Dynamic Virtual Clusters in a Grid Site Manager

Jeffrey S. Chase, David E. Irwin, Laura E. Grit, Justin D. Moore, and Sara E. Sprenkle*
Department of Computer Science
Duke University
Box 90129, Durham, NC 27708, U.S.A.
{chase,irwin,grit,justin,sprenkle}@cs.duke.edu

Abstract

This paper presents new mechanisms for dynamic resource management in a cluster manager called Cluster-on-Demand (COD). COD allocates servers from a common pool to multiple virtual clusters (vclusters), with independently configured software environments, name spaces, user access controls, and network storage volumes. We present experiments using the popular Sun GridEngine batch scheduler to demonstrate that dynamic virtual clusters are an enabling abstraction for advanced resource management in computing utilities and grids. In particular, they support dynamic, policy-based cluster sharing between local users and hosted grid services, resource reservation and adaptive provisioning, scavenging of idle resources, and dynamic instantiation of grid services. These goals are achieved in a direct and general way through a new set of fundamental cluster management functions, with minimal impact on the grid middleware itself.

1 Introduction

The integration of clusters into computational grids is one of several trends driving a shift to *mixed-use* clusters that serve multiple user groups with different needs. Grids enable networked sharing of cluster resources across administrative domains, bringing external users into the cluster. Grid-connected clusters host diverse grid services and software environments, including applications that aggregate resources from multiple sites for a single computational task. Shared mixed-use sites also result from *consolidation* of clusters to gain on-demand access and economies of scale in deployment and administration.

*This work is supported in part by the U.S. National Science Foundation (EIA-9972879, EIA-9870728), by HP Labs, and by IBM through a SUR equipment grant and an IBM faculty research award. Sprenkle is supported by an NSF Graduate Fellowship, and Grit is supported by a National Physical Sciences Consortium Fellowship.

Mixed-use sharing creates new challenges for cluster management, particularly for grid-connected clusters. The outlines of the next-generation grid are now visible [9, 10, 12, 13, 25]: it will include policy-based resource management, distributed authorization, dynamic instantiation of software environments and services, resource reservations for predictable application service quality, and dynamic adaptation to changing load and system conditions. As grid computing extends to include long-lived network services [13], it is increasingly important to provision resources for performance targets embodied in Service Level Agreements or SLAs [10].

This paper presents a new cluster management architecture for grid-ready mixed-use clusters, and experimental results from a prototype implementation called *Cluster-on-Demand* (COD). COD incorporates the best practices of the current generation of cluster managers: database-driven network installs from predefined software distributions [3, 19, 23, 31], software customization and automated remote upgrades [23], and load-balancing task schedulers [5, 24, 27]. Most clusters today follow the Beowulf model of homogeneous clusters with a single system image. While COD is complementary to this model and builds on its success, it is a fundamental departure from it. In particular, COD manages a cluster as a multi-purpose *modular* resource that hosts different user groups and software environments in isolated partitions called *virtual clusters* or vclusters. It exports an external interface to instantiate vclusters and resize them according to dynamic conditions and site policies.

The central point of this paper is that dynamic virtual clusters are a fundamental enabling abstraction for the next-generation grid. In a grid setting, COD acts as a *site manager* that controls local resources and exports a resource negotiation interface to local grid service middleware and external resource brokers on the grid. Vclusters are the basis for dynamic policy-based allocation of cluster resources across different application environments and user communities within each grid site. Vclusters can encapsulate grid

Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)
1082-8907/03 $17.00 © 2003 IEEE

services such as batch pool schedulers or clustered Web services containers (e.g., IBM WebSphere); in this way, sites can control the resources associated with each grid service, without placing ever more complex resource management and access control functions into grid service middleware and applications.

To illustrate the role of dynamic virtual clusters in grid site management, we present experimental results from multiple instances of the Sun GridEngine (SGE) batch pool scheduler running in separate vclusters under COD. We implemented a simple wrapper for SGE to monitor load and negotiate for resources from the COD manager, which allocates resources to the batch pools according to configured policies. The experiments show that the architecture enables dynamic provisioning and differentiated service policies unified at the COD level, without requiring the batch pool scheduler (or, by extension, other grid service middleware) to support these functions. The experiments use a simple priority allocation policy with guaranteed minimum reservations; the purpose of the experiments is to illustrate the power and practicality of vclusters as a *mechanism* for dynamic resource management, and not to evaluate the policies, which are replaceable.

This paper is structured as follows. Section 2 discusses the role of dynamic virtual clusters in grid site management. Section 3 outlines relevant aspects of the design and implementation of the COD cluster manager. Section 4 discusses extensions to grid service middleware to support dynamic provisioning under COD, using the SGE batch pool wrapper as an illustrative example. Section 5 presents experimental results with dynamic resizing of multiple SGE batch pools under trace-driven load. Section 6 sets COD in context with related systems, and Section 7 concludes.

2 Overview

We initiated the Cluster-on-Demand project with the goal of building an "operating system" for a large, shared, mixed-use cluster. COD enables rapid, automated, on-the-fly partitioning of a physical cluster into multiple independent virtual clusters (vclusters). A vcluster comprises a subset of cluster nodes configured for a common purpose, with associated user accounts and storage resources, a user-specified software environment, and a private IP address block and DNS naming domain. Once a configuration is defined, applying it to a node is *automatic*: this makes it easy to redeploy cluster nodes among user communities and software environments under programmatic control.

The basic elements of COD are similar to other systems that manage clusters using database-driven network installs, most notably Oceano [3] and Emulab [31] (see Section 6). Our design for COD leverages widely used open-source components to support diverse hardware platforms and software configurations, and to evolve rapidly with new technology. It is deployable on any modern off-the-shelf cluster (i.e., the nodes support PXE remote boot and can run Linux) with an Ethernet/IP network, remote management using DHCP and NIS, and shared network storage, e.g., through NFS. COD can be deployed incrementally in a bottom-up fashion: its use at a site is transparent to any other software at the site, unless that software uses the COD service interfaces to allocate and configure resources.

As a cluster manager, the COD architecture addresses three key goals:

- **Secure isolation of multiple user communities.** Independent sets of user identities may be active within each vcluster, removing the need for a common space of user accounts and IDs across the system. For example, it is possible to open a segment of a large cluster to external users on a temporary basis, without (in principle) compromising the primary user community.

- **Custom software environments.** Each vcluster may run software tailored to the needs of its users, all the way down to the operating system. With Gigabit Ethernet and a typical SCSI disk, it takes about two minutes to install Linux from a bit image hosted on a network server. Users may select software from a library of configuration templates. Authorized users may upload new template images to instantiate new software environments on "raw" resources exported by the site.

- **Dynamic policy-based resource provisioning.** Vclusters are the subjects of resource allocation for clusters in the same way that resource containers [4] and related resource control mechanisms serve that role on individual servers. Site administrators may specify policies to control the resources assigned to each vcluster. Vclusters are a powerful basis for adaptive resource management when combined with continuous load monitoring within each vcluster.

These functions are also useful as a basis for advanced resource management for grid-connected clusters. One benefit of the approach is that it enables safe and flexible affiliation with grids by encapsulating each grid service in a separate isolated vcluster whose size is controlled by site policies. It becomes easier and safer to donate resources to a grid, because grid services and their external users cannot interfere with portions of the cluster that are assigned for other purposes. Also, since multiple software environments may coexist in different vclusters, it is possible to multiplex grid points-of-presence (e.g., Globus, Avaki) on a single physical cluster. This approach is useful because there is no "one-size-fits-all" grid: rather, this space has several competing, evolving software packages and multiple logi-

Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)
1082-8907/03 $17.00 © 2003 IEEE

cal grids established by peering arrangements among institutions.

More generally, the COD site manager may itself export a grid service interface for resource negotiation and other advanced resource management functions. COD is designed to allow sites to delegate control over shares of their local resources to external policy managers and resource brokers in the grid, as described in Section 3. External managers may obtain rights to cluster resource shares for specific time intervals, and bind them to vclusters under their control. Thus dynamic vclusters can serve as the underlying mechanism to support a wide range of policies for sharing and managing resources. As a grid site manager, the COD architecture can provide three key functions:

- **Balancing local vs. global resource use.** Sites can dynamically control how much of their resource is exposed to the grid, and may hold some resource in reserve for local use. For example, a site can give priority to local uses and allocate idle machines for best-effort grid service. This structure is an alternative to resource scavenging systems such as Condor [20], which are installed on every node, making them more difficult to use safely in heterogeneous software environments.
- **Controlled provisioning for grid services.** The site manager can provision resources dynamically across multiple hosted grid services. Provisioning will become increasingly important as grids host Web services whose request loads vary with time. The site management policies can reflect priority or peering arrangements, as well as feedback about load and/or resource demands to meet service quality targets (SLAs).
- **Reservations.** Distributed grid applications are sensitive to their mappings onto grid resources, creating a need to reserve collections of grid resources at multiple sites to deliver predictable performance [12]. The COD interface allows external managers to obtain a lease on physical resources bound to a vcluster for specific time intervals.

Although COD can serve as a key component of a grid architecture, it diverges from the accepted model of grid computing middleware in one critical respect. One role of middleware is to mask differences in the underlying operating systems by exposing only a middleware API to applications. In contrast, COD views operating systems and other elements of the software environment as components to be manipulated and configured at will. Because COD manages cluster resources at the granularity of nodes, it is possible to customize the vcluster environment by replacing the software on recruited nodes down to the "bare metal".

Several technology trends lead us to this fundamental shift: decreasing cost of cluster nodes, increasing scale of clusters, industry manageability initiatives enabling full control of cluster nodes from the network, and I/O speeds increasing at a faster rate than software size. These trends undermine the longstanding assumption that software environments and applications are bound to specific computer systems that are manually configured to run them on a semi-permanent basis. The node configuration cost—which is on the order of seconds, or minutes for a full "wipe it clean" install [23]—is amortized across long runs of resource-intensive applications, which are typical in a grid setting. Also, vclusters may run Web service containers (e.g., IBM WebSphere), batch pools, or other grid services that schedule multiple tasks on the vcluster's resources over a long period of time (see Section 4).

Customizing node software can advance the potential of grid computing beyond the subset of applications that have been "grid-enabled" by recoding them to use grid middleware APIs. This capability can extend the grid toward true reconfigurable on-demand network computing, in which a pool of network servers act as generic caches for software environments and applications, and are automatically configured to instantiate them wherever resources are available and demand exists.

3 Dynamic Virtual Clusters in COD

Figure 1 illustrates the COD framework. A site administrator issues credentials authorizing access to an external service interface for the COD site. Using this interface, external agents may define user groups, delegate access rights to the members of a group, create vclusters on behalf of a group, and define hardware requirements and software configuration templates for those vclusters. This section gives a brief overview of the essentials of COD; a more detailed discussion can be found in a recent technical report [22] (*issg.cs.duke.edu/cod*).

3.1 Node Management

COD leverages the Dynamic Host Configuration Protocol (DHCP) to take control of cluster nodes through Intel's Preboot eXecution Environment (PXE). At the node configuration level, COD interposes on network management services—NIS and DNS—to control each node's view of its environment according to its vcluster membership. The DHCP, DNS, and NIS servers tie into a unifying back-end database of node states and configurations. Since all servers are stateless, the COD service is as reliable as its database. Our prototype uses MySQL.

When a node boots, the DHCP server queries its status from the database. If the node is switching to a new configuration, the DHCP server loads a minimal *trampoline* OS to install the user-specified software. The trampoline includes

Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)
1082-8907/03 $17.00 © 2003 IEEE

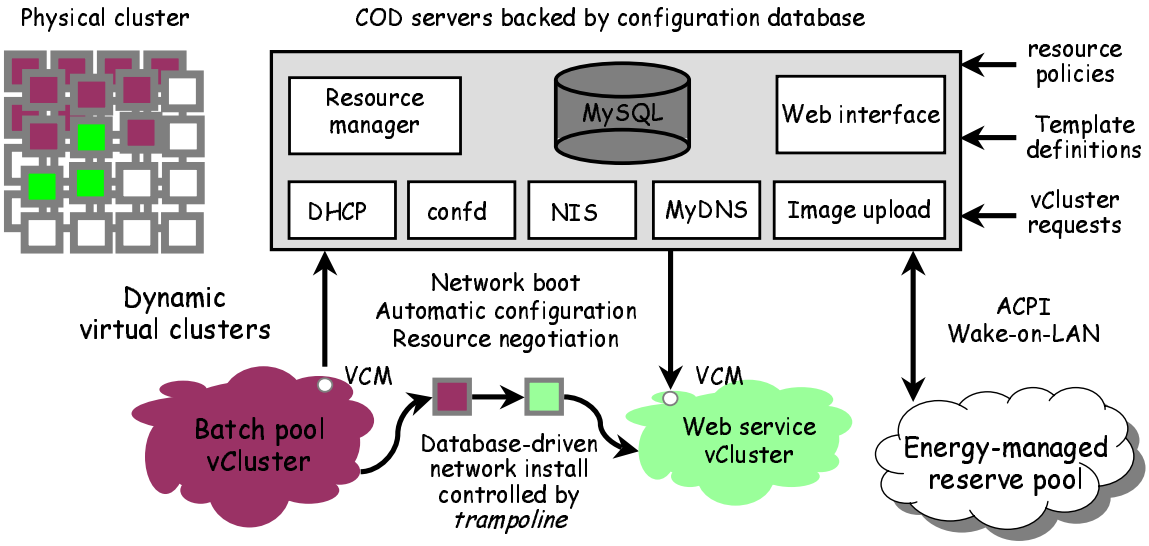

Figure 1. Cluster-on-Demand (COD) partitions a physical cluster into multiple virtual clusters (*vclusters*). Vcluster owners specify the operating systems and software for their vclusters through an XML-RPC interface. The vclusters in this example run a batch scheduler and a Web server cluster, which can resize dynamically to respond to load changes.

a minimal generic x86 Linux kernel and small RAM-based root filesystem. This trampoline probes the node hardware and sends a summary of the installed hardware to a configuration daemon, or *confd*. The confd then directs the trampoline to partition the local drives and to fetch and install software images. As the node boots, the COD servers shape its view of its environment:

- COD assigns node IP addresses within a subnet for each vcluster. Nodes obtain their IP and router addresses from the DHCP server.
- Each vcluster occupies a private DNS subdomain derived from the vcluster's symbolic name assigned at creation time. Nodes obtain their hostnames through DHCP and use DNS or NIS to map between hostnames and IP addresses. Our prototype uses MyDNS, an open-source SQL-enabled DNS server.
- Each vcluster executes within a predefined NIS domain, which enables access for user identities and netgroups enabled for the vcluster.
- COD exports NFS file storage volumes as groups and vclusters are defined. Nodes obtain an NFS mount map through NIS; only those NFS file volumes authorized for access by the vcluster's group are exported to it.

3.2 Virtual Cluster Managers (VCMs)

COD enables fluid assignment of nodes to vclusters according to load and site policies. While some vclusters may maintain a static size, others may benefit from dynamic resizing. For example, a vcluster hosting a pool of network servers or a grid batch scheduler will face varying demand over time.

COD resizes each dynamic vcluster in cooperation with the service hosted within it. A key premise of the COD architecture is that the system can negotiate resource provisioning by interfacing with an application-specific service manager—a *Virtual Cluster Manager* or VCM—for each vcluster that hosts a dynamic service. The VCMs contain the logic for monitoring load and changing membership in the active server set for the specific application environment. Note that the service may itself host applications (such as Web services or compute jobs) that are unaware that this resizing is taking place. This hierarchical division of resource management functions is a cornerstone of the COD architecture for dynamic provisioning.

Our hypothesis is that the VCM functions are simple extensions for well-structured grid services, and rarely require modifying the grid service middleware itself. In many cases, it is possible to leverage support that must be present in any robust cluster service to handle membership changes resulting from node failures and incremental growth. Examples of application services amenable to this approach in-

Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)
1082-8907/03 $17.00 © 2003 IEEE

clude load-leveling batch schedulers, enterprise application monitors [29], cluster-based network services (e.g., [6, 16, 18, 26]), and cluster-based network storage [1, 28]. While removing nodes from a service involuntarily may have a significant performance cost, e.g., if the service hosts parallel applications tuned for a specific degree of parallelism, many cluster services can handle even this case gracefully [11].

Our prototype includes a generic VCM server that handles the details of resource negotiation with the COD manager, as described below. The service-specific aspects of the VCM comprise three pluggable modules: *add_nodes*, *remove_nodes*, and *resize*. These are arbitrary programs or scripts executed from the VCM process with simple command line arguments. The *add_nodes* and *remove_nodes* modules handle the mechanics of changing vcluster membership. The *resize* module contains the VCM policy to monitor load and resource status and request changes to the vcluster size. This simple modular structure makes it easy to implement VCMs for specific services.

Section 4 illustrates these ideas by describing VCM modules for the Sun GridEngine (SGE) batch pool service. These modules use sequences of standard SGE configuration commands to interact with the batch scheduler. We did not modify SGE itself to install a VCM wrapper.

3.3 Resource Negotiation

A key goal of COD is to support flexible, extensible policies for resource management. COD is designed to allow secure external control of site resources through a resource negotiation protocol that separates the *policies* for cluster resource management from *mechanisms* for dynamic virtual clusters in COD itself. The XML-RPC negotiation protocol (called Secure Highly Available Resource Peering or SHARP) is based on soft-state reservations representing *claims* on resource shares for specific time intervals, following the classical leases model [17, 30]. SHARP resource claims are cryptographically signed to make them unforgeable, and may be securely delegated to third parties.

The details of the framework are beyond the scope of this paper. SHARP resource negotiation may be viewed as closely similar to the recent SNAP proposal for Globus [10], extended with support for secure delegation of resource rights; this makes it possible to manage grid resources through a network of interacting brokers managing various shares of the resource pool. For the experiments in this paper, the policy engine is integrated with the COD site manager, and resource rights are redeemed immediately without generating or validating a signed claim.

The COD manager accepts resource requests to create virtual clusters and allocate nodes to virtual clusters. Each request originates from a VCM, which is an XML-RPC network server certified to act on behalf of a vcluster and its group. At a high level, an allocation request is a 4-tuple of the form *(vcluster, template, count, attributes)*: allocate *count* nodes to *vcluster*, selecting from nodes matching the specified *attributes*, and apply the specified configuration *template*. COD handles all requests asynchronously, and uses callbacks to notify the requesting VCM of the outcome of each request. If the request was at least partially satisfied, the COD manager issues a `Grant` callback containing a *lease*. The lease is an XML object asserting that the holder controls a named set of nodes over some specific time interval (its *term*). The COD manager configures any new nodes for the specified *template* before returning the lease. The VCM inspects each lease to identify any changes to its node set, and invokes the service-specific VCM modules to inform the service.

Each VCM periodically invokes its *resize* module to evaluate its load. The *resize* module examines its internal vcluster status measures, executes its policy, and outputs one or more XML commands to the VCM server. The VCM may voluntarily relinquish nodes under light load, or request additional nodes as its load increases. COD may force the VCM to relinquish within a bounded time by refusing to extend a lease if the resources are needed elsewhere. If the VCM does not relinquish a reclaimed node, the COD manager may seize it by forcing it onto the trampoline and rebooting it into a different vcluster.

4 An SGE Batch Pool Manager

To demonstrate and evaluate the COD model for dynamic virtual clusters, we implemented a VCM wrapper for Sun's GridEngine (SGE). SGE is widely used to run compute-intensive batch jobs on large clusters, and is similar to other local task schedulers for grid sites.

In a typical SGE cluster, a single master host runs a scheduler (*sge_schedd*) that dispatches submitted batch jobs across an *active set* of execution hosts. Users submit jobs by executing the SGE *qsub* command on any host in an SGE vcluster. To maintain a uniform environment across the active set, as required by SGE, each vcluster configuration template defines a set of user identities eligible to use the batch pool, and a shared network file volume mounted through NFS. The NFS volume includes the SGE distribution and master status files (the SGE_ROOT directory) and all program and data files for the user jobs.

The core of the SGE VCM is the *add_node*, *remove_node*, and *resize* modules as described in the previous section. Each module has a defined interface and outputs XML commands to the VCM server. These modules encapsulate administrative complexities for assigning nodes to the service. The modules are stateless and replaceable even in a running system, e.g., to modify VCM policies.

The VCM modules for GridEngine are simply scripts

Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)
1082-8907/03 $17.00 © 2003 IEEE

that execute sequences of SGE administrative commands. The *add_node* and *remove_node* modules remotely start up and shut down the proper SGE daemons. To add a host, *add_node* executes the SGE *qconf* command with a standard template to activate the node by its domain name and establish a job queue for the node. After enabling the queue, the VCM executes the SGE daemon processes—*sge_commd* and *sge_execd*—on the node. To remove a node, *remove_node* executes SGE commands—*qconf* and *qmod*—to disable the node's job queue, reschedule any jobs on the queue, destroy the queue, and deactivate the node.

The "brain" of the VCM is the *resize* module, which encapsulates the policy for resizing the vcluster according to load. Our prototype uses a simple policy: request a new node for every X pending jobs and relinquish a node after it has been idle for Y seconds, where X and Y are user-defined parameters. We configured SGE to schedule at most one job on each active execution host. The policies are stable and effective when each batch pool serves compute-bound sequential jobs that run for longer than the reconfiguration times.

The VCM server invokes the *resize* function every $epoch$ seconds to check the status of the batch pool. The *resize* module uses the SGE *qstat* command to obtain a list of queues and the jobs scheduled to them. If there are queued jobs that have not yet started, *resize* requests a new execution node for every X queued jobs. It then records any nodes that are idle and timestamps them, and caches recent history in a file. If there are no queued jobs, it issues a relinquish request to return each idle node to the COD manager after it has been idle for Y seconds, stopping when it reaches a minimum idle reserve of k nodes.

Using the SGE VCM, we can instantiate SGE batch pools within vclusters, and dynamically resize each vcluster according to batch load. The system can isolate users of different batch pools, and apply arbitrary policies to allocate resources to the pools under constraint. Our prototype assigns a priority and a minimum guaranteed reservation to each batch pool. A low-priority batch vcluster is similar to the Condor resource-scavenging model [20]; that is, COD allocates nodes to the batch pool only when they are idle. The COD approach ensures a consistent environment across the batch pool, at the price of a higher node allocation cost to reinstall. The COD model also protects users with fixed leases against interference from the batch pool.

Many competing batch schedulers support multiple batch queues, and some schedule jobs according to priority. The COD approach provides these and other resource management features (e.g., reservations) at the system level, without relying on support within the batch scheduler middleware. The vcluster approach also allows more flexible policies for allocating resources between the batch pool and other competing vclusters. These features extend easily to other middleware services as well.

We are currently considering how to extend the SGE VCM to run with the Avaki Grid manager, which coordinates batch job scheduling across multiple batch pools (including SGE pools). These pools may reside at multiple sites across a wide-area network, with local autonomy over resource management. In the Grid, local site managers pass information about their available resources to a global grid manager, which makes informed global scheduling decisions about where to route jobs [15]. In the COD framework, the local VCM must notify the grid manager of changes in vcluster size and may also pass global requests for additional resources through to the local COD manager. Our intent is that this approach will allow for multiple Grid points-of-presence (e.g., Globus, Avaki) to run as separate vclusters within a shared COD physical cluster, trading resources across the grids according to site-specific policies.

We are exploring more sophisticated resource management policies to handle more complex cases within this framework. What is important is that this example illustrates the feasibility of dynamic cluster resizing and the power and generality of the COD framework. In this case, implementing a VCM was straightforward and required no modifications to SGE itself to support a rich set of resource management capabilities.

5 Experimental Results

This section presents experiments with SGE batch pools in a prototype COD cluster under trace-driven batch load. The results demonstrate dynamic provisioning behavior with simple policies. We ran our experiments with multiple similarly configured SGE batch pools on a testbed of 80 rackmount IBM xSeries-335 uniprocessor servers within the "Devil" Cluster at Duke Computer Science.

5.1 Traces

To stress the prototype under high levels of contention and resource constraint, we construct test trace segments drawn from a nineteen-month trace of production batch queue activity on the Devil Cluster. The full trace starts in September 2001 and continues until mid-April 2003. Each trace entry contains a submit timestamp, job start time, job completion time, and the user identity and executable file name for the job. We divide the SGE users into three user groups, select a three-month window of activity for each group, and combine the trace segments to form the test trace. The three user groups are:

- **Systems.** Researchers in networking and peer-to-peer systems submitted large numbers of short jobs, usually no more than a few minutes long. Activity from

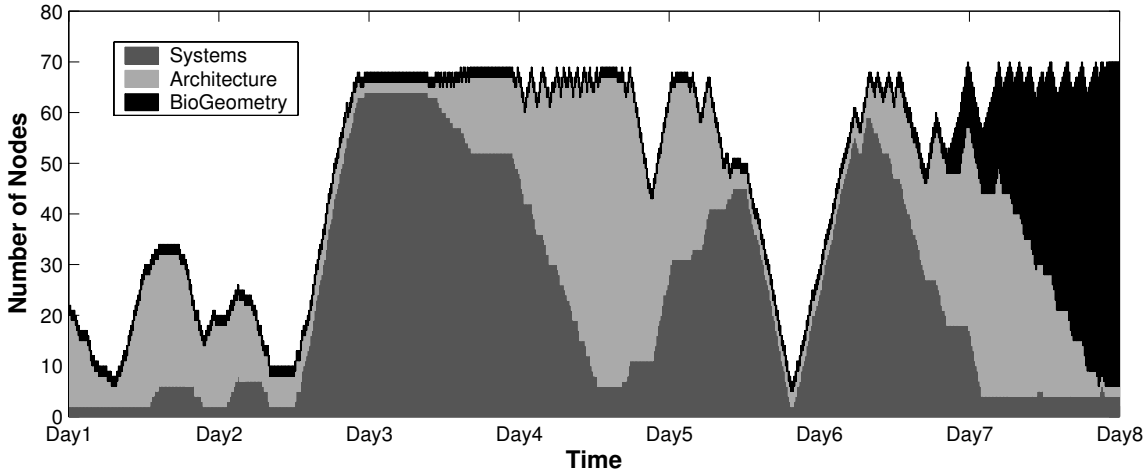

Figure 2. Number of nodes in each of three virtual clusters over time during the live deployment.

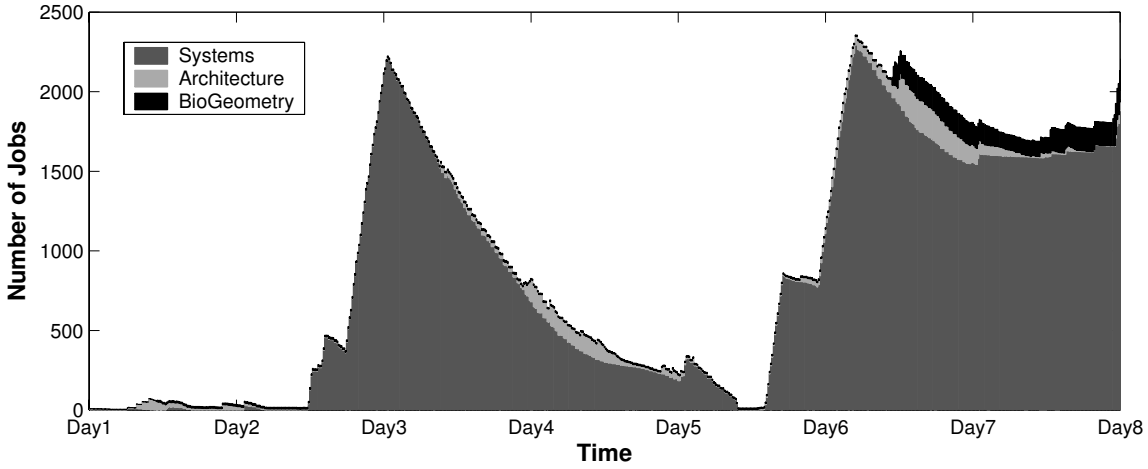

Figure 3. Number of SGE jobs in each batch queue over time during the live deployment.

this group was highly bursty. The Systems trace segment runs from 17:01:52, 2002/02/01 until 00:00:00, 2002/05/01.

- **Architecture.** This team submitted a large number of computer architecture simulations, each consuming many hours of CPU time. Job arrival rate for this group was relatively stable. The Architecture trace section runs from 23:43:50, 2001/09/14 until 00:00:00, 2001/12/01.

- **BioGeometry.** These jobs evaluate new algorithms to predict protein folding and docking. Submitted jobs ran for days or weeks. This group submitted jobs with steadily increasing frequency. The BioGeometry trace segment runs from 18:34:47, 2002/10/03 until 00:00:00, 2003/01/01.

5.2 Live Trace Experiment

In the first test we ran the system on a testbed of seventy-one servers for twelve hours to examine the behavior of the provisioning policies. The test instantiates three SGE batch queues in separate vclusters, then replays a trace segment to each batch queue in real time. All trace records execute an application shell that spins in a busy loop for a specified time. To accelerate the experiment, we sped up the submission and completion of jobs by a factor of 160. This speedup allows a full three-month trace to complete in under twelve hours. While speeding up the trace introduces significant error in the flip times of nodes, the general trends of node allocations are not affected.

Each SGE-VCM performs a *resize* every seven seconds to negotiate for resources with the COD resource manager

Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)
1082-8907/03 $17.00 © 2003 IEEE

according to the policies outlined in Section 4. The *resize* policy requests one node for every 15 jobs still in the queue and relinquishes a node after being idle for 60 seconds. The COD resource manager uses a fixed priority ordering for the batch pools, but guarantees each pool a minimum of two nodes. In our experiment, the Architecture group has the highest priority, the BioGeometry group has middle priority, and the Systems group has lowest priority.

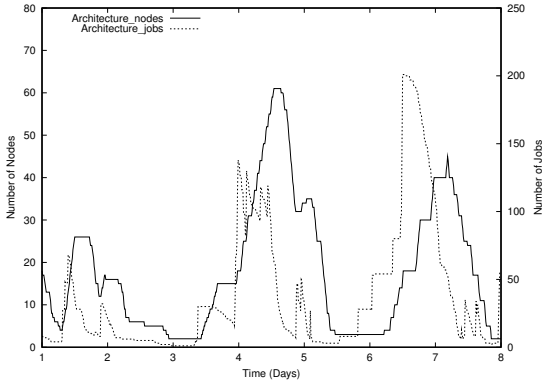

Figure 4. Combined size of the Architecture running and pending job queues, and vcluster size over an eight-day period.

Figure 2 and Figure 3 show the number of active nodes and queued jobs, respectively, for all three groups over a selected eight-day period. The graphs show time in days, where each day represents approximately nine minutes in real time. We examine the stages of resource contention and negotiation visible during this time period.

Initially the cluster is underutilized for approximately two days. A brief second stage involves a large spike in demand from the Systems group, as the number of queued jobs increases over 2000. To accommodate this burst, the COD manager pulls nodes from the idle pool and allocates them to the Systems vcluster. While the Systems vcluster is growing, the Architecture group submits over 100 jobs to its SGE pool. Due to the load spike from Systems, there are no free nodes available. Since the Architecture group has the highest priority, the COD manager reclaims nodes from the other vclusters, primarily the low-priority Systems vcluster, and transfers them to Architecture.

Figure 4 focuses on the Architecture group activity from the same experiment to clearly illustrate the relationship between the length of the job queue and the number of nodes in the vcluster. As the queue length grows, the SGE-VCM obtains more nodes from COD to deliver a faster turnaround time on Architecture jobs. SGE distributes the jobs to the new nodes, restoring equilibrium and causing the queues to shrink. As nodes become idle, the SGE-VCM relinquishes them back to the global resource pool. If the size of the queue is below the request threshold, X — for example, midway through day two to midway through day three — the SGE-VCM leaves the vcluster at roughly constant size.

Starting on day six, the Systems vcluster receives a burst of over 2000 jobs, and requests nodes from the COD idle pool. It keeps these nodes until the higher-priority Architecture and BioGeometry receive new job bursts and start competing for nodes. Since Architecture is higher priority than BioGeometry, it acquires more nodes and retires its jobs faster, eventually relinquishing its nodes to BioGeometry.

While this experiment uses simple policies, it illustrates dynamic policy-based provisioning with differentiated service for SGE batch pools within the COD framework, without any special support for advanced resource management in the batch scheduler itself.

5.3 Policies and Scalability

To experiment with different policies and scalability with larger traces, we replaced the SGE batch pools with an emulated SGE environment to short-circuit the actual job execution. We modified the COD manager to disable node configuration, and replaced each SGE batch pool with a small C application that reads the batch job trace as input, replays the trace, and responds to requests from the VCM modules. Using the emulator, we can run months-long traces on the order of hours. Use of the emulator is transparent to all other system components, including the COD resource manager, MySQL database server, VCM server, and SGE VCM modules.

The emulator moves forward in time in constant steps, or *epochs*. For each epoch, the emulator replays the trace over that epoch by removing completed jobs and inserting submitted jobs on a per-vcluster basis. It maintains lists of the jobs in each queue, jobs to be run in the future, and jobs currently running on nodes. A key component of the emulator is a suite of tools to emulate the SGE administrative commands used by the VCM modules (primarily *qstat*). Their output is identical to the real VCM commands, allowing our SGE-VCM component to plug into a live SGE pool or an emulated trace without modifying the SGE-VCM. The tools connect to the emulator core through a socket to read and modify the batch pool status. After each SGE-VCM uses *qstat* to read state from the emulator, the emulator updates the node and job data structures and advances the clock one epoch.

In this section, we use the emulator to explore two variations of the priority-based resource allocation policy. In the first case, when there are outstanding requests from a high-priority vcluster, COD systematically steals nodes from lower-priority vclusters to meet the requests of higher-priority vclusters, but does not reclaim nodes from a victim

below a minimum guaranteed level of two nodes (as in the previous experiment). This policy has the potential to starve low-priority vclusters if higher priority vclusters are under heavy load.

In the second case, we supplement this policy with variable guaranteed minimum reservations. Instead of the flat minimum size, COD guarantees each vcluster a minimum size based on its priority. In this experiment, 90% of the 80-node resource set is divided among the vclusters: Systems, Architecture, and BioGeometry are guaranteed 12, 20, and 32 nodes, respectively.

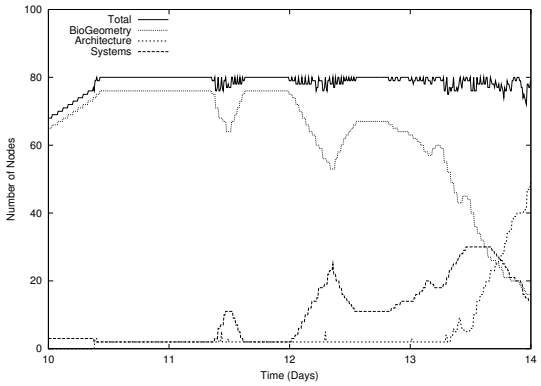

Figure 5. Vcluster size with a pure priority-based resource allocation policy.

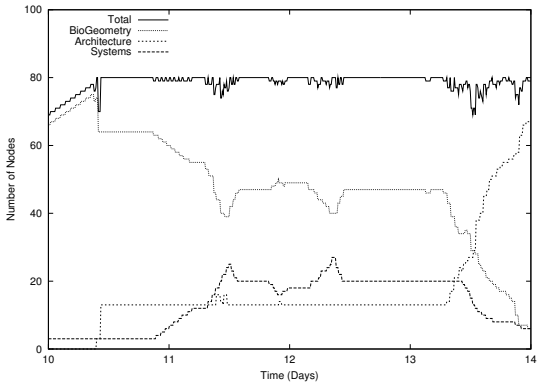

Figure 6. Vcluster size with priority allocation and minimum reservations.

Figures 5 and 6 show the results of our experiments for the two policies. We select a four-day segment of the three-month trace experiment when resources were constrained for an extended period of time. In this experiment BioGeometry has the highest priority, Architecure has the next highest priority, and Systems has the lowest priority. During the four-day segment, BioGeometry is under load and holds most of the resources. By comparing the two graphs, we see that by the middle of day ten, Systems requests resources and is granted its guaranteed twelve nodes by the minimum reservation policy but is denied resources with the priority-only policy. The same is true for Architecture at the middle of day eleven. The minimum reservations allow Architecture and Systems to make forward progress even when BioGeometry is heavily loaded. This experiment illustrates that the mechanisms for dynamic virtual clusters can enable alternative resource management policies.

We ran experiments with larger emulated clusters and more demanding trace segments to test scalability of the system up to 1000 nodes. We amplified the number of jobs in our trace so the system used all 1000 nodes. We also modified the submission of jobs in the trace to create a large number of node transitions. The emulation ran a thirty-three day trace with 1000 nodes in 42 minutes making all database transitions that would have occurred over the thirty-three days. There were 3.7 node transitions per second resulting in approximately 37 accesses to the database per second. The database transactions are not the bottleneck of the system. The emulator is not able to put enough load into the system to limit the performance of the database. This offers initial evidence that key elements of the COD system (e.g., the database) are scalable to large clusters.

6 Related Work

There has been a great deal of research and progress in managing clusters since the early days of the NOW project [2]. The most successful systems today maintain a homogeneous software environment for a specific class of applications. These systems — including Beowulf [5], load-leveling batch schedulers [24, 27], Millennium [7], Rocks [23], and other elements of the NPACI grid toolset — target batch computations written for common OS or middleware APIs. COD adds user-specified control over the software environments, and a unified architecture for dynamic site resource management.

Several companies are marketing products to automate server management for enterprises and Internet hosting providers. Prominent players in this space include TerraSpring, Opsware (Loudcloud), IBM, and HP through its Utility Data Center (UDC) product and related research. While few details of these systems are published, they reflect the concept of policy-based management of resources and configurations in large shared server clusters.

COD was inspired by Oceano [3], an IBM Labs project to automate a Web server farm. Like Oceano, COD leverages remote-boot technology to reconfigure cluster nodes using *database-driven network installs* from a set of user-specified configuration templates, under the direction of a policy-based resource manager. Emulab [31] uses a similar

approach to configure groups of nodes for network emulation experiments on a shared testbed. Both of these systems target specific application environments: network emulation for Emulab, and Web service hosting for Oceano. COD applies the ideas in these systems to a general framework for dynamic sharing of cluster resources across arbitrary user-defined software environments and applications. In particular, COD's hierarchical approach incorporates local resource managers within each vcluster or vcluster group, with a site resource manager to coordinate resource usage across multiple dynamic vclusters. Our goal is to extend these ideas to a general architecture for reconfigurable clustering and dynamic provisioning for a full range of grid services and cluster applications, with "pluggable" application-specific Virtual Cluster Managers (VCMs).

Resource discovery and negotiation are widely recognized as important elements of Grid architectures. The Grid community has responded with initiatives for site management, resource discovery and mapping, policy definitions, and job scheduling. Our site manager architecture for node allocations and reservations complements the Grid Information Services architecture [9] implemented in the Globus metadata service (or MDS), which provides "hints" for resource discovery. For example, a VCM for a dynamic cluster may notify the MDS as its available resource changes due to resource trading. Our approach can also serve as a foundation for the Globus GARA proposal for resource reservations [12] or the SNAP proposal for SLA-based resource allocation [10]. Even so, COD enables a fundamental alternative to the SNAP approach. SNAP proposes to meet end-to-end SLA targets by negotiating SLAs for an application's consituent components with the sites running them, and then combining the SLAs in some way. Our goal is to allow a resource manager (VCM) to directly obtain and control the resources needed to meet SLA targets for end-to-end service quality.

7 Conclusion

This paper presents a new cluster management architecture for shared mixed-use clusters. The key feature of Cluster-on-Demand is support for configurable dynamic virtual clusters, which associate variable shares of cluster resources with application service environments, e.g., batch schedulers and other grid services. The COD site manager assigns nodes to vclusters according to demand and site policies, based on dynamic negotiation with a pluggable service manager for each dynamic vcluster.

Experimental results with the COD prototype and a service manager for the SGE batch service demonstrate the potential of dynamic virtual clusters and resource negotiation as a basis for dynamic provisioning and other advanced resource management for future grid systems. Our results give evidence that the key needs for grid resource management can be met directly by *generic* site management features that are independent of any specific application or middleware environment.

Flexible site management using the COD model will take a key step toward dynamic, adaptive, automatic provisioning of network services from pools of shared server resources dispersed through the Internet and "outsourced" or leased by third parties. COD nodes act as generic "caches" for software environments and applications; COD configures nodes automatically to instantiate them where resources are available and demand exists. This can enable the grid to evolve toward the utility computing model of a shared pool of computing resources—servers, storage, and network capacity—that is automatically provisioned and assigned (or sold) according to demand.

References

[1] D. C. Anderson, J. S. Chase, and A. M. Vahdat. Interposed request routing for scalable network storage. *ACM Transactions on Computer Systems (TOCS) special issue: selected papers from the Fourth Symposium on Operating System Design and Implementation (OSDI), October 2000*, December 2001.

[2] T. Anderson, D. Culler, D. Patterson, and the NOW Team. A Case for NOW (Networks of Workstations). *IEEE Micro*, 15(1):54–64, February 1995.

[3] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D. P. Pazel, J. Pershing, and B. Rochwerger. Oceano - SLA Based Management of a Computing Utility. In *Proceedingsof the 7th IFIP/IEEE International Symposium on Integrated Network Management*, 2001.

[4] G. Banga, P. Druschel, and J. C. Mogul. Resource Containers: A New Facility for Resource Management in Server Systems. In *Third Symposium on Operating Systems Design and Implementation*, February 1999.

[5] Beowulf. Beowulf. http://www.beowulf.com.

[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 18th ACM Symposium on Operating System Principles (SOSP)*, pages 103–116, October 2001.

[7] B. N. Chun and D. E. Culler. User-centric performance analysis of market-based cluster batch schedulers. In *2nd IEEE International Symposium on Cluster Computing and the Grid*, May 2002.

[8] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC)*, August 2001.

[9] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In

Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC), August 2001.

[10] 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.

[11] M. J. Feeley, B. N. Bershad, J. S. Chase, and H. M. Levy. Dynamic node reconfiguration in a parallel-distributed environment. In *Proceedings of the 1991 ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, April 1991.

[12] 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 (IWQoS)*, June 1999.

[13] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the Grid: An Open Grid Services Architecture for distributed systems integration. In *Open Grid Service Infrastructure Working Group, Global Grid Forum*, June 2002.

[14] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, January 2002.

[15] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal on Supercomputer Applications*, 15(3), 2001.

[16] A. Fox, S. Gribble, Y. Chawathe, and E. Brewer. Cluster-Based Scalable Network Services. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, Saint-Malo, France, October 1997.

[17] C. Gray and D. Cheriton. Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency. In *Proceedings of the 12th ACM Symposium on Operating Systems Principles*, pages 202–210, 1989.

[18] S. D. Gribble, E. A. Brewer, J. M. Hellerstein, and D. Culler. Scalable, distributed data structures for Internet service construction. In *Proceedings of the Fourth Symposium on Operating System Design and Implementation (OSDI)*, pages 319–332, October 2000.

[19] J. Hsieh, T. Leng, and Y.-C. Fang. OSCAR: A turnkey solution for cluster computing. *Dell Power Solutions*, pages 138–140, January 2001.

[20] M. Litzkow, M. Livny, and M. Mutka. Condor - A Hunter of Idle Workstations. In *Proceedings of the 8th International Conference on Distributed Computing Systems*, pages 104–111, 1988.

[21] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor - a hunter of idle workstations. In *8th International Conference on Distributed Computing Systems*, pages 104–111. IEEE Computer Society, June 1988.

[22] J. Moore, D. Irwin, L. Grit, S. Sprenkle, and J. Chase. Managing mixed-use clusters with Cluster-on-Demand. Technical report, Duke University, Department of Computer Science, November 2002.

[23] P. M. Papadopoulous, M. J. Katz, and G. Bruno. NPACI Rocks: Tools and techniques for easily deploying manageable Linux clusters. In *IEEE Cluster 2001*, October 2001.

[24] PBS. PBS. http://www.openpbs.org/.

[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] Y. Saito, B. Bershad, and H. Levy. Manageability, Availability and Performance in Porcupine: A Highly Scalable Internet Mail Service. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, December 1999.

[27] Sun. Sun's GridEngine. http://gridengine.sunsource.net/.

[28] C. A. Thekkath, T. Mann, and E. K. Lee. Frangipani: A scalable distributed file system. In *Proceedings of the Sixteenth ACM Symposium on Operating System Principles (SOSP)*, pages 224–237, October 1997.

[29] W. Vogels, D. Dumitriu, K. Birman, R. Gamache, M. Massa, R. Short, and J. Vert. The design and architecture of the Microsoft Cluster Service – a practical approach to high-availability and scalability. In *Fault-Tolerant Computing Symposium (FTCS)*, June 1998.

[30] J. Waldo. The Jini architecture for network-centric computing. *Communications of the ACM*, 42(7):76–82, July 1999.

[31] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.

Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC-12)
1082-8907/03 $17.00 © 2003 IEEE