RC24991 (W1005-053) May 7, 2010
Computer Science

IBM Research Report

On the Design of a Deep Computing Service Cloud

Zon-Yin Shae, Hani Jamjoom, Huiming Qu, Mark Podlaseck,
Yaoping Ruan, Anshul Sheopuri
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

On the Design of a Deep Computing Service Cloud

Zon-Yin Shae, Hani Jamjoom, Huiming Qu, Mark Podlaseck, Yaoping Ruan, Anshul Sheopuri

IBM T.J. Watson Research Center
Hawthorne, NY 10590, USA

ABSTRACT

This paper presents a detailed view of a system called Deep Cloud, which enables the offering of High Performance Computing (HPC) capabilities as a service, and the design decisions and architectural considerations involved in building such a system. Our system straddles traditional Grid computing and emerging Infrastructure-as-a-service (IaaS) systems. Similar to Grid computing, our goal is to enable scalable scientific applications. However, we extend the key principles from IaaS to enable predictable resource allocation of HPC resources with a pay-as-you-go model. Our approach uses dynamic pricing to shape demand for HPC resources, while providing access to HPC resources for both interactive and batched scientific workloads. We also describe a novel approach to HPC resource catalog creation: the conversion of an HPC 3D topology into an easily calculable resource inventory that supports predictable reservation and scheduling. This enables a dynamic resource placement mechanism for reservation requests which we call Just-In-Time placement. Just-In-Time placement collectively performs resource placement to achieve optimal HPC resource utilization. Finally, we describe the need for a coherent, integrated user experience when interacting with the system.

1. INTRODUCTION

High performance computing (HPC) continues to play an important role in scientific investigations. Despite this role, the use of HPC has been limited to a relatively small number of research and commercial institutions. The reasons for its limited deployment are simple: the huge capital expense required setting up and maintaining the underlying cyber infrastructure and what would be considered by today's standards a suboptimal user experience. Meanwhile, cloud computing has emerged as a potentially disruptive paradigm that gives users easy access to large numbers of virtualized machines. While, in theory, these computing clusters could provide HPC capabilities, access to supercomputers in a cloud computing model is virtually nonexistent.

In this paper, we provide a detailed view of a system called Deep Cloud, where supercomputing capabilities can be offered as a service. We implement this architecture on top of a HPC system currently used by hundreds of researchers on a daily basis. The work outlined in this paper builds on research in grid computing and places it the context of a highly consumable services model. Grid computing, in theory, shares many aspects of cloud computing, but has not gained wide-scale adoption, primarily because it optimizes for system use, rather than user expectations. In contrast, cloud computing has focused on meeting user expectations of underlying system resources: offering users what they need, when they need it, and charging them for what they use.

We thus want to look at scientific applications in a new perspective. By enabling a cloud model, we hope to (1) create opportunities for new applications that can take advantage of the underlying computing capability, and (2) accelerate the adoption of scientific applications from the research community into commercial spaces. Our model expands on the two key cloud concepts in Infrastructure-as-a-service (IaaS): (1) on demand fulfillment and (2) incremental payment. In mainstream IaaS, on demand fulfillment enables users to dynamically provision a flexible number of computing capacities (typically expressed in VM instances or elastic storage) with the fulfillment completing in minutes. The second concept is the pay-as-you-go model, charging users with resources they use, where usage might be defined in terms of powered-up VMs. These two concepts (dynamic pricing and incremental payment) enable new and innovative system architectures.

In giving an overview of the key design decisions and architectural considerations in building an HPC cloud, we relate our architecture to existing cloud architectures across four dimensions: (1) management API, (2) user experience, (3) resource pricing, and (4) resource management. In this paper, Section 2 describes the architecture of Deep Cloud. Section 3 describes the management API that encapsulates the underlying HPC resources. Section 4 describes HPC 3D topology mapping into resource catalog for reservation. Section 5 describes the role of HPC pricing for demand shaping. Section 6 then describes Just-In-Time resources placement. In Section 7, we describe key aspects of improving user experience. We end the paper describing related work in Section 8.

2. SYSTEM ARCHITECURE

The Deep Cloud system architecture, shown in Figure 1, follows Service Oriented Architecture (SOA) principles. The design goal is to streamline complex job scheduling to support various application workloads. All system components are loosely coupled with well-defined APIs, enabling third party components to plug in as SOA services.

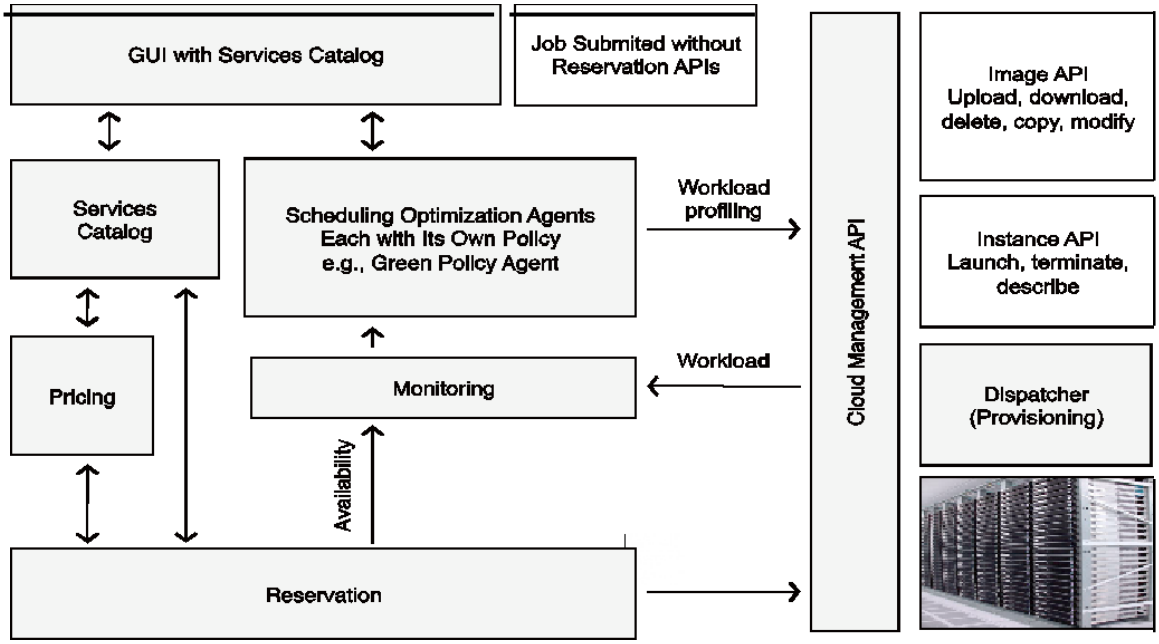

Figure 1: System architecture

The reservation is the key mechanism for resource management. Reservations allows users (or scheduling agents) to reserve some or all of the underlying system resources for a specified duration. Users interact with the reservation component, making reservations for interactive sessions, for example, through the Web-based service catalog. As with other HPC systems, users also expect to batch their jobs without an explicit reservation. Unlike traditional implementations, our system supports such batching through scheduling agents. These agents allow different scheduling policies to coexist, each optimizing for a different user objective, but all leveraging the underlying reservation and pricing components. For example, a green batch agent can schedule jobs to run only during night hours to help reduce peak power consumption. Another agent can optimize for resource utilization and will schedule the jobs whenever it sees there are unused resources available.

The main challenge in enabling agent-based scheduling is to ensure proper coordination among all agents for scheduling jobs. Our architecture addresses this challenge by the centralized resource reservation mechanism. Specifically, all the agents need to schedule the jobs through the reservation component. This coordinates all agents and synchronizes the explicit user reservations coming from the Web-based service catalog. We call this type of job scheduling macro-scheduling. In contrast to traditional micro-schedulers, which are closely coupled to the physical resource, macro-schedulers rely on the reservation mechanism and are independent of physical resources. Consequently, they can be implemented as independent and scalable SOA components.

The dispatcher component implements the APIs for resource provisioning services. Similar to IaaS cloud APIs, they include three types of calls: (1) image management, (2) instance management, and (3) availability management. The image management APIs are used to register and manage the application images as well as the data required by the application. The instance management APIs are used to launch the application. Finally, the availability management APIs are used for querying resource availability. Note that, image management API are also used to support the pre-loading of the application image and data. This is especially important in the HPC applications, which can consume large amounts of data (easily reaching the peta-byte range). Our system pre-loads application data and ensure its availability before the scheduled reservation start time.

3. DEEP CLOUD APIs

We provide a set of Deep Cloud APIs to manage HPC's underlying resources. Our API is consistent with the cloud API in the market [1]. Deep cloud APIs abstract HPC's native execution commands and data interfaces. This abstraction enables complete control over the system (both compute and storage resources) and frees other Deep Cloud components (e.g., the reservation subsystem) from dealing with low-level system commands. By having APIs compatible with industry standards [3], we also expect easy portability of commercial management tools to HPC clouds.

Deep Cloud API	Function
Reservation Tools	Create, list, describe, destroy, change
Instance Tools	Describe, launch/terminate (interactive mode)
Availability Tools	Get available CPUs
Security Tools	Front-node access control
Collaboration Tools	Get user name of a resource at a given time for change reservation negotiation

Table 1: Deep Cloud API

Table 1 illustrates the Deep Cloud API tools. Reservation APIs provide users the resource reservation. In addition to make or cancel the reservation commands, our system supports users to change their reservations. It is well known that HPC applications run time are very difficult to predict. It is therefore very often that at the end of the user's reservation time slot, the running application is still not completed yet. In this situation, the change reservation API becomes very useful to extend the time slot. However, in a high percentage of resource utilization system, there is a very good chance that the next time slot needed for the current application execution extension had already been reserved by other person. As a result, the change reservation will only become possible when the other person who reserved the targeted time slot agreed to change reservation as well. The change reservation actions will ripple out centered from a single change reservation. Change reservation can be more successful if user can know whom to negotiate with the change reservation. The Collaboration API provides the user name of a resource at a given time. Instances tools are used to launch, terminate, and monitor the application instances. Availability tools provide the

current resource inventory information. In a HPC environment, usually, system provides a console for users to login to run the application. Deep Cloud Security tools prevent users from accessing the HPC resources without reservation first. Comparing our image management APIs to cloud images refer to virtual machines (VMs) where Deep Cloud images refer to compiled applications. Thus, applications act as 'appliances' that are managed by the various subsystems in Deep Cloud. In the first phase, we have only implemented the APIs that are critical to other Deep Cloud subsystems. However, we expect that other services will also be implemented in the future.

4. HPC RESOURCE MANAGEMENT

In general, HPC has a very powerful, yet complex, 3D resource topology which can make resource management extremely difficult. We describe, in this section, an innovative resource mapping methodology to enable the comprehensive resource management of the deep computing capability. In particular, this mapping methodology abstracts the location of the resource from users and applications. It also provides the required flexibility and scalability for integrating pricing and cataloging in an HPC cloud.

There are a number of challenges in creating a resource catalog that captures availability of HPC resources. They arise from HPC's large number of configuration possibilities, enabled by its three-dimensional connectivity fabric. There are two views to this resource catalog: users and resource management. From a user's point of view, Deep Cloud need to provide the ability to reserve HPC resources across various possible configurations. For example, a user can request 32-nodes for 2 days, where another can request 2 racks for 3 hours. From a resource management point of view, this translates to the need to track all possible configurations over an infinite time horizon, with the goal of optimizing resource allocation to maximize system utilization. It is ultimately a packing problem, constrained by the underlying system configuration and wiring.

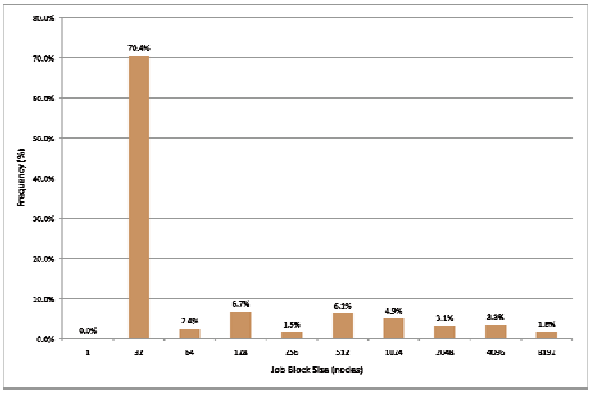

Figure 2: Histogram of job size

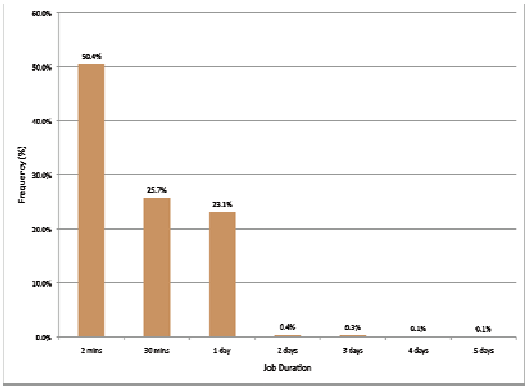

Figure 3: Histogram of job duration

To design a catalog that meets both criteria, we analyzed existing HPC workloads across two dimensions: job size and job duration. Figures 2 and 3 show a histogram for job size and duration, respectively, covering a three month period. We also looked at other configuration parameters, including connectivity, storage needs, etc. These distributions all point to two distinct user communities: (1) development jobs, requiring a relatively small number of nodes; (2) production jobs, requiring a very large percentage of system resources. Based on the above, Deep Cloud maintains a simplified resource catalog that can fulfill most (if not all) of expected user configurations.

5. BALANCING WORKLOAD OF RESERVATION SYSTEM WITH PRICING

In this section, we discuss the relationship between the reservation subsystem and dynamic pricing engine. As discussed, the reservation subsystem allows users to reserve system resources for a fixed time duration. Optimization schemes are applied to minimize the system fragmentation (by the vertical reordering of boxes) as long as the reservations have not started (e.g., switching 8 and 3 in Figure 4). On the time horizon, we rely on the dynamic pricing engine to shape demands, e.g., to provide incentives for users that reserved mid-day hours (box 0 in Figure 4) to shift

to mid-night hours. In order to discourage users from reserving more resources than they need, a mechanism is required to provide disincentives. Pricing serves as an effective mechanism to ensure that users that have the willingness and ability to pay to use resources only when they require them. Furthermore, by pricing resources differently during different blocks of time (for example, peak and off-peak times), we can balance the workloads better, thereby improving customer satisfaction metrics, such as the number of fulfilled requests.

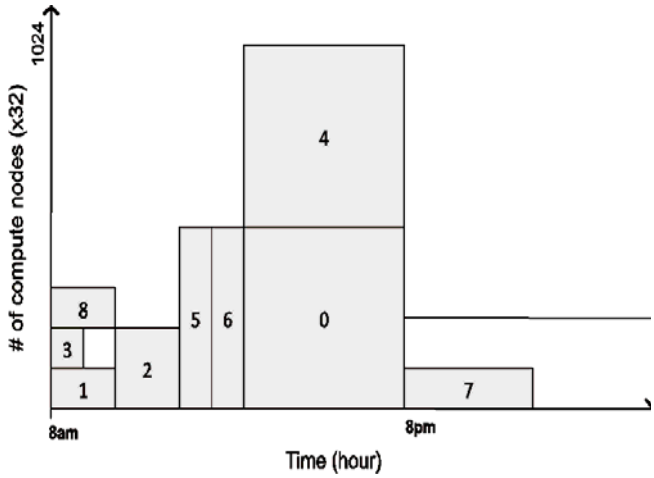

Figure 4. Illustrative example of workload balancing

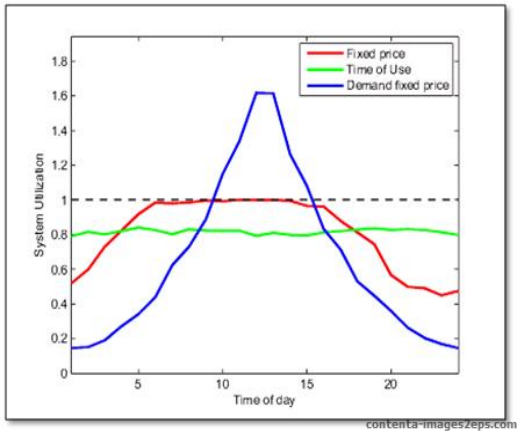

Figure 5. Workload

In our system, we implement a token-based scheme that works as follows: Computing resources may be purchased in exchange for tokens. For a given number of user requests, the scheme's objective is to maximize system performance by (dynamically) adjusting resource prices in tokens per unit of time. We consider two kinds of pricing schemes: pre-paid and post-paid. In the prepaid scheme, each user is allocated a limited number of tokens (in lieu for dollars at the beginning of each period) which can be exchanged for computing time. In the post paid-scheme, users may purchase tokens in real-time (using dollars) to reserve computing resources; however, a cap limits the amount of computing resources that can be purchased by a user. Our pricing strategies "smooth demand" over the horizon and improve customer satisfaction with respect to static pricing strategies.

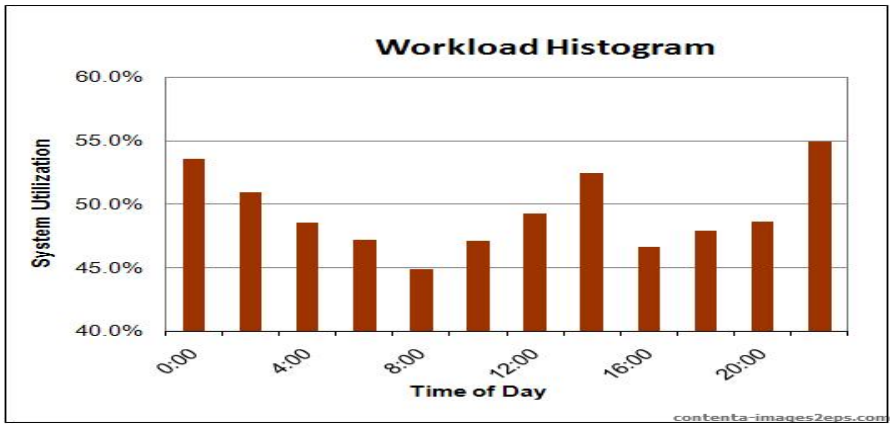

Figure 6: Utilization

To perform a preliminary evaluation of the proposed pricing strategy, we obtained a four-month historical system log from the HPC. The system log contains 22,801 fulfilled resource requests. Requests are characterized by start time, duration, the number of compute nodes requested, as well as the time when the request is made. Figure 5 shows the histogram of the average workload over a 24-hour period. The workload does not fluctuate by a large margin, but

we do observe periodic highs (e.g., midnights) and lows (e.g., early mornings) which indicate that time of use pricing will help flatten this user resource demand over time. Although we have access to the real system log, we cannot use it directly for simulation because only accepted requests are logged (some requests may be rejected due to system capacity). To simulate the real demand and demonstrate the effect of time of use, we generate a trace according to the distribution (of request size and duration) of the system log. Figure 6 shows a preliminary simulation result averaged over 200 days. The blue line depicts the demand with fixed price which exceeds the system capacity by over 50% around noon. The red line depicts the system utilization with fixed price within a day. Notice that some of the demands shift to earlier or later time of the day because the system is full in the middle of the day. Finally, the green line depicts the system utilization with time of use pricing. As expected, time of use pricing flattens demand and provides more predictable system utilization. In this way, user satisfaction enhanced by a reduced number of dropped requests. In addition, we anticipate more opportunities for better capacity management, especially for supercomputers with special hardware wiring constraints. While we are working actively to generate more experiments with simulation data, we are planning for a real-world deployment, as illustrated in the next section.

6. JUST-IN-TIME PLACEMENT

A reservation request is a 4-tuple consisting of (1) start time, (2) duration, (3) number of compute nodes, and (4) connectivity. Using our 2-dimensional resource model, a reservation request can be translated into a rectangle whose edges are duration and number of computer nodes. Therefore, to allocate the requested resource at a particular time, we must insert this rectangle into a 2-dimensional resource map. Typically, each of the request boxes has a fixed start time which is specified by the user. The assignment of particular computing nodes to perform a requested task is optimized by our reservation subsystem, so as to achieve maximum utilization of the underlying hardware. For example, in Figure 4, when a new request, box #8, arrives, our reservation system will attempt to stack it on top of the current reservations, box #1 and #3.

In order to achieve optimal resource placement at a particular time, we use a global optimization mechanism which we call “Just-In-Time” resource placement. Specifically, we aggregate a group of reservation requirements for placement optimization instead of making an individual resource placement when a user makes a reservation. Just-In-Time placement is particularly important to achieve effective scheduling when requests are mixed with both interactive and batch modes. Requests for interactive sessions usually specify a strict time requirement so that users can log into the allocated computing resources from a console and perform interactive operations, such as programming and debugging. This would cause fragmentation of the computing resources and will result in sub-optimal resource utilization.

One of the challenges of the Just-In-Time placement mechanism is to provide a resource availability indication at the time a reservation request is submitted (since the actual placement will be performed at a later stage). A system with a moderate to somewhat high request load can easily run into conflict and result in unsatisfied requests. This is particularly common in the HPC environment where resources are limited with highly dependent 3-dimensional topology. Using the space and time packing provisioning mechanism described in Section 4, we degenerate the complex 3D topology into a simple 1D resource mapping with the number of computing nodes associated at a particular time. The resource catalog resulting from this resource packing mechanism is a resource inventory that can be easily tracked. Thus, the availability of a reservation is predictable, even though the placement is performed in the future.

7. USER EXPERIENCE

The Deep Cloud user interface is a web-based client for activities related to the development and execution of HPC applications, such as making reservations and submitting jobs.

As discussed in Section 2, reservations are a key component of system resource management. Consequently, a primary focus of the Deep Cloud user interface is the reservation catalog. Figure 7 depicts the reservation catalog for interactive sessions. One design goal of this interface is to display a relatively large number of calendar days and existing reservations on a single screen. We accomplish this by representing all days (except one; more on this one below) as stacked horizontal timelines, one timeline for each 24- hour period. Each day is segmented into 48 30-minute
cells. Cells corresponding to confirmed reservations are highlighted, so as to enable a user to see all upcoming reservations in the months ahead. When a 24-hour timeline is clicked, it expands to display the entire range of resource

configurations (from 32 nodes to 16k nodes) available for that day (Monday 14 December in the figure). In the this expanded view of a day, variable pricing for various times is indicated by color: dark blue cells are more expensive, while lighter blue cells are less expensive. (Unavailable cells are disabled and displayed as dark gray.) To make a reservation, a user drags a small floating lens across the expanded grid and configures its length to match the duration of the desired session. The lens responds to its location by displaying the price for the interval indicated. If more than one thirty-minute cell is included in the specified duration, the price for each component cell is itemized. Discounts, if any, are displayed, explained, and applied to the total.

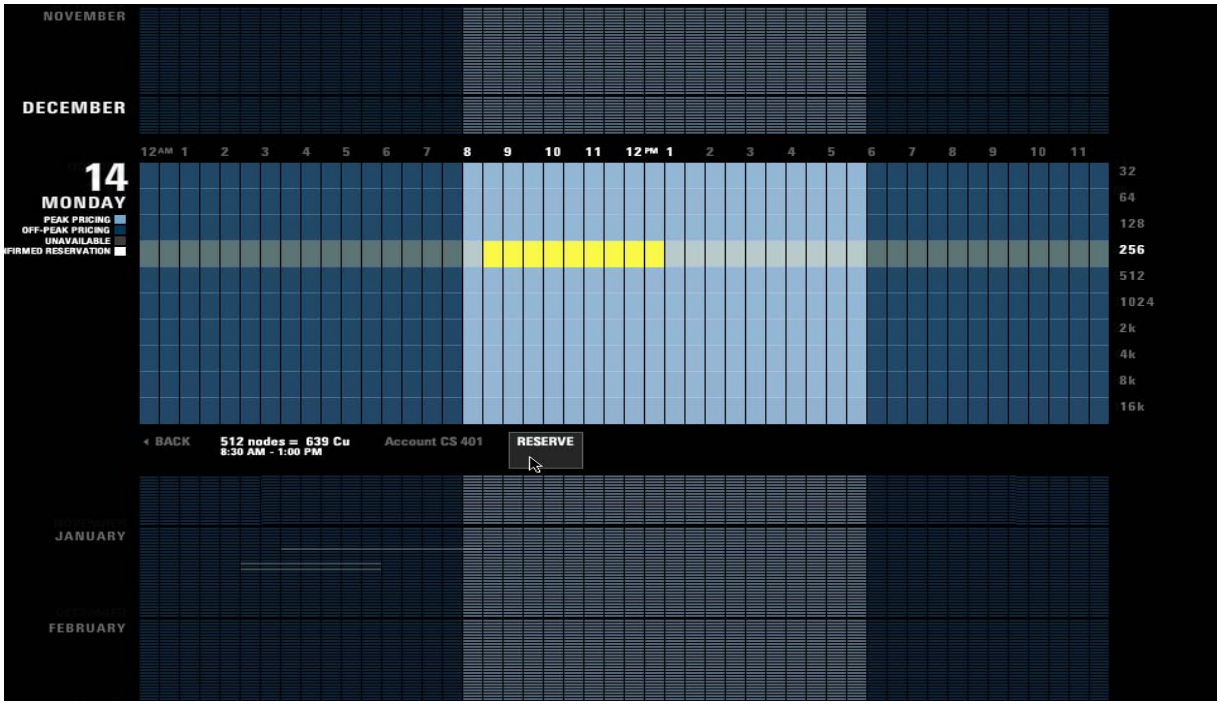

Figure 7: Illustrative example of user experience

8. RELATED WORK

Our paper investigates scalable deep computing system architecture that employs a combination of cloud computing and SOA mechanism for system-wide integration and expansion. Our work relates to a number of research areas, including Grid Computing, Cloud Computing, scheduling, distributed management, dynamic pricing, I/O optimization. Given the breadth of related work, we focus here on areas that we extended, not directly leveraged. That said, in this section we primarily focus on contrasting our work with Grid Computing and Cloud Computing, noting that other related references have been sited throughout the paper.

Compared to Cloud Computing, our approach is based on a different assumption for resource availability. Cloud Computing assumes that resources can be unlimited. This view was recently reiterated by Michael Armbrust et al. [4], where the they state because of the illusion of infinite computing resources that are available on demand, users no longer need to plan ahead for provisioning. This is a sharp contrast to our work, where when analyzing production demand, we found that HPC resources continue to be constrained relative to potential demand.

There is a large body of work looking at Grid Computing. A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities. We compare our approach to Globus and TeraGrid as two popular grid implementations. Ian Foster started the Globus Toolkit [5] since the late 1990s to support the development of service oriented distributed computing applications and infrastructures. Core Globus Toolkit components address fundamental issues relating to security, resource access, resource management, data movement, resource discovery. Recently, it grows into the TeraGrid [2] to additionally support wide-area parallel file system, workflow, and scientific visualization support.

That said, there are similarity between our work and Grid Computing. In particular, our design choices have been motivated by earlier work showing the benefits of resource planning and relative homogeneousness of underlying resources. For example, Lingyun Yang et al. [11] observed that when resources are very heterogeneous, a resource

with larger capacity will show higher variance In performance and, therefore, will more strongly influence the execution time of an application than a machine with less variance. Their work tries to predict resource capacity over future time intervals of running applications to determine the proper resource mapping. Matthias Hovestadt et al. [6] presented planning systems for the present and future demand by assigning a start time to all requests. They show that advanced reservation eases the process of negotiating the resource usage between the system and agents. Warren Smithy et. al [10] used runtime prediction techniques to predict application wait times and to improve the performance of least-work-first and backfill scheduling algorithms. David O'Hallaron et al. [9] studied a new scheduling concept for the TeraGrid, suggesting the use of a market-based auction mechanism to create a uniformly fair environment, whereby all users have the same opportunity to reveal which jobs are actually time-critical.

A number of studies looked at the expected impact of reservation and scheduling on underlying system utilization. Elie Krevat et al. [8] has studies scheduling on HPC's cellular architectures with 3D interconnects. They show that restrictions for job partitions to be both rectangular and contiguous introduce fragmentation issues that affect the utilization of the system and wait time of the jobs. Additionally, their results show that FCFS with backfilling provide better utilization. Also, James Patton Jones et al. [7] investigated the operation data of various HPC centers including the Intel iPSC/860, Intel Paragon, Thinking Machines CM- 5, and Cray Origin 2000. They discovered that the utilization of these machines shows three general trends: (1) scheduling using a naive FCFS first-fit policy results in 40-60% utilization, (2) switching to the more sophisticated dynamic backfilling scheduling algorithm improves utilization by about 15 percentage points (yielding about 70% utilization), and (3) reducing the maximum allowable job size further increases utilization.

9. CONCLUSIONS & CHALLENGES

We have given an overview of the key design decisions and architectural considerations in building a cloud that provides High Performance Computing (HPC) to a large population of users as a service. We have implemented this architecture on top of a HPC system. We then converted a 3D topology into an easily calculable resource catalog on which a scalable just-in-time resource placement scheduling mechanism is implemented – providing guaranteed resource availability and efficient resource utilization. As a result, we extended key principles from IaaS to enable predictable resource allocation of HPC resources with a pay-as-you-go model, while providing access to HPC resources for interactive and batched scientific workloads. We also described how access to this system looks from an end-user's perspective. An interesting challenge is how to offer HPC computing as a platform-as-a-service (PaaS) to create new services for the science community. These HPC services have requirements that are distinct from those of commercially available cloud services. For example, to offer data-as-a-service, HPC applications usually process very large data sets. The data can be very sensitive (e.g., oil seismic data) and may need to be hosted on a restricted area in a specific country. The communication latency for HPC computing nodes to access the application data might dramatically degrade the HPC performance. Another example is to offer higher-level computing applications (like seismic processing) as a service. One possible solution might establish an economically viable ecosystem for the HPC computing cloud where data is shared and HPC algorithms are reused. We are currently researching these topics.

REFERENCES

[1] Amazon elastic compute cloud (amazon ec2).

[2] http://www.teragrid.org/

[3] Open cloud manifesto.

[4] M. Armbrust, A. Fox, R. Grifth, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. In Berkeley Reliable Adaptive Distributed Systems Laboratory, 2009.

[5] I. Foster. Globus toolkit version 4: Software for service-oriented systems. In IFIP International Conference on Network and Parallel Computing, pages 2–13. Springer-Verlag LNCS, 2006.

[6] M. Hovestadt, O. Kao, A. Keller, and A. Streit.Scheduling in hpc resource management systems: Queuing vs. planning. In JSSPP Springer-Verlag Berlin Heidelberg, pages 1–20, 2003.

[7] J. P. Jones and B. Nitzberg. Scheduling for parallel supercomputing: A historical perspective of achievable utilization. In Job Scheduling Strategies for Parallel Processing, 1999.

[8] E. Krevat, J. G. Castanos, and J. E. Moreira. Job scheduling for the bluegene/l system. In Lecture Notes in Computer Science.

[9] D. O'Hallaron, V. Conitzer, and A. Scheller-Wolf. Auction-based scheduling for the teragrid. In DARPA/NSF proposal, 2007.

[10] W. Smithy, V. Taylory, and I. Foster. Using run-time predictions to estimate queue wait times and improve scheduler performance. In Springer-Verlag, 1999.

[11] L. Yang, J. Schopf, and I. Foster. Conservative scheduling: Using predicted variance to improve scheduling decisions in dynamic environments. In Supercomputing, 2003.