Critical Survey on Multideployment and Multisnapshotting on Clouds

S. Komal Kaur[#1], K J Sarma[#2], P Raja Prakasha Rao[#3]
[#]*Computer Science Department, JNT University*
TRR Engineering College, Hyderabad
[1]komalkaur13@gmail.com
[2]jskalavendi@gmail.com

Abstract— With Infrastructure-as-a-Service (IaaS) cloud economics getting increasingly complex and dynamic, resource costs can vary greatly over short periods of time. Therefore, a critical issue is the ability to deploy and snapshot, for that it require to boot and terminate VMs very quickly, which enables cloud users to exploit elasticity to find the optimal trade-off between the computational needs (number of resources, usage time)and budget constraints. This paper proposes a concept which reduces the time required to simultaneously boot a large number of VM instances on clouds from the same initial VM image (multi-deployment). Our proposal was not only to deploy large number of client system but also to snapshot the large number of Virtual images which is done concurrently. Large scale experiments under concurrency on hundreds of nodes show that introducing such a technique which improves infrastructure service by using or by sharing the resources.

Keywords— *virtual machine, virtual image, Multideployment, Multisnapshotting.*

I. INTRODUCTION

Cloud computing is the use of computing resources (hardware and software) that are delivered as a service over a network (typically the Internet). The name comes from the use of a cloud-shaped symbol as an abstraction for the complex infrastructure it contains in system diagrams. Cloud computing entrusts remote services with a user's data, software and computation.

1.1 There are many types of public cloud computing:[1]

- Infrastructure as a service (IaaS),
- Platform as a service (PaaS),
- Software as a service (SaaS)
- Storage as a service (STaaS)
- Security as a service (SECaaS)
- Data as a service (DaaS)
- Business process as a service (BPaaS)
- Test environment as a service (TEaaS)
- Desktop as a service (DaaS)
- API as a service (APIaaS)

End-user access cloud based applications through a web browser or a light-weight desktop or mobile app while the business and user's data are stored on servers at a remote location. Proponents claim that cloud computing allows enterprises to get their applications up and running faster, with improved manageability and less maintenance, and enables IT to more rapidly adjust resources to meet fluctuating and unpredictable business demand.[2][3].Cloud computing relies on sharing of resources to achieve coherence and economies of scale similar to a utility (like the electricity grid) over a network

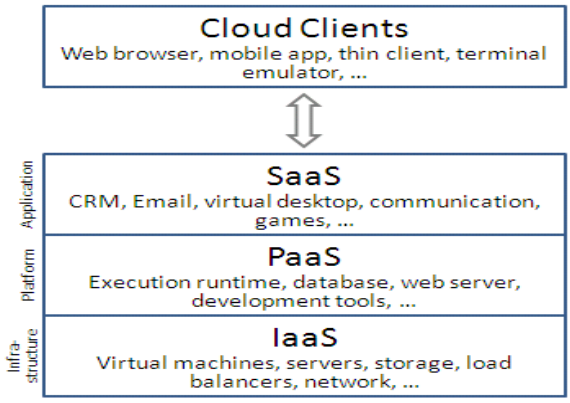


Fig1. Types of Cloud Computing

II. INFRASTRUCTURE AS A SERVICE

In this most basic cloud service model, cloud providers offer computers, as physical or more often as virtual machines, and other resources. The virtual machines are run as guests by a hypervisor, such as Xen or KVM. Management of pools of hypervisors by the cloud operational support system leads to the ability to scale to support a large number of virtual machines. Other resources in IaaS clouds include images in a virtual machine image library, raw (block) and file-based storage, firewalls, load balancers, IP addresses, virtual (VLANs), and software bundles.[4] IaaS cloud providers supply these resources on demand from their large pools installed in data centers. For wide area connectivity, the Internet can be used or—in carrier clouds -- dedicated virtual private networks can be configured.

- To deploy their applications, cloud users then install operating system images on the machines as well as their application software. In this model, it is the cloud user who is responsible for patching and maintaining the operating systems and application software. Cloud providers typically bill IaaS services on a utility computing basis, that is, cost will reflect the amount of resources allocated and consumed.
- IaaS refers not to a machine that does all the work, but simply to a facility given to businesses that offers users the leverage of extra storage space in servers and data centers.
- Examples of IaaS include: Amazon CloudFormation (and underlying services such as Amazon EC2), Rackspace Cloud, Google Compute Engine, and RightScale.

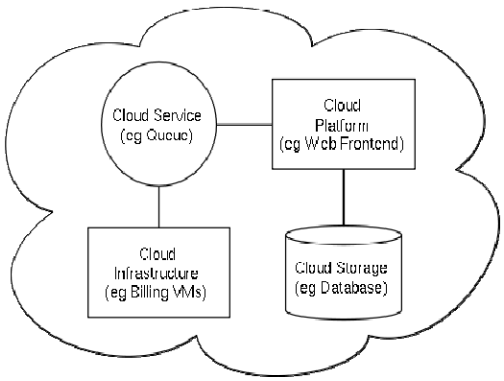


Fig .2 Cloud Infrastructure

ADVANTAGES:

More and more companies are moving from traditional servers to virtual servers in the cloud, and many new service-based deployments are starting in the cloud. However, despite the overwhelming popularity of the cloud here, deployments in the cloud look a lot like deployments on traditional servers. Companies are not changing their systems architecture to take advantage of some of the unique aspects of being in the cloud.

The key difference between remotely-hosted, virtualized, on-demand-by-API servers (the definition of the "cloud" for this post) and any other hardware-based deployment (e.g., dedicated, co-located, or not-on-demand-by-API virtualized servers) is that servers are software on the cloud.

APPLICATION STATE

The state of the VM deployment is defined at each moment in time by two main components: the state of each of the VM instances and the state of the communication channels between them. For VM instances that need large amounts of memory, the necessary storage space can explode to huge sizes. For example, saving 2 GB of RAM for 1,000 VMs consumes 2 TB of space, which is unacceptable for a single one point-in-time deployment checkpoint.Therefore, can further be simplified such that the VM state is represented only by the virtual disk attached to it ,which is used to store only minimal information about the state, such as configuration files that describe the environment and temporary files that were generated by the application. This information is then later used to reboot and reinitialize the software stack running inside the VM instance.

Such an approach has two important practical benefits:

(1) huge reductions in the size of the state, since the contents of RAM, CPU registers, and the like does not need to be saved; and

(2) portability, since the VM can be restored on another host without having to worry about restoring the state of hardware devices that are not supported or are incompatible between different hypervisors.

Since Model is the most widely used checkpointing mechanism in practice, we consider the multisnapshotting pattern.

APPLICATION ACCESS PATTERN

A VM typically does not access the whole initial image. For example, it may never access some applications and utilities that are installed by default with the operating system. In order to model this aspect, it is useful to analyze the life-cycle of a VM instance, which consists of three phases:

- Boot phase: involves reading configuration files and launching processes, which translates to random small reads and writes from/to the VM disk image acting as the initial state.
- Application phase: translates to either negligible virtual disk access (e.g., CPU-intensive applications that do not require persistent storage or data intensive applications that rely on dedicated storage services such as Amazon S3 [6]).
- Shutdown phase: generates negligible disk access to the image and is completely missing if the VM instance was terminated prematurely.

III. DESIGN MODEL

We rely on four key principles: aggregate the storage space, optimize VM disk access, reduce contention, and optimize multisnapshotting.

AGGREGATE THE STORAGE SPACE

We propose to aggregate the storage space from the compute nodes in a shared common pool that is managed in a distributed fashion, on top of which we build our virtual file system. This approach has two key advantages. First, it has a potential for high scalability, as a growing number of compute nodes automatically leads to a larger VM image repository, which is not the case if the repository is hosted by dedicated machines. Second, it frees a large amount of storage space and overhead related to VM management on dedicated storage nodes, which can improve performance and/or quality-of-service guarantees for specialized storage services that the applications running inside the VMs require and are often offered by the cloud provider (e.g., database engines, distributed hash tables, special purpose file systems, etc.).

OPTIMIZE VM DISK

When a new VM needs to be instantiated, the underlying VM image is presented to the hypervisor as a regular file accessible from the local disk. Read and write accesses to the file, however, are trapped and treated in a special fashion. A read that is issued on a fully or partially empty region in the file that has not been accessed before (by either a previous read or write) results in fetching the missing content remotely from the VM repository, mirroring it on the local disk and redirecting the read to the local copy. If the whole region is available locally, no remote read is performed. Writes, on the other hand, are always performed locally.

REDUCE CONTENTION BY STRIPING THE IMAGE

Each VM image is split into small, equal-sized chunks that are evenly distributed among the local disks participating in the shared pool. When a read accesses a region of the image that is not available locally, the chunks that hold this region are determined and transferred in parallel from the remote disks that are responsible for storing them. Under concurrency, this scheme effectively enables the distribution of the I/O workload, because accesses to different parts of the image are served by different disks.

OPTIMIZE MULTISNAPSHOTTING BY MEANS OF SHADOWING AND CLONING

Saving a full VM image for each VM is not feasible in the context of multisnapshotting. Since only small parts of the VMs are modified, this would mean massive unnecessary duplication of data, leading not only to an explosion of utilized storage space but also to an unacceptably high snapshotting time and network bandwidth utilization.

IV. ARCHITECTURE

The simplified architecture of a cloud that integrates our approach is depicted in Figure3.The typical elements found in the cloud are illustrated with a light background, while the elements that are part of our proposal are highlighted by a darker background. A distributed versioning storage service that supports cloning and shadowing is deployed on the compute nodes and consolidates parts of their local disks into a common storage pool. The cloud client has direct access to the storage service and is allowed to upload and download images from it. Every uploaded image is automatically striped. Furthermore, the cloud client interacts with the cloud middleware through a control API that enables a variety of management tasks, including deploying an image on a set of compute nodes, dynamically adding or removing compute nodes from that set, and snapshotting individual VM instances or the whole set.

The cloud middleware in turn coordinates the compute nodes to achieve the afore mentioned management tasks. Each compute node runs a hypervisor that is responsible for running the VMs. The reads and writes of the hypervisor are trapped by the mirroring module, which is responsible for on-demand mirroring and snapshotting and relies on both the local disk and the distributed versioning storage service to do so. The cloud middleware interacts directly with both the hypervisor, telling it when to start and stop VMs, and the mirroring module, telling it what image to mirror from the repository, when to create a new image clone (CLONE), and when to persistently store its local modifications (COMMIT).

Both CLONE and COMMIT are control primitives that result in the generation of a new, fully independent VM image that is globally accessible through the storage service and can be deployed on other compute nodes or manipulated by the client. A global snapshot of the whole application, which involves taking a snapshot of all VM instances in parallel, is performed in the following fashion. The first time the snapshot is taken, CLONE is broadcast to all mirroring modules,followed by COMMIT. Once a clone is created for each VM instance, subsequent global snapshots are performed by issuing each mirroring module a COMMIT to its corresponding clone.

ZOOM ON MIRRORING

One important aspect of on-demand mirroring is the decision of how much to read from the repository when data is unavailable locally, in such way as to obtain a good access performance.

A straightforward approach is to translate every read issued by the hypervisor in either a local or remote read, depending on whether the requested content is locally available. While this approach works, its performance is questionable. More specifically, many small remote read requests to the same chunk generate significant network traffic overhead (because of the extra networking information encapsulated with each request), as well as low throughput .

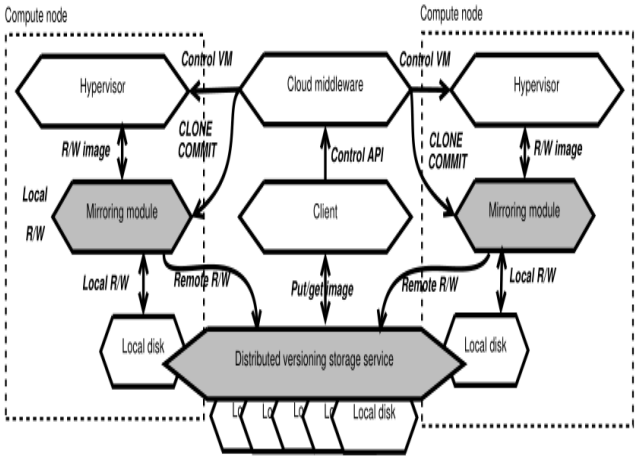
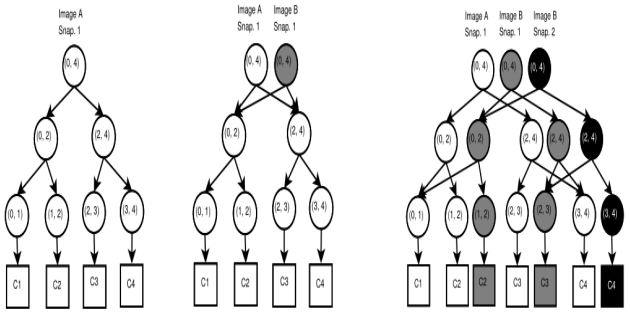


FIG 3. ARCHITECTURE OF A CLOUD

V. EVALUATION

PERFORMANCE OF MULTIDEPLOYMENT

The first series of experiments evaluates how well our approach performs under the multideployment pattern, when a single initial VM image is used to concurrently instantiate a large number of VM instances.



(a) Segment tree and chunk composition of the initial VM image A

(b) Segment trees and chunk composition after VM image A was cloned into VM image B using CLONE

(c) Segment trees and chunk composition after VM image B was snapshotted two times consecutively using COMMIT

Fig 4 : Cloning and Shadowing by means of Segment Trees

Prepropagation

It is the most common method used on clouds. It consists of two phases. In the first phase the VM image is broadcast to the local storage of all compute nodes that will run a VM instance. Once the VM image is available locally on all compute nodes, in the second phase all VMs are launched simultaneously. Since in this phase all content is available locally, no remote read access to the repository is necessary.

Qcow2 over PVFS

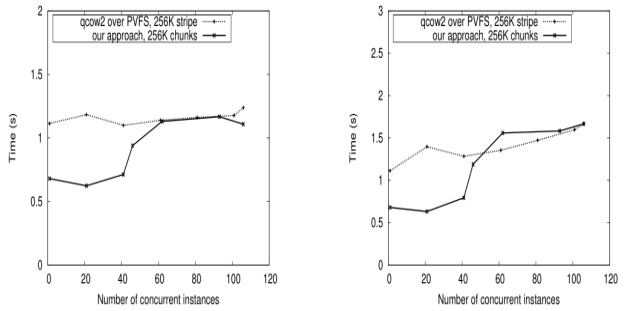
The second method we compare against is closer in concept to our own approach. We assume that the initial VM image is stored in a striped fashion on a distributed file system. We have chosen to use PVFS [9] to fill this role, as it is specifically geared to high performance and employs a distributed metadata management scheme that avoids any

potential bottlenecks due to metadata centralization. PVFS is deployed on all available compute nodes, as is our approach, and is responsible for aggregating their local storage space in a common pool. To instantiate a new set of VM instances on the compute nodes, in a first initialization phase we create a new qcow2 [12] copy-on-write image in the local file system of each compute node, using the initial raw 2 GB VM image stored in PVFS as the backing image.

MULTISNAPSHOTTING PERFORMANCE

This evaluates the performance of our approach in the context of the multisnapshotting access pattern. Since it is infeasible to copy back to the NFS server the whole set of full VM images that include the local modifications done by each VM instance, we limit the comparison of our approach with qcow2 over PVFS only.

The experimental setup is similar to the one used in the previous section: BlobSeer and PVFS are deployed on the compute nodes, and the initial 2 GB VM image is stored in a striped fashion on them, in chunks of 256 KB. The local modifications of each VM image are considered to be small, around 15 MB; this corresponds to the operating system and application writing configuration files and contextualizing the deployment, which simulates a setting with negligible disk access. In the case of qcow2 over PVFS, the snapshot is taken by concurrently copying the set of qcow2 files locally available on the compute nodes back to PVFS. In the case of our approach, the images are snapshotted in the following fashion: first a CLONE, followed by a COMMIT is broadcast to all compute nodes hosting the VMs. In both cases, the snapshotting process is synchronized to start at the same time.



(a) Average time snapshot an instance under concurrency (b) Completion time to snapshot all instances

Figure 5: Multisnapshotting: our approach compared with qcow2 images using PVFS as storage backend. Diff for each image is 15 MB.

The average time to snapshot per instance is depicted in Figure 5(a). As can be observed, both in our approach and qcow2 over PVFS, average snapshotting time increases almost imperceptibly at a very slow rate. The reason is that an increasing number of compute nodes will always have at least as many local disks available to distribute the I/O workload, greatly reducing write contention. Since BlobSeer uses an asynchronous write strategy that returns to the client before data was committed to disk, initially the average snapshotting time is much better, but it gradually degrades as more concurrent instances generate more write pressure that eventually has to be committed to disk. The performance level is closing to the same level as qcow2 over PVFS, which essentially is a parallel copy of the qcow2 files.

VI. CONCLUSIONS

As cloud computing becomes increasingly popular, efficient management of VM images, such as image propagation to compute nodes and image snapshotting for checkpointing or migration, is critical. The performance of these operations directly affects the usability of the benefits offered by cloud computing systems. This paper introduced several techniques that integrate with cloud middleware to efficiently handle two patterns: multideployment and multisnapshotting. We demonstrated the benefits of our approach through experiments on hundreds of nodes using benchmarks as well as real-life applications. Compared with simple approaches based on prepropagation, our approach shows a major improvement in both execution time and resource usage: the total time to perform a multideployment was reduced by up to a factor of 25, while the storage and bandwidth usage was reduced by as much as 90%.

ACKNOWLEDGMENT

The experiments presented in this paper were carried out just a survey on how the cloud computing can be able to deploy large number of virtual machines simultaneously and also snapsnot of all the deployment even concurrently, using the amazon.org.

REFERENCES

[1] Amazon elastic block storage (ebs).http://aws.amazon.com/ebs/.
[2] File system in userspace (fuse).http://fuse.sourceforge.net.
[3] Nimbus. http://www.nimbusproject.org/.
[4] Opennebula. http://www.opennebula.org/.
[5] Amazon Elastic Compute Cloud (EC2).http://aws.amazon.com/ec2/.
[6] Amazon Simple Storage Service (S3).http://aws.amazon.com/s3/.
[7] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia..
[8] A. Bar-Noy and S. Kipnis. Designing broadcasting algorithms in the postal model for message-passing systems. In SPAA '92: Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures, pages 13–22, New York, 1992. ACM.
[9] P. H. Carns, W. B. Ligon, R. B. Ross, and R. Thakur. Pvfs: A parallel file system for Linux clusters. In Proceedings of the 4th Annual Linux Showcase and Conference, pages 317–327, Atlanta, GA, 2000. USENIX Association.Symposium on Operating Systems Principles, pages 205–220, New York, 2007. ACM.
[10] B. Claudel, G. Huard, and O. Richard. Taktuk, adaptive deployment of remote executions. In HPDC '09: Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing, pages 91–100, New York, 2009. ACM.
[11] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. In SOSP '07: Proceedings of 21st ACM SIGOPS
[12] M. Gagné. Cooking with Linux—still searching for the ultimate Linux distro? Linux J., 2007(161):9, 2007.
[13] J. G. Hansen and E. Jul. Scalable virtual machine storage using local disks. SIGOPS Oper. Syst. Rev.
[14] M. Hibler, L. Stoller, J. Lepreau, R. Ricci, and C. arb. Fast, scalable disk imaging with Frisbee. In ATC '03: Proceedings of the 2003 USENIX Annual Technical Conference, pages 283–296, San Antonio, TX, 2003.
[15] Y. Jégou, S. Lantéri, J. Leduc, M. Noredine, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, and T. Iréa. Grid'5000: A large scale and highly reconfigurable experimental grid testbed. International Journal of High Performance Computing Applications, 20(4):481–494, November 2006.