RC2 – A Living Lab for Cloud Computing

Kyung Dong Ryu, Xiaolan Zhang, Glenn Ammons, Vasanth Bala, Stefan Berger, Dilma M Da Silva
Jim Doran, Frank Franco, Alexei Karve, Herb Lee, James A Lindeman, Ajay Mohindra, Bob Oesterlin
Giovanni Pacifici, Dimitrios Pendarakis, Darrell Reimer, Mariusz Sabath

IBM TJ Watson Research Center
Yorktown Heights, NY

Abstract

In this paper we present our experience in building the Research Compute Cloud (RC2), a cloud computing platform for use by the worldwide IBM Research community. Within eleven months of its official release RC2 has reached a community of 631 users spanning 34 countries, and serves on average 350 active users and 1800 active VM instances per month. Besides offering a utility computing platform across a heterogeneous pool of servers, RC2 aims at providing a living lab for experimenting with new cloud technologies and accelerating their transfer to IBM products. This paper describes our experience in designing and implementing a flexible infrastructure to enable rapid integration of novel ideas while preserving the overall stability and consumability of the system.

1 Introduction

Cloud Computing has become synonymous with ways to contain and manage IT costs for enterprises. Cloud Computing is a paradigm where compute capacity is made available to users in an on-demand fashion through a shared physical infrastructure. The expectation is that sharing hardware, software, network resources, and management personnel would reduce per unit compute cost for enterprises. Several vendors such as Amazon EC2, Google, and Rackspace have been providing commercial Cloud offerings. Though not enterprise-grade level yet, Cloud Computing has piqued the interest of several large enterprises, which have started deploying and experimenting with the technology for their test and development environments. IBM Research has developed and deployed a Cloud Computing platform called Research Compute Cloud (RC2) for use by the worldwide IBM Research community. The goals of RC2 are to establish an "innovation" platform for the IBM Research community and to serve as a "living" lab for the research technologies developed by the IBM Research community. The platform has been purposefully architected to facilitate collaboration among multiple research groups and encourage experimentation with cloud computing technologies. The platform also serves as a showcase of new research technologies to IBM customers and business partners.

The IT infrastructure of the IBM Research division resembles that of a global enterprise having many different lines of business spread across multiple geographies. IBM Research is a geographically distributed organization, consisting of several thousand research personnel spread across 9 research laboratories worldwide. Each IBM research lab operates its own local data center that is used predominantly for lab-specific research experiments. In addition, a portion of the data center infrastructure is collectively used for production workloads such as email, employee yellow pages, wikis, CVS servers, LDAP servers, etc. Critical production workloads can be replicated across different lab data centers for purposes of failover. This infrastructure is a substantial investment built over many years, and is very heterogeneous in its make up. For instance, IBM's POWER series systems and System Z mainframes are mixed with many generations of commodity x86 blade servers and IBM iDataplex systems.

The Research Compute Cloud (RC2) is an infrastructure-as-a-service cloud built by leveraging the existing IT infrastructure of the IBM Research division. Its goals were two-fold: (a) create a shared infrastructure for daily use by the research population, and (b) provide a living lab for experimenting with new cloud technologies. There were several challenges that the team needed to address to meet the two goals. The first challenge was to design and build a consistent infrastructure-as-a-service interface over a heterogenous infrastructure to meet the needs of the Research user community. The second challenge was to enable a true living lab where the Research community could develop and test new technologies in the cloud. The architecture of RC2 had to be flexible enough to enable experimentation with different cloud technologies at the management, platform, and application layers. All this had to be achieved without any disruptions to the stability and consumability of the overall infrastructure.

In this paper, we present our experience in building RC2. In Section 2, we present the architecture of RC2 to meet the two design goals. Next, we discuss the implementation of RC2 in Section 3. Section 4 presents our experience specifically in the context of pluggability and extensibility of the environment. We conclude the paper by discussing related work in Section 5 and future work in Section 6.

2 Architecture

As mentioned in Section 1, RC2 aims to provide a research platform where exploratory technologies can be rapidly introduced and evaluated with minimal disruption on the operation of the cloud. This requirement calls for a componentized, extensible cloud architecture.

Figure 1 shows the architecture of RC2 which consists of a cloud dispatcher that presents an external REST API to users and a collection of managers that provide specific services. For each manager, the dispatcher contains a proxy whose role is to marshal requests to and responses from the manager itself.

This architecture enables a loose coupling between the managers. Any manager only knows its corresponding proxy; there is no direct communication between managers. Different groups within IBM Research can work on different managers without anyone mandating how their code should integrate. Groups only need to agree on the APIs that the manager proxies will expose within the dispatcher.

The dispatcher is driven by an extensible dispatch table that maps request types to manager proxies. When a request enters the dispatcher (whether from an external source like an RC2 user or an internal source like one of the managers), the dispatcher looks up the request's signature and dispatches it to the manager proxy responsible for that type of request. A new manager can be added simply by adding its request type and mapping information to the table.

Another benefit of this design is that, because all requests pass through the dispatcher, features such as admission control, logging and monitoring can be implemented easily in the dispatcher. A potential drawback is that the dispatcher becomes a bottleneck, but this problem can be solved by distributing requests among multiple dispatcher instances.

Figure 1 shows the managers that currently exist in RC2. The user manager authenticates users. The image manager catalogs, accesses, and maintains virtual-machine images. The instance manager creates, deploys, and manipulates runnable instances of the image manager's images. The security manager sets up and configures the network isolation of cloud tenants' security domains for communication both outside the cloud and with other security domains inside the cloud.

Distribution of functionality implies distribution of the system's state among individual components. This distribution makes it difficult to obtain a complete and consistent view of the system state during long-running requests (for example, instance creation), which complicates failure detection and recovery. Our architecture tackles this problem by requiring each manager to maintain and communicate the states of the objects it manages. For example, both images and instances have associated states, which can be queried by sending a "describe image" or "describe instance" request to the appropriate manager. Long-running requests are processed in two stages. The first stage synchronously returns an object that represents the request's result, and the second stage asynchronously completes the time-consuming remainder of the request and updates the object's state accordingly. Request completion (or failure) can be determined by querying the object's state.

Another challenge was to design a set of infrastructure-as-a-service APIs that could be implemented consistently across a heterogeneous pool of servers. Differences among platforms can be huge. For example, consider two different server platforms: an IBM xSeries blade server and an IBM pSeries blade server. The former runs a software virtual-machine monitor (in RC2, Xen or KVM) on commodity x86 hardware, while the latter runs a firmware hypervisor (PHYP), on IBM Power hardware (also referred to as System P). These two platforms support different operating systems, different image formats, and different mechanisms for creating instances from images.

For example, for Xen and KVM based VM instances, the images exist in raw (block) disk format. Deploying those images into new VM instances requires copying the disks onto the host's storage and mounting the file system of those disks to customize the images. The process is completely different for the AIX operating system that runs on the pSeries servers – the images exist in a special backup (tar-like) format, and are referred to as *mksysb* backups. Deploying those images into new PHYP instances requires booting off an initial *install kernel* which in turn drives the installation of the image files into the newly booted VM. The installation is achieved through a special central server called the Network Installation Manager (NIM), which creates filesystems for the instance, with files restored from the *mksysb* backups.

Our design supports multiple platforms by requiring

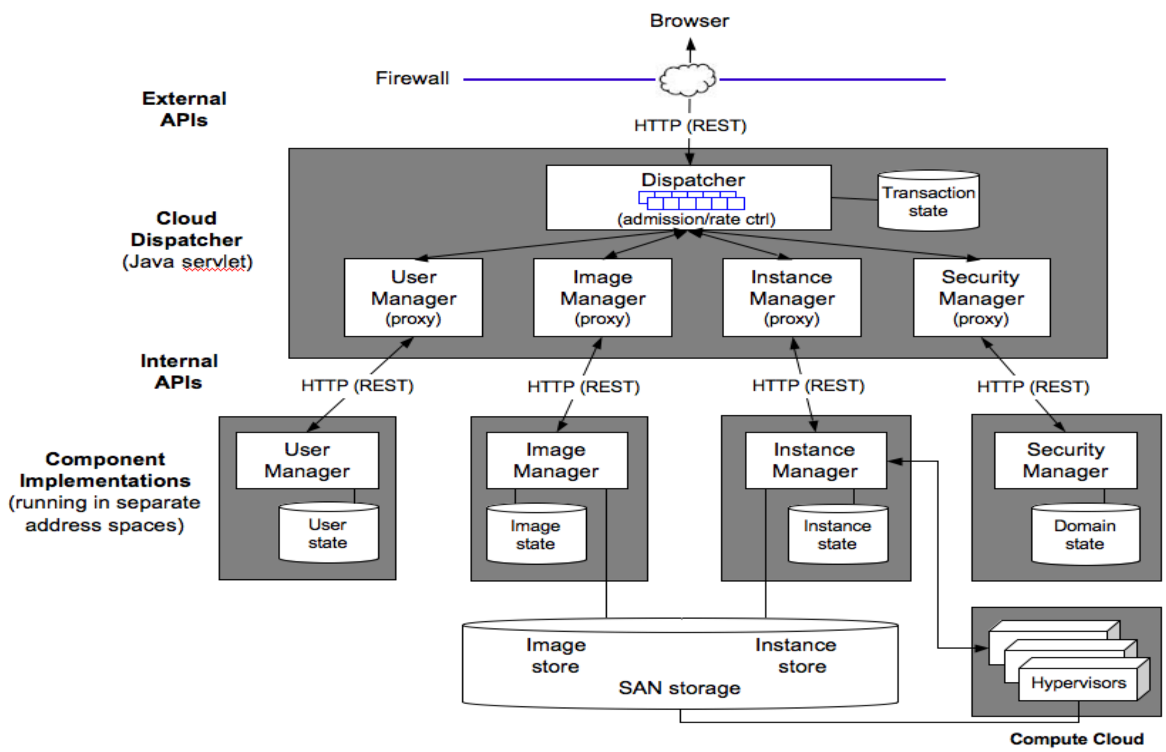

Figure 1: RC2 Architecture

that requests avoid platform-specific parameters. For example, both types of images are stored in the repository with the same format, has identical list of attributes, and can be queried in an identical manner. Similarly, the same instance creation API is used to start an instance for both image types (although the parameter values vary). The requester is not required to know the specific platform type of the image or instance that she is operating on. This approach minimizes the complexity of supporting multiple platforms, as only the instance manager, which receives requests for creating instances, must concern itself with differences among platforms.

3 Implementation

The RC2 hardware infrastructure is comprised of management servers, the host server pool, and the storage subsystem. The management servers host RC2 management services, provision and capture of virtual machines, and http access for users. The host pool houses the provisioned instances and consists of a collection of IBM iDataplex blades varing in size from 32GB-4way to 128GB-8way systems. The storage pool consists of a SAN subsystem that is shared across both the host servers and the management servers.

3.1 Dispatcher

The RC2 cloud dispatcher is composed of three layers: a REST servlet, the cloud manager, and several manager proxies. The REST servlet provides an HTTP-based REST interface to cloud operations. The interface can be used by programs as well as through a web-based graphical user interface. The manager proxies decouple interactions between user and cloud dispatcher and communication between the dispatcher and managers. This separation promotes flexibility of managers while allowing uniform cloud interfaces to users. Although, in the current implementation, all managers are accessed through REST APIs, they can be easily replaced with implementations that use different communication mechanisms such as Java Message Service (JMS).

The cloud manager sits between the REST servlet and the manager proxies, providing admission control and rate control using dispatch queues and request-handler threadpools. There are currently two types of dispatch queues: synchronous request queues and asynchronous request queues. The former handles short-lived cloud requests such as looking up image information and listing an owner's instances whereas the latter handles long-lived cloud requests such as creating an instance or capturing an image. The threadpool size of the synchronous request queue is typically set to a large value to allow more requests to be processed concurrently while that of the asynchronous request queue is limited to a number that matches the rate at which the back-end manager can process requests. The configuration of dispatch queues such as queue length and threadpool size can be changed at runtime through an administrative REST interface, which is designed to allow feedback-based rate control in the future.

3.2 Instance Manager

The instance manager keeps track of the cloud's virtual-machine instances. An instance-manager implementation must provide several basic services: "start instance", which adds a running instance to the cloud; "stop instance", which stops an instance; "delete instance", which removes an instance from the cloud; and a query service for listing instances and their state. Only the implementation of "start instance" is described here because it is the least straightforward.

Starting an instance involves four tasks: selecting a target host, creating and configuring an instance on that host (which includes choosing a security domain), retrieving and configuring the virtual-machine image, and finally starting the instance. Each task is implemented by plugins, so as to support a variety of host and image types.

The instance manager selects a host with the proper resources to run the user-requested instance. The current implementation uses a best-fit algorithm [7] that considers memory, cpu, disk, network connectivity, and host-specific requirements such as the host's architecture and virtual-machine monitor. Selecting the host also binds some instance parameters, including the IP address of the new instance.

The instance manager retrieves the image from the image manager and configures it for execution. Image configuration sets both user-specific parameters, such as ssh keys, and instance-specific parameters, such as the IP address. Some parameters are set by modifying the retrieved image before startup while others are set at startup-time by embedding an "activation engine" [3] in the image that runs the first time the instance boots. The instance-specific parameters are provided through a virtual floppy drive. The activation engine is designed for extensibility and can configure operating system, middleware, and application parameters.

Next, the instance manager instructs the chosen host to allocate a new instance. The details are host-specific; the current implementation includes plugins for AIX hosts and for x86 hosts based on Xen and KVM.

Finally, the instance manager starts the instance. The user is notified and a description of the new instance is sent to a database for compliance tracking.

3.3 Image Manager

The image manager maintains a library of images. The image manager cooperates with the user manager to control access to images and with the instance manager to create runnable instances of images and to capture images of runnable instances as images.

Each image has a unique image identifier, which names the image for access-control purposes. The library stores one or more versions of each image and each version has a version identifier, which names both data and metadata. The data consists of a set of files, including disk images and other files required to create a runnable instance. The metadata is a set of version attributes, such as a name, a description, and the identifier of the version's parent.

Version data is immutable. Therefore, if a disk image is modified by a running instance, it can be captured back to the library only as a new version, whose parent will be the version from which the instance was created. Some version attributes are mutable, such as the description, while others are immutable, such as the parent identifier. The access-control information associated with an image is mutable.

The most important image manager services are "checkout" and "checkin". Given a version or image identifier and a target URL, checkout creates a runnable instance from a version; if an image identifier is supplied, the most recent version of that image will be checked out. The target URL identifies a directory on the SAN where the instance manager expects to find the runnable instance and to which the image manager copies the version's data files. The image manager also places control files in the directory that, among other things, identify the source version.

Given a source URL, which identifies a directory on the SAN that was populated by a checkout, checkin creates a new version. Note that the directory's data files, including its disk images, may have been modified by instance execution. There are two kinds of checkin calls: the first creates a new version of the original image while the second creates the first version of a new image. Currently, only the second kind is exposed to RC2 users.

Both checkout and checkin are asynchronous calls. The instance manager invokes these two interfaces and tests for completion by polling a status file, which the image manager updates on completion, or by supplying the URL of a callback, which the image manager invokes on completion.

The image manager controls access to images in the library. Each image has an owner, a list of users and groups with checkout access, and a list of users and groups with both checkout and checkin access. Only the owner may update the lists. Each image manager call includes an owner and a list of groups to which the owner belongs, which the manager uses to verify that the caller has the required access for the call. The image manager assumes that a call's owner and group list is correct: the user manager is responsible for user authentication and the cloud dispatcher ensures that calls do not forge user or group names.

The image manager provides other services besides checkin and checkout. These include calls that list, describe, and delete images and versions, plus calls that update access-control lists. Deleted versions retain their metadata but lose their data files.

The image manager uses a file-granularity, content-addressable store (CAS) to maintain the image content [9]. The CAS saves space by guaranteeing that the same item is never stored twice. It also keeps the reference information necessary to garbage collect deleted image data.

3.4 Security Manager

RC2 has been architected with several mechanisms to provide security in a cloud environment. In particular, the security manager provides support for isolation between different cloud user's workloads in a heterogeneous, multi-tenant environment. The isolation model follows our established concepts of Trusted Virtual Domains (TVDs) [2] and a Trusted Virtual Data Center (TVDc) [10]. A TVD is a grouping of (one or more) VMs belonging to the same user that share a trust relation and common rules for communicating among themselves as well as with the outside world.

The security manager exports a broad API through the cloud dispatcher and provides extensive functionality for life-cycle management of security domains and their runtime configuration.

The security manager is currently built on top of modifications to the Xen daemon for the establishment and runtime configuration of firewall rules on virtual machines' interfaces in Domain-0. Our architecture makes use of the fact that in the Xen hypervisor all virtual machines' network packets pass through the management virtual machine (Domain-0) and firewall rules can be applied on the network interface backends that each VM has in that domain. This allows us to filter network traffic originating from and destined to individual virtual machines.

The extensions to the Xen daemon provide functionality for the application of layer 2 and layer 3 network traffic filtering rules using Linux's ebtables and iptables support. While a VM is running, its layer 3 network filtering rules can be changed to reflect a user's new configuration choices for the security domain a virtual machine is associated with. We support a similar network traffic filtering architecture with the Qemu/KVM hypervisor where we implemented extensions to the libvirt management software providing equivalent functionality as the extensions to the Xen daemon.

Functionality that the security manager provides for support of security domain life cycle management involves the following:

- Creation and destruction of security domains.
- Renaming and configuration of parameters of security domains.
- Retrieval of security domain configuration data.
- Modifications of security domains' network traffic rules.
- Establishment of collaborations between security domains of the same or different cloud tenants.

Altogether, the security manager adds 17 new commands to the dispatcher API.

The realization of the security domains concept drove extensions to several other existing components in the architecture. Extensions were implemented in the cloud dispatcher layer to make the new APIs visible to other management components as well as external entities. The instance-manager request that creates a virtual machine instance was extended with optional parameters describing the security domain into which a virtual machine is to be deployed. A new internal request was added to the instance manager for deployment of filtering rules associated with VM instances. Several previously existing workflows, which are part of the instance manager, were modified to notify the security manager of VMs' life cycle events as well as to perform configuration in the Xen management virtual machine (Domain-0).

3.5 Chargeback

We implemented a simple allocation-based pricing model to experiment with users' behavior in resource consumption under different pricing models. Users are charged for compute capacity based on a billable unit of "per instance hour consumed". This begins with instance creation and ends when an instance is destroyed. At this time, the same charges are incurred whether the instance is active (that is, running) or inactive (stopped). Rates differ by system type (XEN, PHYP, and KVM) and configuration (small, medium, large, and extra large). In addition, there are separate charges for end-user initiated transactions that lead to state changes of their instances (Initialize, Start, Stop, Reboot, Destroy). Charges are calculated on an hourly basis and are integrated with IBM's existing internal billing systems.

3.6 User Manager

The user manager authenticates users by invoking services available in the IBM infrastructure and associating authentication with sessions. It also manages user-specific information, such as ssh public keys, that can be queried by other managers during provisioning.

4 Experience

RC2 was released in a Beta version in early April, 2009, and officially released to IBM Research world-wide in early September, 2009. In this section, we present global usage statistics of RC2 and our experiences using RC2 as a research platform to experiment with new cloud technologies.

4.1 RC2 Usage

Within 11 months of its official production release, RC2 has served 631 distinct users spanning 34 countries. The image library has accumulated a collection of 2286 images, all of which derive from just three root images that were imported into the libary at the beginning. The number of images in the library grew starting about a week after the Beta release. The library grew modestly during the Beta testing period but has been experiencing faster growth since the official release in early September. The number of instances has grown at a similar rate; Figure 2 shows this growth since the production release.

The average number of active instances per month is also growing, reaching 1800 in the most recent month. This includes 102 instances of the System P type. On the average there are about 350 distinct active users per month, who consume a total of 600,000 virtual-machine hours.

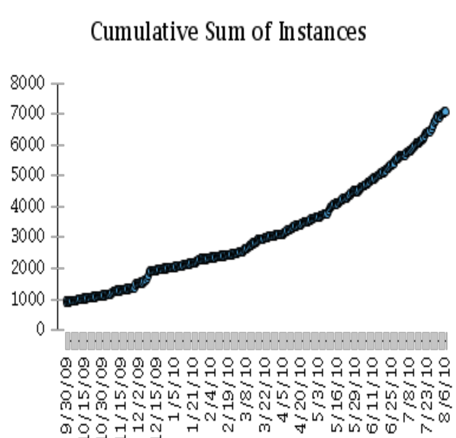

Figure 2: Instance Growth

RC2 was first released free of charge. When charges for instance ownership were introduced in early October, it had a dramatic impact on user behavior, as shown in Figure 3. There was a significant drop in the number of instances right after users received their first statements, leading to a drop in memory utilization. Interestingly, the number quickly bounced back, and memory utilization again approached pre-chargeback levels. We consider this to be a strong endorsement from our user community about the value of the service provided by RC2.

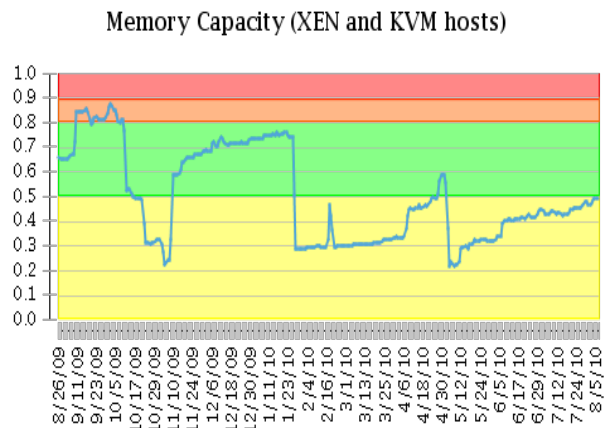

Figure 3: Cloud Memory Utilization
Percentage of memory allocated for instances as a ratio of total available memory.

4.2 RC2 as a Living Lab

In addition to its role as a production-quality IT offering for IBM's research organization, RC2 also serves as an experimental testbed for innovative cloud management technologies. To this end, we show how RC2's architectural emphasis on extensiblity and pluggability has helped facilitate these experimental activities.

The initial version of RC2 consisted of three managers: the image manager, the instance manager, and the user manager. The security manager was added to provide stronger isolation between multiple tenants' workloads in the cloud. While the security manager presented significant functionality enhancements, the core RC2 architecture remained essentially the same given that it was designed to be extensible from the start and most changes were contained at the cloud dispatcher.

The pluggable cloud dispatcher architecture enabled us to deploy an image manager based on the Network File System (NFS) for Research sites that lack a SAN storage environment. For these sites, we reimplemented the image manager interfaces using NFS as the backing store. As with the SAN, the file system is mounted on each hypervisor node so that images are locally accessible. The instance manager required no change as the NFS-based image manager supports the same set of requests as does the SAN-based image manager. The flexibility of RC2 allowed researchers to experiment with alternate implementations without requiring changes to other components.

Being a living lab implies that sometimes RC2 needs to deal with unusual infrastructure-level changes that are typically not seen in production systems. One such example is change of supported hypervisor types. Initially, RC2 adopted Xen as its only x86 hypervisor. Later on there was a strategic decision to switch to KVM, which means that RC2's entire Xen image collection needs to be converted to KVM.

Because RC2 is a production system, the conversion needs to be accomplished with minimal disruption to the user. This translates into two concrete requirements. First, the contents of all existing Xen images as well as instances need to be preserved. Users should just be able to start their existing images as usual without even noticing that the images will be in fact running on a KVM hypervisor. Similarly, when existing Xen instances are captured, they should automatically be converted to KVM without any user intervention. Second, conversion of the entire Xen image/instance collection needs to be achieved with zero downtime (except for the regularly scheduled maintenance window). During the conversion period, both Xen and KVM provisioning must be supported.

Our solution required multiple enhancements to be made to both the instance manager and the image manager. The instance manager, upon receiving a capture request, performed an on-the-fly conversion of the captured Xen image to a KVM image. The image manager was enhanced with a *locking* functionality that hid newly converted KVM images from the end user until the images were ready to be seen by the users. Again, the decoupled architecture of the RC2 system allowed individual component to be separately tested and replaced, making it possible to achieve the conversion without any disruption of the system.

The RC2 team successfully converted the entire Xen image collection (419 images) to KVM. The migration process started on May 6th, 2010 and ended on June 14th. During this whole period, the RC2 production system was continuously running with all functionalities enabled and no noticeable performance slowdown. The process was also completely transparent to the users. All conversion activities were shielded from the end users. End users did not notice any change of their images until the "conversion" day, at which point the newly converted images (with new image numbers) appeared on user's login view. Advance notice was sent to the users a few days earlier so they were prepared for this change on "conversion" day.

5 Related Work

Current cloud computing offerings focus on building an optimized, homogeneous environment for delivery of compute services to customers. Amazon's EC2 [1] and IBM's Developer Cloud [4] are examples of such offerings. By contrast, our work focuses on heterogeneity and providing a pluggable and extensible framework to serve as a living lab for cloud technologies. The open source project Eucalyptus [8] provides capabilities similar to those of Amazon's EC2 and could be used in a living lab, as developers can modify and extend the source. However, the project lacks support for heterogeneous platforms and a pluggable architecture.

6 Conclusion and Future Work

The RC2 project succeeded in achieving its two main goals: (1) it delivers high-quality cloud computing services for the IBM Research community and (2) it provides an effective framework for integration of novel ideas into a real cloud platform, rapidly enriching the evaluation of new technologies by offering meaningful, realistic user experience and usage/performance data. Many of these new technologies were adopted by newly announced IBM products in 2009 such as Websphere Cloudburst Appliance [6] and VM Control [5].

The current RC2 system is implemented only in the New York area data center. However, the RC2 services are available to all of the worldwide IBM Research Labs. In 2010, we plan to create RC2 zones in at least two other labs on two different continents.

The current RC2 production system has numerous monitoring probes installed at different points in the infrastructure and in the management middleware that runs the data center. These probes provide a rich set of real-time monitoring data, which is itself available as a service provided through a collection of REST APIs. We plan to use this feature to provide a simulated data center environment over the RC2 production environment, for experimental purposes. The simulated environment will behave as if it is the actual production environment underneath, by tapping into the real-time monitoring data provided by the probes.

7 Acknowledgments

To keep a system such as RC2 in production as it evolves rapidly and drastically can not be done without the contributions of many talented engineers and IT specialists. We thank the entire RC2 team for the intense effort.

We also thank our anonymous reviewers, and our shephard Matthew Sacks, for their insightful reviews and comments.

References

[1] AMAZON. Amazon Elastic Compute Cloud (Amazon EC2). `http://aws.amazon.com/ec2`.

[2] BUSSANI ET AL. Trusted Virtual Domains: Secure Foundations for Business and IT Services. Technical Report RC23792, IBM Research, November 2005.

[3] HE, L., SMITH, S., WILLENBORG, R., AND WANG, Q. Automating deployment and activation of virtual images. White paper, IBM developerWorks, August 2007. `http://www.ibm.com/developerworks/websphere/techjournal/0708_he/0708_he.html`.

[4] IBM. IBM Cloud Computing. `http://www.ibm.com/ibm/cloud`.

[5] IBM. VM Control Enterprise Edition. `http://www-03.ibm.com/systems/management/director/plugins/syspools/index.html`.

[6] IBM. Websphere Cloudburst Appliance. `http://www-01.ibm.com/software/webservers/cloudburst/`.

[7] KWOK, T., AND MOHINDRA, A. Resource calculations with constraints, and placement of tenants and instances for multi-tenant SaaS applications. *Services-Oriented Computing – ICSOC 5364* (2008).

[8] NURMI, D., WOLSKI, R., GRZEGORCZYK, C., OBERTELLI, G., SOMAN, S., YOUSEFF, L., AND ZAGORODNOV, D. The Eucalyptus open-source cloud-computing system. In *Proceedings of CC-Grid'09: the 9th IEEE International Symposium on Cluster Computing and the Grid* (Shangai, China, May 2009).

[9] REIMER, D., THOMAS, A., AMMONS, G., MUMMERT, T., ALPERN, B., AND BALA, V. Opening black boxes: Using semantic information to combat virtual machine image sprawl. In *The 2008 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '08)* (March 5-7, 2008).

[10] STEFAN BERGER ET AL. Security for the cloud infrastructure: Trusted virtual data center implementation. *IBM Journal of Research and Development 53*, 4 (2009).