2011 IEEE International Conference on Cluster Computing

Heterogeneous cloud computing

Steve Crago*, Kyle Dunn, Patrick Eads, Lorin Hochstein†, Dong-In Kang,
Mikyung Kang, Devendra Modium, Karandeep Singh, Jinwoo Suh, John Paul Walters
University of Southern California / Information Sciences Institute
3811 N. Fairfax, Suite 200. Arlington, VA, 22203
{*crago**, *lorin*†}*@isi.edu*

***Abstract*—Current cloud computing infrastructure typically assumes a homogeneous collection of commodity hardware, with details about hardware variation intentionally hidden from users. In this paper, we present our approach for extending the traditional notions of cloud computing to provide a cloud-based access model to clusters that contain a heterogeneous architectures and accelerators. We describe our ongoing work extending the OpenStack cloud computing stack to support heterogeneous architectures and accelerators, and our experiences running OpenStack on our local heterogeneous cluster testbed.**

***Keywords*-cloud computing; accelerators; high-performance computing;**

I. Introduction

Cloud computing has become increasingly prevalent, providing end-users with temporary access to scalable computational resources. At a conceptual level, cloud computing should be a good fit for technical computing users, such as scientists who often need to run computational intensive jobs as part of their work. Indeed, scientists are already beginning to take advantage of cloud computing resources to execute scientific workflows [1], [2], [3].

However, since the current cloud computing market evolved from the IT community, it is often not a good match for the needs of technical computing end-users from the high-performance computing (HPC) community. Providers such as Amazon and Rackspace provide users with access to a homogeneous set of commodity hardware, with details of the hardware obscured through virtualization technology and little or no control of locality (except sometimes by geographic region). By contrast, technical computing end-users may want to obtain access a heterogeneous set of resources, such as different accelerators, machine architectures, and network interconnects.

Nevertheless, while most clouds in their current form are not a great match for traditional HPC users, the cloud computing model of providing access to computational resources provides advantages over the traditional batch-scheduled model typically used at HPC centers. In this paper, we describe how traditional cloud computing infrastructure can be extended to support technical computing users by providing access to a heterogeneous set of computing resources, and discuss our ongoing work in extending the OpenStack Compute [4] cloud computing framework to support heterogeneous clouds.

II. Why a heterogeneous cloud?

Data and computational centers are often limited by power density and efficiency, as well as compute density. While general-purpose microprocessor and server manufacturers are working to improve power efficiency, heterogeneous processing resources can provide an order of magnitude or more improvement using these metrics. These improvements are likely to be persistent, since specialized devices can be optimized for specific kinds of computations, and this optimization can be performed for efficiency. There are numerous examples for problems well suited to specific architectures. Examples of such architectures include digital signal processors, network packet processors, graphical processing units (GPUs, also known as GPGPUs, general-purpose GPUs, in this context), symmetrical multiprocessors (SMPs), and conventional CPUs.

Today's cloud infrastucture, with a few notable exceptions (e.g. SGI Cyclone, R Systems, Amazon Cluster GPUs), generally focuses on commodity hardware, with no control over target architectures aside from choosing from a fixed number of memory/CPU sizes. If cloud users are to be able to take advantage of the performance and efficiency advantages of heterogeneous computing, the cloud infrastructure software must recognize and handle this heterogeneity.

In the past, grid computing and batch scheduling have both been commonly used for large scale computation. Cloud computing presents a different resource allocation paradigm than either grids or batch schedulers. In particular, Amazon EC2 [5] is equipped to handle many smaller compute resource allocations, rather than a few, large requests as is normally the case with grid computing. The introduction of heterogeneity allows clouds to be competitive with traditional distributed computing systems, which often consist of various types of architectures as well. When combined with economies of scale, dynamic provisioning and comparatively lower capital expenditures, the benefits of heterogeneous clouds are numerous.

Cloud computing allows individual users to have administrative access to a dedicated virtual machine instance. The capability to separate users is superior compared to a batch

978-0-7695-4516-5/11 $26.00 © 2011 IEEE
DOI 10.1109/CLUSTER.2011.49

scheduling approach, where it is common for multiple jobs to share a single operating system. The advantages of this are apparent from the perspectives of security as well as flexibility for users, offering a variety of operating systems. Some batch-scheduling implementations, such as LSF [6], depend on nearly identical configuration of compute nodes across a cluster, a potentially daunting task for system administrators.

III. OpenStack

To support heterogeneity in a cloud computing environment, we chose to extend the OpenStack software stack. *OpenStack* refers to a collection of open-source software packages designed for building public and private clouds. In this paper, we focus on *OpenStack Compute* (also referred to as *Nova*), which provides an infrastructure-as-a-service [7] model of cloud computing access[1]. OpenStack provides similar functionality to other open-source cloud projects such as Eucalyptus [8], OpenNebula [9], and Nimbus [10]. Two notable OpenStack deployments in scientific organizations are NASA's Nebula cloud [11] and the U.S. Department of Energy's Magellan cloud [12].

OpenStack is implemented as a set of Python services that communicate with each other via message queue and database. Figure 1 shows a conceptual overview of the OpenStack architecture, with the OpenStack Compute components bolded and OpenStack Glance components (which stores the virtual machine images) shown in a lighter color.

We chose to extend OpenStack because of its modularity and scalability, as well as its Apache 2.0 based license model and open governance.

The *nova-api* service is responsible for fielding resource requests from users. Currently, OpenStack implements two APIs: the Amazon Elastic Compute Cloud (EC2) API [5], as well as its own (OpenStack) API.

The *nova-schedule* service is responsible for scheduling compute resource requests on the available compute nodes.

The *nova-compute* service is responsible for starting and stopping virtual machine (VM) instances on a compute node.

The *nova-network* service is responsible for managing IP addresses and virtual LANs for the VM instances.

The *nova-volume* service is responsible for managing network drives that can be mounted to running VM instances.

The *queue* is a message queue implemented on top of RabbitMQ [13] which is used to implement remote procedure calls as a communication mechanism among the services.

The *database* is a traditional relational database such as MySQL that is used to store persistent data that is shared across the components.

The *dashboard* implements a web-based user interface.

[1]Subsequently, we will use the term *OpenStack* to refer to *OpenStack Compute*.

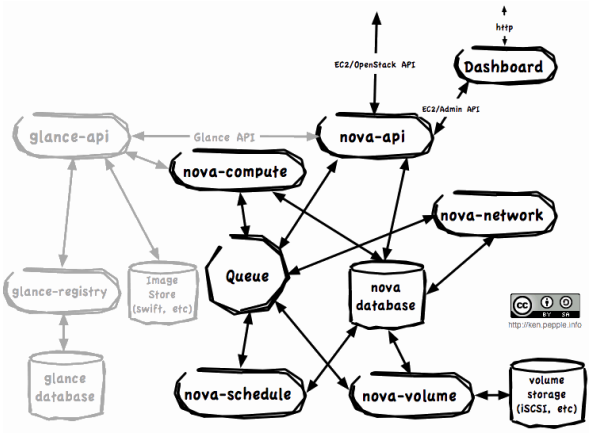

Figure 1. OpenStack Architecture. Credit: Ken Pepple [14]

IV. Supporting heterogeneity

In a traditional cloud computing service such as Amazon's EC2, the user has a limited range of options when requesting virtual machine instances, namely:

- Amount of memory
- Number of cores
- Amount of local storage
- 32-bit (x86) or 64-bit (x86_64) platform

Amazon offers a set of instance types with different amount of resources at different price points. For example, the user might request a "Standard Large" or *m1.large* instance, which has 7.5GB of RAM, 2 virtual cores, 850 GB of RAM, on a 64-bit platform.

To support a wider range of heterogeneity, a user must be able to request an instance with a configuration of computing resources that meets the user's needs. The user can choose it through the user interface by using predefined instance types or an extra string appended to an instance type. The user information is relayed to scheduler that chooses the computing resources that meets the user's request. In our current implementation, if the user's architectural requirements cannot be satisfied, then no resources will be provisioned.

A. Choosing architecture and other configurations

A simple way of choosing an instance with a configuration of computing resources including architecture choice is using predefined instance types. This is a typical way in other cloud computing interfaces. For example, as implemented in our software stack, if *cg1.small* instance type maps to a configuration that includes one virtual CPU and one GPU, then by using *cg1.small*, a user can choose the predefined configuration.

Another way of specifying the user's choice is by appending an extra string to an instance type. This enables a user to choose a configuration that cannot easily done

through predefined instance types. Note that heterogeneity exists not only among different processor architectures but also among the same processor architectures having different features. For example, the x86 architecture has added new features as its micro-architecture has evolved. Intel Nehalem micro-architecture has added SSE4.2 instruction sets. The recent x86 micro-architecture Westmere has added Carry-less Multiplication (CLMUL) instruction to improve some encryption applications. Making use of those features is critical to some high performance applications.

These features are very diverse, and it is not easy to capture all combinations of these using predefined instance types. To express these features in a flexible way, we provide another way of specifying desired features: using an extra string in the instance type string. For example, *cg1.small; cpu_info=SSE4.2* means that the requested instance type is cg1.small and CPU has SSE4.2 feature. By using this approach, we maintain compatibility with the existing EC2 API implemented by OpenStack, which allows us to use the command-line euca2ools [15] tools unmodified.

A user can specify a CPU architecture that is different from x86. For example, TILEPro64 architecture can be chosen using *t1.small; cpu_arch= TILEPro64*.

The syntax of the extended image type string is:
$image\_type[;keyname = keyvalue[,keyvalue]]$

A key may have one or more values. If there are multiple values for a key, all of them must be satisfied. There may be zero or more $keyname = keyvalue$ pairs. Again all of those keys must match. Some keys we use are:

- xpus: number of accelerators (*e.g.*, GPU) to use
- xpu_arch: architecture of the accelerator (*e.g.*, Fermi)
- cpu_arch: architecture of CPU (*e.g.*, X86_64, TILEPro64)
- cpu_features: required features of CPU (*e.g.*, SSE4.2)
- hypervisor_type: hypervisor of the VM (*e.g.*, Xen, KVM, LXC)

We plan to extend the syntax of the instance type string to express more general heterogeneous architectures and features: expressing OR operations of constraints (*e.g.*, cpu_arch=x86_64 OR TILEPro64) and making the $keyvalue$ specify a range of values.

B. Implementation

The user choice is conveyed to the API server (nova-api), which retrieves detailed information from a database table (instance_type) such as cpus, network, memory, and disk requirements. Additional information such as accelerator and number of accelerator is retrieved from another table (instance_type_extra_spec). The two tables are separated such that the instance_type table covers all common case fields such as the number of CPUs. Another table (instance_type_extra_spec) is used to store data that may appear on some instances.

When the scheduler gets the information, the scheduler compares the requested information to the available resource information. The available resource information is available in the zone manager, which stores the latest information that is periodically sent by compute nodes. Each requested data element is compared to find a compute node that has all the features that the user requested. In particular, the architecture information is compared to find the right architecture. Additional information, such as the accelerator type and the number of resources are compared to choose a compute node that meets the request. When a compute node is found, the compute node is selected and instantiation of the virtual machine starts.

On the compute node, the compute service needs to collect the capability of the node, which is sent to zone manager. For this, the libvirt library is used on x86 machines. Libvirt internally keeps the exact feature list of each CPU, but it returns only the commonly found features as the capabilities. The underlying philosophy of this design is to maximize the possibility of migration of virtual machines. However, to provide high performance computing in the cloud, exposing the detailed capability of the host machine to the cloud manager is necessary so that user can choose not only CPU architecture but also specific features.

To expose the detailed capabilities of the host machine, we disabled the part of libvirt library that masks the original capabilities to return common capabilities. This affects only the capabilities of the host. The capabilities of the guests are not affected. We patched libvirt such that it adds "-cpu host" flag to the "qemu-kvm" command such that the virtual machine spawned by qeum-kvm can use all the features of the host CPU.

V. Supporting accelerators

The challenge to using accelerators in the cloud is that they were never intended to support virtualization. In this section, we describe our GPU-based accelerator support and the challenges faced in accessing accelerators within a virtualized high performance environment.

A. Virtualized Accelerators

Accelerator support in general, and GPU support in particular, are in their infancy within the virtualization community. The primary challenge is that current GPUs do not support virtualization. That is, they cannot be shared by multiple virtual machines and must instead be dedicated to a specific virtual machine.

Strategies such as PCI-passthrough are effective for many low-end GPUs and network cards, but few high performance GPUs are compatible and support varies widely depending on the virtualization strategy in use. Xen [16] and Parallels Desktop [17], for example, support NVIDIA GPUs that explicitly support hardware SLI Multi-OS. KVM [18], on the other hand, currently has no GPU support.

gVirtuS [19] and vCUDA [20] have been proposed as alternatives to PCI-passthrough. Rather than passing the physical hardware through to the virtual machine, both vCUDA and gVirtuS rely on a split driver model with an interposing software API used within the virtual machine, and physical GPU binding occurring on the host system. The interposing layer intercepts calls into the CUDA environment, marshals the data, and then forwards the CUDA library calls to the host.

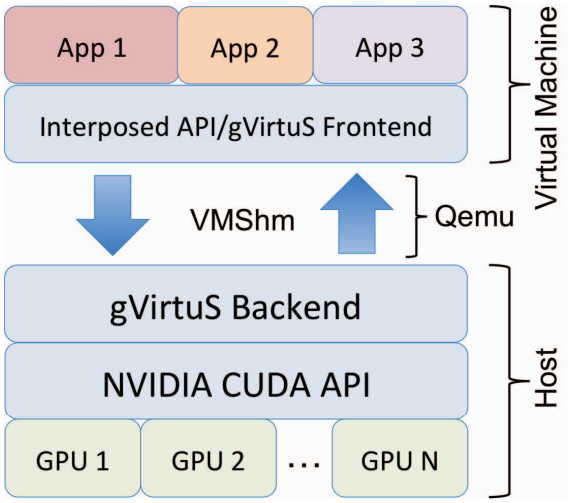

Figure 2. Overview of gVirtuS.

B. Overview of gVirtuS

We selected gVirtuS for integration into OpenStack due to its open source availability and its compatibility with KVM. gVirtuS implements a split frontend/backend model where the frontend resides within the virtual machine, and the backend resides on the host system and executes CUDA library calls on behalf of the virtual machine (see Figure 2).

The gVirtuS frontend serves two primary functions. First, it provides a CUDA API and the shared library libcudart.so. This allows programs to be compiled against gVirtuS, and also allows pre-built binaries to run unmodified using the shared library. Second, the frontend provides data marshalling for the CUDA library calls, which allows gVirtuS backend to accurately reproduce the intended CUDA call.

The gVirtuS backend, residing on the host, provides the frontend with the necessary remote execution support. The backend receives CUDA library requests and their associated data and executes them on the physical hardware as shown in Figure 2. Unlike the frontend, which provides a mock libcudart.so shared library, the gVirtuS backend is built against the NVIDIA-supplied libcudart.so shared library, allowing the backend to interface directly with the physical GPU hardware.

gVirtuS provides several user-selectable *Communicators* that allow the frontend and backends to communicate. We selected the *VMShm* communicator, which uses a shared memory segment exposed through QEMU [21]. VMShm exposes a device named */dev/vmshm0* within the virtual machine and a POSIX shared memory segment on the host. Both the host and virtual machine *mmap()* this shared memory segment and use it for all CUDA communication.

C. GPU Support and Implementation in OpenStack

Intuitively, adding gVirtuS support to OpenStack is straightforward. A CUDA-enabled virtual machine is simply an x86 with the gVirtuS shared library pre-loaded and a working gVirtuS backend residing on the host. There were, however, several challenges we encountered in enabling GPU access within a heterogeneous cloud environment. In this section we describe the challenges and our OpenStack implementation.

Our first challenge was that gVirtuS's default action is to allow a virtual machine to access any GPU within the host machine. By contrast, our OpenStack implementation should provide exclusive access to GPUs that have been dedicated to a specific virtual machine. Adding this functionality required modification to the backend.

The second challenge was that gVirtuS' backend process consumes host resources while servicing a virtual machine. Typically the resource utilization of gVirtuS is low; however, crashed CUDA processes within the virtual machine often resulted in runaway backend processes that pinned a CPU's utilization to 100%. This was due to the spinlocks used during host to virtual machine communication. The solution to this was to introduce a heartbeat mechanism to gVirtuS that killed off runaway backend processes in the event of software crashes within the virtual machine.

With these changes we integrated gVirtuS into OpenStack's Nova compute service via the libvirt connection. Scheduling a job to a node is a two step process. First the Nova scheduler identifies a host with an available GPU. Second, the host that was assigned the job chooses the GPU to assign to the virtual machine. Specific GPU $\rightarrow$ VM assignments are managed through a hash table within the Nova compute service.

Once the node and the GPU have been identified, Nova spawns the virtual machine. During this process the job parameters are identified, and if a GPU is requested, a gVirtuS backend process is started for the virtual machine, and the job otherwise starts as usual. Similarly, when a job is destroyed, the virtual machine is torn down, the gVirtuS backend is killed, and the hash table entry mapping the virtual machine to a GPU is deleted.

VI. Supporting non-x86 architectures

Some (non-x86 based) machine architectures of interest to technical computing users have either poor or non-existent support for virtualization. For example, our heterogeneous

target, Tilera [22] Linux (MDE-2.1.2) does not yet support KVM or Xen virtualization.

One alternative to using virtualization to provision hardware in a cloud environment is to do *bare-metal* provisioning: rebooting the machine to a fresh system image before handing over control to the user, and wiping the local hard drive when the user is done with the resources.

To support the Tilera architecture through OpenStack, we developed a *proxy compute node* implementation, where our customized nova-compute service acts as a front-end that proxies requests for nodes to a Tilera-specific back-end that does the bare metal provisioning of the nodes as needed.

Our intention is to ultimately support different provisioning back-ends. Several provisioning tools are available, such as Dell's crowbar [23], as an extension of opscode's Chef system [24], Argonne National Lab's Heckle [25], xCat [26], Perceus [27], OSCAR [28], and ROCKS [29]. These tools provide different bare-metal provisioning, deployment, resource management, and authentication methods for different architectures. These tools use standard interfaces such as PXE (Preboot Execution Environment) [30] boot and IPMI (Intelligent Platform Management Interface) [31] power cycle management module. For boards that do not support PXE and IPMI, such as the TILEmpower board, specific back ends must be written.

To support provisioning TILEmpower boards, the proxy compute node is designed as follows. An x86 proxy compute node is connected to the TILEmpower boards through the network such that a cloud user can ssh into them directly after an instance starts on the TILEmpower board. A proxy compute code may handle multiple TILEmpower boards. A TILEmpower board is configured to be tftp-bootable. The proxy compute node acts as the tftp server for the TILEmpower boards. After the proxy compute node receives instance images from the image server, it wakes up a TILEmpower board and passes the images to the TILEmpower board using tftp protocol, and controls their booting. Once a TILEmpower board is booted, the proxy compute node does not do anything except control board power. Once an instance is running, a user can access the board through ssh. The block diagram shown in Figure 3 describes the procedure in detail.

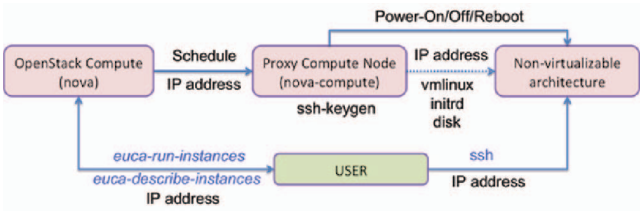

Figure 3. Block diagram of proxy bare-metal compute node

The TILEmpower compute node shown in Figure 4 is an example of proxy compute node.

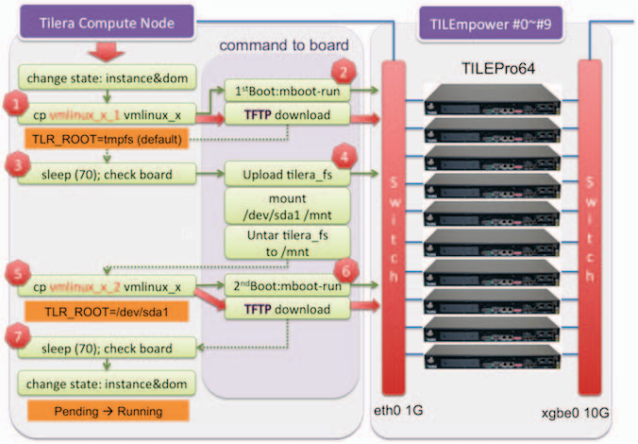

Figure 4. TILEmpower compute node, an example of a proxy compute node

We are currently working on the design and implementation of a general bare-metal proxy compute node that can be used not only for TILEmpower boards but also other architectures. This design includes three parts: default connection functions (spawn/reboot/destroy), bare-metal domain-related information, a and bare-metal node-specific *driver* part.

VII. User Interface

Nova strives to provide API-compatibility with popular systems like Amazon and Rackspace to enable compatibility with existing tool sets created for interaction with offerings from these vendors. This broad compatibility and support for multiple APIs prevents vendor lock-in. These APIs provide all the common commands used to interface with and administer Nova, including for running instances, terminating instances, rebooting instances, attaching volumes, detaching volumes, getting console output etc.

The Euca2ools (EC2) API [15] is the most popular cloud API, and is used in our reference implementation. The community is currently also working on the open source OpenStack API [32], which would be compatible with the Rackspace cloud API. There are plans to add new features to the OpenStack API and to make it extensible to suit specific installations. This would allow it to work better with heterogeneous clouds, and we plan to use it in future deployments.

The key component that implements OpenStack's user interface is the *API server*. A messaging queue brokers the interaction between the API server, compute nodes, volumes, networking controllers and the scheduler. A request from a user is received by the API server, which first authenticates the user with an access and secret key. The users access key needs to be included in the request, and the request must be signed with the secret key. Availability of objects required to fulfill the request is evaluated and, if available, the request is routed to the queuing engine for the relevant

workers. Workers continually listen to the queue, and when such listening produces a work request, the worker takes assignment of the task and begins its execution. Upon completion, a response is dispatched to the queue, which is received by the API server and relayed to the originating user. Database entries are queried, added, or removed as necessary throughout the process.

The *OpenStack Dashboard* provides a web interface into OpenStack Compute to give end users, application developers, and DevOps staff similar functionality to the command-line API. The OpenStack Dashboard is implemented using Django [33], which is a web application framework written in Python. The example Dashboard installation provided by OpenStack is called *openstack-dashboard* [34]. It uses a sqlite3 database and the default Django server. To create a more robust, production-ready installation that supports heterogeneity, we configured Dashboard to use the Apache web server and MySQL database. The Dashboard web interface has several views that make it easy to work with and administer the cloud.

The *Instances* page shows the list of current instances and their state. We added a column to the instances table in this view that shows the architecture on which the instance is running. The *Images* page shows all the images that are available to the user. We added an additional field that shows the architecture for which an image is built. The *Keys* page shows the currently available key-pairs and provides an option to create new keys as required. The *Volumes* allows a user to create and manage volumes. The *Lauch Image* page provides the interface to launch new instances. We added new options on this page for the user to select more heterogeneity components as explained below.

As described in Section IV, we let the user choose their instance with a much finer granularity than the Amazon EC2. For example, we let the user specify the CPU architecture, specific CPU features, accelerators such as GPUs, hypervisor type etc. To support this heterogeneity, we need to specify these additional choices as strings that are appended to the instance type string. These are passed as usual to the API server and are serviced by the scheduler. This scheme does not require any change to the EC2 API. To make the dashboard more user-friendly, we present these choices as selectable options in the dashboard on the Launch Instance page.

VIII. Initial Deployment

We deployed OpenStack with the extension of our heterogeneous architecture support on the compute cluster at USC/ISI (Arlington campus). Our extension is applied to the OpenStack Cactus release, and we plan to merge our extension to the upstream OpenStack Diablo release that is due Q3 2011.

Our compute cluster has three heterogeneous compute elements:

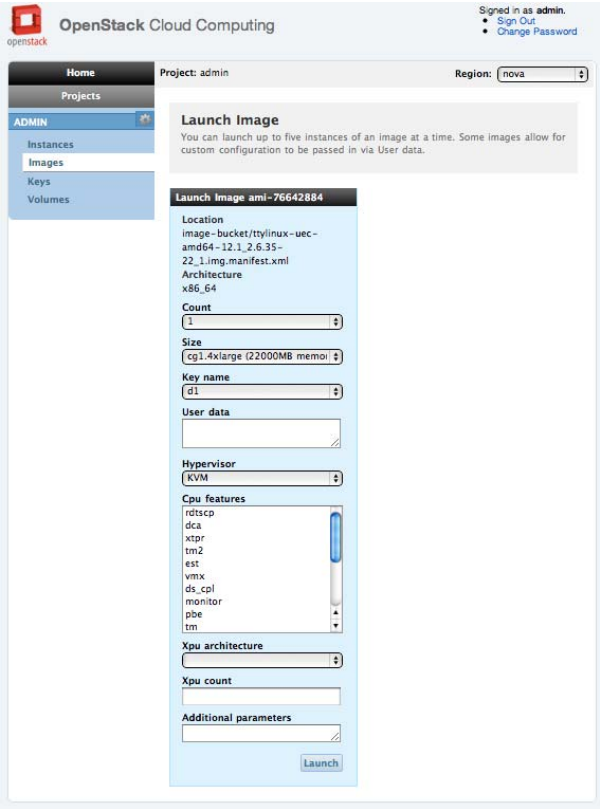

Figure 5. Launch image page of Dashboard.

- one SGI UV100 with 128 cores and 512GB of main memory
- three GPU-enabled x86_64 machines, each equipped with an NVIDIA Tesla S2050
- ten TILEmpower boards with TILEPro64 processors

One x86_64 machine that we call the *head node* is dedicated to run most of OpenStack nova services except nova-compute, which include mysql, rabbitmq-server, nova-api, nova-network, nova-objectstore and nova-scheduler. Another x86_64 machine is dedicated for a proxy node for all the TILEmpower boards. All of them are connected to 10Gig Ethernet switches. All machines except the head node and TILEmpower boards run the OpenStack nova-compute service. All x86_64 machines including the SGI UV100 run 64-bit SUSE Linux Enterprise 11 SP1. The TILEmpower boards run Linux for TILEPro64 processor. A user logs into the head node and launches instances on the other compute nodes.

IX. Related Work

In research, universities have expressed interest in massively scaled computing infrastructure for exploring new approaches to computationally-intensive problems that would otherwise be too difficult given ordinary computing infrastructure [35]. Recently, the National Science Foundation has

awarded nearly $5 million in grants to fourteen universities through its Cluster Exploratory (CLuE), and in conjunction with IBM/Google Cloud Computing University Initiative [35]. One award recipient, the University of Maryland Institute for Advance Computer Studies (UMIACS), is conducting research for gene sequencing analysis that involves vast quantities of information [36]. Researchers at the University of Massachusetts-Amherst Center for Intelligent Information Retrieval (CIIR), also funded by NSF CLuE, are exploring relationships among words, such that search results may be more effectively ranked [35], [37]. Research methods for analysis of tens of terabytes of data for anomaly detection, classification and motion tracking are being developed at the University of Washington Survey Science Group (SSG) [38]. SSG is also funded through the CLuE initiative.

Studies show that clouds are a viable option for HPC in comparasion to local clusters [39] [40] [41]. Cloud providers are equipping their clouds with HPC hardware to meet the needs of users from scientific community. Amazon EC2 with its Cluster Compute and Cluster GPU instances provides GPU processing power for users to run HPC applications on the cloud [5]. National Center for Supercomputing Applications(NCSA) also deployed high-performance computing cluster,dubbed iForge consisting of mix of servers with different archtectures and different processing powers to meet the needs of its private cloud users [42].

X. Conclusion and future work

Cloud computing is quickly becoming a dominant model for end-users to access centrally managed computational resources. Through our work in extending OpenStack, we have demonstrated the feasibility of providing technical computing users with access to heterogeneous computing resources using a cloud computing model.

The initial work to date has been to build a proof-of-concept prototype. In the next phase of our research, we will focus more on performance issues. Since program performance is a high-priority for technical computing users, it is important to understand the performance impact of cloud-related technologies such as virtualization, and to optimize the system configuration for program performance where possible. In particular, we will be evaluating alternative mechanisms for allowing GPU access from an instance, and we will be examining the impact of virtualization on the SGI UV hardware.

For enhancing the functionality of our heterogeneous cloud, we will be looking at supporting locality requests to improve network communication performance on multinode applications, as well as incorporating scientific workflow frameworks such as Pegasus [43], as well as supporting dynamic network provisioning, by integrating frameworks such as DRAGON [44].

References

[1] K. Keahey, "Cloud computing for science," in *Proceedings of the 21st International Conference on Scientific and Statistical Database Management*, New Orleans, LA, June 2009.

[2] E. Deelman, G. Singh, M. Livny, J. B. Berriman, and J. Good, "The cost of doing science on the cloud: the montage example," in *Supercomputing Conference*, 2008.

[3] L. Wang, J. Tao, M. Kunze, A. Castellanos, D. Kramer, and W. Karl, "Scientific cloud computing: Early definition and experience," in *High Performance Computing and Communications, 2008. HPCC '08. 10th IEEE International Conference on*, sept. 2008, pp. 825 –830.

[4] "OpenStack." [Online]. Available: http://www.openstack.org

[5] "Amazon elastic compute cloud (amazon EC2)." [Online]. Available: http://aws.amazon.com/ec2/

[6] P. C. Corporation. (2011) Platform lsf. [Online]. Available: http://www.platform.com

[7] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A Berkeley view of cloud computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, Feb 2009. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html

[8] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus open-source cloud-computing system," in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, ser. CCGRID '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 124–131. [Online]. Available: http://dx.doi.org/10.1109/CCGRID.2009.93

[9] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Capacity leasing in cloud systems using the OpenNebula engine," in *Proceedings of the 2008 Workshop on Cloud Computing and its Applications (CCA08)*, October 2008.

[10] K. Keahey and T. Freeman, "Contextualization: Providing one-click virtual clusters," in *eScience, 2008. eScience '08. IEEE Fourth International Conference on*, dec. 2008, pp. 301 –308.

[11] "NASA Nebula." [Online]. Available: http://nebula.nasa.gov

[12] "DOE Magellan." [Online]. Available: http://magellan.alcf.anl.gov/

[13] "RabbitMQ." [Online]. Available: http://www.rabbitmq.com/

[14] K. Pepple. (2011, Apr.) OpenStack nova architecture. [Online]. Available: http://ken.pepple.info/openstack/2011/04/22/openstack-nova-architecture

[15] "Euca2ools." [Online]. Available: https://launchpad.net/euca2ools

[16] "Xen hypervisor." [Online]. Available: http://xen.org/

[17] "Parallels desktop." [Online]. Available: http://www.parallels.com/products/desktop/pd4wl/

[18] "Kernel based virtual machine (KVM)." [Online]. Available: http://www.linux-kvm.org

[19] G. Giunta, R. Montella, G. Agrillo, and G. Coviello, "A GPGPU transparent virtualization component for high performance computing clouds," in *Proceedings of the 16th international Euro-Par conference on Parallel processing: Part I*, ser. EuroPar'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 379–391. [Online]. Available: http://portal.acm.org/citation.cfm?id=1887695.1887738

[20] L. Shi, H. Chen, and J. Sun, "vCUDA: GPU accelerated high performance computing in virtual machines," in *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–11. [Online]. Available: http://portal.acm.org/citation.cfm?id=1586640.1587737

[21] F. Bellard, "QEMU, a fast and portable dynamic translator," in *Proceedings of the annual conference on USENIX Annual Technical Conference*, ser. ATEC '05. Berkeley, CA, USA: USENIX Association, 2005, pp. 41–41. [Online]. Available: http://portal.acm.org/citation.cfm?id=1247360.1247401

[22] T. Corporation. (2010, Jan.) Tilera. [Online]. Available: http://www.tilera.com/

[23] R. Hirschfeld. (2011, May) Cloud installer (system wide, bare metal, system prereqs, components, networks). [Online]. Available: https://blueprints.launchpad.net/openstack-common/+spec/installer-crowbar

[24] OPSCODE. (2011, Jun.) Opscode chef - open source sysems integration framework. [Online]. Available: http://www.opscode.com/

[25] A. N. Laboratory. (2010, Jul.) Heckle. [Online]. Available: http://trac.mcs.anl.gov/projects/Heckle/

[26] xCat Open Source Project. (2011, May) xCat extreme cloud administration toolkit. [Online]. Available: http://xcat.sourceforge.net/

[27] P. O. S. Project. (2010, Apr.) Perceus provision enterprise resources and clusters enabling uniform systems. [Online]. Available: http://www.perceus.org/

[28] M. J. Brim, T. G. Mattson, and S. L. Scott, "OSCAR: open source cluster application resources," in *Proceedings of the 3rd Annual Linux Symposium*, 2001.

[29] P. M. Papadopoulos, M. J. Katz, and G. Bruno, "NPACI rocks: tools and techniques for easily deploying manageable linux clusters," *Concurrency and Computation: Practice and Experience*, vol. 15, no. 7-8, pp. 707–725, 2003. [Online]. Available: http://dx.doi.org/10.1002/cpe.722

[30] Wikipedia. (2011, Jun.) Preboot execution environment. [Online]. Available: http://en.wikipedia.org/wiki/$Preboot\_Execution\_Environment$

[31] ——. (2011, May) Intelligent platform management interface. [Online]. Available: http://en.wikipedia.org/wiki/$IPMI$

[32] "Openstack api." [Online]. Available: http://wiki.openstack.org/OpenStackAPI_1-1/

[33] "Django." [Online]. Available: https://www.djangoproject.com/

[34] "OpenStack dashboard." [Online]. Available: https://launchpad.net/openstack-dashboard

[35] (2009, Apr.) National science foundation awards millions to fourteen universities for cloud computing research. [Online]. Available: http://www.nsf.gov/news/news_summ.jsp?cntn_id=114686

[36] White, James, Navlakha, Saket, Nagarajan, Niranjan, Ghodsi, Mohammad-Reza, Kingsford, Carl, Pop, and Mihai, "Alignment and clustering of phylogenetic markers - implications for microbial diversity studies," *BMC Bioinformatics*, vol. 11, no. 1, p. 152, 2010. [Online]. Available: http://www.biomedcentral.com/1471-2105/11/152

[37] G. Druck and A. McCallum, "High-performance semi-supervised learning using discriminatively constrained generative models," in *International Conference on Machine Learning*, 2010. [Online]. Available: http://www.cs.umass.edu/ gdruck/pubs/druck10high.pdf

[38] Wiley, Connolly, Gardner, Krughoff, Balazinska, Howe, Kwon, and Bu, "Astronomy in the cloud: Using MapReduce for image co-addition," *Publications of the Astronomical Society of the Pacific*, vol. 123, no. 901, pp. 366–380, 2011. [Online]. Available: http://www.jstor.org/stable/10.1086/658877

[39] Z. Hill and M. Humphrey, "A quantitative analysis of high performance computing with Amazon's EC2 infrastructure: The death of the local cluster?" in *Grid Computing, 2009 10th IEEE/ACM International Conference on*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 26–33.

[40] M. Fenn, M. A. Murphy, and S. Goasguen, "A study of a KVM-based cluster for grid computing," in *ACM-SE 47:Proceedings of the 47th Annual Southeast Regional Conference*, 2009, pp. 1–6.

[41] J. Ekanayake and G. Fox, "High performance parallel computing with clouds and cloud technologies," in *First International Conference CloudComp on Cloud Computing*, 2009.

[42] "NCSA deploys new high-performance cluster dedicated to industrial use." [Online]. Available: http://www.ncsa.illinois.edu/News/11/0531NCSAdeploys.html

[43] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz, "Pegasus: a framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming Journal*, vol. 13, no. 3, pp. 219–237, 2005.

[44] T. Lehman, J. Sobieski, and B. Jabbari, "DRAGON: a framework for service provisioning in heterogeneous grid networks," *Communications Magazine, IEEE*, vol. 44, no. 3, pp. 84–90, March 2006.