Design and Implementation of the Workflow of an Academic Cloud

Abhishek Gupta, Jatin Kumar, Daniel Mathew, Sorav Bansal, Subhashis Banerjee and Huzur Saran

IIT Delhi

Abstract. In this work, we discuss the design and implementation of an academic cloud service, which we call Baadal. Tailored for academic and research requirements, Baadal bridges the gap between a private cloud and the requirements of an institution where request patterns and infrastructure are quite different from commercial settings. For example, researchers typically run simulations requiring hundreds of Virtual Machines (VMs) all communicating through message-passing interfaces to solve complex problems. We describe our experience with designing and developing a cloud workflow to support such requirements. Our workflow is quite different from that provided by other commercial cloud vendors (which we found not suited to our requirements).
Another salient difference in academic computing infrastructure from commercial infrastructure is the physical resource availability. Often, a university has a small number of compute servers connected to shared SAN or NAS based storage. This may often not be enough to service the computation requirements of the whole university. Apart from this infrastructure, universities typically have a few hundred to a few thousand "workstation" which are commodity desktops with local disk-attached-storage. Most of these workstations remain grossly underutilized. Our cloud infrastructure utilizes this idle compute capacity to provide higher scalability for our cloud implementation.

Keywords: Virtualization, Hypervisors

1 Introduction

Cloud Computing is becoming increasingly popular for its better usability, lower cost, higher utilization, and better management. Apart from publicly available cloud infrastructure such as Amazon EC2, Microsoft Azure, or Google AppEngine, many enterprises are setting up "private clouds". Private clouds are internal to the organization and hence provide more security, privacy, and also better control on usage, cost and pricing models. Private clouds are becoming increasingly popular not just with large organizations but also with medium sized organizations which run a few tens to a few hundreds of IT services.

An academic institution (university) can benefit significantly from private cloud infrastructure to service its IT, research, and teaching requirements. In

this paper, we discuss our experience with setting up a private cloud infrastructure in Indian Institute of Technology (IIT) Delhi, which has around 8000 students, 450 faculty members, more than 1000 workstations, and around a hundred server-grade machines to manage our IT infrastructure. With many different departments and research groups requiring compute infrastructure for their teaching and research work, and other IT services, IIT Delhi has many different "labs" and "server rooms" scattered across the campus. We aim to consolidate this compute infrastructure by setting up a private cloud and providing VMs to the campus community to run their workloads. This can significantly reduce hardware, power, and management costs, and also relieve individual research groups of management headaches.

We have developed a cloud infrastructure with around 30 servers, each with 24 cores, 10 TB shared SAN-based storage, all connected with 10Gbps fibre. We run Virtual Machines on this hardware infrastructure using KVM[4] and manage these hosts using our custom management layer developed using Python and libvirt[1].

1.1 Salient Design Features of Our Academic Cloud

While implementing our private cloud infrastructure, we came across several issues that have previously not been addressed by commercial cloud offerings. We describe some of the main challenges we faced below:

Workflow : In an academic environment we are especially concerned about simplicity and usability of the workflow for researchers (e.g., Ph.D. students, research staff, faculty members) and administrators (system administrators, policy makers and enforcers, approvers for resource usage).

For authentication, we integrate our cloud service with a campus-wide LDAP server to leverage existing authentication mechanisms. We also integrate the service with our campus-wide mail and Kerberos servers.

A researcher creates a request which which should be approved by the concerned faculty member before it is approved by the cloud administrator. Both the faculty member and cloud administrator can change the request parameters (e.g., number of cores, memory size, disk size, etc.) which is followed by a one click installation of the virtual machine. As soon as the virtual machine is installed, the faculty and the students are informed about the same with a VNC console password that they can use to access the virtual machine.

Cost and Freedom : In an academic setting, we are most concerned about both cost and freedom to tweak the software. For this reason, we choose to rely solely on free and open-source infrastructure. Enterprise solutions like those provided by VMware are both expensive and restrictive.

Our virtualization stack comprising of KVM[4], Libvirt[1], and Web2py[2] is open-source and available freely.

Workload Performance : Our researchers typically need large number of VMs executing complex simulations communicating with each other through message-passing interfaces like MPI[3]. Both compute and I/O performance is critical for such workloads. We have arranged our hardware and software to provide the maximum performance possible. For example, we ensure that the bandwidths between the physical hosts, storage arrays, and external network switches are the best possible with available hardware. Similarly, we use the best possible emulated devices in our Virtual Machine Monitor. Whenever possible, we use para-virtual devices for maximum performance.

Maximizing Resource Usage : We currently use dedicated high-performance server-class hardware to host our cloud infrastructure. We use custom scheduling and admission-control policies to provide maximal resource usage. In future, we plan to use the idle capacity of our lab and server rooms to implement larger cloud infrastructure at minimal cost. We discuss some details on this below.

A typical lab contains tens to a few hundred commodity desktop machines, each having one or more CPUs, a few 100 GBs of storage, connected over 100Mbps or 1Gbps ethernet. Often these clusters of computers are also connected to a shared Network-Attached Storage (NAS) device. For example, there are around 150 commodity computers in the Computer Science department. Typical utilization of these desktop computers is very low (1-10%). We intend to use this "community" infrastructure for running our cloud service. The VMs will run in background, causing no interference to the applications and experience of the workstation user. This can significantly improve the resource utilization of our lab machines.

1.2 Challenges

Reliability : In lab environments, it is common for desktops to randomly switch-off or become disconnected. These failures can be due to several reasons including manual reboots, pulling out of network cables, power outages, or physical hardware failures. We are working on techniques to have redundant VM images to be able to recover from such failures.

Network and Storage topology : Most cloud offerings use shared storage (SAN/NAS). Such shared storage can result in a single point of failure. Highly-reliable storage arrays tend to be expensive. We are investigating the use of disk-attached-storage in each computer to provide a high-performance shared storage pool with built-in redundancy. Similarly, redundancy in network topology is required to tolerate network failures.

Scheduling : Scheduling of VMs on server-class hardware has been well-studied and is implemented on current cloud offerings. We are developing scheduling algorithms for commodity hardware where network bandwidths are lower, storage

is distributed, and redundancy is implemented. For example, our scheduling algorithm maintains redundant copies of a VM in separate physical environments.

Encouraging Responsible Behaviour : Public clouds charge their users for CPU, disk, and network usage on per CPU-hour, GB-month, and Gbps-month metrics. Instead of a strict pricing model, we use the following model which relies on good community behaviour:

- Gold : The mode is meant for virtual machines requiring proportionally more CPU resources than other categories and are well suited for compute-intensive applications. We follow a provisioning ratio of 1:1, that is we don't overprovision as it is expected that the user will be using all the resources that he has asked for. We expect Gold instances to run only during actual experiments and simulations. The user is sent daily reminders about the running Gold instance, and requested to switch it off if not in heavy use.
- Silver : This mode is recommended for moderately heavy jobs. We typically follow a overprovisioning ratio of 2:1 implying that we allocate twice as much as resources as the server should ideally host. The user is sent a reminder every week for a running Silver instance. We expect researchers to use this mode during testing their software.
- Bronze : The mode is meant for virtual machines with a small amount of consistent CPU resources typically required when we are working on some code and before the actual run of the code. We expect users performing development activities (e.g., text editor, etc.) to use this mode (this mode is also called edit-mode for this reason). We follow a 4:1 provisioning ratio which means that we typically allow the resources to be overprisioned by a factor of four.
- Shut-Down : In this mode user simply shut down the virtual machine and the user is charged minimally.

The user can switch between the modes with the ease of a click and no rebooting of the virtual machine is required.,We use live migration capabilities of the hypervisor to implement these modes seamlessly.

The rest of this paper is structured as follows: in Section 2 we talk about our experiences with other Cloud Offerings. Section 3 describes key aspects of our design and implementation. Section 4 evaluates the performance of some relevant benchmarks on our virtualization stack over a range of VMs running over different hosts. Section 5 reviews related work, and Section 6 discusses future work and concludes.

2 Experiences with Other Cloud Offering

We tried some off-the-shelf cloud offerings before developing our own stack. We describe our experiences below.

2.1 Ubuntu Enterprise Cloud

Ubuntu Enterprise Cloud is integrated with the open source Eucalyptus private cloud platform, making it possible to create a private cloud with much less configuration than installing Linux first, then Eucalyptus. Ubuntu/Eucalyptus internal cloud offering is designed to be compatible with Amazon's EC2 public cloud service which offers additional ease of use.

On the other side, there is a need to familiarize with both Ubuntu and Eucalyptus, as were frequently required to search beyond Ubuntu documentation following the Ubuntu Enterprise Cloud's dependence on Eucalyptus. For example, we observed that Ubuntu had weak documentation for customizing images, which is an important step in deploying their cloud. Further even though the architecture is quite stable and worth using, it doesn't serve the requirements of a custom tailored interface which should suit an academic or research environment like ours.

2.2 VMware vCloud

VMware vCloud offers on demand cloud infrastructure such that end users can consume virtual resources with maximum agility. It offers consolidated datacenters and an option to deploy workloads on shared infrastructure with built-in security and role-based access control. Migration of workloads between different clouds and integration of existing management systems using customer extensions, APIs, and open cross-cloud standards serves as one of the most convincing arguments to use the same for a private cloud.

Despite these features and one of the most stable cloud platforms VMware vCloud might not be an ideal solution to be deployed by an academic institution owing to the high licensing costs attached to it, though it might prove ideal for an Enterprise with sufficiently good budget.

3 Baadal: Our Workflow Management Tool for Academic Requirements

Currently Baadal is based on KVM as the hypervisor and the Libvirt API which serves as a toolkit to interact with the virtualization capabilities. The choice of libvirt is guided by the fact that libvirt can work on variety of hypervisors including KVM, Xen, and VMWare. Thus, we can change the underlying hypervisor technology at any later stage with minimal efforts.

We export our management software in two layers namely Web based and Command-line interface (CLI). While our web based interface is built using web2py, a MVC based python framework, we continue to use python for the command line interface as well. The choice of the python as the primary language for the entire project is supported by the excellent support and documentation by libvirt community.

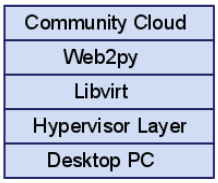

Fig. 1. Virtualization Stack.

3.1 Deconstructing Baadal

Baadal consists of four components:

Web Server : The web server provides a web-based interface for management of the virtual machines. Our implementation is based on web2py.

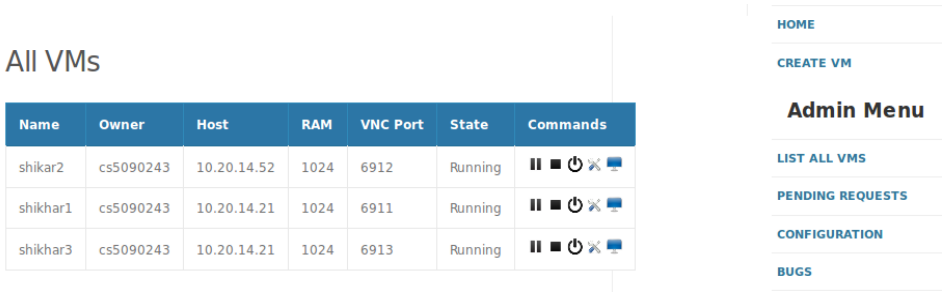

Fig. 2. List of VMs in Baadal's database along with their current status and some quick actions.

Hosts : Multiple hosts are configured and registered in the Baadal database using the web server interface. The hosts run the virtual machines and a common storage based on NAS provides seamless storage to allow live migration of VMs.

Clients : Any remote client which would access the virtual machines using remote desktops or for example say ssh connections.

VNC Server : The server receives requests from the clients regarding the VNC console access. IPtables have been setup for port forwarding so the requests that came to the server are forwarded to the appropriate hosts, and consequently served from there. The server can be same or different from the web server based on the traffic that is needed to be handled.

Table 1. Some tests performed on different kind of Hardware Infrastucture.

Test[1]	**KVM+Desktop**[2]	**KVM+Blade Server**[3]	**VMware+Blade Server**[4]
Empty Loop(10000000)	21840μs	44321μs	44553μs
Fork(1000000)	29.72 s	6.88 s	3.97 s
Wget(685.29 MB)	54.09 s	20.36 s	9.5 s
cp(685.29 MB)	71.97 s	11.65 s	26.07 s
iscp(685.29 MB)	29.64 s	52.34 s	4.75 s
oscp(685.29 MB)	73.54 s	83.68 s	4.86 s
Ping Hypervisor(5 packets)	.2886 s	.3712 s	.1204 s

Note: 1-All the Virtual Machines under test have 1GB RAM, 1 vCPU and 10 GB Harddisk.
2-VM with KVM as hypervisor running on Lab Desktop (4GB RAM, C2D, 500GB HDD, 1Gbps Network)
3-VM with KVM as hypervisor running on HP Proliant BL460c G7 (16GB RAM, 24 CPU, 10Gbps Network)
4-VM with VMware as hypervisor runningon Dell PowerEdge R710 (24GB RAM, 16 CPU, 10Gbps Network)

3.2 Workflow

Client requests a VM from Baadal using the web/command-line interface. The request once approved by administrator leads to spawning of a VM on any of the hosts. The host selected for spawning is determined by the Scheduling algorithm as described in the following section.

Once the VM has been setup the VM can be administered by the user which includes changing the run-level of the VM apart from normal operations like shutting down, rebooting the VM.

4 Implementation

While designing Baadal following have been implemented:

4.1 IPTables Setup

For accessing the graphical console of the VM users can use VNC console. Due to migrations of VMs the host of a VM may change and it can be troublesome if we provide a fixed combination of host IP address and port to for connecting to the VNC console. Baadal uses IP Tables and thus setup port forwarding connections to the VNC server. Clients can connect to the VNC console with the IP address of the VNC Server and a dedicated port which will be forwarded to the appropriate host which is currently hosting the VM of client. Incase of migration we change the port forwarding tables in background without causing any kind of inconvenience or delays to the user. So the user always connects to the VNC server with a fixed port number. The packets from user are forwarded by the VNC server to the appropriate host and all requests are served from there.

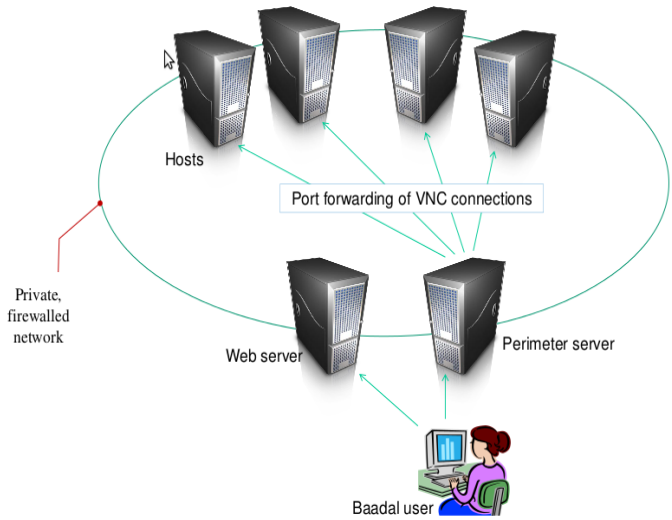

Fig. 3. Workflow of the working of Baadal.

4.2 Cost Model

We have been observing that in an academic environment some people tend to reserve VMs with high resources which are never used optimally by them. To reduce such number of occurrences we have implemented a cost model accounting for the usage case put up by the user (which can be dynamically changed by him) and the time the machine is running. We have defined three levels 1,2,3 with 1:a,1:b,1:c as the over-provisioning ratios respectively and have associated a decreasing order of cost with each of them. So if a user defines his run-level to be one then we will schedule his VM with an over-provisioning ratio a. So if k GB of ram is left then I can provision this VM if my requirements for the ram on the VM are less than k/a GB. The user is expecting to switch between different run-levels according to his requirement. The overall process is defined in a way leading to better utilization without any need for policing. Since, the run levels are associated with different cost factors users tend to follow the practice.

4.3 Scheduler

When the run-level for any VM is switched by the user we need to schedule his VM into an appropriate host. So we have designed and tested a scheduling algorithm which uses the greedy strategy for finding the host satisfying the given constraints (VM run-level and configuration of the hosts and the VM).

As a general observation it is hardly the case when all the VMs are used optimally used. The usage is reduced to a further extent during the off-peak hours when we can probably save on our costs and energy by trying to condense the number of hosts actually running and switching off the others. While doing

this proper care is taken so as to ensure that the VM doesn't see a degradation of the services when the migration is done.

5 Cost and Performance Comparisons

As both libvirt and KVM have undergone a rigorous testing phase before they are released as stable releases (which we are using), we need not do rigorous benchmark tests against the standard tests. We have subjected our scheduling algorithms to rigorous testing in an order to see if they are behaving as intended. The testing has also lead us to further optimization of the algorithms as we are sometimes introduced to some cases that we didn't take proper care of.

A second part of testing/experimentation involved in identifying the constants a,b and c so as to optimize the cost model. The constants may vary from institution to institution but generally tends to be closer to 1,2 and 3 respectively.

6 Future Work and Conclusions

6.1 Future Work

In a laboratory setup of any academic institution is genearlly observed to be as low as 1-10%. Thus quite a few of the resources goes underutilized. If we can run a community based cloud model on these underutilized community infrastructure we would be able to over-provision resources (like providing each student with its own VM), thereby improving the overall utilization of the physical infrastructure without compromising on the user's experience with the desktop. A significant rise as high as from 1-10% to 40-50% is expected in the utilization of the resources.

It is common in such environment for desktops to randomly be rebooted/switched-off/disconnected. Also, hardware/disk failure rates are higher in these settings, compared to tightly-controlled blade server environments. Being able to support VMs with a high degree of reliability is a challenge. The solution we intend to investigate is to run redundant copies of VMs simultaneously to provide much higher reliability guarantees, than what the physical infrastructure can provide. Doing this requires the ability to efficiently run multiple copies of identical VMs simultaneously and seamlessly switching between them. We at IIT Delhi have implemented Record/Replay feature in Linux/KVM (an open source Virtual Machine Monitor) which allows efficient synchronization of virtual machine images at runtime. We intend to use this implementation to provide higher reliability guarantees to cloud users on community infrastructure.

Currently, we support VMs that run atop the KVM hypervisor, but plan to add support for KVM/QEMU, VMware, and others in the near future. Also, we plan to optimize the software with storage specific plugins. For example, if one is using Netapp storage for his cloud he can take advantage of the highly optimized copy operation provided by Netapp rather than using the copy operation provided by an Operating system.

Due to the diversity in hardware characteristics and network topologies, we expect new challenges in performance measurements and load balancing in this scenario.

6.2 Conclusions

Baadal, our solution for private cloud for academic institutions, will allow administrators and researchers to deploy an infrastructure where users can spawn multiple instances of VMs and control them using a web-based or command line interface atop existing resources. The system is highly modular, with each module represented by a well-defined API, enabling researchers to replace components for experimentation with new cloud-computing solutions.

To summarize this work illustrates an important segment of cloud computing that has been filled by Baadal by providing a system that is easy to deploy atop existing resources, that lends itself to experimentation by the modularity that is inherent in the design of Baadal and the virtualization stack that is being used in the model.

References

1. Libvirt, the virtualization api. http://www.libvirt.org.
2. Massimo DiPierro. *Web2py Enterprise Web Framework, 2nd Ed.* Wiley Publishing, 2nd edition, 2009.
3. Fagg Gabriel, Edgar, Graham, Bosilca, George, Angskun, Thara, Dongarra, Jack, Squyres, Jeffrey, Sahay, Vishal, Kambadur, Prabhanjan, Barrett, Brian, Lumsdaine, Andrew, Castain, Ralph, Daniel, David, Graham, Richard, Woodall, and Timothy. Open mpi: Goals, concept, and design of a next generation mpi implementation. In Dieter Kranzlmller, Pter Kacsuk, and Jack Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 3241 of *Lecture Notes in Computer Science*, pages 353–377. Springer Berlin / Heidelberg, 2004.
4. Laor Kivity, Kamay, Lublin, and Liguori. kvm: the linux virtual machine monitor. *Virtualization Technology for Directed I/O. Intel Technology Journal*, 10:225–230, July 2007.