Beyond Virtual Data Centers: Toward an Open Resource Control Architecture*

Jeff Chase, Laura Grit, David Irwin,
Varun Marupadi, Piyush Shivam and Aydan Yumerefendi
Department of Computer Science
Duke University
Durham, NC

ABSTRACT

This paper summarizes recent research on networked virtual computing in the NICL lab at Duke. Over the past few years, we have constructed a service-oriented substrate for networked sharing and adaptive middleware environments based on a virtual on-demand computing. The goal of the project is to develop protocols and tools that can link together virtual computing clusters at different sites, and incorporate other kinds of resources (network tunnels, storage, and high-end computing resources) into a unified resource control plane for virtual computing. A key focus is to do this in a way that integrates well with existing middleware systems, to enable dynamic control of advanced virtual computing environments.

Our approach is based on foundational abstractions for leasing resources and factoring control over leased resources across the infrastructure providers, application controllers, and brokering intermediaries, interacting through common leasing protocols in an Open Resource Control Architecture (ORCA). The paper outlines the use of our resource leasing prototype (called Shirako) in several systems: as a foundation for manageable grid computing systems (based on Globus), in an interactive Web-based laboratory for autonomic services research (called Automat), and as a basis for virtualized cluster computing environments.

1. INTRODUCTION

Virtual computing offers a fundamentally new approach to sharing networked resources, enabled by advances in virtualizing server infrastructure. A notable example is the resurgence of virtual machine (VM) technology (e.g., Xen [2, 8] and VMware [25]). A recent Forrester Research report [13] states that 50% of global 2000 firms have server virtualization deployed or planned to deploy by mid-2007. More generally, clusters, blade arrays, and other scalable computing systems enable controlled resource partitioning on a coarser physical basis.

It is common today to manage sharing of networked resources through "middleware" such as grid computing software and job managers for networked clusters. The architectural choice to virtualize at the infrastructure level is a fundamentally new direction that is complementary to the wide range of existing middleware approaches. The value proposition is that it offers safe and effective sharing at a lower level of abstraction. For example, a user can obtain an on-demand private machine instance rather than the service of queuing their job to run on someone else's machine. In this example, the "raw" machine abstraction offers users on-demand access and control over personal, customized computing environments. Virtual computing systems can also offer assurances of predictable behavior and isolation from other users, using virtualization technologies to control the binding of guest environments to physical hosting resources.

Virtual computing depends on a new foundational layer of control software that instantiates and manipulates the contexts in which the applications, middleware, and machine operating systems run. Since it controls the bottom of the software stack closest to the hardware, we refer to this new virtualization and control layer as *underware* to distinguish it from classical middleware. A key characteristic of underware is that it is not visible to users or applications at the top of the software stack, other than to provide a more stable and controlled environment for them to run in. Its role is to protect and enhance the investments in higher-level software (including middleware), simplify infrastructure management, and facilitate access.

Virtual computing systems are already mature enough to offer tangible and compelling benefits in practical deployments today. Even so, the technology is at an early stage. Virtual computing "underware" is a new market segment and also a rich area for computing systems research. It offers a rare opportunity to rethink the software architecture and leverage the new capabilities in a way that applies the benefits broadly to a full range of computing environments—including existing middleware systems for grid and cluster computing, Web services and systems not yet imagined.

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

Here is a partial list of functional demands that shape the key architectural choices:

- *Managing thousands or millions of dynamic machine instances.* The leading VM systems support live migration and checkpoint/restart, strong performance isolation, and fine-grained allocation of server resources as a measured and metered quantity. Machines are no longer bound to physical hardware; VMs may be created in seconds using techniques such as flash cloning of stored images, and then paused, saved, restored, or migrated at the push of a button. How should the underware control plane expose, leverage, and manage these new capabilities?
- *Adaptive applications.* The "guests" hosted on virtual computing utilities will include long-running applications and services. Some of these will be mission-critical services that must maintain service quality 24x7 across flash crowds, faults, and other changes. The underware should export programmatic, service-oriented interfaces for self-managing guests to negotiate for resources and configure them on-the-fly.
- *Autonomous infrastructure providers.* Virtual computing underware should offer a unified interface for guests to obtain resources from many infrastructure providers contributing resources to a shared pool. Because of the difficulty of managing federation, first-generation systems such as PlanetLab [3] have limited their initial deployments to centrally controlled infrastructure. But we want to build resource sharing federations that grow organically, and any such system must protect the autonomy of infrastructure providers to control how their resources are used.
- *Resource control policy.* On-demand access is wonderful until the resource runs out. A key challenge today is to develop practical *policies* for adaptive and reliable allocation of networked computing resources from a common pool. Good arbitration policies would assign resources to their "highest and best use" while balancing the needs and interests of resource providers and resource consumers (guests). Independent policy control by each participant is the foundation for sustainable economic structures with contractual quality of service and accountability.

Virtual computing underware should meet these needs in a way that integrates well with existing middleware systems. For example, grid computing will not truly break out of the lab until we have a new generation of grid systems that offer reliable and predictable performance in a manageable package. We believe that the key is to combine job execution middleware with virtual computing underware—but the "right" factoring of functions across the layers is still a research topic.

Finally, the software architecture for virtual computing should generalize to the full range of infrastructure assets. Virtual computing today is mostly about servers, but virtualization techonologies are also advancing for wide-area networks (lambda provisioning and virtual layer-2 links such as MPLS) and for network storage (e.g., [18, 15]).

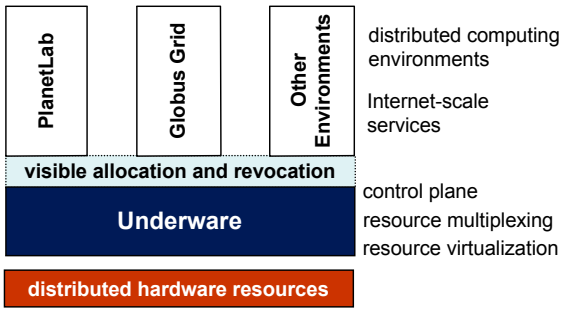

Figure 1: Multiple software environments (guests) are hosted on a common pool of virtualized resources from infrastructure providers. The "underware" control plane exports protocols and interfaces for guest "controllers" to negotiate for resource lease contracts on the shared infrastructure.

Our research envisions an underware control plane linking providers and consumers of infrastructure resources, including virtual computing clusters but also other on-demand resources such as storage, network channels, and even software licenses. Our approach is based on a common core of services for the foundational abstraction of *resource leasing*. Guests use open leasing protocols to acquire and coordinate underlying resources for efficient and reliable execution, optionally assisted by brokering intermediaries. The system is based on a service-oriented framework that reflects the dynamic trust relationships and factoring of policy control across the various stakeholders—an Open Resource Control Architecture (Orca).

This paper outlines the architecture and rationale for an Orca control plane, describes our current prototype, and summarizes ongoing work. We focus on our experiments in using lease-based resource management as a common foundation for different middleware systems and usage scenarios for programmable hosting centers. To illustrate our approach to dynamic configuration and adaptive resource control, we describe Automat, an interactive Web-based laboratory for autonomic services research using the common control plane. We also summarize how we have used the prototype as a substrate for a manageable grid hosting system (using Globus), a private PlanetLab deployment, and JAWS (Job-Aware Workspace Service), a new approach to job execution for virtual cluster computing.

2. OVERVIEW AND DESIGN

The Orca vision may be viewed as a sort of "Internet operating system" to multiplex diverse user environments on a common pool of hardware resources, which may be contributed by multiple providers. The hosted environments (guests) may range from simple personal computing systems to large-scale network services to programming platforms for distributed computing, such as grid middleware. The guest software systems are typically designed to run on dedicated hardware, but can just as easily run on virtualized resources. Virtual computing allows a given resource pool to host any of these diverse environments, possibly at different times, and with changing resource assignments to respond to changing demands. The control plane exposes programmatic leasing

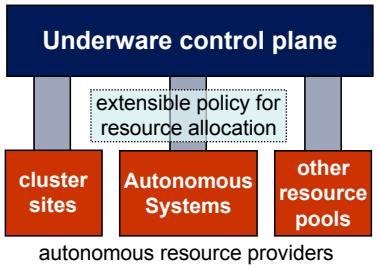

Figure 2: Policies for allocating hardware resources are outside the underware resource control plane. Resource providers control the policies by which resources are used.

interfaces to request, release, grant, and withdraw resources, as depicted in Figure 1.

The behavior of the guests is determined by the software stacks installed within their virtual execution contexts, which are customized for each guest. The underware layer exports "raw" resource abstractions and concerns itself narrowly with managing physical resources and coordinating resource configuration steps. The means to specify, implement, and harmonize policies for local resource management are left to the infrastructure providers. Figure 2 depicts resource sharing and arbitration policies residing outside the control plane and within the providers.

To enable this structure, the Orca architecture embodies four key design principles:

- *Sustainable structure and autonomy.* All participants have the ability to quantify and control what they contribute to a system and what they obtain from it through time. The structure protects their autonomy to exercise this control according to local policies.
- *Negotiated contracts with varying degrees of assurance including but not limited to strong isolation (reservations).* The control plane provides a means to form *contracts* for specific resource allocations at specific times. For example, many mission-critical applications depend on differential service quality assurances (e.g., Internet 911 and disaster response). Different providers and virtualization technologies may offer varying degrees of isolation, but the contracts are explicit about the assurances they offer.
- *Neutrality for applications and resources.* The architecture generalizes to a wide range of hosted applications or services. It provides extensible programmatic interfaces to instantiate, configure, and control a wide range of software environments on a wide range of resources through an elemental leasing abstraction (the "narrow waist" of the architecture).
- *Policy neutrality with a balance of local autonomy and global coordination.* Orca must be free of resource allocation policy: needs and capabilities evolve with time. Consumers determine their policies for requesting resources, and providers determine arbitration policy for the resource pools under their control. Providers may delegate resources to groups or communities, who may subdivide their holdings via brokering intermediaries that represent their policies. The system has no central point of trust or control.

Cluster-on-Demand and Shirako. The starting point for Orca is a pair of integrated systems created in our previous research: Cluster-on-Demand (COD) and the Shirako leasing core, described below. The COD project began as an outgrowth of work on dynamic feedback-controlled resource provisioning in hosting centers [6, 9]. COD exports a service to allocate and configure *virtual clusters* from a shared server cluster. Each virtual cluster comprises a dynamic set of nodes and associated resources assigned to some guest 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. Typically the leased virtual clusters have an assurance of performance isolation: the nodes are either physical servers or Xen [2] virtual machines with assigned shares of node resources. The first COD protoype [7] was funded by IBM and the NSF Middleware Initiative.

CODv1 had an *ad hoc* leasing model with built-in resource dependencies, a weak separation of policy and mechanism, and no ability to delegate or extend provisioning policy or to coordinate resource usage across multiple sites. Our experience with the COD prototype convinced us of the need for an automated and service-oriented approach to resource management and configuration for virtual on-demand computing environments. In particular, these systems must incorporate sound control policies to assign and provision resources dynamically, arbitrate contending requests, repair faults, and adapt to changing conditions. Much of our follow-on work to COD addresses these policy and control issues as a cornerstone of our research agenda in autonomic services and autonomic infrastructure, particularly in the context of infrastructures with many owners, e.g., a federation of networked on-demand clusters.

Leases and Actors. In 2003 we began to rearchitect the COD system around a general leasing contract model that is resource-independent and policy-neutral (Sharp [12]). A *resource lease* is a signed contract granting the holder rights to exclusive control over some quantity of a resource over a specific period of time. Leases are dynamic and renewable by mutual consent between the resource provider and holder. The leasing abstraction applies to any set of computing resources that is "virtualized" in the sense that it is partitionable as a measured quantity. For example, an allocated instance of a resource might comprise some amount of CPU power, memory, storage capability, and/or network capability, measured by some standard units, and with attributes to describe resources and the degree of isolation.

The participants in the leasing protocols are services representing different stakeholders in the system, and incorporating their local policies. Actors can take any of three roles within the system:

- Each guest is represented by an actor that monitors application demands and resource status, and negoti-

ates to acquire leases for the mix of resources needed to host the guest. Each *service manager* requests and maintains leases on behalf of one or more guests, driven by its own knowledge of application behavior and demand. The *guest controller* is a policy module in the service manager that controls these choices.

- An *authority* controls resource allocation for an infrastructure provider with direct control over a pool (aggregate) of resources in some *site* or administrative *domain*. The authority is responsible for enforcing isolation among guests hosted on the resources under its control. COD runs as a site authority.
- *Brokers* maintain inventories of resources offered by domains, and match requests with their resource supply. A domain may maintain its own broker to keep control of its resources, or delegate partial, temporary control to third-party brokers that aggregate resource inventories from multiple sites.

We built the second-generation COD system around an open leasing core called Shirako [17]. Shirako is a Java-based toolkit for building resource leasing services using Web services technologies. It is designed as a common core with lease state machines, and plugin interfaces for application-specific, resource-specific, or policy-specific components. Shirako is one technology for developing Orca actors, but Orca is "open" in the sense that any actor may join the network if it speaks the leasing protocols (SOAP with WS-Security, or XMLRPC) and common conventions for describing resources and their configuration. The protocols as implemented in Shirako are compatible with self-certifying secure tickets and accountable delegation developed in Sharp [12].

Shirako actors may be deployed programmatically to a Web container, and a Web portal offers controls for authorized users to create and manipulate actors. These actors may represent different trust domains and identities, and may enter into various trust relationships or contracts with other actors. Actors establish trust in a decentralized fashion. Broker operators may establish trust pairings with sites or guests, to serve resources on behalf of a site, or to authorize a guest to access its resource holdings according to a broker arbitration policy. Where manual control is desired, operators may use the Web portal to register public keys of approved peers, grant or deny resource requests, and specify the types, quantities, and time periods of resources shared with peers.

Resources, handlers, and drivers. The leasing abstraction can apply to a wide range of physical assets. Virtualized infrastructure exports control points to partition and configure the underlying physical resources. Some network and storage elements export control actions to the network via protocols such as SNMP, Web Services protocols, or (more frequently) by remote login to a command-line interface from a privileged IP address.

In Shirako, these underlying control operations are invoked by *handler* and *resource driver* plugins invoked from the core on lease state transitions. Many control actions are invoked through a standard *node agent* that runs on a node, e.g., in a guest VM, a control VM (Xen dom0), or a control node for some other substrate component. A *node driver* is a packaged set of actions that run under the node agent to perform resource-specific configuration on a node. The node agent accepts signed requests from an authorized actor on the network, e.g., using SOAP/WS-Security or XMLRPC, including requests to install, upgrade, and invoke driver packages. *Handlers* are action scripts that run within an actor to configure or tear down a logical resource unit, which may involve multiple physical resources, possibly covered by separate leases. The core upcalls prologue/epilogue handler actions on lease state transitions to notify the actor of changes to the resource allotment; these map to a registered handler action for the given resource type. We script handlers using *ant*, which can be invoked directly from the Java core and supports direct invocation of SNMP, LDAP, Web container management, and node driver actions on the node agents. The various actors control the behavior of handlers and drivers by means of property lists passed through the leasing protocols and core.

Cluster-on-Demand as a plugin. The rearchitected COD [7, 17] comprises a set of handlers, drivers, and other plugins specific to managing clusters of physical hosts and virtual machines. It handles imaging, booting, and node post-install, allocation of IP subnets and addresses, and DNS subdomains and names. COD includes drivers and handlers to coordinate with standard services for site administration (DHCP, DNS, and client-side PAM/NSS and NFS automount) patched to read configuration information from an LDAP-based Network Information Service (RFC 2307). It supports VM configuration through drivers that run under a node agent in the Xen control domains (dom0). At the Duke site we flash-clone images on a Network Appliance filer and export them to VMs over the network; with this technology COD can instantiate a Xen VM in about 20 seconds.

Controllers. Shirako provides interfaces to define and compose *controller* plugins for generic actors for the various roles. A controller is a clocked policy module that makes choices to meet its objectives under changing conditions: what resources to request (in the guest's service manager) and what configuration options to use for them, which requests to grant and how to schedule them (in a broker), and how to map granted requests on to actual resources (in a site authority). Controllers specify the handlers they use for each type of resource.

A fundamental issue is the factoring of complex control functions across the actor roles. For example, Shirako/COD supports fine-grained sliver resizing on lease renewal boundaries, and policy-driven suspend, resume, checkpoint, and migration [8] of VM instances. These capabilities enable rapid VM creation, more robust computing, and the flexibility to adapt to changing conditions while retaining continuity and liveness. However, migration may be driven from a service manager, broker, or authority for different reasons. The factoring must reflect access to the underlying resource control points.

3. USAGE SCENARIOS AND SYSTEMS

Figure 3: View for service controller of the Rubis e-commerce service in the Automat Web portal.

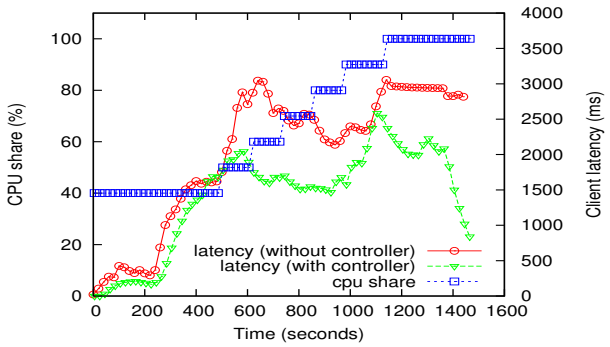

Figure 4: Comparing behavior of controllers for the Rubis e-commerce service under dynamic load.

This section outlines several examples of uses of the prototype as a substrate for advanced middleware environments. These systems are works-in-progress that illustrate the power and generality of flexible resource leasing as a foundation for a unified resource control plane.

3.1 Automat

We are using Shirako as a substrate for a web-based interactive laboratory targeted for research into the mechanisms and policies for *autonomic hosting centers* that configure, monitor, optimize, diagnose, and repair themselves under closed-loop control (a collaboration with Shivnath Babu). The prototype, called Automat[1], allows users to allocate resources from an on-demand server cluster to explore the interactions of application services, workloads, faultloads, and controllers [26]. Users interact with a Web portal to upload and deploy guests and controllers, subject them to programmed test scenarios, and record and display selected measures as the experiment unfolds. Users can install their own services, and experiment with pluggable controller modules for sense-and-respond monitoring and adaptation by hosted services and, crucially, by the hosting center itself. Users exploring self-management at the hosting center level can instantiate *virtual data centers* and define controllers that govern resource arbitration, resource selection, and dynamic migration. To drive feedback control policies, Automat provides a common, but replaceable, instrumentation plane based on Ganglia [21] to capture, propagate, and store streams of system-level and application-level metrics.

One goal of Automat is to improve researcher productivity by enabling researchers to focus on the novel aspects of their work, rather than on the infrastructure needed to realize their objectives. Our premise is that a standard "harness" to package and share test applications, workloads, faultloads [4], and system prototypes—and deploy them at the push of a button—will catalyze the growth of a repository of shared test cases and approaches within the community. A successful testbed can enable repeatable experiments, provide a common means to measure progress toward shared goals, facilitate sharing of research outputs, and establish paths for technology transfer.

Automat defines a common framework for developing, installing, upgrading, and composing Shirako controllers written in Java. Controllers are driven by a clocking signal and may use Automat-provided classes for calendar scheduling, resource allocation, processing of instrumentation streams, and common heuristics for optimization and placement. In addition, each controller registers to receive upcalls as new resources join or leave its domain, and to notify it of changes in resource status. Guest controllers may also invoke driver actions in the guest, either directly from Java or using an XML scripting interface for Shirako handlers based on Ant.

We have prototyped the basic mechanisms for an extensible Automat Web portal using Velocity and Java under Tomcat, as a front end to site authorities, brokers, and service managers based on the Shirako leasing core. Automat controllers may include *view* extensions as plugins to the portal interface, enabling users to monitor and interact with controllers during an experiment. Controllers, views, and guests are packaged in *extension packages* containing Java archives, presentation templates (e.g., Velocity scripts), guest installation files, etc. Extensions are uploadable through the Automat portal, which offers menus to instantiate and configure the registered packages.

For example, Rubis [5] is a Web application that implements a simple auction website. We developed a guest controller for Rubis that acquires resources from a virtual data center, instantiates the Rubis service, subscribes to the service's performance metrics, and uses feedback control to adapt resource allocation under varying workload. The workload is driven by another guest controller that provisions, launches, and controls a workload generator for Rubis. The controllers interact to sequence the experiment. Figure 3 shows a screenshot of the view for a guest controller, showing the status of the server allocated to this guest, metrics subscribed to by the controller, and control actions that can be taken on the guest. The screenshot in Figure 3 was taken from an experiment on Automat in which we studied controller effectiveness under a varying workload.

Users may upgrade controller modules and their views dy-

[1]An Automat is an automated vending facility introduced in the 1930s, in which clients select from menus of precooked items and compose them to form a meal. In this case, the "meal" is a reproducible experiment drawn from a menu of packaged test scenarios and dynamic controllers instantiated on demand in a virtual data center. The Automat project is funded through CNS-0720829.

namically. Our design uses a custom classloader in a Java-based Web application server to introduce user-supplied controller code into the portal service. Views interact with the controller using a management proxy supplied as part of the controller implementation. The proxy always runs within the portal service, but the controller code itself and its actor may run under a separate JVM communicating using SOAP or XMLRPC. In our experiments, the portal and control domain with all controllers run within a single JVM under a Tomcat instance.

3.2 Globus

One goal of our work is to instantiate complete middleware environments within isolated logical containers (workspaces) comprising sets of virtual machines, and show how add-on controllers for each hosted environment can invoke the leasing services to grow and shrink the computing resources assigned to their containers. In a collaboration with Lavanya Ramakrishnan at RENCI, we demonstrated adaptive hosting of multiple Globus grid instances on networked clusters using a Shirako resource control plane [23]. This work was demonstrated in the RENCI booth at Supercomputing/SC06.

Globus grids support familiar abstractions: they allow users to run their jobs and workflows on somebody else's operating system. Globus also provides services to establish a common notion of identity, a common distributed middleware for routing jobs and scheduling them on local resources. Globus provides a uniform sharing and execution model, but because it is middleware, and does not control operating systems or hardware resources, it has limited control over resource management. QoS, reservations, and flexible site control are important for grid computing, but they have been elusive in practice. For example, Globus can only control when to submit jobs to queues or operating systems; it cannot control what resources are allocated by the lower layer, unless the lower layer provides those hooks. In our approach, the underware control plane offers those hooks to a controller for the hosted grid environment.

We implemented a guest controller called a Grid Resource Oversight Controller (GROC) that invokes Globus interfaces to monitor and control a Globus grid, and issues leasing requests through the control plane to modulate the resources available to the grid. If a backlog of jobs accumulates in the job queues, the GROC leases additional server resources according to its policy, and integrates them into the batch job service instance at the grid's point-of-presence at the hosting site. Figure 5 illustrates this dynamic resource recruitment in a typical scenario.

The underware layer makes it possible for a controller to support dynamic provisioning, advance reservations, and site control over resource sharing without modifying Globus itself. In the Globus experiment we found it is useful for each guest controller to pass supplementary information to the resource controller with their requests. For example, the lower-priority Globus guest controller indicates that it will accept fewer resource units if the resource controller cannot satisfy its full request. We use an extensible property list mechanism: the Globus guest controller sends a property, `elastic`, to indicate its willingness to accept fewer resources

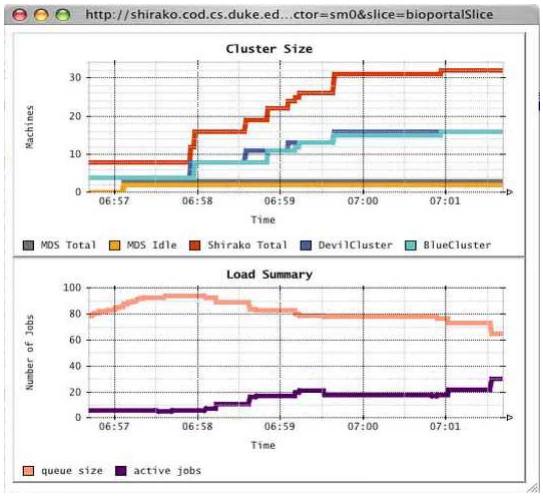

Figure 5: Screenshot showing how a Globus guest controller leases additional server resources to clear a backlog of jobs.

to the resource controller. Other useful properties include `deferrable` to specify that the resource controller can defer the start time of the lease, and `deadline` to indicate to the arbitration policy that a request must be filled by some deadline.

3.3 Jaws

The simple approach described for the Globus experiment above allows multiple user communities and middleware environments to share computing resources with policy-based adaptive resource control. However, it is limited in that virtual machine control interfaces (e.g., checkpoint/, restore, migrate) do not apply to individual jobs, but only to complete middleware environments. The middleware controls job scheduling on its machines; the mapping of jobs to machines is not known outside of the middleware, and the scheduler may place multiple jobs on a machine. Thus it is not possible for a controller to migrate a job by migrating the containing VM, or use VM control interfaces to suspend/resume, save/restore, or reserve resources for individual jobs.

Virtual machines introduce new opportunities for flexible job management in batch computing systems. If VM instantiation is cheap, it becomes possible to run each job within a private virtual machine workspace [10, 20, 19, 11]. With this model, it becomes possible to customize the workspace for the needs of specific jobs, and essential checkpointing and migration features are supported in a general way at the VM level. For example, future systems must deal with more diverse software configurations, e.g., different library versions, operating systems, or middleware services. To harness computing resources for a given task, it is necessary to install and configure the correct software stack. A virtual workspace provides access to a collection of resources (physical or virtual) and a software environment suitable for the execution of a job or a collection of jobs.

Our collaborators in the Globus Workspaces project are inte-

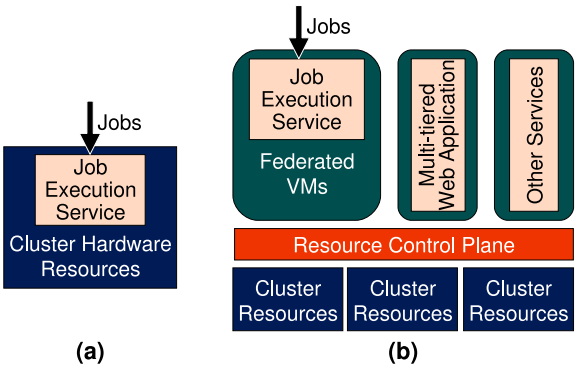

Figure 6: A classic job execution service running on a cluster's physical machines (a) compared to a job execution service running on federated virtual machines above a generic resource control plane and sharing underlying resources with other cluster services (b).

grating support for virtual computing into Globus [19]. Condor middleware system are taking a similar approach [24].

Virtualization technology offers a rare opportunity to rethink the software architecture from the ground up. We are developing a streamlined job execution system called JAWS (Job-Aware Workspace Service) that uses the resource control plane to obtain isolated virtual workspaces sized for each job. The key architectural principle is to separate control over resource sharing from job execution and management. Although it manages job execution, JAWS is not a job scheduler: all functions to schedule and arbitrate shared resources migrate to the underware layer. In this way, a JAWS job service can share a common pool of networked cluster resources with other cluster services, including other middleware environments such as grid software or applications such as Web services, as depicted in Figure 6. Decoupling sharing control and job management simplifies the job execution service. The underware layer takes care of the underlying trust and authorization issues, and resource arbitration policy.

JAWS executes each job within the customized workspace instantiated by the site authority. Our JAWS prototype uses Plush [1] to execute a job on newly instantiated VMs. Job workflows are described in an XML specification language that details data staging, software installation, and process sequencing. Plush monitors the execution of the job and reports, via XML-RPC callbacks, job completion or failure.

Each JAWS job executes in one or more virtual machines, each bound to a "sliver" of resources provisioned for the job. A sliver is the partition of a physical machine's resources; leases in JAWS are for a set of slivers which form a virtual workspace. After the initial slivering, a virtual workspace may require resizing to accommodate for the changing requirements of a job or to meet client performance targets. Virtual machines may be resized along multiple dimensions to enlarge or shrink a job's virtual workspace. Resizing may involve migration to a new host.

Since JAWS jobs run within a leased performance-isolated workspace, they are protected from contention effects while their leases are valid. This predictability is important because, increasingly, high-end computational science and engineering problems are solved on a complex hierarchy of resource and software systems that consist of scientific codes, portals, workflow tools, web services, resource management middleware, and underlying cluster and HPC resources. In this setting, effective execution depends on a complex set of interactions between the end-user, different layers of software infrastructure, and the underlying resources. Contention in one part of a complex workflow can gate the progress of the entire workflow. In contrast, job-level and process-level resource control on mainstream cluster operating systems offer limited predictability and isolation, making it difficult to orchestrate large composite applications effectively, particularly for deadlines or dynamic behavior.

A Shirako broker acts as the resource scheduler for JAWS. In our prototype the broker schedules leases for virtual machines bound to varying amounts of CPU, memory, bandwidth, and local storage from an inventory of physical hosts. In many respects the broker behaves like a conventional job scheduler: we have implemented many commonly used scheduling algorithms as pluggable policies for Shirako brokers, including Earliest Deadline First, FCFS, and a new integrated proportional share/backfill algorithm called Winks.

JAWS can leverage software environments packaged as *appliances*: complete bootable images containing both application and (custom) operating system components. The appliance model is well-suited to virtualized infrastructure, and has revolutionary potential to reduce integration costs for software producers and to simplify configuration management of software systems. For example, rPath [2] is packaging appliances for key driving applications, including the STAR application from Brookhaven and the Community Climate System Model, and supporting infrastructure such as the Open Science Grid toolset.

3.4 PlanetLab and Plush

PlanetLab [3] is a widely used network testbed for large-scale network services (`http://www.planet-lab.org`). It controls a set of dedicated hardware resources, which it manages and images centrally. PlanetLab provides abstractions and system services (e.g., distributed virtualization, resource discovery, monitoring) to enable deployment of widely distributed applications that control their own communication patterns and overlay topology. Much of the common API is provided by a Linux kernel flavor mandated by PlanetLab. PlanetLab applications may run continously and adaptively on varying resource allotments. PlanetLab established a "best effort" resource management model as a fundamental architectural choice.

We adapted the PlanetLab platform to use Shirako interfaces to a resource control plane to lease shared hardware resources. MyPLC is a downloadable PlanetLab software package for use in creating private PlanetLab instances. Each PlanetLab instance has a "PLC" central coordinator server that monitors and controls all participating nodes and co-

[2] `http://www.rpath.com`

ordinates application deployment on the testbed. We completed the integration to run PlanetLab kernels on Xen virtual machines; this required minor modifications to the PlanetLab boot process along with a recent patch to enable *kexec* in Xen-capable kernels. We wrote a small wrapper that leases resources from a Shirako broker and instantiates a MyPLC central server and one or more PL hosts, using the new PLC API to add and remove nodes from the MyPLC instance. With support from the PlanetLab organization, this system could also be used to add and remove local machines from the public PlanetLab, enabling cluster sites to contribute resources to PlanetLab on a temporary basis.

Because PlanetLab resource allocations are uncertain and unstable, PlanetLab applications must already be capable of adapting to changing resource availability. To provide application adaptivity we again use Plush [1], which was originally developed to ease application and service deployment on PlanetLab. Plush allows unmodified applications under its control to adapt to PlanetLab's volatile environment. It uses a resource discovery service (SWORD [22]) to match (and re-match) an application to a suitable set of resources, and abstract application deployment and execution using an XML descriptor for software installation, process execution workflows (i.e., the process execution order), and failure modes (i.e., what should I do if something fails). Plush exports an XML-RPC interface that allows control software to add or remove resource instances from a running application at any time; although, applications must include support for using resources once added. We have also extended Plush to obtain resources directly through Shirako, in a collaboration with Jeannie Albrecht. This step allows Plush-based PlanetLab applications to run directly on a Shirako system, which gives them assurances about the resources under their control.

4. RESOURCE CONTROL POLICY

A key premise of our work is that resource control and strong resource contracts are essential to deliver on the promise of distributed virtual computing. For example, consider an on-demand system that instantiates best-effort virtual machines, with no assurance that they will make forward progress. The provisioning problem is trivial in this system: no admission control or resource arbitration is needed. The placement choice might be left to users, e.g., to seek out hosts that appear lightly loaded at any given time. Alternatively, the providers or brokers might coordinate placement to balance the load.

Now consider a system whose goal is to assure a predictable level of service quality for its guests. Best-effort contracts might be adequate if the system is sufficiently over-provisioned. The alternative is to make promises to each consumer about the resources it will receive—how much and/or when. Orca represents such promises in the lease term and in the lease attributes.

The leading VM systems, including Xen, support the resource control mechanisms necessary to back these stronger contracts. Performance-isolating schedulers permit fine-grained allocation of host server resources as a measured and metered quantity: each guest VM is bound to a *sliver* of host resources sized along multiple dimensions (e.g., CPU capacity, memory, and network bandwidth). Slivers are sized to meet performance targets in the guest while minimizing the "crosstalk" from competing VMs on the same host. Support for VM migration [8] provides the flexibility to satisfy new and unforeseen resource demands while retaining the continuity, liveness, and sliver sizing of existing VMs.

These technologies are reasonably mature, although refinement continues. They motivate a stronger focus on the policy implications and the significant research challenges that they raise. Resource control features are essential mechanisms, but someone or something must control how they are used, e.g., how to size the slivers, where to place the VMs, and when and where to migrate. These capabilities create a rich policy space for system management infrastructures. It is straightforward to expose these choices to human operators through a control panel, but the grand challenge is a self-managing compute utility in which the control functions are automated and respond to changes in real time. Resource management policy is difficult for at least three reasons:

- *It is heuristic.* Resource management involves projections under uncertainty and optimization problems that are NP-hard in their general form, forcing us to adopt heuristics tailored for specific needs and settings. There is no "one size fits all" solution.
- *It is dynamic.* Resource allocation policies must adapt to changing workload and demands in real time.
- *It is organic and emergent.* Policy choices must balance the needs and interests of multiple independent stakeholders, e.g., resource providers and resource consumers or hosted guests. In *federated* systems—in which independent providers contribute resources to a shared pool—brokering intermediaries may also play a role to supplant the need for pairwise peering arrangements and/or to implement community sharing policies. In general, the resource assignment emerges from choices taken by each of these actors, and the complex interactions of those choices.

In the Orca architecture, resource controller modules in brokers control resource *provisioning* (allocation of quantities of resources over time), while site authorities control and *assignment* of specific resource units to approved leases, e.g., *placement* of active VM images within the server network. We recently extended Shirako/COD to support fine-grained sliver sizing along multiple dimensions (e.g., the amount of CPU cycles and network bandwidth bound to a sliver), sliver resizing on lease renewal, and policy-driven migration to adapt to changing demands [14]. This required minor changes to the Shirako policy interface, as well as new features to protect the accountability of contracts, which are beyond the scope of this paper. To support virtual machine migration and resizing we extended the event handler set for site authority resource drivers with a new event: *modify*. The *modify* event handler for Xen virtual machines invokes Xen primitives to resize or migrate a target VM, guided by a property list passed from a policy module.

The basic structure of resource management policy is common to a range of visions for networked virtual computing, encompassing managed data centers, network testbeds, grid computing systems, and market-based utilities. With the right factoring of policy and mechanism, these systems can build on the same underlying resource management substrate. Most of the differences among competing approaches to distributed virtual computing boil down to matters of policy, or questions of who the players are and how much power they have, or differing application assumptions that ultimately have little impact on the underlying resource management requirements. These superficial differences leave open the opportunity for a common management "fabric". In particular, as networked utilities increase in scale, market-based control becomes attractive as a basis for resource allocation that is fair, flexible, adaptive, and decentralized. Strong resource contracts enable negotiated, accountable exchanges in a marketplace. We have used Shirako to experiment with market-based virtual machine hosting based on a self-recharging virtual currency [16].

It is an open question what range of resource contracts are practical with virtualization technologies, e.g., with existing and future VM schedulers. Like many of our colleagues, we are evaluating the fidelity of resource control schemes in the Xen hypervisor and interactions with guest OS kernels (e.g., for dynamic memory provisioning).

5. CONCLUSIONS AND ONGOING WORK

Virtualization promises to change the way the server backbone is managed at the lowest level. Orca has potential to advance the foundations for networked resource sharing systems that can support a wider range of resource types, management policies and services, relationships among resource providers and consumers, and service models including but not limited to best-effort service.

Our premise is that a common Orca substrate can meet the evolving needs of several strains of systems for networked resource sharing—whether the resources are held in common by a community of shareholders, offered as a commercial hosting service to paying customers, or contributed in a reciprocal fashion by self-interested peers.

Orca can reduce barriers to sharing networked resources and services. Infrastructure providers may contribute resources in a controlled way, and potential users can harness those resources effectively for a wider range of activities by automated install of suitable software stacks on a temporary basis. The resource owners may be isolated from the management responsibilities for the guest services that they host, which devolve to the lease holder (e.g., endorsement and mapping of user identities). It also becomes possible to factor many resource management and security concerns out of the middleware, and address them in a common foundational layer.

An important goal of the project is to develop new handlers, drivers, and controllers to broaden the range of resource types beyond cluster nodes, VMs, and simple storage object. Examples include network name space management, repository space for storage objects and saved images, dynamic configuration of storage systems (e.g., Lustre) on collections of leased disk partitions or luns, constructing and managing virtual networks (overlays) consisting of multiple points of presence (virtual clusters at different sites) linked by tunnels. It may be possible to linking supercomputing resources into an Orca control plane.

We are continuing to investigate fundamental architectural issues raised by our approach: sensitivity of the leasing model to a global notion of time; accountability for contracts; protection of guests from faulty hosts and vice versa; techniques and overheads for resource partitioning and virtualization; coordinated configuration of complex aggregate resource sets (e.g., network overlays) within the leasing framework; automatic control policies for resource arbitration and adaptation, and their interactions; attribute-based resource matching and locality; decentralized trust management; resource ontologies; scalability to very large infrastructures (e.g., by emulation experiments); and policy control for complex choices enabled by virtual machine technology, e.g., migration and save/restore as a basis for robust, adaptive computing. Our development agenda in the core will focus on robust multi-actor state management and event tracking, since failures may occur at many interacting points and levels.

6. REFERENCES

[1] J. Albrecht, C. Tuttle, A. Snoeren, and A. Vahdat. PlanetLab Application Management Using Plush. *ACM Operating Systems Review (SIGOPS-OSR)*, 40(1), January 2006.

[2] 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 Nineteenth ACM Symposium on Operating Systems Principles (SOSP)*, October 2003.

[3] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating System Support for Planetary-Scale Network Services. In *Proceedings of the First Symposium on Networked Systems Design and Implementation (NSDI)*, March 2004.

[4] G. Candea, M. Delgado, M. Chen, and A. Fox. Automatic Failure-Path Inference: A Generic Introspection Technique for Internet Applications. In *Proceedings of the Third Workshop on Internet Apllications (WIAPP)*, June 2003.

[5] E. Cecchet, J. Marguerite, and W. Zwaenepoel. Performance and Scalability of EJB Applications. In *Proceedings of the Seventeenth Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, November 2002.

[6] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing Energy and Server Resources in Hosting Centers. In *Proceedings of the Eighteenth ACM Symposium on Operating System Principles (SOSP)*, October 2001.

[7] 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)*, June 2003.

[8] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live Migration of Virtual Machines. In *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation (NSDI)*, May 2005.

[9] R. P. Doyle, O. Asad, W. Jin, J. S. Chase, and A. Vahdat. Model-based resource provisioning in a Web service utility. In *Proceedings of the Fourth USENIX Symposium on*

Internet Technologies and Systems (USITS), March 2003.

[10] R. J. Figueiredo, P. A. Dinda, and J. A. B. Fortes. A Case For Grid Computing On Virtual Machines. In *Proceedings of the Twenty-third International Conference on Distributed Computing Systems (ICDCS)*, May 2003.

[11] I. Foster, T. Freeman, K. Keahey, D. Scheftner, B. Sotomayor, and X. Zhang. Virtual Clusters for Grid Communities. In *Cluster Computing and Grid (CCGRID)*, May 2006.

[12] 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.

[13] F. E. Gillett and G. Schreck. Server virtualization goes mainstream. Technical Report 2-22-06, Forrester Research Inc., February 2006.

[14] L. Grit, D. Irwin, A. Yumerefendi, and J. Chase. Virtual Machine Hosting for Networked Clusters: Building the Foundations for "Autonomic" Orchestration. In *Proceedings of the First International Workshop on Virtualization Technology in Distributed Computing (VTDC)*, November 2006.

[15] L. Huang, G. Peng, and T. Chiueh. Multi-Dimensional Storage Virtualization. In *Proceedings of Joint International Conference on Measurement and Modeling of Computer Systems*, June 2004.

[16] D. Irwin, J. Chase, L. Grit, and A. Yumerefendi. Self-Recharging Virtual Currency. In *Proceedings of the Third Workshop on Economics of Peer-to-Peer Systems (ECONP2P)*, August 2005.

[17] 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.

[18] W. Jin, J. S. Chase, and J. Kaur. Interposed proportional sharing for a storage service utility. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems (ACM SIGMETRICS/Performance)*, June 2004.

[19] K. Keahey, K. Doering, and I. Foster. From Sandbox to Playground: Dynamic Virtual Environments in the Grid. In *Proceedings of the Fifth International Workshop in Grid Computing (Grid)*, November 2004.

[20] B. Lin and P. A. Dinda. VSched: Mixing Batch and Interactive Virtual Machines Using Periodic Real-time Scheduling. In *Proceedings of the Eighteenth Annual Supercomputing Conference (SC)*, November 2005.

[21] M. L. Massie, B. N. Chun, and D. E. Culler. The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. *Parallel Computing*, 30(7), July 2004.

[22] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Design and Implementation Tradeoffs for Wide-Area Resource Discovery. In *Proceedings of the Fourteenth IEEE Symposium on High Performance Distributed Computing (HPDC)*, July 2005.

[23] L. Ramakrishnan, L. Grit, A. Iamnitchi, D. Irwin, A. Yumerefendi, and J. Chase. Toward a Doctrine of Containment: Grid Hosting with Adaptive Resource Control. In *Proceedings of the Nineteenth Annual Supercomputing Conference (SC)*, November 2006.

[24] S. Santhanam, P. Elango, A. Arpaci-Dusseau, and M. Livny. Deploying Virtual Machines as Sandboxes for the Grid. In *Proceedings of the Second Workshop on Real, Large Distributed Systems (WORLDS)*, December 2005.

[25] C. A. Waldspurger. Memory Resource Management in VMware ESX Server. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.

[26] A. Yumerefendi, P. Shivam, D. Irwin, P. Gunda, L. Grit, A. Demberel, J. Chase, and S. Babu. Towards an Autonomic Computing Testbed. In *Workshop on Hot Topics in Autonomic Computing (HotAC)*, June 2007.