Harnessing Virtual Machine Resource Control for Job Management

Laura Grit, David Irwin, Varun Marupadi, Piyush Shivam
Aydan Yumerefendi, Jeff Chase, and Jeannie Albrecht[†]
Duke University *University of California, San Diego*[†]
{grit,irwin,varun,shivam,aydan,chase}@cs.duke.edu, jalbrecht@cs.ucsd.edu

Abstract

Virtual machine technology promises important benefits for grid computing and cluster batch job systems, including improved isolation, customizable workspaces, and support for checkpointing and migration. One way to gain these benefits is to "drill holes" in existing batch computing systems; however, we believe these new capabilities warrant a rethinking of the architectures of existing systems. We propose separating resource control for VMs into a new foundational layer that focuses narrowly on resource management.

We present JAWS*, a new batch computing service that is built as a thin-layer above a resource control plane that enables it to share a common pool of networked cluster resources with other cluster services.* JAWS *executes jobs within isolated virtual machine workspaces. We discuss how exposing resource control allows* JAWS *to leverage VM-based resource isolation as a means to learn models of application behavior, and use those models to guide scheduling policies for efficient resource sharing.*

1 Introduction

Virtualization technologies continue to mature, exposing new resource management control to existing batch computing systems. Nearly all virtual machine software supports new controls such as strong performance isolation, save/restore of running VMs, and live migration. Moreover, VMs may be created and destroyed within a matter of seconds by using techniques such as flash cloning of stored images. Virtualization promises to change the way cluster and data center resources are managed at the lowest level. Machines are no longer bound to physical hardware; they may be created, destroyed, started, stopped, saved, restored, or migrated at the push of a button. How should we expose and leverage this new control for cluster and grid computing environments?

In recent work, we explored a simple technique to leverage virtualization: instantiate complete middleware environments (e.g., Globus [17] and Sun Grid Engine [8]) within isolated logical containers (workspaces) comprising sets of virtual machines. We showed how add-on controllers for each hosted environment could grow and shrink the computing resources assigned to their containers by invoking resource leasing services in an underlying *resource control plane* to obtain resources according to demand. In essence, we introduce a new software layer to manage virtualized resources below the middleware and below the operating systems: we refer to the resource control plane as "underware". The architecture and current implementation (Shirako) of the underware layer is outlined in Section 3 and described in more detail in an earlier paper [8].

This simple approach allows multiple user communities and middleware environments to share computing resources with policy-based adaptive resource control. However, it is limited in that the new control interfaces do not apply to individual jobs, but only to complete middleware environments. The middleware controls job scheduling on its machines; the mapping of jobs to machines is not known outside of the middleware, and the scheduler may place multiple jobs on a machine. Thus it is not possible for a controller to migrate a job by migrating the containing VM, or use VM control interfaces to suspend/resume, save/restore, or reserve resources for individual jobs. An alternative is to retrofit VM support as an option into existing middleware for job management [11, 20]. However, virtualization technology offers a rare opportunity to rethink the software architecture from the ground up.

In this paper we propose a streamlined job execution system called JAWS (Job-Aware Workspace Service) that uses the Shirako resource control plane to run each job within an isolated virtual container sized for the job. Although it manages job execution, JAWS is not a job scheduler: all functions to schedule and arbitrate shared resources migrate to the underware layer. In this way, a JAWS job service can share a common pool of networked cluster resources with other cluster services, including other middleware environments such as grid

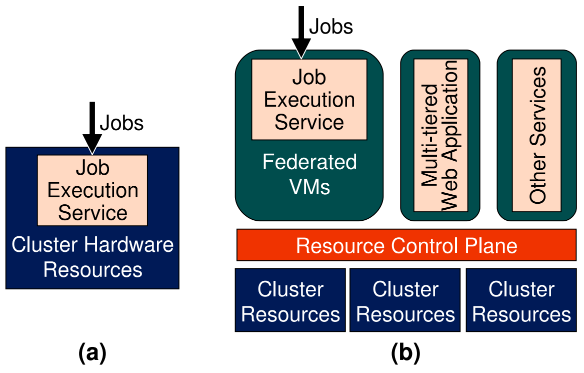

Figure 1: A classic job execution service running on a cluster's physical machines (a) compared to a job execution service running on federated virtual machines above a generic resource control plane and sharing underlying resources with other cluster services (b).

software or applications such as Web services. The control plane underware provides common facilities to reserve, allocate and configure resources and manage VM workspaces, and to control sharing of resources across multiple autonomous resource providers. These functions apply to all hosted environments (see Figure 1), including the JAWS job service.

JAWS has access to all VM control functions exposed by the underware layer on a per-job basis. We explore one example of how to use this control: we integrate *active learning* techniques developed in the NIMO project [21] to construct performance models that are useful for proactive resource management. We built an automated NIMO controller for JAWS that manipulates VM control "knobs" to learn models to predict job runtime for a given assignment of resources (e.g., CPU capacity and memory). Before deploying JAWS , NIMO collected job data from a specialized testbed with diverse machines; the new JAWS-based system uses VM resource control to explore the space of resource assignments automatically on any VM hosting cluster.

In this paper we describe the benefits of exposing resources in a new foundational layer and using it as the basis for a job execution service (Section 2). We illustrate the benefits of the architecture through the design and implementation of JAWS (Section 3). We also discuss one benefit of exposing resource control by integrating active learning into JAWS (Section 4).

2 Overview

Increasingly, high-end computational science and engineering problems are solved on a complex hierarchy of resource and software systems that consist of scientific codes, portals, workflow tools, web services, resource management middleware, and underlying cluster and HPC resources. In this setting, effective execution depends on a complex set of interactions between the end-user, different layers of software infrastructure, and the underlying resources.

These systems are typically built on grid middleware that, in turn, is layered upon traditional, mainstream cluster operating systems and batch schedulers. The job-level and process-level resource control of these systems offers limited predictability and isolation and makes it difficult to orchestrate large composite applications effectively, particularly for deadlines or dynamic behavior [5, 20].

In addition, future systems must deal with more diverse software configurations: e.g., different library versions, operating systems, or middleware services. To harness computing resources for a given task, it is necessary to install and configure the correct software stack.

Virtual machines or workspaces (VWs) [5, 11, 14] have been proposed as a means to address the challenges of isolation, customization, and management. A virtual workspace provides access to a collection of resources (physical or virtual) and a software environment suitable for the execution of a job or a collection of jobs.

2.1 Workspace Architecture

In this paper we address the architectural questions of creating, managing, and using virtual workspaces for job execution. Our main architectural principle is to separate control over resource sharing from job execution and management. Similarly to [11] we delegate control over a workspace to the job owner, but extend this concept by factoring sharing control out of the middleware into a general resource control plane (underware). Factoring sharing control into the underware layer allows operators to control resource assignment and usage while insulating them from the details of any specific environment or service.

Decoupling sharing control and job management simplifies the job execution service. The underware layer takes care of the underlying trust and authorization issues, and resource arbitration policy.

2.2 Benefits of VM Control

The proposed architecture supports important capabilities that are difficult to obtain in many of today's cluster/grid environments, but are important for HPC as it evolves and seeks to incorporate new virtualization technology.

Resource Control. Resource owners need control over how their resources are used and users must know what resources are available to them. The resource control plane uses *leases* to represent contracts over resource commitments that grant the holder rights to exclusive control over some quantity of resource over a specified period of time. Leases are dynamic and renewable by mutual consent between a resource provider and a guest (e.g., JAWS). The leasing abstraction applies to any set of computing resources that is "virtualized" in the sense that it is partitionable as a measured quantity. For example, by using performance-isolating schedulers, a resource might comprise some amount of CPU capacity, memory, storage capacity, and/or network capacity, measured by some standard units, and with attributes to describe them.

A sliver is the partition of a physical machine's resources; leases in JAWS are for a set of slivers which form a virtual workspace. Slivering may result in the collocation of two complementary sized virtual workspaces on the same host machine. After the initial slivering, a virtual workspace may require resizing to accommodate for the changing requirements of a job or to meet client performance targets. Virtual machines may be resized along multiple dimensions to enlarge or shrink a job's virtual workspace. Resizing may involve migration to a new host.

Resource Federation. The underware control plane may coordinate resource allocation and deployment across multiple autonomous clusters—this is an example of resource *federation*. Independent clusters may donate resources to third-party brokers. The underware control plane uses these brokering intermediaries to implement resource sharing policies.

Robust Computing. Virtual workspaces in JAWS enable a more robust computing environment for jobs including: saving VM images as they execute for checkpoint or recovery, snapshotting file system images, and VM migration to balance load or perform system maintenance. In addition, our underware control plane uses a database to persist all state pertaining to deployed virtual machines and pending resource requests as advocated by [18]. Persistent state makes it possible to recover after a crash and restore the system into operation with minimal consequences.

Advance Reservations. Sophisticated applications may want to know what resources they have control over and when, while some applications may require resources at specific times (e.g., for the coordination of a workflow). To support intelligent applications the resource control plane must provide resource reservations, both immediate and advanced.

Allowing advance reservations is difficult in any system where applications may require resource for an unpredictable amount of time. For example, in JAWS and classic job execution services, it is possible that job runtime estimates are inaccurate [13] and the job may require resources longer than expected. When such a system allocates advanced reservations, there is the risk that resources allocated to a job may be reclaimed before the job completes to satisfy an advanced reservations. The use of leases in our architecture is beneficial in these situations because they offer a measure of assurance to JAWS: resources are revoked at a well-known time allowing JAWS to checkpoint the state (VM image and disk) of unfinished jobs and re-bind them to resources at a later time.

Support for Diverse Environments. Within a grid or a cluster, jobs may require different and complex execution environments. Each virtual machine on a host can run its own customized software stack from the operating system up to application services.

JAWS can leverage software environments packaged as *appliances*: complete bootable images containing both application and (custom) operating system components. The appliance model is well-suited to virtualized infrastructure, and has revolutionary potential to reduce integration costs for software producers and to simplify configuration management of software systems. For example, rPath [1] is packaging appliances for key driving applications, including the STAR application from Brookhaven and the Community Climate System Model, and supporting infrastructure such as the Open Science Grid toolset.

Resource Sharing. The position of the resource control plane in the software stack makes it possible to host multiple services from a common pool of resources. The resource control exposed by the underware layer, enables a service to completely customize its workspace and to adjust dynamically the resources bound to its workspace. Compared to traditional batch schedulers that are configured statically to cluster resources, JAWS is dynamic in the number of resources it uses, making it possible to reallocate unused resources to other cluster service that may require more resources at that time.

Application Management. The type of jobs running in HPC job management environments are evolving from self-contained entities to long-running, multi-component workflows that can more closely resemble services than classical HPC jobs. As long-running and complex application workflows become more prevalent, a need for long-lived application managers to monitor resource usage and application status as well as allow user interaction with a running application. These managers may take advantage of resource controls exposed by the underware layer. For example, the manager may adjust the resources associated with the application's virtual workspace, or may control other functions like checkpointing a job or migration. JAWS acts as the application manager to its client jobs.

3 JAWS Design and Prototype

JAWS is designed as a thin layer above Shirako: an implementation of an underware control plane architecture [8, 17]. Shirako is a toolkit for distributed resource management based on resource leasing. There are three

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

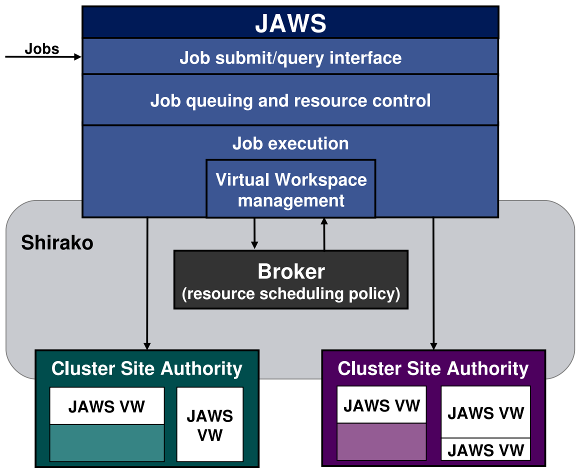
JAWS
Jobs
Job submit/query interface
Job queuing and resource control
Job execution
Virtual Workspace management
Shirako
Broker
(resource scheduling policy)
Cluster Site Authority
JAWS VW
JAWS VW
Cluster Site Authority
JAWS VW
JAWS VW
JAWS VW

Figure 2: JAWS uses Shirako to obtain resources and to configure virtual workspaces (VWs). JAWS determines how to queue and manage its submitted jobs. The resource control policy determines what resources to request from a Shirako broker.

different roles Shirako servers may take: service manager (resource consumer), site authority (resource provider), or broker (resource arbitrator and provisioner). [8] details Shirako's protocols and design; we give a brief summary here as it relates to JAWS.

Figure 2 illustrates the basic design of JAWS. JAWS acts as a Shirako service manager, or resource consumer, that requests resources for jobs from a broker: brokers are granted resource rights by Shirako site authorities for subsets of their resources for a specified duration. Brokers may aggregate resource rights from multiple site authorities to coordinate cross-site allocation. Brokers enable a controlled form of federation: sites explicitly delegate resource rights to brokers for fixed periods of time.

Brokers follow resource scheduling policies that determine how to partition their resources among multiple services. If a broker grants a resource request a signed ticket is returned to the JAWS service manager which is then redeemed at the site authority. The site authority instantiates the resources and returns a lease to JAWS. A lease signifies a capability to use a resource for a period of time. Shirako site authorities use Cluster-on-Demand (COD [3]) as a back-end resource manager for allocating virtual machines to JAWS. COD includes support to configure and instantiate OS images and IP/DNS namespaces on physical servers and Xen virtual machines.

While our current implementation uses Xen, our goal is to remain independent of any specific virtualization and to support any technology that offers accurate, low-overhead performance isolation, migration, and save/restore of VM images. To this end our design factors out technology-specific code into resource drivers that take as input a set of technology-independent properties that describe the attributes of the machine (e.g., IP address, CPU share, Memory, etc.). Resource drivers are not limited to VMs, and we have implemented drivers that re-image and configure physical machines.

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

JAWS defines a *resource control* policy that determines *how much* resource is needed for a job (e.g., one machine or multiple machines, how much CPU capacity and memory to bind to each machine, and how much local disk space to incorporate into the workspace) and *how long* the resource is needed. The resources controlled by JAWS may vary with time due to contention from other systems using the same resource control plane. Under contention the control plane can redirect resources following a predefined resource scheduling policy, *independent* of the policy used by JAWS. However, since each JAWS job runs within a leased performance-isolated workspace, instantiated jobs are protected from contention and cannot be affected by other systems sharing the same control plane.

A Shirako broker acts as the resource scheduler for JAWS. In many respects a Shirako broker behaves like a conventional job scheduler: we have implemented many commonly used scheduling algorithms as pluggable policies for Shirako brokers, including Earliest Deadline First, Proportional Share, and Backfill. In our prototype the broker schedules leases for virtual machines bound to varying amounts of CPU, memory, bandwidth, and local storage from an inventory of physical hosts. The default broker policy module for our prototype defines a simple greedy algorithm that fills resource requests, in the form of 4-tuples (CPU, memory, bandwidth, storage), in a FCFS worst-fit fixed order.

4 Resource Control with Active Learning

A job in JAWS is a specific instance of an application with certain input parameters and input dataset. Since instances of an HPC application may execute multiple times, JAWS has an opportunity to learn the resource demands of HPC applications from prior application runs. The information gathered from previous executions of an application is captured in a performance model that predicts the application's performance (e.g., runtime) as a function of specific virtual workspace specification (e.g., amount of CPU, memory, network bandwidth). Explicit resource control in JAWS enables integration of active learning techniques to learn such models accurately and

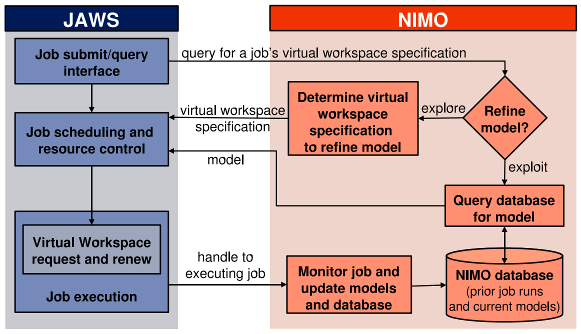
JAWS
NIMO
Job submit/query interface
query for a job's virtual workspace specification
Determine virtual workspace specification to refine model
explore
Refine model?
virtual workspace specification
Job scheduling and resource control
model
exploit
Query database for model
Virtual Workspace request and renew
Job execution
handle to executing job
Monitor job and update models and database
NIMO database (prior job runs and current models)

Figure 3: JAWS interacts with NIMO to determine an effective and efficient virtual workspace specification for its jobs. For each request, NIMO must decide if it will explore a new virtual workspace to enhance an application's performance model, or exploit an existing model. NIMO monitors the execution of each job to collect the training data for updating its models.

quickly (Section 4.1). JAWS can use learned models in several ways.

Reservations and deadlines. Prior knowledge of a job's runtime helps JAWS select an appropriate lease length for the job's workspace. JAWS must estimate job completion times so that it can schedule job requests in the future (advance reservations). Correct runtime estimation help JAWS minimize the risk that resources will be revoked from a job before it completes. We are experimenting with providing "minimal-impact" deadline scheduling: performance models can be used to predict the minimal amount of resources a job needs to complete before a given deadline (i.e., "just in time"). This is useful in situations where unplanned events require jobs to be completed before a deadline, but the job scheduler seeks to minimize the impact on other jobs.

Informed workspace collocation and coscheduling. Understanding the relationship between resource allotments and application performance enables JAWS to assist the resource control plane to make intelligent decisions about efficiently assigning virtual workspaces to underlying hardware resources. For instance, using information supplied by JAWS, the resource control plane may collocate a virtual workspaces with complementary resource profiles: a CPU-intensive workspace may be collocation with an I/O intensive one.

4.1 Active Learning of Models in NIMO

In our current JAWS prototype, the job scheduling policy enhances its understanding of job resource demands by interacting with NIMO [21]: a system that uses active learning techniques to learn accurate performance models. Previous work on NIMO learned models that predict application performance on physical resources; here we use NIMO to learn application performance models for jobs running inside virtual workspaces.

NIMO leverages the explicit resource control exposed in JAWS to learn accurate models proactively and quickly. Figure 3 outlines the interactions between JAWS and NIMO.

NIMO builds a performance model to estimate the performance of an application running within a virtual workspace. The performance model estimates application runtime using three inputs: (1) the *resource profile* of the resources assigned to the workspace (e.g., CPU cycles and network bandwidth for VMs), (2) the *data profile* of the input dataset, and (3) the *application profile* that captures the behavior of the application resulting from the interaction of the application, the resources assigned to it, and its data profile.

NIMO uses active learning to expose systematically and proactively the relevant training data samples required for building an accurate performance model. To collect one sample training data point, NIMO requests a specific virtual workspace within which to run the job. While a job is executing, NIMO gathers instrumentation data from passive instrumentation streams using common tools (e.g., sar, tcpdump) with no changes to application or system software. The instrumentation data is then processed to produce a training sample data point. NIMO collects multiple samples to build its models by proactively running applications within virtual workspaces with different resource profiles.

The choice of virtual workspace specifications for collecting the training data points in NIMO is guided by *design of experiments* (DOE) [9] active learning [4] branches of statistics to do automated and intelligent gathering of training samples from a potentially large space of resources and input datasets for a given application. DOE offers off-the-shelf algorithms for quickly identifying relevant parameters and interactions that affect application performance. Active learning algorithms can identify the next application run, i.e., the virtual workspace specification and the input dataset for the execution, that will maximize the accuracy of the model learned from previous runs of the application. Further details on active learning algorithms are in [21].

JAWS provides an ideal environment for NIMO to apply active learning techniques to learn application performance models. Explicit resource control in JAWS through the resource control plane enables NIMO to obtain a desired workspace configuration, e.g., JAWS can easily configure a virtual workspace for NIMO with a specific CPU and memory configuration. Since JAWS configures such workspaces in a matter of seconds, it enables NIMO to collect a vast set of training data samples simultaneously and with ease.

In prior work, NIMO used physical resources to collect the training sample data to learn the performance models. With physical resources, the data collection re-

Application	Description
CardioWave [16]	Cardiac electrophysiology simulation (MPI)
BLAST [2]	Search genomic dataset for matches on proteins and nucleotides
NAMD [15]	Simulation of large biomolecular systems (MPI)
Terraflow [23]	Computes flow routing and flow accumulation on massive grid-based terrains

Table 1: Applications submitted to JAWS.

quired a tedious and time consuming set up of physical machines with varying CPU, memory, network, and storage configurations. Moreover, the range of configurations was also limited by the available diversity and the number of physical resources. It took a month to learn and validate the performance models for the applications discussed in Section 4.3.

In this work, NIMO uses JAWS to collect the model training data. Since JAWS can configure virtual workspaces with strict limits on resource usage, it enables NIMO to request a range of resource configurations to collect sufficient training data for learning accurate models. Using JAWS, NIMO accomplished the model learning and validation within a day.

4.2 JAWS interaction with NIMO

To determine the resource specifications for a virtual workspace, JAWS queries NIMO for the application performance model. NIMO answers such queries by either returning the model, or suggesting a new virtual workspace specification within which the job must run. Figure 3 illustrates both alternatives:

- *Exploit model* - NIMO returns learned models to JAWS. JAWS uses the model to determine the virtual workspace specification within which to run the job. However, if NIMO chooses to return the model before an accurate model is ready, the model exploitation may result in an inefficient virtual workspace specification for the job.

- *Explore model* - In the absence of application performance model or if the model is still inaccurate, NIMO returns a virtual workspace specification to JAWS and collects proactively a new training data sample for learning the model. However, exploration may result in longer job runtimes and inefficient assignment of resources to the job.

Exploration versus exploitation is a classic dilemma in active machine learning. Exploration improves the performance model by gathering more training data points and may ultimately improve job runtimes and resource efficiency. However, excessive exploration provides diminishing returns at a cost: resources may be wasted to improve performance marginally. NIMO deals with the exploration versus exploitation dilemma in several ways:

- In addition to requesting virtual workspaces for a job submitted to JAWS by another client, NIMO itself may become a JAWS client, and proactively request multiple virtual workspaces with different resource profiles to run the application. This allows NIMO to collect quickly the training data samples to converge to an accurate model. To minimize the impact of these exploration runs on other clients, the job management policy may assign these requests low priority in the request queue.

- Since NIMO can monitor job execution, it can compare the model-predicted runtime to the observed runtime of a job and decide if its models are sufficiently accurate. If NIMO decides that its models need to be refined, it can re-enter the exploration phase to collect more sample points.

- The frequency of job submissions for a given application may help determine the right balance between exploration and exploitation; for instance, NIMO may decide to do extensive exploration for an application that is run thousands of times as opposed to one that is only run a few times.

4.3 Initial Results

Our initial results show that instantiating one (or more) VMs for a job is practical for common job lengths. Our prototype testbed consists of 72 IBM x335 rackmount servers with 2.8Ghz Intel Xeon processors and 1GB of memory each running Xen 3.0.4. VMs use a flash-cloned root filesystem from a Network Appliance FAS3020 filer connected via iSCSI. In total, it takes 12 seconds to flash-clone a root image and bring up (e.g., login via ssh) a new VM.

We use JAWS to execute streams of jobs drawn from a mix of three bio-medical and one GIS application. Table 1 describes the applications. We present some key results that demonstrate that : (a) informed provisioning of virtual workspaces can enable efficient use of resources in the resource control plane; and (b) using active learning NIMO learns accurate application models.

Figure 4 shows BLAST's runtime for different virtual workspaces, comprised of a single VM, with a different CPU and memory allotments. The figure shows that there exists a CPU and memory threshold that re-

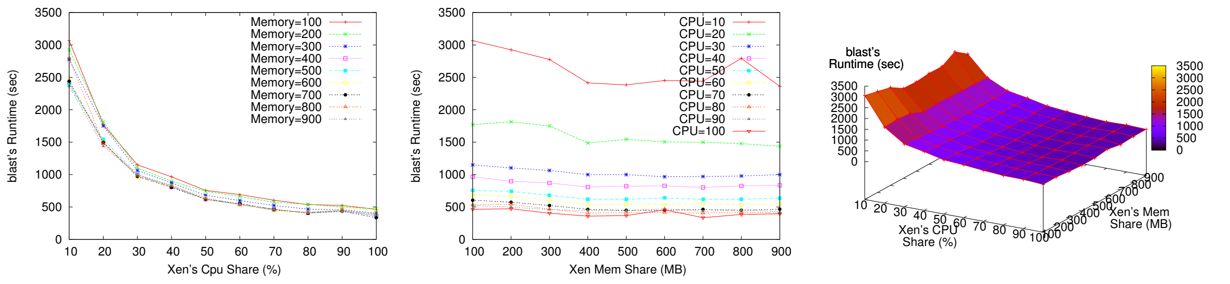

Figure 4: BLAST's runtime for different virtual workspace sizes. Allocating more than 300-400 MB of memory, and 60-70% CPU share does not improve BLAST's runtime. NIMO active learning automatically learns models that predict such knees, allowing efficient and effective assignment of resources.

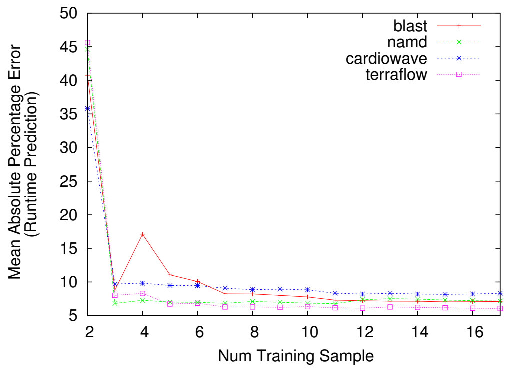

Figure 5: Accuracy of NIMO models. Using active learning NIMO quickly converges to accurate performance models with a small number of training samples.

sult in diminishing improvements to BLAST's completion time. The point of diminishing returns occurs at the knee of the curve: allocating resources past the knee results in resource waste since job runtime is unaffected. Prior knowledge of such knees enables multiple jobs to be collocated on the same physical host without affecting the runtime of either. The other applications from Table 1 show similar behavior. NIMO learns application models that can predict such knees.

Figure 5 shows the accuracy of the application models learned by NIMO. The models predict the runtime of an application as a function of CPU, and memory shares assigned to it. 10 CPU $\times$ 9 memory configurations comprise the total number of virtual workspaces. NIMO uses active learning to collect the relevant training data samples by running the application within a few virtual workspaces. The accuracy of the model is evaluated on all virtual workspaces that are not used for training. The details of active learning algorithms, and model accuracy evaluation methodology can be found in [21]. The figure shows that NIMO can learn a fairly accurate performance model with a few training sample runs. Of course, the number of samples will increase as the number of parameters that affect the job runtime increase. However, active learning systematically explores the parameter space, collects the relevant training samples, and enables convergence to accurate models quickly.

5 Related Work

Multiple groups have proposed using virtual machines for running jobs in the grid and other HPC settings [5, 7, 11, 20]. Most previous work with virtual machines in the grid has focused on the infrastructure required to support grid-type job execution environments [12, 19]. There has also been work aimed at enabling virtual workspaces to isolate grid applications from both other applications as well as custom execution environments [6, 11].

Some effort has been made to build clusters of virtual machines, but such work has largely been focused on virtualizing the network layer [22], for example, to enable transparent migration [10]. The resource control offered by VMs can be utilized by batch computing systems. For instance, collocation decisions may be made to increase the utilization of underutilized servers.

6 Conclusion

This paper proposes to factor resource control out of traditional batch scheduling services into a resource control layer called underware. Underware makes it possible to share a common pool of federated resources among multiple services. We present a new batch computing service, called JAWS, that uses the underware control plane to execute jobs within customized isolated virtual workspaces. We detail the advantages of separating resource management and job management and demonstrate using the control exposed by underware for active learning of application performance models in JAWS. Initial results show that the JAWS approach is promising: VM overheads are modest compared to common job lengths, applications have knees where increasing a resource allotment does not improve job runtime, and active learning on VMs can be fast and effective.

Future work includes study of end-to-end scheduling of jobs using NIMO's performance models to improve efficiency through model-guided collocation. This requires balancing the exploitation and exploration tradeoff. We are also investigating model-uses for "minimal impact" deadline scheduling as described in Section 4.

References

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

[2] S. Altschul, T. Madden, A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. Lipman. Gapped BLAST and PSI-BLAST: A New Generation of Protein Database Search Programs. *Nucleic Acids Research*, 25:3389–3402, 1997.

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

[4] V. Fedorov. *Theory of Optimal Experiments*. Academic Press, 1972.

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

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

[7] W. Huangy, J. Liuz, B. Abaliz, and D. K. Panda. A Case for High Performance Computing with Virtual Machines. In *Proceedings of the 20th ACM International Conference on Supercomputing (ICS)*, June 2006.

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

[9] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons, May 1991.

[10] X. Jiang and D. Xu. Violin: Virtual Internetworking on Overlay Infrastructure. In *Proceedings of the Third International Symposium on Parallel and Distributed Processing and Applications (ISPA)*, July 2003.

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

[12] I. Krsul, A. Ganguly, J. Zhang, J. A. B. Fortes, and R. J. Figueiredo. VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing. In *Proceedings of the Seventeenth Annual Supercomputing Conference (SC)*, November 2004.

[13] C. B. Lee, Y. Schwartzman, J. Hardy, and A. Snavely. Are user runtime estimates inherently inaccurate? In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, June 2004.

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

[15] J. C. Phillips, R. Braun, et al. Scalable Molecular Dynamics with NAMD. *Journal of Computational Chemistry*, 26:1781–1802, 2005.

[16] J. Pormann, J. Board, D. Rose, and C. Henriquez. Large-Scale Modeling of Cardiac Electrophysiology. In *Proceedings of Computers in Cardiology*, September 2002.

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

[18] E. Robinson and D. J. DeWitt. Turning Cluster Management into Data Management: A System Overview. In *Proceedings of the Third Biennial Conference on Innovative Data Systems Research (CIDR)*, January 2007.

[19] P. Ruth, X. Jiang, D. Xu, and S. Goasguen. Virtual Distributed Environments in a Shared Infrastructure. In *Computer*, May 2005.

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

[21] P. Shivam, S. Babu, and J. Chase. Active and Accelerated Learning of Cost Models for Optimizing Scientific Applications. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, September 2006.

[22] A. Sundararaj and P. Dinda. Towards Virtual Networks for Virtual Machine Grid Computing. In *Proceedings of the Third USENIX Virtual Machine Technology Symposium (VM)*, May 2004.

[23] L. Toma, R. Wickremesinghe, L. Arge, J. S. Chase, J. S. Vitter, P. N. Halpin, and D. Urban. Flow computation on massive grids. In *Proceedings of the ACM Symposium on Advances in Geographic Information Systems*, November 2001.