Managing Mixed-Use Clusters with Cluster-on-Demand

Justin Moore, David Irwin, Laura Grit, Sara Sprenkle, and Jeff Chase*
Department of Computer Science
Duke University
{justin,irwin,grit,sprenkle,chase}@cs.duke.edu

Abstract

Although clusters offer inexpensive computing power, they are difficult and expensive to manage, particularly for user communities with diverse software needs. This paper presents *Cluster-on-Demand* (COD), a cluster operating system framework for *mixed-use* clusters. COD interposes on standard network management services — DHCP, NIS, and DNS — to partition a cluster into dynamic *virtual clusters* (vclusters) with independent installed software, name spaces, access controls, and network storage volumes. COD allocates nodes to vclusters on-the-fly, reconfiguring them as needed with PXE network boots. A key element of COD is a protocol to resize vclusters dynamically in cooperation with pluggable middleware components such as batch schedulers. The COD framework is a key building block for automated management of computing utilities and grids.

1 Introduction

Clustering inexpensive computers is an effective way to obtain reliable, scalable computing power for network services and compute-intensive applications. Since clusters have a high initial cost of ownership, including space, power conditioning, and cooling equipment, leasing or sharing access to a common cluster is an attractive solution when demands vary over time. Shared clusters enable more effective use of resources by multiplexing, and they offer economies of scale in administration as personnel costs grow even as hardware costs decline.

There has been a great deal of research and progress in managing clusters since the early days of the NOW project [5]. The most successful systems today maintain a homogeneous software environment for a specific class of applications. These systems — including Beowulf [1], load-leveling batch schedulers [2, 3], Millennium [10], Rocks [21], and other elements of the NPACI grid toolset — target batch computations written for common OS or middleware APIs. These are powerful tools, but one size does not fit all: users of a shared cluster should be free to select the software environments that best support their needs, which may involve multiple operating systems, multiple batch classes, Web applications, and multiple Grid points-of-presence, each serving a different segment of the user community. Tools to manage *mixed-use* clusters are still lacking.

This paper describes the architecture and implementation of *Cluster-on-Demand* (COD), a system to enable rapid, automated, on-the-fly partitioning of a physical cluster into multiple independent *virtual clusters*. A virtual cluster (vcluster) is 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. COD vclusters are *dynamic*; their node allotments may change according to demand or resource availability.

COD was inspired by Oceano [7], 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 [26] uses a similar approach to configure groups of nodes for network emulation experiments on a shared testbed. Section 2.2 sets COD in context with these and other related systems.

The primary contribution of COD is to extend these techniques to a general framework for a cluster operating system. Like a conventional OS, COD allocates resources to its users, isolates user environments from one another, mediates interactions with the external environment, and manages shared resources dynamically as demands change. Rather than allocating slivers of each node's memory and CPU to user processes, COD allocates complete machines to vclusters shared by a group of users. Users assume full control of their machines down to the bare metal: COD installs user-specified software in each vcluster, analogously to a conventional OS instantiating a user program in a process. Most impor-

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

tantly, COD enables flexible, decentralized, dynamic resource management across vclusters; vclusters manage their own resources internally, and interact with COD to obtain or release resources as needed. Our design leverages widely used open-source components to support diverse hardware platforms and to evolve rapidly with new technology.

2 Overview

Figure 1 illustrates the COD framework. A site administrator issues credentials authorizing access to a Web service interface for the COD site. Using this interface, external users may define user groups, delegate access rights to the members of a group, request vclusters on behalf of a group, and define hardware requirements and software configuration templates for those vclusters. COD interposes on the Dynamic Host Configuration Protocol (DHCP) to take control of cluster nodes through Intel's Preboot eXecution Environment (PXE) [13], and install the user-specified software under the control of a minimal *trampoline* OS. Once a node is active in a vcluster, DHCP and other standard network management services — the Network Information Service (NIS) and Domain Name Service (DNS) — coordinate to assign node hostnames and IP addresses, update the visibility of network storage, and set user access rights. These services tie into a unifying back-end database of node states and configurations.

COD enables fluid assignment of resources according to site policies and resource demands within the vclusters. While some vclusters may maintain a static reservation for a fixed purpose, others may benefit from dynamic resizing. For example, a vcluster hosting a pool of network servers or a grid scheduler will face varying demand over time. COD resizes dynamic clusters in cooperation with a middleware component associated with each vcluster. In recent years, a wide range of cluster managers have emerged for specific application environments (see Section 2.2). Each of these systems supports reconfiguration for changing cluster size [15] to allow for node failures and incremental growth. A key premise of COD is that each of these systems may run as a local manager – a *Virtual Cluster Manager* or VCM — for a virtual cluster, negotiating vcluster size with the global COD manager. This hierarchical division of resource management functions between the global COD manager and the VCMs is a cornerstone of the COD architecture. Section 5.3 describes the resource management framework in detail, and Section 6 illustrates with an example of an extended SGE batch scheduler.

COD maintains surplus nodes in a common energy-managed *reserve pool* and deploys them to vclusters as needed. A resource manager updates the database to assign nodes to vclusters according to defined policy rules and vcluster resource demands. It manages the reserve pool to improve energy efficiency under light load (see Section 3.4).

The shared reserve pool offers each vcluster backup capacity to handle node failures or load swings. During periods of light load, a high-demand vcluster may obtain more nodes than a privately owned cluster would allow. Transitioning a node between vclusters may require a software reinstall whose cost is measured in minutes (see Section 3.3). This "context switch" cost is acceptable in a large, shared cluster, when subsets of nodes are reserved for specific purposes over periods of hours, days, or longer.

2.1 Design Goals and Choices

Five key goals drive the COD design.

Hardware-independence. COD is designed to work with any computer that supports Linux and remote-boots using PXE/DHCP. When a node first powers up in a COD cluster, COD loads the trampoline, which is a minimal, generic, memory-based OS that boots quickly on a wide range of servers. The trampoline is built with components from standard Linux distributions, including Red Hat's *kudzu* hardware-probing program and driver modules for all Linux-supported devices. Once booted, the trampoline scans the hardware and transmits a record of the node's physical configuration for later use by the COD resource manager.

OS-independence. Although the trampoline leverages the comprehensive platform support from Linux, COD itself is OS-agnostic. For full generality, COD configurations may specify operating system images as raw bitfile partition images, or specify an alternate installer program to interpret OS-specific image formats. COD users may select from predefined partition images, request the system to clone images from a manually installed node, or use arbitrary OS-specific tools to prepare images and upload them to the configuration database through the image upload server. To support heterogeneous clusters, configuration templates may include rules to select among multiple images by matching on hardware attributes.

User control. A vcluster owner has the same degree of control over its virtual cluster as it would over a physical cluster dedicated for its own use. Owners may obtain superuser access on their nodes, control the user accounts enabled for login privileges, or replace any element of the software environment — including the operating system — for the nodes assigned to them.

Automated management. Once configurations are defined, site administration overhead (other than hardware maintenance) is independent of the number of hosts and

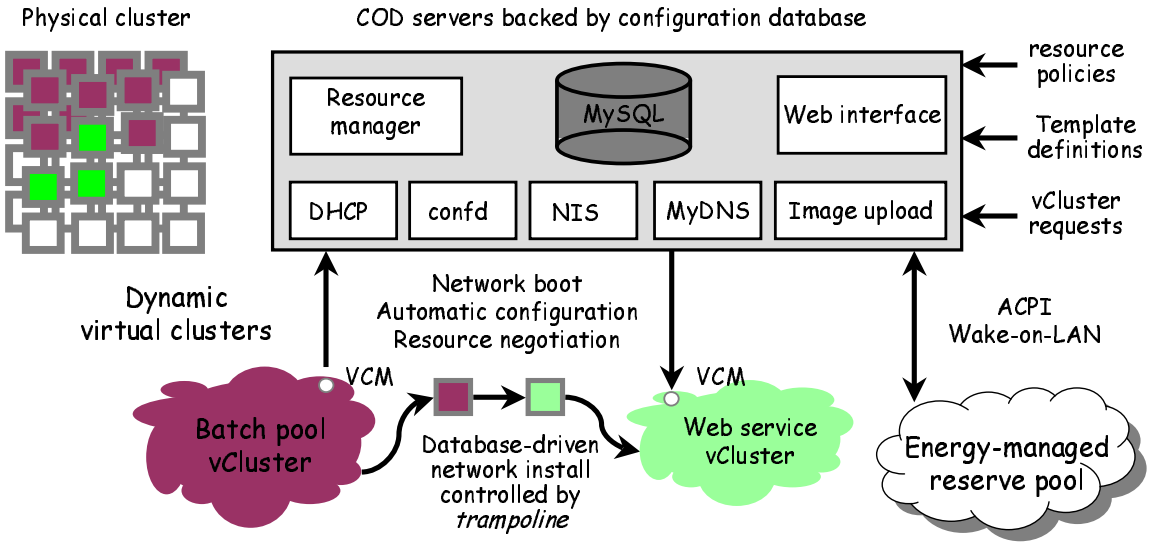

Figure 1: Cluster-on-Demand (COD) partitions a physical cluster into multiple virtual clusters (*vclusters*). Vcluster owners specify the operating systems and software for their vclusters through a Web interface. The vclusters in this example run a batch scheduler and a Web server cluster, which can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

the number of users. Everyday administrative tasks such as user account management devolve to vcluster owners through a PHP/SQL Web interface. COD fully automates the steps to track multiple configurations, track the allocation status of cluster nodes, change a node's role or reassign it to a different vcluster, and/or upgrade a node to a new predefined software configuration. To add a PXE-enabled server to a COD cluster, simply plug it into the network and power it up.

Isolation and safety. COD must prevent users from monopolizing resources or interfering with other users or the system itself. A key benefit of the vcluster concept is that node access is controlled on a vcluster basis. Moreover, COD maintains a consistent environment across each vcluster. Nodes joining a vcluster always install a clean configuration of verifiable integrity; this helps protect against corruption by previous users and leakage of sensitive data between vclusters. This "wipe it clean" philosophy derives from Rocks [21]. Finally, IP-enabled managed power components allow COD to seize control of any node by interrupting its power supply, forcing it to restart and boot onto the trampoline. VLANs enable a cluster manager to isolate each vcluster on a virtual private Ethernet segment (although the COD prototype does not yet configure VLANs). These existing mechanisms are sufficient for basic isolation, but mixed-use clusters will also benefit from secure BIOS functions and configurable QoS shares for VLANs and network storage.

2.2 Related Work

Manageability is increasingly recognized as an important challenge for computer systems design and research. The industry has responded with key initiatives for automated management, including PXE remote-boot technology [13], the ACPI [11] and Wake-on-LAN [12] standards for server power management, device interface standards that promote interoperability, and components such as network switches and IP-enabled power distributors. On the system software side, most systems now obtain network names, network addresses, user identities, and host-specific information from network-administered services such as NIS, DNS, LDAP, and DHCP. These are key enablers for the COD approach.

Several companies are marketing products to automate server management for enterprises and Internet hosting providers. Prominent players in this space include TeraSpring, 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. One contribution of this paper is to describe in detail how to address these goals in a general and flexible way by combining widely used open-source components.

Many systems use PXE network installs to manage node configurations. For example, the popular BpBatch tool from *bpbatch.org* uses PXE boots to install, update, or select local software on each node; the BpBatch boot im-

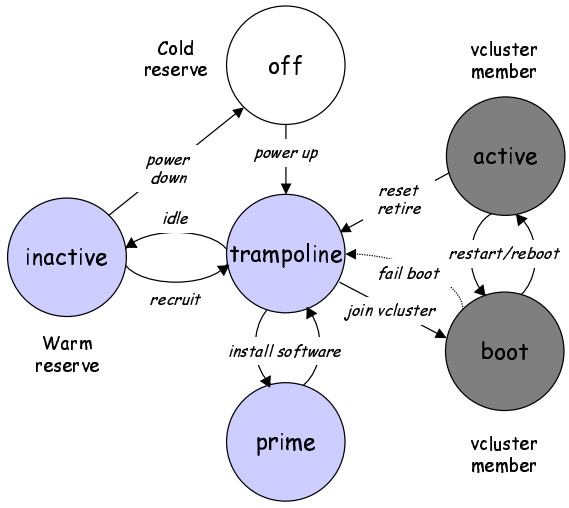

Figure 2: The lifecycle of a COD node.

age is an interpreter for scripts that can examine or partition the disk, fetch a software distribution and install it locally, and/or load a new boot image from a selected local partition. COD selects PXE actions dynamically by querying a configuration database; this key technique of database-driven network installs was described previously for Oceano and Emulab. These systems share the primary goals of automated management, user control, and safe isolation for mixed-use clusters. They also configure VLANs to isolate virtual clusters. 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 global resource manager to coordinate resource usage across multiple dynamic vclusters. The COD premise is that the key features of Oceano and Emulab — Oceano's Web service resource manager and Emulab's rich support for configuring emulation experiments — could apply at the vcluster level in the COD framework. Similarly, many other application cluster managers could be adapted to run as VCMs within the COD framework. Examples include Beowulf, load-leveling batch schedulers, grid managers, enterprise application monitors [25], cluster-based network services (e.g., [23, 17, 18, 9]), and cluster-based network storage [19, 24, 6, 4].

3 Managing Nodes

Figure 2 depicts the states and transitions in the lifecycle of a COD node. When a node first powers on, it loads the trampoline, which takes control of the node, scans it, and prepares it to join a virtual cluster selected by the COD manager. Before assigning a server to a vcluster, COD must first *prime* the node by applying a configuration template, which typically installs one or more partition images on a local disk. When a node is assigned to a vcluster, subsequent reboots load its user-specified software; a node running this user software is **active** in its vcluster. A node leaves its vcluster when the VCM voluntarily releases the node or COD forcibly reclaims it with a power reset. In either case, the node boots back onto the COD trampoline, which may idle the node in a warm (**inactive**) or cold (**off**) reserve pool, or accept commands from the COD manager to return the node to its previous vcluster or prime it for a new configuration.

This section outlines COD's handling of these node states and transitions in more detail. Section 4 then discusses a node's environment while it is active in a vcluster.

3.1 Remote Boot

Every COD cluster node has a PXE-capable network interface and a BIOS configured to boot from the network at power-on or restart. In a typical network boot, the node's ethernet NIC broadcasts a DHCP request with a vendor tag indicating that it is a PXE client; the DHCP server responds with IP information as well as the name and location of a bootloader image to download via TFTP.

The COD DHCP server queries the node status database to determine which image reference to return on a PXE boot request. If the node is **active**, DHCP returns a command to boot from its primed image, e.g., from the primary local partition, or from an image server for a diskless boot. DHCP commands the node to boot the trampoline if the node was **off**, or if it has returned to the **trampoline** state after previous membership in a vcluster. This occurs when a reboot into a vcluster fails (detected by a timeout or a repeated PXE request from a node in the **boot** state), when a vcluster voluntarily relinquishes a node, or when the COD manager forcibly reclaims a node from a vcluster by interrupting its power supply. These correspond to the transitions into the **trampoline** state from the dark grey states in Figure 2.

To boot the COD trampoline, the DHCP server first returns a reference to a small first-stage bootloader called PXELinux (from *syslinux.org*). The PXE card uses TFTP to fetch and boot PXELinux from the specified image server. A PXELinux script then fetches a COD-generated configuration file, also via TFTP. The configuration file contains boot parameters, including the name of the com-

pressed trampoline image and the network location of a COD configuration server (*confd*). PXELinux then fetches the trampoline kernel itself, and boots it.

3.2 On the Trampoline

The role of the COD trampoline is to scan the hardware, idle the node on warm reserve until it is needed, and prepare the node to join a vcluster selected by the resource manager. The trampoline is a minimal Linux kernel image built to run on any standard Intel-based server, and incorporating a minimal ramdisk — currently sixteen megabytes uncompressed, six megabytes compressed — mounted as its root file system. The trampoline can probe and recognize any attached hardware device for which Linux drivers are available. This generality allows a COD cluster to incorporate any standard PXE-capable server or PC supported by standard Linux. The trampoline does not depend on a local disk, and it does not modify local storage unless the node enters the **prime** state.

After booting, the trampoline executes the *kudzu* program to probe the hardware and load drivers, then initializes its primary network interface using DHCP. A user-mode program called *ctramp* then contacts the COD configuration server (*confd*) indicated in the trampoline boot parameters, and transmits a compact XML summary of the node's physical configuration, including partition structures, memory capacity, and CPU speed extracted from *proc*. The trampoline then disconnects and awaits further configuration instructions from *confd*.

3.3 Priming

When the resource manager recruits a node for a new vcluster, *confd* connects to the (*ctramp*) port and transmits image layout information from the selected configuration template in the cluster database. In the **prime** state, a user-mode program called *cprime* configures the node according to the image layout information, optionally partitioning the local disk. The image layout includes a list of URLs for compressed, checksummed configuration images available from an HTTP server, and may include a separate image for each disk partition. To prime the node, *cprime* fetches the images and uncompresses, installs, and verifies them by executing a pipeline of standard programs (*wget*, *gunzip*, *dd*, and *mdsum*). To complete the priming, *cprime* installs a master boot record (MBR), informs *confd* that the node is entering the **boot** state, and initiates a reboot into the newly installed OS.

In our current prototype, each vcluster node independently fetches its images from an HTTP server. To distribute images efficiently we plan to use *pcp*, which pipelines data in parallel through a secure n-ary multicast tree. Emulab's *frisbee* disk loader uses a related multicast approach to distribute images to large vclusters.

Because COD uses bit-copy partition images by default, priming is independent of the operating system and other node software. In particular, the configured OS may use file system formats unknown to the Linux-based trampoline. However, in its simplest form, the bit-copy approach may cause the system to write unnecessary data to the disk, e.g., to zero out blocks that contain no useful data and will later be overwritten by the software. One benefit of paying this cost is that it helps secure the system against data leakage when a physical node transitions from one vcluster to another. Writing complete bit-copy images also allows *cprime* to write the data sequentially at the disk spindle speed. However, COD configuration templates may specify an alternative Linux executable to replace *dd* as the image installer. The installer may interpret the image and place it on the disk in a manner specific to the target OS or file system. For example, Emulab's *frisbee* uses filesystem-specific image compression to avoid zeroing disk blocks that are uninitialized in the image; this is similar to Partition Image (*partimage.org*) and other filesystem-aware image installers.

3.4 The Reserve Pool

One goal of COD is to define common facilities to improve energy efficiency for clusters when they are running below capacity. Today's standard low-end rackmount servers consume on the order of 1-2 MWh per year in electricity ($100-$200), counting cooling costs. The cluster can reduce its power demand by stepping unallocated servers down to low-power states, reactivating them with Wake-on-LAN when they are needed. VCM middleware for time-varying workloads such as network services can concentrate load on a minimal set of servers, releasing unused servers to the energy-managed reserve pool. Our previous work has demonstrated the benefits of this technique for energy management in Web server farms [9].

Surplus nodes idle in an energy-managed reserve pool. If a node in the **trampoline** state is not needed, *confd* commands it to idle in the **inactive** state. When a sufficient warm reserve exists in this state, it commands surplus nodes to step down to the cold-reserve (**off**) state. COD leaves previously installed software (if any) cached for future use. If a reserve node is assigned to reactivate the same configuration template, and the template was not modified since the node's last priming, then COD skips the **prime** phase and enters **boot** to reboot the node directly into its vcluster.

4 The Virtual Cluster Environment

This section discusses the mechanisms to customize vcluster operating environments. When a node is **active** in a vcluster, it runs a user-specified operating system;

Sections 4.1 and 4.2 discuss the means to define and initialize the OS. COD interposes on well-established network management services — DHCP, NIS, and DNS — to shape the node's view of its context, including its IP address and hostname, authorized user accounts and passwords, and access to network storage volumes, as described in Section 4.3. The COD prototype requires that the OS understand and use these services to configure; we have not yet extended COD for alternative services such as LDAP or Active Directory.

4.1 Constructing Images

Users construct partition images for COD configuration templates using tools outside of COD. To simplify creation of new templates, the COD trampoline can extract and save partition image bitfiles and partition maps from a COD cluster node. Thus the usual way to configure a virtual cluster is to instantiate a node using a standard configuration, use arbitrary tools to modify the configuration directly on the node (e.g., by logging in and installing new software from the network), then command COD to save the resulting configuration into a template. The Millennium *rootstock* tool and commercial systems such as ImageCast (*phoenix.com*) use a similar approach.

Importantly, image extraction is the only way to preserve changes to the local file system across node reallocations in COD. Nodes are stateless: any long-term durable data are saved on network storage volumes accessible to the group and assigned to the vcluster.

4.2 Initializing the Node OS

COD does not specify or constrain the mechanisms to perform node-specific configuration in an active node. Some applications may require additional configuration steps, which are necessarily OS-specific and application-specific. To complete these steps, the vcluster owner or installed software may run arbitrary programs or scripts using standard remote execution tools such as *ssh*. For example, a key component of Emulab is the *TMCC* program, which configures Linux/Unix hosts for network emulation experiments. Rocks and Linux/Beowulf compute clusters execute Linux-specific configuration tools such as *kickstart* to initialize nodes and load packages for specific roles within the vcluster. The VCM may trigger these steps when COD notifies it that a node has joined the vcluster. COD vcluster owners may employ other standard tools such as *Cfengine* [8] to update or modify configurations within a COD vcluster.

4.3 Host Configuration

COD interposes on network management services to control each node's view of its environment according to its vcluster membership.

- COD assigns node IP addresses within a subnet for each vcluster. Nodes obtain their IP and router addresses through the COD DHCP server as they boot.
- COD assigns node domain names derived from the vcluster's symbolic name assigned at creation time. Each vcluster occupies a private DNS subdomain. 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.
- Conceptually, each vcluster executes within its own NIS domain, which enables access for specific user accounts and netgroups. Our current implementation uses a common NIS domain and an NIS server shim that queries the configuration database to filter lookups for users and groups.
- COD exports NFS file storage volumes as groups and vclusters are defined. The node configuration template may include an NFS mount map supplied through NIS. The mount map contains a list of NFS file volumes authorized by the site administrator for access by the vcluster's group. The map directs the node to attach these volumes at specified points in the node's file name space. When a vcluster is created, COD issues export commands to file servers to export the selected file volumes to a netgroup containing the vcluster's complete DNS name space.

5 Dynamic Virtual Clusters

The COD resource manager assigns nodes to satisfy resource requests from users. All allocation requests are lists of 4-tuples 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 satisfies or rejects each request as an atomic unit. Note that different nodes in the same vcluster may use different templates, to allow for specialized roles within the vcluster. As a simple basis for allocating resources under constraint, all requests inherit a *priority* level assigned to the owning group by the site administrator.

A request to create a vcluster takes a similar form. If COD accepts a vcluster create request, it establishes a *lease* expiration time on the vcluster according to the requesting group's default lease time or a requested lease time, whichever is lower. Vclusters may return nodes to COD voluntarily or in response a resource reclamation request from the COD manager. COD reclaims resources from a vcluster only the vcluster's current size exceeds its create

size, or if the lease is no longer valid, i.e., because it has expired or the site administrator has canceled it.

5.1 Node Selection

The nodes selected to satisfy a request tuple must match certain attribute constraints. First, they must capable of accepting images associated with the specified configuration template. Second, they must meet or exceed the specified request attributes, e.g., to meet a minimum memory or storage requirement. The desired goal state for the resource manager is to satisfy high-priority node requests using low-cost nodes — preferably idle nodes — while simultaneously matching the maximum number of unique nodes to highly constraining requests. For example, assume we have a node with multiple network cards and a one gigahertz processor. This node may be used to satisfy a request specifying a gigahertz processor. However, if fast processors are more common than multi-interface nodes, the multi-interface request is more constraining than the request for a gigahertz processor and we would prioritize this node to fulfill a multi-interface node request over one for a gigahertz processor request.

Our node selection approach employs a matching scheme similar to Condor's *classads* [22]. Node attributes are specified using XML. XML is a natural choice because it allows for a tree-like structure for node attributes. XML standards specify methods for attribute matching and transformations from one XML object to another based on a set of rules. XML libraries allow COD components to store, parse, and retrieve data easily in a standard format with cross-platform support. COD uses simple operators to determine if two attribute specifications are *equal* or if one *satisfies* the other. The attributes of a selected node — obtained by hardware probing each time the node boots the trampoline — must *satisfy* attributes uploaded with the template and specified in the request itself.

To simplify request matching, COD uses the *equal* operator to group nodes into equivalence classes or *node types* as they are discovered. For each request tuple, COD matches the request and template attributes against the type attributes to determine which types are eligible to satisfy the request. This is a significant savings because clusters typically have a small number of node types, although they may have a large number of nodes of each type. The site administrator may group the nodes of a given type into *blocks* based on network proximity, and assign each node type a *cost* to reflect their value in the system (e.g., assigning higher cost to special node types, such as multi-interface nodes). Currently these block and cost assignments are manual.

The COD resource manager satisfies pending requests in order of their vcluster priority, highest to lowest. To satisfy a request tuple, it first identifies the eligible (satisfying) node types with blocks large enough to handle the request, then selects the least-cost eligible type, and finally allocates nodes from the largest block of that type. If no suitable blocks are available, it repeats this process with the nodes allocated to each vcluster priority level, to find a suitable block of nodes to reclaim from a lower-priority vcluster. The request blocking mechanism ensures that nodes allocated together are close to each other on the network

5.2 Stability

The nature of resource allocation in COD leads to a large overhead in transferring a node from one virtual cluster to another. Reallocation overhead is measured in minutes. Accordingly, the resource manager must take care to avoid thrashing if reported resource demands change quickly relative to the reallocation time.

COD uses a sliding time window that grants a node immunity from reallocation after COD assigns it to a virtual cluster. The size of this window represents the *scheduling inertia*. A mapper with a large inertia allocates stable virtual clusters, allowing low-priority customers to retain new nodes regardless of short-lived spikes in resource demands. A mapper with a smaller inertia responds better to a constant stream of high-priority requests, each wanting nodes for a relatively short period of time. However, too small an inertia could lead to thrashing. Ideally, the scheduling inertia should allow stable, long-term virtual clusters and highly dynamic, short-lived virtual clusters to share the same hardware base.

5.3 Virtual Cluster Managers (VCMs)

Dynamic resizing requires coordination between the COD manager and cluster middleware. A VCM controls each vcluster that is capable of adaptive resizing. The VCM is a network server certified to act on behalf of the vcluster's group. The VCM may run within the vcluster on its minimal leased allotment, or outside of the vcluster. COD does not specify the mechanism to instantiate the VCM, e.g., as a side effect vcluster creation. A node template's post-boot init sequence may start a VCM, or the vcluster creator may launch the VCM after COD notifies it (currently by e-mail to the cluster group) that the vcluster was created. The prototype assumes the VCM runs within the vcluster, and authenticates it by IP address.

VCMs negotiate node allocations with the global COD resource manager. In general, we expect that each dynamic virtual cluster is under the control of middleware such as a batch scheduler or other cluster manager. Note that the middleware may itself host applications (such as Web services or compute jobs) that are unaware that this resizing is taking place; the middleware is responsible for subcontracting its allotted resources to specific tasks

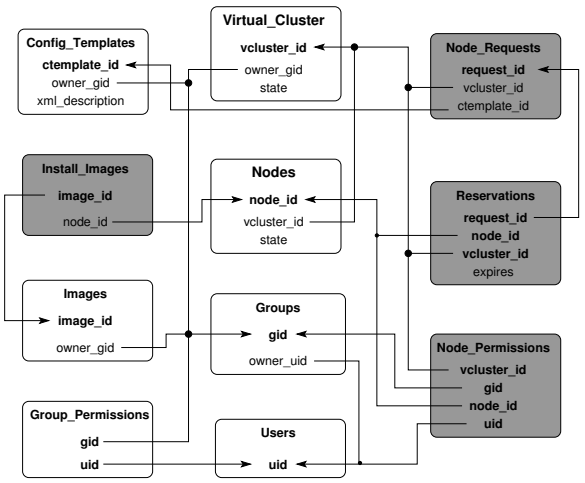

Figure 3: An overview of the database schema

without further involvement from COD. The VCM mediates interactions between the local cluster manager and the COD manager using COD's *Node Management Protocol* (NMP), NMP allows either party to request nodes from the other or transfer node control to the other. The NMP uses XML as the data payload over a TCP/IP transport, with HTTP-like headers.

During periods of resource contention, COD may reclaim nodes from a low-priority vcluster. NMP reclamation requests include an $attribute$ description for the requested nodes, just as node requests to COD. If the VCM holds multiple nodes that *satisfy* the request attributes, it is free to select which of these nodes to return to COD. This allows COD to reallocate nodes with minimal disruption to affected virtual clusters.

5.4 Coordination

The COD resource manager and servers communicate and synchronize through the configuration database, which stores all information about users, physical nodes, and virtual clusters. Our prototype uses MySQL.

Figure 3 shows a simplified view of the key database tables and their relationships. The primary keys are in boldtype, and the arrows represent foreign keys in other tables indexed by that primary key. Tables in white boxes are persistent tables, while transitional tables—tables used to coordinate components—are in shaded boxes.

Consider a typical Web form request to create a vcluster with a single configuration template. The Web interface validates the user through the `Group_Permissions` table, inserts a new virtual cluster into the `Virtual_Clusters` table with **pending** status, and adds a request for nodes into the `Node_Requests` table. The next time it executes, the resource manager attempts to fill the request from the resources in the `Nodes` table, using the policy described above. If the allocation succeeds, it assigns nodes to the vcluster in the `Nodes` table, and updates the vcluster's status in `Virtual_Clusters`. The DHCP daemon and *confd* examine this state to instantiate the trampoline on the node and direct it to apply the assigned configuration template.

When the configuration template is applied to a node, the resource manager updates the node's permissions in the `Node_Permissions` table and inserts the node's `Install_Images` information in the pending state. After the image is installed, the resource manager updates the node's status in `Install_Images` and `Nodes`. The resource manager inserts completed reservations into the `Node_Reservations` table and updates the state of each newly-reserved node in the `Nodes` table. The resource manager clears reservations when a node leaves the vcluster.

This simple example illustrates the role of the tables in coordinating interactions of the COD servers.

Persistent tables are divided into two categories: environment-initialized and user-initialized. The `Nodes` table is an environment-initialized table that depends on the hardware resources. Users populate the remainder of the persistent tables. Through the Web interface, authorized users can create or modify user accounts, groups—and, therefore, update group permissions, images, virtual clusters, and configuration templates.

Components access the `Group_Permissions` and `Node_Permissions` tables to authorize requests. We described a few examples in the above scenario. The Web interface must validate all create requests, and the Image Server and Web interface use the permissions tables to validate changes to configuration templates.

6 Example VCM: SGE Batch Pool

To experiment with dynamic virtual clusters, we implemented a COD VCM extension for Sun GridEngine (SGE), which is widely used to run compute-intensive batch jobs on large clusters. 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. The VCM extension allows SGE to run within a COD vcluster that grows and shrinks with demand and resource availability. To simplify the experiments, we configured SGE to schedule at most one job on each active execution host. Users submit jobs by executing the SGE *qsub* command on any host in an SGE vcluster.

We implemented the COD-enabled SGE VCM with about 600 lines of C code that runs as a server daemon process

on the master host. The VCM interacts with the COD manager through a network connection, and executes sequences of SGE administrative commands to query the active set status and change its membership. To maintain a uniform environment across the active set, as required by SGE, the vcluster configuration template defines a common space of user identities 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 SGE VCM uses simple policies to size the active set. The VCM executes a *resize* function every $epoch$ seconds to check the status of the batch pool, using 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, the VCM requests a new execution node for every n queued jobs. The *resize* also records any nodes that are idle and timestamps them. If there are no queued jobs, the VCM returns each idle node to the COD manager after it has been idle for $maxidle$ seconds, down to a minimum idle reserve of k nodes.

The SGE VCM resizes the active set as follows. To add an execution host, the VCM executes the SGE *qconf* command with a standard template to activate the node by its DNS name and establish a job queue for it. Before enabling the queue, the VCM remote-executes the SGE daemon processes (*sge_commd* and *sge_execd*) on the node. To remove a node, the VCM 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 COD manager may unilaterally reclaim nodes from the batch pool to meet demands from higher-priority vclusters. In this case, the VCM negotiates with the COD manager to select each victim node. First the VCM uses *qstat* to search for idle nodes. If all nodes are busy, the VCM selects the busy node with the job that started most recently. SGE currently has no checkpoint facility, so this policy limits the amount of wasted computation when the VCM is forced to restart a job.

The VCM allows SGE batch pools to benefit from COD's support for energy management and controlled cluster resource sharing. COD isolates other cluster users from interference by batch jobs, and ensures a consistent environment across the batch pool. Our current VCM policies are stable and effective when each batch pool serves a single priority class of single-node compute-bound jobs that run for longer than the reconfiguration times. 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.

A low-priority batch vcluster is similar to the Condor resource-scavenging model [20]; that is, COD allocates only idle nodes to the batch pool. 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. Finally, COD allows more flexible priority schemes for allocating nodes to the batch pool and other vclusters.

6.1 Other VCMs

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 [16]. 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 have previously demonstrated a system to dynamically resize clustered Web server allocations to adapt to load changes [9]. While this system ran as a standalone cluster manager, it is natural to run it as a COD VCM.

7 Experimental Results

The COD prototype is approximately thirty thousand lines of code, consisting of a DHCP server, the configuration daemon (confd), the resource mapper, the VCM network negotiation interface, and a MySQL implementation of the database API. Other services, such as NIS and NFS, obtain configuration files from scripts that pull state out of the database. The pieces of the COD framework function as described in this paper. One simplification is that our test configurations include only two node types, with one block per node type. We have observed successful dynamic node reassignments under VCM control, but do not have graphs of complete coordinated experiments with dynamic allocation at this time. This section describes our initial results with the COD prototype, including performance measurements and observations on node priming.

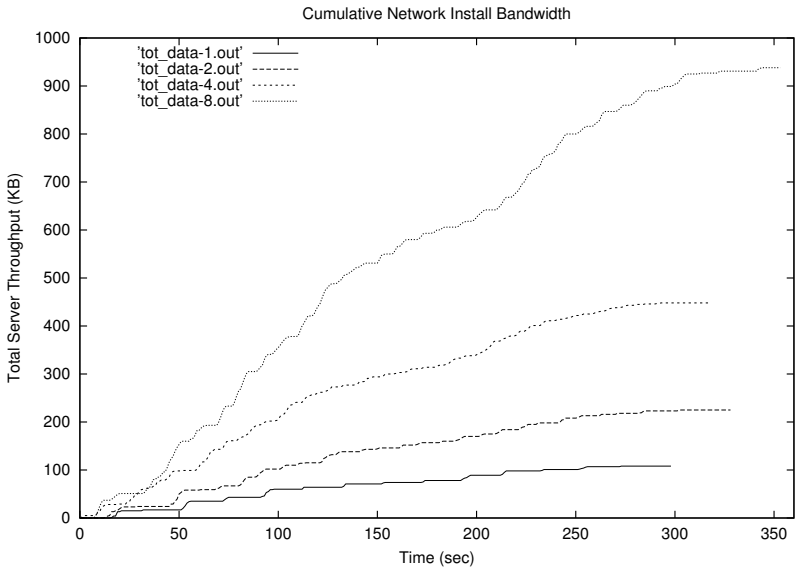

Figure 4: Cumulative image server bandwidth as a function of time during node priming with the Debian Linux 3.0 configuration template, showing the behavior of parallel node installations.

Image	Size (MB)	Size (gz)	Time (s)
Duke Linux	2,048	105	122
Duke BSD	2,048	137	129
Debian Linux	1,951	422	127

Table 1: Elapsed priming time for various images (sec)

7.1 Experimental Setups

In our first experiment, all our management services ran on a single node; these include the DHCP server, confd, the resource mapper, a TFTP server, and an apache web server. All hosts in the experiment were single processor IBM e-Server xSeries 330 machines, each with a 1 GHz Pentium III processor and an 18 gigabyte IBM Ultra Wide SCSI drive. The network was switched 100 Mbps Ethernet, with Intel EtherExpress Pro NICs in all nodes. All COD servers ran under Debian Linux version 3.0.

The first experiment measures the time it takes to prime a node with a configuration template. Table 1 shows the results of using COD to install three different images. The first image is a snapshot of a standard Linux installation configured for the Duke Computer Science cluster. This image includes standard clustering software such as monitoring tools, a batch queue, and software development libraries. The filesystem is second extended (ext2). The second image is a FreeBSD installation configured for the Duke Computer Science cluster with similar software. The filesystem is an extended primary partition containing four UFS logical volumes. The final image is an out-of-the-box installation of Debian Linux 3.0 [14]. The filesystem is ext2 and contains a full suite of development tools and utilities. Images are stored in compressed format on the server; they are decompressed client-side before they are written to disk. Write speed to disk for all images is approximately 16 MB/sec, approaching the limits of the SCSI drives.

The testbed for our second experiment is a ten-node testbed consisting of IBM x330 and x335 e-Server nodes. The x335 nodes have a 2 GHz Pentium IV processor, a 40 gigabyte IDE drive, and two Broadcom 5703 gigabit NICs. The network was switched 100 Mbps Ethernet. Two x330 nodes hosted the management services, with an eight-node cluster of x335s.

Figure 4 shows total installation progress for parallel node priming measured in megabytes downloaded from the image server. Eight nodes easily install in parallel in under six minutes. As the images install, the management services update state to reflect the new vcluster member of the nodes. This allows users to access new nodes immediately following priming without the need for post-install configuration.

Figure 5 examines the installation progress as a function of combined server throughput to all nodes. The bursty nature of install traffic is due to image compression. Large portions of empty disk compress well, resulting in small network utilization, but decompress client-side into large blocks. The network traffic stalls as the

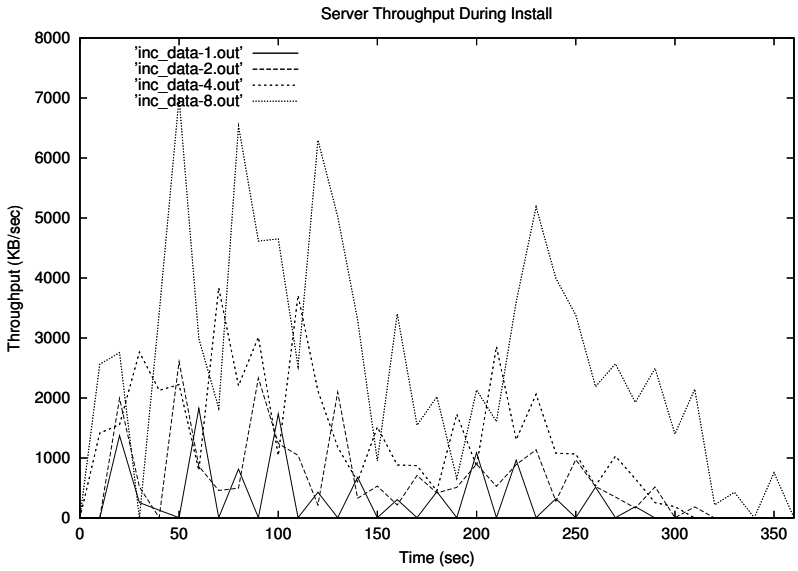

Figure 5: Bandwidth variations through time during the install, showing compression effects for a bit-image install.

node writes these blocks to disk, while performing a cryptographic checksum in parallel. This further reinforces the fact that wipe-it-clean bit-image installation times are bound by the write speed of the client disks.

8 Conclusion

Cluster-on-Demand shows how PXE remote-boot technology can serve as the basis for a cluster operating system framework, enabling flexible, automated, dynamic sharing of a cluster by independent groups of users. COD can combine with virtual machine technology to enable vclusters to share individual physical servers at a fine grain.

Our work with COD addresses key challenges that oppress cluster users today: staffing costs for cluster administrators, loss of productivity due to configuration errors and delays, brittle software environments that do not meet the full range of user demands ("should we run Linux or Windows on our cluster"?), and the difficulty of capacity planning and resource scheduling in the presence of bursty and unpredictable demand. COD is designed to give cluster users powerful "push-button" control over their software environments, enable cluster managers to specify policies controlling the amount of resource allocated to each group, and improve productivity and return on investment for cluster infrastructure.

Flexible site management using the COD model will also 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. Utility computing envisions services and applications that float freely in a global pool of raw servers and storage, which are automatically provisioned and distributed (or sold) according to demand, much as electricity is today. 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. Our approach has the potential to use resources effectively and scale incrementally by plugging in more resources, while holding human administrative burdens constant.

Availability

We intend to release COD as open source when it is ready.

References

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

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

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

[4] 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.

[5] 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.

[6] T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, and R. Wang. Serverless network file systems. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 109–126, December 1995.

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

[8] M. Burgess. Cfengine: a system configuration engine. *USENIX Computing Systems*, 8(3), 1995.

[9] 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.

[10] 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.

[11] C. Corp., I. Corp., M. Corp., P. Technologies, and T. Corp. Advanced Configuration and Power Interface Specification. `http://www.acpi.info/spec.htm/`.

[12] I. Corp. Intel Networking Wake-On-LAN. `http://www.intel.com/network/connectivity/resources/technologies/`

[13] I. Corp. Intel Wired For Management. `http://www.intel.com/labs/manage/wfm/tools/pxeplus.htm`.

[14] Debian. Debian GNU/Linux 3.0 Release Information. `http://www.debian.org/releases/testing/`.

[15] 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.

[16] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid. *International Journal: Supercomputer Applications*, 2001.

[17] 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.

[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. Extended version appears in ACM Transactions on Computer Systems (TOCS).

[19] E. Lee and C. Thekkath. Petal: Distributed virtual disks. In *Proceedings of the International Symposium on Architectural Support for Programming Languages and Operating Systems*, pages 84–93, October 1996.

[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] 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.

[22] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC)*, July 1998.

[23] Y. Saito, B. N. Bershad, and H. M. Levy. Manageability, Availability and Performance in Porcupine: a Highly Scalable Cluster-Based Mail Service. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, pages 1–15, Kiawah Island, December 1999.

[24] 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)*, October 1997.

[25] 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.

[26] 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.