International Journal of Advanced Information Science and Technology (IJAIST) ISSN: 2319:2682
Vol.1, No.7, November 2012 DOI:10.15693/ijaist/2012.v1i7.54-57

Efficient Multideployment And Multisnapshotting On Clouds

P.JAGALINGAM, P.SARAVANAN and T.B DHARMARAJ
Dept of Computer Science Engg, Christ the king engineering college, karamadai-635109, Tamilnadu, India.

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 the re – sources. This new model raises new challenges in the design and development of IaaS middleware. One of those 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 datacentres 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 project 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: *virtual Machines, Infrastructure as a service, hypervisor .*

Introduction:

This new model raises new challenges in the design and development of IaaS middleware. One of those 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 concurrent deployment and snapshotting that are at the same time hypervisor independent and ensure a maximum compatibility with different configurations.

In this system we are planning to use three servers, one server will act as a gateway server or dispatcher server and other two servers are going to act as computational servers where the VM (virtual machine) is nothing but the mirror image of any application which is going to deploy in cloud servers (Armbrust et al., 2010).. A client machine can able to access any VM through Gateway server only. In Gateway server will have the management module which will decides which server has to respond for the client request based on load balance calculations.

In this work the underlying infrastructure is represented by a large-scale cloud data center comprising heterogeneous physical nodes which are nothing but cloud servers. Each node has a CPU, which can be multicore, with performance defined in Millions Instructions per second (MIPS). Besides that, a node is characterized by the amount of RAM and network bandwidth (Bar-Noy et al., 1992). Users submit requests for provisioning of *m* heterogeneous VMs with resource requirements defined in MIPS, amount of RAM and network bandwidth. SLA violation occurs when a VM cannot get the requested amount of resource, which may happen due to VM consolidation. The software system architecture is tiered comprising a dispatcher, global and local managers. The entire client request will be received by gateway server (i.e.) cloud middleware which will interact with hypervisor and take the decision which server has to respond.

Modules description:

The major modules in our project are,

1. Designing the Virtual File System
2. Optimize VM and Reduce contention
3. Optimize multisnapshotting

Designing the Virtual File System

We propose to aggregate the storage space from the compute nodes in a shared common pool that is managed in a distribution fashion, on top of which we build our virtual file system. This approach has 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 (B.Claudel et al., 2009) 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 system, etc.)

Optimize VM and Reduce contention

In this module we discuss 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 (O.Richard et al., 2009) . If the whole region is available locally, no remote read is performed. Writes, on the other hand, are always performed locally. 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 size and is subject to a trade-off (G.DeCandia et al., 2007) 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.

Optimize Multisnapshotting

We propose a solution that addresses these requirements by leveraging two features proposed by versioning system: 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 first class object that can be accessed independently. We propose to deploy a distributed versioning system that efficiently supports shadowing and cloning, while consolidating the storage space of the local disks into a shared common pool. With this approach, snapshotting can be easily performed in the following fashion.

Store only the incremental differences between snapshots.

Consolidate each snapshot as a standalone entity.

Present a simple raw image format to the hypervisors to maximize migration portability.

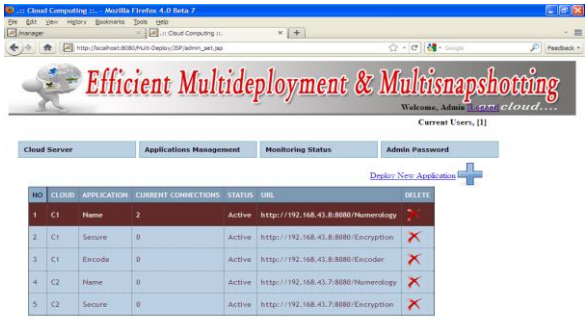
Results and Discussion:

IMPLEMENTATION

In our project we have implemented our approach in the cloud by means of two basic building blocks: a distributed versioning storage service, which supports cloning and shadowing and is responsible for managing the repository, and a mirroring module, which runs on each compute node and is responsible for trapping the I/O accesses of the hypervisor to the image with the purpose of facilitating on – demand mirroring and snapshotting.

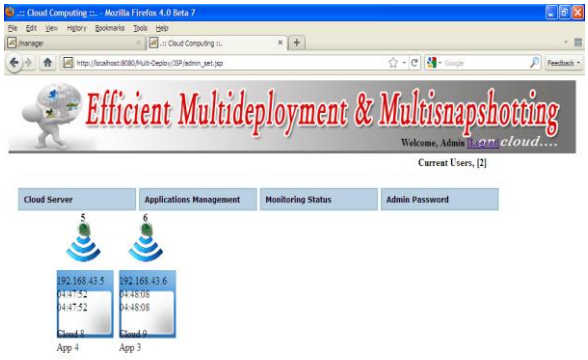
Cloud Server Creation:

Application creation:

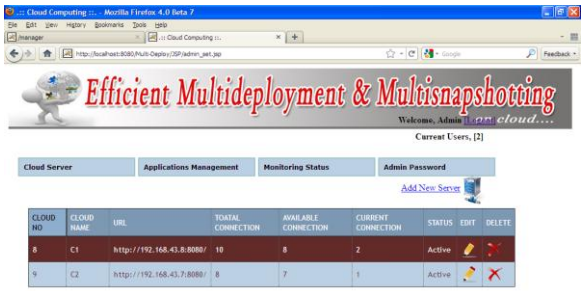


Application Deployment:

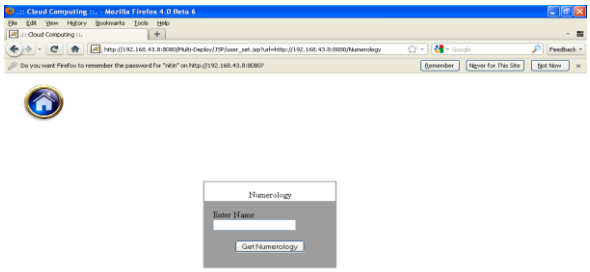
Monitoring Status:



ResourcesAllocation:



Accessing Application:



Monitoring Application:

DESCRIPTON

Admin user who is the super user who will maintain the cloud server configuration details and application deployment details in the cloud. In cloud server session, admin can add the total connections in cloud server. Admin can view the cloud connection details and can edit, delete the clouds. In application Management, he can add the applications which we

are going to deploy in the cloud. He can view, edit and update the application details. In Application deployment, he can deploy the applications in particular cloud which we added already. Admin can view and delete the deployment details. In monitoring status, if any user is using the applications, it will show the cloud details and users system detail in active connections session. Monitoring server shows the different cloud details i.e. how many applications are deployed in particular clouds, how many applications are running in each clouds, how many users are currently using the applications in particular cloud, total connections in each cloud and available connections in each clouds.

Conclusions:

We propose a lazy VM deployment scheme that fetches VM image content as needed 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. 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 us-age was reduced by as much as 90%. Compared with approaches that use copy-on-write images (i.e., qcow2) based on raw backing images stored in a distributed file system (i.e., PVFS), we show a speedup of multideployment by a factor of 2 and comparable multisnapshotting performance, all with the added benefits of transparency and portability.

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. 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. 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