The design and implementation of Horizon desktop service cross-cloud

Abstract

As public cloud prevails, customers tend to operate their applications on multiple clouds. How to simplify management complexity, guarantee security, reduce cost and encourage resource sharing across clouds presents a main challenge. To address this issue, VMware promotes the cross-cloud architecture which aims to provide a common operating environment to manage various clouds. [1][2]

Inspired by this vision, we’d like to build a large scale and reliable Horizon desktop service cross-cloud, providing fluent user experience to large number of users with lower cost. To achieve this goal, we divide the service into two parts, the desktop service and desktop manager service, according to their different characteristics. The desktop manager is actually a http server, which handles http requests from users, manages the status of desktop using database, and replies. We run the http server in docker container, and employ container orchestration tool, docker swarm, to manage the containers across clouds as a cluster. Docker swarm provides functionalities like load balance, scale up, scale down, replication and fault tolerance. Based on these functionalities, we design strategy to migrate service and data across clouds according to some policy to find the optimal place to run the service. [3]

As to the desktop service, we maintain a floating desktop pool which works as a desktop cache. These desktops may reside in different clouds according to some metrics, for example, performance/cost ratio, data security level. User gets a desktop from the cache.   
When user logoff a desktop, it will be freed in order to save cost.

Experiments show that how our migration strategy successfully reduces overall operating cost at the same time maintains the quality of service to users.

The nature of the connection server

End users send HTTP requests to connection server. These requests may be categorized into

1. Authentication
2. Get all services (desktop or app) entitled
3. Request access to any particular service
4. leave a service

To end user, the connection server works as a HTTP server, like Nginx or Node.js. It receives http requests from user, query or modify the database, then reply with a HTTP response. The database records user’s accounts and the services entitled.

Users tend to be the move. They would like to use the service through mobile device during very short fragmented time and they would like the server to respond in less than a second.   
  
Users also expect high availability of the service. They would like to access their desktops anytime, anywhere, on any devices.

All the above quality shall be reached without compromise of security since user may store some critical data through the desktop.

The key challenge is that:

1. extremely fast response
2. Cross cloud

Different clouds may vary in network performance, cost, and geography. Latency-sensitive users may prefer a cloud that provides better network performance; Cost-sensitive users may choose clouds that charge less.

~~The overall performance data…...~~

~~(# of Independent requests in parallel , user data size, Network latency) 🡪 response time~~

1. ~~Scalability~~

~~In on premise setup, the number of users online simultaneously maybe tens of thousands. In public cloud, we expect the number to be ten-fold.~~

~~We’d like to add new servers into the system and the quantity of servers shall be O(N) where N is the number of the users. And the response time is O(logN) to the number of users.~~

~~Besides scale up, the system shall scale down when requests number reduces in order to save cost in public cloud~~

1. ~~Fault tolerance and reliability~~

~~The connection server is the entry point to desktop service. If it fails, end users shall have no access to any desktops. As our server may run on various clouds which provide different SLA, it is of high probability that parts of the servers may fail at any particular time. It is crucial that the whole system still works with acceptable quality when some part of the servers fail.~~



In the above picture, a desktop manager, DM for short, manages desktop status and its connection with users. DM runs in container. Several containers may run in one host. Hosts may spread among clouds. Each host is a docker host, managed as a cluster using tools like docker swarm.

The DM master is also a docker host, but works as the cluster master, and other docker hosts work as slave. The DM master exposes the service from public cloud to users. The DM master decides the number of DM containers running in each slave according to some polices, e.g. round robin, node cpu load. It also works a load balancer. It receives requests from user, picks the slave to serve according to some polices, and forwards the reply.

Several DM may access the distributed data store simultaneously.

*~~When the service load is high, new host or service may be added into the system and put into use in seconds. When load is low, abundant service or host will be removed in order to save cost.~~*

~~Then Will this architecture meet the system’s requirement? We will focus on its performance issue in the first place. The questions:~~

1. ~~What’s the metrics based on which load balancer decides to scale up/down the system?  
   Q？ Load balancer应该用docker swarm提供的？？？还是应该用别的？？~~

~~广泛阅读主要的load balancer的方法。再看看docker swarm采用的是什么方法？~~

1. ~~Which cloud to choose when scale up/down the system?~~
2. ~~How distributed data store works if requests and data bursts?~~

~~Will the load balancer itself be the bottleneck if requests burst?  
你的研究重点是实现cross cloud的负载迁移。而不是实现一个特别好的load balancer. Load balancer这东西，够用就可以了。~~

~~The distributed broker service~~

~~Service and data migration between clouds~~

~~Manual migration without interrupt of service. For some reasons, users may want to migrate service and its data from cloud1 to cloud2 without interrupt of the service.~~

~~Service auto migration according to some metrics (performance/price ratio, or other metrics defined by user). This is similar as process migration between CPUs~~

~~Service auto scaling. Which cloud to choose to scale the service?~~

~~Private cloud + public cloud. Public cloud has unlimited resource. Private cloud has limited resource but lower cost. This may decide how to do auto scale~~

~~Public cloud + public cloud~~

~~迁移的算法   
迁移的时机。 看看linux kernel是如何进行进程迁移的。。。~~

~~迁移的动作：~~

~~数据库如何迁移？？？~~

~~计算部分的迁移，直接交付给docker swarm….~~

The distributed desktop service

We will maintain a floating desktop pool which works as a desktop cache. These desktops may reside in various clouds according to some metrics, like performance/cost ratio.

Once requested, a desktop will be chosen from the cache. Then new desktops will be created from some cloud vendor according to some metrics and filled into the cache in background. When user logoff one desktop, the desktop may be freed in order to save cost. Note, the desktop here is non-persistent. Data volumes shall be mounted once user login. However, this is not in our topic here.

~~Affinity between desktop controller and desktop~~

~~The controller manages the desktop state and authentication information. For efficient communication, the controller and the desktop shall be in the same cloud.~~

~~To achieve this, the load balancer shall choose a fixed controller service to serve a specific user. Then the controller will choose a desktop in the same cloud as itself from the desktop cache.~~

**The desktop allocator service**



Each time end user logins a desktop, it will trigger DM to allocate a desktop. This activity is on the critical path for user to use the desktop service. So the latency shall be as small as possible.

Also, there may be thousands of users from different places sending the request simultaneously, so the allocator shall keep the consistency and be highly scalable.

Keep the above requirements in mind, the system is acting as follows. DM requests a desktop from the desktop allocator. The desktop allocator runs in a centralized server in order to keep consistency. It allocates desktops from various clouds according some policies. These desktops in different clouds form a floating desktop pool.

**Reserved desktop pool**

It usually takes tens of seconds to several minutes to allocate a new desktop from cloud depending on the cloud infrastructure. To make it faster, desktop allocator will allocate some number of free desktops in advance and keep them in the reserved desktop pool.

Then to allocate a desktop, the allocator just picks a desktop from this pool and returns it to DM.

Pool data structure

It takes around 50 bytes to describe one desktop in the pool. We assume there are at most 100,000 desktops in the reserved pool. So the total size is around 5MB. This size is quite moderate to be saved in RAM.

We use redis to manipulate the data base

Data persistence

The pool data is critical data for our system. So it will also be saved in a SSD disk in case the server fails

Algorithm to fill the pool

A thread keeps monitoring the number of the free desktops in the pool. Once the number is below some threshold, the thread will send requests to various cloud for new desktops and fill the pool once requests return successfully.

The admin can choose the policy about the ratio of desktops created from a cloud. By default, the algorithm tends to create more desktops on cloud that provides the best performance/cost. By adjusting the parameters here, the distribution of the desktops tends to achieve the best performance/cost or some other goals set by the admin.

**DM Free desktop cache**

Request desktop in batch

It is very frequent for DM to request a new desktop. It will be highly inefficiently if it sends each request to the allocator. To reduce the latency, the DM will request the desktop in batch in advance. And the free desktop will firstly save in the cache on DM. After cache is setup, most requests for desktop shall be satisfied in the cache.

The desktop allocator will tag the resource status as cached. Once user sends get-desktop-connection request, the allocator will mark the resource status as busy and move it into the busy desktop pool

Cache timeout and invalidation

Some DM may fail or be killed with some desktop cache still available on hands. To prevent resource leak, DM desktop cache will be invalidated if it is free for more than N minutes which is configurable. The resource will be reclaimed and become available again at the desktop allocator after another N minutes.

**Desktop entitlement service**

[1] Pat Gelsinger Introduces VMware’s Cross-Cloud Architecture <https://www.youtube.com/watch?v=owThAMzhMKE>

[2] The VMware Cross-Cloud Architecture <https://www.youtube.com/watch?v=9-OHKRnx8uU>

[3] VMworld Day 1: VMware Cross-Cloud Architecture Enables Organizations to Thrive in the Cloud Era http://www.vmware.com/radius/vmworld-2016-chris-wolf-recap-day-one/

Prototype time schedule

1.

Setup the system 11th to 25th Nov

* 11th Nov to 18th Nov  
   setup the network between private cloud and public cloud   
   use docker swarm to control them  
   run the manager ok in container  
   use docker swarm to manage the service across the hosts   
   Modify the manager so that it don’t contact VADC

Write a client that sent http request to the docker swarm load balancer…..

* 21st Nov to 25th Nov   
  Set up the distributed data base across clouds

… Choose the best data base software that meets our requirement  
… Modify the desktop manager to write the data into the distributed data base

The floating desktop pool and the data base…

As to floating desktop pool, there is a consistent view to the floating desktop pool across desktop managers. Each time, user login, allocate one from the pool…. And create a new one to the pool… the desktop IP address may change. So you also need to modify this mapping in the data base…. so there may be a lot of write/read simultaneously………….

2. The migration of computing resource between clouds 28th Nov to 2nd Dec   
 migrate the desktop manager only  
  
3. Migrate data base between clouds 5th Dec to 9th Dec

4. Design rebalance algorithm to do auto migration 12th Dec to 16th Dec

5. Performance test 19th Dec to 30th Dec

Write paper……….

Zhibin的讨论

跨云的必要性？ ：

customer 可能经常会用到几个云。

这样他们可能会使用几套云的管理平台。 而且云之间的workload不是互相迁移的，数据共享也不容易。

所以我们用跨云的设计，统一了桌面云的管理，实现了数据共享，和workload 的迁移。

同时，可以综合利用各种云的特性。例如某一个云降价了，就可以将更多的负载，迁移过去。

资源抽象层的意义？

这个东西应该具有很大的产品和工程意义。但是不一定方便写论文。

SDDC 已经有了跨云的设计，为什么不基于它来做？

我们的跨云桌面独立于vshpere，使用范围更广了。

Abstract

Cross cloud is important…

Cross cloud is also important for horizon

The key problem: workload/data migration, the latency.

The issue can’t be fixed in traditional on premise horizon. So we redesign the whole system.

Evaluation of the design

Introduction

Similar as above….  
 We support multiple clouds, including public cloud and private cloud.

We can migrate workload between them

The latency is low

Horizon desktop overview

Mainly about the protocol…. What’s the critical path

Horizon cross cloud design

Network connectivity

The desktop manager service

The distributed data base

Floating desktop pool

Evaluation

Migrate the desktop manager from one cloud to another cloud in seconds

The latency is accessing the data base and the desktop service.

Future work

conclusion