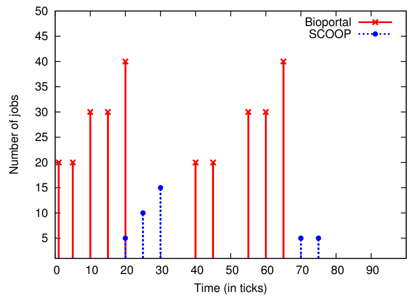


(a) Load signal used for experiment

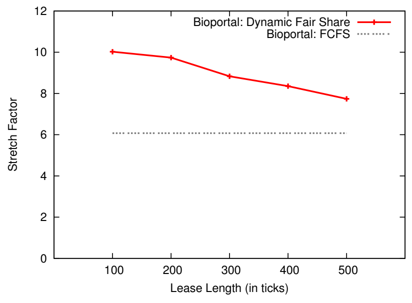


(b) Utilization with varying lease lengths

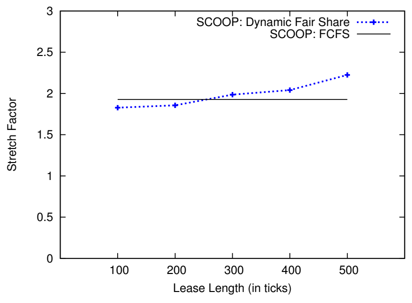
Figure 7: Efficiency of the system. (a) shows the load signal and (b) the variation of efficiency with lease length across multiple cluster sizes.



(a) Load signal used for experiment



(b) Fairness: Bioportal



(c) Fairness: SCOOP

Figure 8: Stretch factor, as a measure of fairness, of two competing GROCs.

FCFS is 100%, since it will always run the next job rather than leave a server idle. In contrast, a local batch scheduler running within a container may hold servers idle, even while another task scheduler in a different container has jobs waiting to run. For the given workload, in a resource constrained case (10 resources per site), the average efficiency across sites is 92% and in the overprovisioned case (30 resources per site) the average efficiency is 78%. As with utilization, efficiency is higher on smaller clusters since the GROCs are more constrained and may make better use of their resources. Efficiency is lower on larger clusters—but of course efficiency is less important when resources are overprovisioned.

Fairness is a closely related issue. One measure of fair resource allocation is the relative stretch factor of the jobs executed at a given provider site. Stretch factor is the ratio of completion time to job duration. That is, we might view a site as "fair" if a job incurs equivalent waiting time regardless of which grid submitted the job to the site. (Of course, the benefits of container-level resource control include support for differentiated service and performance isolation, which are "unfair" by this definition.) Both the FCFS job policy and the Dynamic Fair Share container policy strive to be "fair" in that they do not afford preferential treatment. Even so, these simple policies allow one of the GROCs to grab an unfair share of resources if a burst of work arrives while another is idle.

Figure 8 shows the average stretch factors for two job streams (Bioportal and SCOOP) running under both job-level and container-level resource control. Bioportal submits an initial burst of short jobs, which fill the global FCFS queue (for job-level control) or trigger lease requests for a block of servers (for container-level resource control). A subsequent burst of longer SCOOP jobs must wait for servers to become available. These bursts are followed by another pair of bursts of Bioportal and SCOOP jobs as shown in Figure 8 (a).

The Bioportal (Figure 8 (b)) shows a higher stretch factor than SCOOP (Figure 8 (c)) in all cases. In this particular scenario, the SCOOP bursts submit longer jobs to the queue, increasing the waiting time for the subsequent burst of Bioportal jobs. However, resource leasing can allow either workload to hold its resources longer so that some are still available for the next burst. In this case, longer leases improve the stretch factor for Bioportal and increase the stretch factor for SCOOP, improving fairness of the overall system.

In general, efficiency and fairness properties result from the interaction of the policy choices and the workload; it is less significant whether resource control is implemented at the job level or container level. A rich range of policies could be implemented at either level. The advantage of container-level control is that its policies generalize easily to any middleware environment hosted within the containers. On the other hand, the granularity of that control must be coarser to avoid sacrificing efficiency and utilization.

5 Related Work

To the best of our knowledge there is no prior work that uses dynamic resource pool resizing and multiple policy points to manage application resource requirements and resource provider policies in Grid sites. We provide a summary here of related work that have common elements with our effort.

Infrastructure sharing and Community delegation. Currently most deployed grid sites such as TeraGrid and OSG use static SLAs to enforce sharing policies. These policy choices need to be dynamic and adaptive to allow both providers and consumers to be able to adapt to varying load conditions. The grid hosting architecture provides this ability; the resource allocations result from the interactions of GROC request policies and site arbitration policies. Resource providers today use mechanisms like community accounts or virtual organization management to provide site autonomy and control over resources while trying to manage large number of users through delegation. Our approach is compatible with such approaches: it does not dictate how a hosted VO/grid implements its security policy for its users, or how it enforces policy at its sites.

Virtual execution environments. New virtual machine technology expands the opportunities for resource sharing that is flexible, reliable, and secure. Several projects have explored how to link virtual machines in virtual networks [12] and/or use virtualization to host grid applications, including SoftUDC [21], In Vigo [24], Collective [27], SODA [20], and Virtual Playgrounds [22] and DVC [31]. Shared network testbeds are another use for dynamic sharing of networked resources.

Schedulers, Meta-schedulers, Adaptation. Grid scheduling and adaptation techniques are used to evaluate system and application performance are to make scheduling and/or rescheduling decisions [4, 33]. Heuristic techniques are often used to qualitatively select and map resources to available resource pools [6, 23]. GROC is orthogonal to these specific techniques and can serve as a framework for an application manager to apply one or more of these techniques. Various site selection policies [11] and meta-schedulers [7, 18] are being explored in the context of the Grid. These provide an interface for applications to submit jobs to multiple sites. Our architecture allows application managers to implement policies for resource selection that are tied to the knowledge of the resources and the application requirements in the GROC. Leases in Shirako are also similar to soft-state advance reservations. Several works have proposed resource reservations with bounded duration for the purpose of controlling service quality in a grid. GARA includes support for advance reservations, brokered co-reservations, and adaptation [15, 16].

6 Conclusion

The increasing separation between resource providers and consumers makes resource control in today's grid both more important and more difficult.

This work illustrates the dynamic assignment of shared pools of computing resources to hosted grid environments. It shows how to extend grid management services to use a dynamic leasing service to acquire computational resources and integrate them into a grid environment in response to changing demand. In our prototype, each VO runs a private grid based on an instance of the Globus Toolkit (GT4) middleware running within a network of virtual machines at the provider sites. Each site controls a dynamic assignment of its local cluster resources to the locally hosted grid points of presence.

Our approach addresses resource control at the container level, independently of the grid software that runs within the container. Each GROC represents a hosted grid serving a VO, with points of presence at multiple provider sites. Each grid serves a different user community and controls its own internal environment and policies.

References

[1] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, A. Frohner, A. Gianoli, K. Lorentey, and F. Spataro. VOMS, an Authorization System for Virtual Organizations. In *First European Across Grids Conference, Santiago de compostela*, February 2003.

[2] Ant, September 2005. `http://ant.apache.org/`.

[3] P. Barham, B. Dragovic, K. Faser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, October 2003.

[4] F. Berman, H. Casanova, A. Chien, K. Cooper, H. Dail, A. Dasgupta, W. Deng, J. Dongarra, L. Johnsson, K. Kennedy, C. Koelbel, B. Liu, X. Liu, A. Mandal, G. Marin, M. Mazina, J. Mellor-Crummey, C. Mendes, A. Olugbile, M. Patel, D. Reed, Z. Shi, O. Sievert, H. Xia, and A. YarKhan. New Grid Scheduling and Rescheduling Methods in the GrADS Project. *International Journal of Parallel Programming (IJPP)*, Volume 33(2-3):pp. 209–229, 2005. Special issue on Next Generation Software.

[5] A. Blatecky, K. Gamiel, L. Ramakrishnan, D. Reed, and M. Reed. Building the Bioscience Gateway. In *Science*

Gateways: Common Community Interfaces to Grid Resources Workshop at Global Grid Forum 14 (GGF14), June 2005.

[6] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy. Task Scheduling Strategies for Workflow-based Applications in Grids. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*. IEEE Press, May 2005.

[7] R. Buyya, D. Abramson, and J. Giddy. Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid. In *Proceedings of the Fourth International Conference on High Performance Computing in Asia-Pacific Region (HPC-ASIA)*, May 2000.

[8] J. S. Chase, D. E. Irwin, L. E. Grit, J. D. Moore, and S. E. Sprenkle. Dynamic Virtual Clusters in a Grid Site Manager. In *Proceedings of the Twelfth International Symposium on High Performance Distributed Computing (HPDC-12)*, June 2003.

[9] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke. SNAP: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. In *8th Workshop on Job Scheduling Strategies for Parallel Processing*, July 2002.

[10] M. Degermark, T. Kohler, S. Pink, and O. Schelen. Advance Reservations for Predictive Service in the Internet. *Multimedia Systems*, 5(3):177–186, 1997.

[11] C. Dumitrescu and I. Foster. GRUBER: A Grid Resource SLA-based Broker. In *Proceedings of EuroPar*, September 2005.

[12] R. J. Figueiredo, P. A. Dinda, and F. Fortes. A Case For Grid Computing On Virtual Machines. In *International Conference on Distributed Computing Systems (ICDCS)*, May 2003.

[13] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. In *IFIP International Conference on Network and Parallel Computing*, pages 2–13, 2005.

[14] I. Foster, T. Freeman, K. Keahey, D. Scheftner, B. Sotomayor, and X. Zhang. Virtual Clusters for Grid Communities. In *International Symposium on Cluster Computing and the Grid*, May 2006.

[15] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation. In *Proceedings of the International Workshop on Quality of Service*, June 1999.

[16] I. Foster, A. Roy, and V. Sander. A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation. In *Proceedings of the 8th International Workshop on Quality of Service*, June 2000.

[17] Y. Fu, J. Chase, B. Chun, S. Schwab, and A. Vahdat. SHARP: An Architecture for Secure Resource Peering. In *Proceedings of the 19th ACM Symposium on Operating System Principles*, October 2003.

[18] GridWay Metascheduler. http://www.gridway.org/.

[19] D. Irwin, J. S. Chase, L. Grit, A. Yumerefendi, D. Becker, and K. G. Yocum. Sharing Networked Resources with Brokered Leases. In *Proceedings of the USENIX Technical Conference*, June 2006.

[20] X. Jiang and D. Xu. SODA: A Service-On-Demand Architecture for Application Service Hosting Utility Platforms. In *12th IEEE International Symposium on High Performance Distributed Computing*, June 2003.

[21] M. Kallahalla, M. Uysal, R. Swaminathan, D. Lowell, M. Wray, T. Christian, N. Edwards, C. Dalton, and F. Gittler. SoftUDC: A Software-Based Data Center for Utility Computing. In *Computer*, volume 37, pages 38–46. IEEE, November 2004.

[22] K. Keahey, K. Doering, and I. Foster. From Sandbox to Playground: Dynamic Virtual Environments in the Grid. In *5th International Workshop in Grid Computing*, November 2004.

[23] Y.-S. Kee, D. Logothetis, R. Huang, H. Casanova, and A. Chien. Efficient Resource Description and High Quality Selection for Virtual Grids. In *Proceedings of the Fifth IEEE Symposium on Cluster Computing and the Grid (CCGrid)*, May 2005.

[24] I. Krsul, A. Ganguly, J. Zhang, J. Fortes, and R. Figueiredo. VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing. In *Supercomputing*, October 2004.

[25] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A Community Authorization Service for Group Collaboration. In *Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*, June 2002.

[26] M. Ripeanu, M. Bowman, J. Chase, I. Foster, and M. Milenkovic. Globus and PlanetLab Resource Management Solutions Compared. In *Proceedings of the Thirteenth International Symposium on High Performance Distributed Computing (HPDC-13)*, June 2004.

[27] C. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum. Optimizing the Migration of Virtual Computers. In *5th Symposium on Operating Systems Design and Implementation*, December 2002.

[28] SCOOP website. http://www.renci.org/research/scoop.

[29] L. Smarr, A. Chien, T. DeFanti, J. Leigh, and P. Papadopoulos. The OptiPuter. *Communications of the Association for Computing Machinery*, 47(11), November 2003.

[30] A. Sundararaj and P. Dinda. Towards virtual networks for virtual machine grid computing. In *3rd USENIX Conference on Virtual Machine Technology*, 2004.

[31] N. Taesombut and A. Chien. Distributed Virtual Computers (DVC): Simplifying the Development of High Performance Grid Applications. In *Workshop on Grids and Advanced Networks*, April 2004.

[32] TeraGrid Science Gateway project at U. Chicago. *Special PRiority and Urgent Computing Environment (SPRUCE)*, February 2006. `http://spruce.uchicago.edu/`.

[33] J. S. Vetter and D. A. Reed. Real-time Performance Monitoring, Adaptive Control, and Interactive Steering of Computational Grids. In *International Journal of High Performance Computing Applications*, November 2000.