Study on a novel migration scheme for seamless service in distributed cloud computing

A-Young Son  
Department of Computer Engineering, 
Kyung Hee University 
Suwon, South Korea  
ayths28@khu.ac.kr

Eui-Nam Huh  
Department of Computer Engineering, 
Kyung Hee University 
Suwon, South Korea  
johnhuh@khu.ac.kr

Abstract

Live migration allows a server administrator to move a running virtual machine or application between large-scale datacenter for resource management. However, most of the previous works concentrate on the implementation of migration technology itself while previous works didn’t consider waiting time on migration efficiency. In this paper, we propose an efficient migration method in lightweight environment and considered distributed cloud computing condition.

In the future, we will study on live migration mechanism to improve the live migration efficiency in the large-scale service and variety of datacenter environments.

1. Introduction

Live migration technologies can contribute to efficient resource management in a cloud datacenter [1]. VM migration is important component and useful function for resource management in cloud computing. It is an issue that in the data center using the resources efficiently. In addition to enterprise provide resources in the form of service to users.

Nowadays, datacenters are distributed data center which deploy Micro-Datacenter (MDC) that are geographically distributed over a large number of locations[1]. We proposed an efficient migration method for seamless service to keep service running on a different server in the cluster moves to different area in distributed cloud computing condition. However, most of the previous work didn’t consider waiting time on migration efficiency and live migration such as migration failures due to the insufficient resources in target machine.

The goal is to maximize the migration efficiency by choosing the target machine based on waiting time and resource utilization in distributed environments.

Organization of the paper is as follows: Chapter 2 provides information about related work on previous approaches. Chapter 3 as solution for problems mentioned above and lightweight migration architecture through resource management. Chapter 4 provides performance evaluation and analysis. The last section concludes the paper and future work.

2. Related work

2.1. Live migration

Live migration provides many benefits such as energy saving, load balancing and maintenance. Previous migration work have limits like [Table 1]. So we study on choosing the right target machine considering waiting time through monitoring.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem</td>
<td>Migration noise</td>
<td>Overhead</td>
<td>Downtime, response time</td>
<td>Number of Iteration</td>
</tr>
<tr>
<td>Proposed method</td>
<td>Transfer only Hard page</td>
<td>Compression</td>
<td>Predict using of function</td>
<td>WWS</td>
</tr>
<tr>
<td>Pros</td>
<td>Total migration time</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cons</td>
<td>Previous works didn’t consider waiting time on migration efficiency and live migration such as migration failures due to the insufficient resources in target machine</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2.2. Virtualization technology

Docker and Libvirts bring rapid development in virtualization market. Docker [6] as an open source project, Docker makes the virtual space for running applications without hypervisor. Because Docker is so light, it is highly favored between the developer and operator. Libvirts [7] also as an open source API. It used in the orchestration layer of hypervisors in the development of a cloud-based solution. Virtualization
market is growing. But related work is insufficient in this situation.

3. Proposed Scheme

In this work, we focus on designing a migration system for an efficient migration in lightweight environment using Libvirt API and providing migration scheme. We proposed three scheme as follows.

1) Determines VMs which need to migrate through resource monitoring like hotspot
2) Consider waiting time and service execution time for target machine.
3) Determines target machine under the constraints of resource.

A proposed lightweight architecture for migration in distributed cloud environment is shown in Figure 2.

In this paper, we consider the waiting time and resources in order to solve the previous migration work limits that do not consider waiting time in a distributed environment. When it comes to the request of the user, selecting the target machine via a resource manager. We improves the speed by select the micro-datacenter for a new micro-data service. The goal is to maximize the migration efficiency. Figure 2 is architecture for improvement migration in MDC environment. It consists of three main modules: Monitoring manager, hotspot detector, and Migration manager.

Monitoring manager is responsible for observing the resource status of both virtual machines and physical machines, including the resource utilization, virtual machine configuration information (workload characteristics, VCPU number, memory size, image size) which is essential to make migration decisions. We set the relevant algorithms, which can be used to find the VMs which need to migrate.

Hotspot detector discovers hotspot through resource monitoring in performance degradation situation like hotspot. Hotspot detector continuously monitors these usage profiles to detect hotspot.

If the average usage of cpu and memory exceeds a threshold, we detect hotspots that may exceed the threshold, which not only can't guarantee the QoS, and waste resources.

Migration manager chooses the right target machine from the candidate target machine list. Service waiting time based on our algorithm described in below figure 2. Table 2 is definition of parameters for our algorithm.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t$</td>
<td>waiting time</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>possibility of waiting service</td>
</tr>
<tr>
<td>$r$</td>
<td>arrival rate/service rate = service rate</td>
</tr>
<tr>
<td>$\lambda_i$</td>
<td>arrival rate</td>
</tr>
<tr>
<td>$m$</td>
<td>service rate</td>
</tr>
<tr>
<td>$E[T_r]$</td>
<td>prediction of waiting time</td>
</tr>
<tr>
<td>$E[ST]$</td>
<td>service execution time</td>
</tr>
<tr>
<td>$V_w$</td>
<td>number of waiting VMs</td>
</tr>
</tbody>
</table>

Figure 2 Proposed migration phase

Migration phase consists of three main phase for migration as follows:

Step 1. Migration pre-phase
- Migration decision shown formula 1 is done.
  1) Formula 1: calculating service execution time and We define hotspot threshold through monitoring

$$W = W_q + E[ST]$$  \hspace{1cm} (1)

Step 2. Migration target machine selection phase
- Target machine selected through formula 2 and 3
- If you want service movement, Monitoring confirm the resource consumption for target VMs.
We will select the target MDC server using below formula based on waiting time.

1) Formula 2: prediction of waiting time
   \[
   \min_{y=1} \frac{1}{y(y-1)} \tag{2}
   \]

2) Formula 3: calculation of average waiting time in server
   \[
   V_q = \frac{1}{t_0} \int_{t_0}^{t_0} q(t) \, dt
   \]
   \[
   \therefore P(\tau) \leq \frac{\lambda}{\mu} \tag{3}
   \]

Step 3. Migration phase

- Performs the migration using formula 4
- It is considered service time and waiting time

1) Formula 4: resource allocation
   \[
   P_{\text{resource}} = \frac{E[ST] + E[ST]}{E[ST]} \tag{5}
   \]

Migration significantly consumes resource on the source and target machine. Hence it need to resource allocation for both source and target machine. Algorithm 1 is based on queuing theory. It is used service and arrive rate according to poisson distribution and exponential distribution. We can calculate number of waiting VMs through algorithm 3. If the hotspot is detected, we decision migration or not, through monitoring. And then put the virtual machine by ascending order using waiting time and execution time.

We will pre-set for create the host resource. When perform the migration, it will be possible to control in host-level through setting of new server resource. But we will use algorithm in lightweight environment. FIFO is able to occurs waiting time at long process time. So we proposed priority of service execution for the target machine selecting based on HRN scheduling method shown in formula 4. It is based on service time assigned priority according to waiting time and execution time. In the future, if we apply this method to the previous work, it will contribute efficiency.

4. Evaluation

Table 3 Comparision of lightweight environment and Openstack Environment

<table>
<thead>
<tr>
<th></th>
<th>Lightweight environment</th>
<th>Openstack environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel core 4</td>
<td>Intel core 4</td>
</tr>
<tr>
<td>RAM</td>
<td>4GB</td>
<td>4GB</td>
</tr>
<tr>
<td>OS</td>
<td>Ubuntu 12.04 64bit</td>
<td>Ubuntu 12.04 64bit</td>
</tr>
<tr>
<td>API</td>
<td>Libvirt v.0.9.8</td>
<td>Libvirt v.0.9.8</td>
</tr>
<tr>
<td>benchmark</td>
<td>ab(web Server)</td>
<td>rally</td>
</tr>
<tr>
<td>sysbench(DB)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In this section, we analyze and evaluate performance in proposed lightweight environment compared with Openstack environment. Both the two environments were used basically the same OS and the hypervisor and the same API version. To compare migration time, we use benchmarking tool called Rally. Table 3 shows the experiment environment.

4.1. Lightweight Environment

We create flavor type VMs as shown in Table 4 according to VM type in Openstack. In Figure 4, we can see the migration time depending on the type of the virtual machine. It is measured for migration time and downtime using benchmarking tool rally in Openstack. In Figure 4, the bigger memory size, total migration time is increased. But there is no change in downtime. In this paper, we focused on an efficiency of a total migration time, without considering the downtime. Because of performance degradation like Figure 4, we proposed an improved migration scheme in lightweight environment.
4.2. Performance of Proposed algorithm
MDC servers are constantly checked servers and the six server which the target machine to be migrated for resource through the monitoring. Because MDC is small of server size, algorithm execution time in this state is negligible. We can see that proposed migration performance improvements 9.6% compared with the baseline.

However, proposed algorithm will be given priority based on average waiting time and execution time. The larger memory, migration time is increased. But increase rate of migration time is small for existing algorithm. If apply this change at previous work, we will see improve efficiency for migration performance. Because proposed algorithm is considered also execution time.

4.3 Docker migration for MDC

<table>
<thead>
<tr>
<th>Table 5 Docker migration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>snapshot</td>
</tr>
<tr>
<td>reboot</td>
</tr>
<tr>
<td>Transfer</td>
</tr>
</tbody>
</table>

Table 5 present Docker migration efficiency in distributed environment through comparison of migration element. And it shows a possibility in distribution environment through Figure 9. Therefore, our proposed architecture will bring huge improvements to the migration in the large-scale service and variety of datacenter environments.

5. Conclusion
This paper proposed an efficient migration method in lightweight environment through resource monitoring in performance degradation situation like hotspot. Moreover, we have propose three stages of improved migration.

The goal is to maximize the migration efficiency by choosing the target machine consider waiting time and resource utilization. In this work, we focus on designing a migration system and providing migration scheme for resource utilization an efficient migration based on lightweight environment.

1) Determines VMs which need to migrate through resource monitoring like hotspot.
2) Selects target machine considering waiting time and service execution time for target machine.

We demonstrate by experiment in lightweight environment the superiority of our scheme compared with Openstack environment. Performance evaluation result presents proposed algorithm about 9.5% less than scheduling algorithm in lightweight environment. Therefore, our proposed architecture will bring huge improvements to the migration process while still guaranteeing quality of service.

In the future, we will study on live migration mechanism to improve the live migration efficiency in the large-scale service and variety of datacenter environment.

6. Acknowledgment
This work was supported by the IT R&D program of MSIP/IITP [B0101-15-0535, Development of Modularized In-Memory Virtual Desktop System Technology for High Speed Cloud Service]

7. References