Open access Journal International journal of Emerging Trends in Science and Technology

Multideployment and Multisnapshotting on IAAS Cloud Environments

Authors

D. Amulya[1], Dr.N.Chandra Sekhar Reddy[2], B. Srinivas Reddy[3]

[1]Student,M.Tech CSE Dept,Institute of Aeronautical Engineering, Hyderabad-500043, Telangana, India.

Email: *amulya292@gmail.com*

[2]Professor, CSE Dept Institute of Aeronautical Engineering, Hyderabad -500043, Telangana, India
Email: *naguchinni@gmail.com*

[3]Associate.Professor, H&BS Dept. Institute of Aeronautical Engineering, Hyderabad -500043,Telangana, India.
Email: *bsreddy5555@gmail.com*

ABSTRACT

Infrastructure as a Service (IaaS) cloud computing has revolutionized the way we think of acquiring resources by introducing a simple change: allowing users to lease computational resources from the cloud provider's datacenter for a short time by deploying virtual machines (VMs) on these resources. This new model raises new challenges is the need to deploy a large number (hundreds or even thousands) of VM instances simultaneously. Once the VM instances are deployed, another challenge is to simultaneously take a snapshot of many images and transfer them to persistent storage to support management tasks, such as suspend-resume and migration. With datacenters growing rapidly and configurations becoming heterogeneous, it is important to enable efficient concurrent deployment and snapshotting that are at the same time hypervisor independent and ensure a maximum compatibility with different configurations. This paper addresses these challenges by proposing a virtual file system specifically optimized for virtual machine image storage. It is based on a lazy transfer scheme coupled with object versioning that handles snapshotting transparently in a hypervisor- independent fashion, ensuring high portability for different configurations.

Keywords: *Cloud computing, Infrastructure as a Service, Multideployment, Multisnapshotting, Virtual machine.*

1. INTRODUCTION

Cloud computing can be viewed as a model for distributing information technology. In order to gain access to the resources from Internet without depending on direct connection with the server, the model can easily retrieve resources via web-based tools and applications. Here, the information which is to be accessed is stored in clouds and it gives the privileged to the user to access the information whenever every they want. Thereby, allowing the users to work remotely. In general cloud computing resources such as hardware and software which are distributed as a service across the network. It centralizes the data storage, processing and bandwidth which in turn provides efficient computing process to the users. Cloud computing entrusts remote services with a user's data, software and computation.

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 [1]. Cloud computing relies on sharing of resources to achieve coherence and economics of scale similar to a utility (like the electricity grid) over a network.

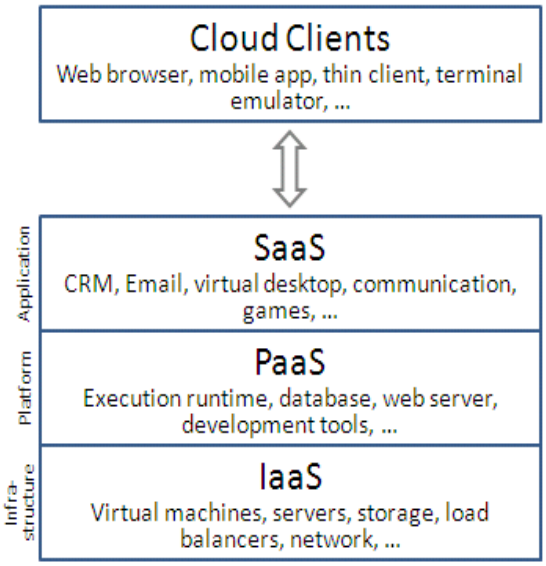


Figure1.Cloud computing services

2. INFRASTRUCTURE AS A SERVICE

In this most basic 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 hypersior, 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. IaaS cloud providers supply these resources on demand from their large pools installed in datacenters. 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 business that offers users the leverage of extra storage space in servers and datacenters.
- Examples of IaaS include: Amazon CloudFormation (and underlying services such as Amazon Ec2), Rackspace Cloud, Google Compute Engine, and Right Scale.

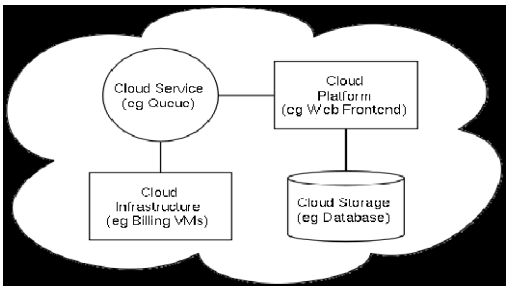


Figure2. Cloud infrastructure

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 2GB of RAM for 1000 VMs consumes 2TB 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 as 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 state: 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).
- Shutdown phase: Generates negligible disk access to the image and is completely missing if the VM instance was terminated prematurely.(e.g., because of hardware failure).

Objectives:

Main aim of this paper is:

1.With IaaS, users can lease storage and computation time from large datacenters. Leasing of computation time is accomplished by allowing users to deploy virtual machines(VMs) on the datacenter's resources. Since the user has complete control over the configuration of the VMs using on-demand deployments.

2. One of the commonly occurring patterns in the operation of IaaS is the need to deploy a large number of VMs on many nodes of a datacenter at the same time, starting from a set of VM images previously stored in a persistent fashion.

3. To optimize the multisnapshotting**.**

3. RELATED WORK

Multideployment that relies on full broadcast-based prepropagation is a widely used technique. While this technique avoids read contention to the repository, it can incur a high overhead in both network traffic and execution time. Furthermore, since the VM images are fully copied locally on the compute nodes, multisnapshotting becomes infeasible: large amounts of data are unnecessarily duplicated and cause unacceptable transfer delays, not to mention huge storage space and network traffic utilization[1].

Closer to our approach is Lithium, a fork-consistent replication system for virtual disks. Lithium supports instant volume creation with lazy space allocation and instant creation of writable snapshots. Unlike our approach, which is based on segment trees, Lithium is based on log structuring[5], which can potentially degrade read performance when increasing the number of consecutive snapshots for the same image: the log of incremental differences starts growing, making it more expensive to reconstruct the image. Cluster volume managers for virtual disks such as Parallax enable compute nodes to share access to a single, globally visible block device, which is collaboratively managed to present individual virtual disk images to the VMs. While this enables efficient frequent snapshotting, unlike our approach, sharing of images is intentionally not supported in order to eliminate the need for a distributed lock manager, which is claimed to dramatically simplify the design. Several storage systems, such as Amazon S3(backed dynamo), have been specifically designed as highly available key- value repositories for cloud infrastructures. They can be valuable building blocks for block-level storage volumes that host virtual machine images; however, they are not optimized for snapshotting[4].

Our approach is intented to complement existing cloud computing platforms, both from industry (Amazon Elastic Compute Cloud: EC2) and from academia (Nimbus , OpenNebula[10]). While the details for EC2 are not publicly available, it is widely acknowledged that all these platforms rely on several of the techniques presented above. Claims to instantiate multiple VMs in "minutes", however, are insufficient for meeting our performance objectives; hence, we believe our work is a welcome addition in this context.

4. EXISTING SYSTEM

In our existing cloud computing platforms, both from industry (Amazon Elastic Compute Cloud) and from academia(Nimbus[9]). While the details for EC2 are not publicly available, it is widely acknowledged that all these platforms rely on several of the techniques presented above. Claims to instantiate multiple VMs in "minutes", however, are insufficient for meeting our performance objectives; hence, we believe our work is a welcome addition in this context. In addition to incurring significant delays and raising manageability issues, these patterns may also generate high network traffic that interferes with the execution of applications on leased resources and generates high utilization costs for the user.

5. PROPOSED SYSTEM

This paper proposes a distributed virtual file system specifically optimized for both the multideployment and multisnapshotting patterns. Since the patterns are complementary, we investigate them in conjunction. Our proposal offers a good balance between performance, storage

space, and network traffic consumption, while handling snapshotting transparently and exposing standalone, raw image files(understood by most hypervisors) to the outside.

Our contributions are can be summarized as follows:

- We introduce a series of design principles that optimize multideployment and multisnapshotting patterns and describe how our design can be integrated with IaaS infrastructures.
- We show how to realize these design principles by building a virtual file system[4] that leverages versioning-based distributed storage services.
- We evaluate our approach in a series of experiments, each conducted on hundreds of nodes provisioned on the Grid'5000 test bed, using both synthetic traces and real-life applications.
- We propose a solution that addresses these three requirements by leveraging two features proposed by versioning systems: shadowing and cloning. Shadowing means to offer the illusion of creating a new standalone snapshot of the object for each update to it but to physically store only the differences and manipulate metadata in such way that the illusion is upheld. This effectively means that from the user's point of view, each snapshot is a first-class object that can be accessed independently.

6. ARCHITECTURE

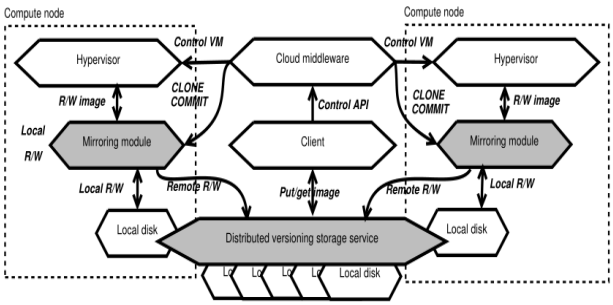


Figure3: Cloud architecture that integrates our approach(dark background)

7. 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 locally available on the compute nodes

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 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 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 and 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. While splitting the image into chunks reduces contention, the effectiveness of this approach depends on the chunk size and is subject to a trade-off. A chunk that is too large may lead to false sharing; that is, many small concurrent reads on different regions in the image might fall inside the same chunk, which leads to a bottleneck. A chunk that is too small, on the other hand, implies a higher access overhead, both because of higher network overhead, resulting from having to perform small data transfers, and because of higher metadata access overhead, resulting from having to manage more chunks.

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.

We propose a solution that addresses these three requirements by leveraging two features proposed by versioning systems: shadowing and cloning[3]. Shadowing means to the illusion of creating a new standalone snapshot of the object for each update to it but to physically store only the differences and manipulate metadata in such way that the illusion is upheld. This effectively means that from the user's point of view, each snapshot is a first-class object that can be accessed independently. For example, let's assume a small part of a large file needs to be updated. With shadowing, the user sees the effect of the update as a second file that is identical to the original expect for the updated part. Cloning means to duplicate an object in such way that it looks like a standalone copy that can evolve in a different direction from the original but physically shares all initial content with the original.

Zoom on mirroring

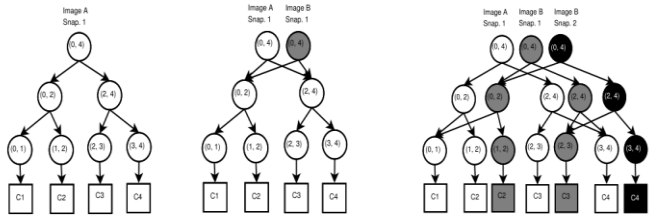
One important aspect of on-demand mirroring is the decision of how much to read from the repository when data is unavailable locally, in such way as to obtain a good access performance. A straightforward approach is to translate every read issued by the hypervisor in either a local or remote read, depending 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 track overhead(because of the latencies of the requests that add up). Moreover, in the case of many scattered small writes, a lot of small fragments need to be accounted for, in order to remember what is available locally for reading and what is not. Fragmentation is costly in this case and incurs a significant management overhead, negatively impacting access performance. For this reason, we propose two strategies that aim to limit the negative impact of small reads and writes.

First, a read operation on a region that is not fully available locally triggers remote reads that fetch the full minimal set of chunks that cover the requested region. While this leads to more network track than is strictly required, it improves the performance of correlated reads(i.e., a read on one region that is followed by a read "in the neighborhood") at a minimal cost when using chunk sizes. The second strategy we propose limits fragmentation by forcing a single contiguous region to be mirrored locally for each chunk. More specifically, a second write that falls on the same chunk as a previous write such that the gap between them is not available locally will trigger a remote read that will fill the gap. With this approach only the limits of a single contiguous region need to be maintained for each chunk, which places an upper limit on fragmentation overwritten to.

8. 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 concurrency 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

Figure4. Cloning and Shadowing by means of segment trees

Prepropagation

Prepropagation 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 contents 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 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 instance on the compute nodes, in a first initialization phase we create a new qcow2[2] copy-on-write image in the local file system of each compute nodes, using the initial raw 2GB 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[4] and PVFS are deployed on the

compute nodes, and initial 2GB VM image stored in a striped fashion on them, in chunks of 256KB. The local modifications of each VM image are considered to be small, around 15MB; 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, snapshot is taken by concurrently copying the set of qcow2 files locally available on the compute nodes back to PVFS. In 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.

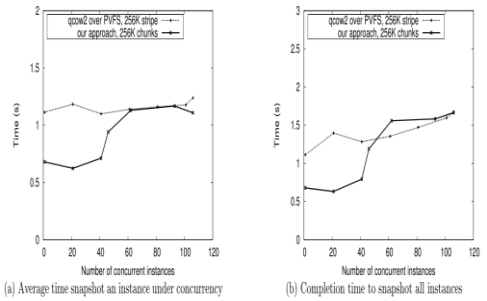


Figure 5: Multisnapshotting: our approach compared with qcow2 images using PVFS as storage backend. Diff for each image is 15 MB.

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 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.

9. CONCLUSION

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%.

10. FUTURE ENHANCEMENTS

The propose a lazy VM deployment scheme that fetches VM image content as need by the application executing in the VM, thus reducing the pressure on the VM storage service for heavily concurrent deployment requests. Furthermore, we leverage object versioning to save only local VM image differences back to persistent storage when a snapshot is created, yet provide the illusion that the snapshot is a different, fully independent image. This has two important benefits. First, it handles the management of updates independently of the hypervisor, thus greatly improving the portability of VM images and compensating for the lack of VM image format standardization. Second, it handles snapshotting transparently at the level of the VM image repository, greatly simplifying the management of snapshots. With respect to multisnapshotting, interesting reductions in time and storage space can be obtained by introducing reduplication schemes. We also plan to fully integrate the current work with Nimbus and explore its benefits for more complex HPC applications in the real world.

ACKNOWLEDGEMENT

We thank our H.O.D Prof. Dr. N. Chandra Sekhar Reddy for giving us the eminent facilities to perform my project work. I am obliged to of CSE department, IARE for their timely help and support.

11. REFERENCES

1. M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. Commun. ACM, 53:50–58, April 2010.
2. 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.
3. 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.

4. 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.
5. 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 Symposium on Operating Systems Principles, pages 205–220, New York, 2007. ACM.
6. M. Gagn´e. Cooking with Linux—still searching for the ultimate Linux distro? Linux J., 2007(161):9, 2007.
7. J. G. Hansen and E. Jul. Scalable virtual machine storage using local disks. SIGOPS Oper. Syst. Rev., 44:71–79, December 2010.
8. M. Hibler, L. Stoller, J. Lepreau, R. Ricci, and C. Barb. Fast, scalable disk imaging with Frisbee. In ATC ’03: Proceedings of the 2003 USENIX Annual Technical Conference, pages 283–296, San Antonio, TX, 2003.
9. Y. J´egou, S. Lant´eri, J. Leduc, M. Noredine, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, and T. Ir´ea. Grid’5000: A large scale and highly reconfigurable experimental grid testbed. International Journal of High Performance Computing Applications, 20(4):481–494, November 2006.
10. K. Keahey and T. Freeman. Science clouds: Early experiences in cloud computing for scientific applications. In CCA’08: Proceedings of the 1st Conference on Cloud Computing and Its Applications, 2008.