An Efficient Virtual Machine Migration Algorithm Based on Minimization of Migration in Cloud Computing

Nguyen Khac Chien[1(✉)], Vo Sy Giang Dong[2], Nguyen Hong Son[3], and Ho Dac Loc[4]

[1] The People's Police University, Ho Chi Minh City, Vietnam
nkchienster@gmail.com
[2] FPT Software, Ho Chi Minh City, Vietnam
giangdongptit@gmail.com
[3] Post and Telecommunication Institute of Technology, Ho Chi Minh City, Vietnam
ngson@ptithcm.edu.vn
[4] Ho Chi Minh City University of Technology, Ho Chi Minh City, Vietnam
hdloc@hcmhutech.edu.vn

Abstract. Virtual machine (VM) migration in the cloud computing (CC) environment is an important issue to solve many problems, such as: Load Balancing, accomplished by migrating VMs out of overloaded/overheated servers, and Server Consolidation, where servers can be selectively brought down for maintenance after migrating their workload to other servers. In this paper, we propose an efficient VM migration algorithm based on minimization of migrations in CC to improve efficiency and response the requirements for user and restrict of service level agreements (SLA) violation. Experimental results showed the effectiveness of the proposed algorithm compared with existing algorithm.

Keywords: Virtual machine · Datacenter · Cloud computing · Live migration · Cloudsim

1 Introduction

In CC, storage, application, server and network devices can be virtualized. Virtualization can make many benefits, such as resource utilization, portability, and application isolation, reliability of system, higher performance, improved management ability and fault tolerance.

During operation of the datacenter in CC, many issues will arise, where energy issue plays an important role. Energy consumption in the datacenter will be very large, occupying a large part of operating costs. Greater power consumption leads to emit more CO_2 emissions, causing the greenhouse effect, the effect posed on the environment. The consolidation of VMs on a physical server that has light loads, use live migration, then state transition of idle servers into a state with low energy consumption (sleep, hibernate), which will help service providers optimize cloud resource usage and reduce energy consumption. However, it is very difficult to optimize the use of energy as the programs' workload keeps changing all the time which requests for the supply of dynamic resources. Therefore, the consolidation of VMs may badly affect performance. If the resources required by an application that is not fully allocated, then the responses to the application of user may be slow, or not available.

In addition, overloading of the services on the VM leads to a fact that we need to migrate to the VM has greater processing capacity to ensure quality of services (QoS) of cloud computing provider with customer commitment through the SLAs.

Determining when to implement VM migration in the CC is a major challenge to researchers and providers of CC services. Making a decision on implementing VM migration effectively and ensuring the performance of the system as well as the most efficient system resources usage are the ultimate goals of CC. This paper proposes a method of improving VM algorithm based on minimization of migrations to improve utilization, as well as satisfying the requirements for users.

The rest of the paper is organized as follows: Sect. 2 discusses related work, in Sect. 3, we propose method to improve VM migration algorithm, the experiment is described in Sect. 4, and the conclusion is presented in Sect. 5.

2 Related Work

Live migration facilitates online maintenance, load balancing and energy management [12]: (1) Online maintenance: Improving system's reliability and availability a system must be connected with the clients and the upgradation and maintenance of the system is also a necessary task so for this all VMs are migrated away without disconnections. (2) Load Balancing: VMs can be migrated from heavy loaded host to light loaded host to avoid overloading of any one server. (3) Energy Management: VMs can be consolidated to save the energy. Some of the underutilized server VM's are switched down, and the consolidated servers ensure power efficient green cloud.

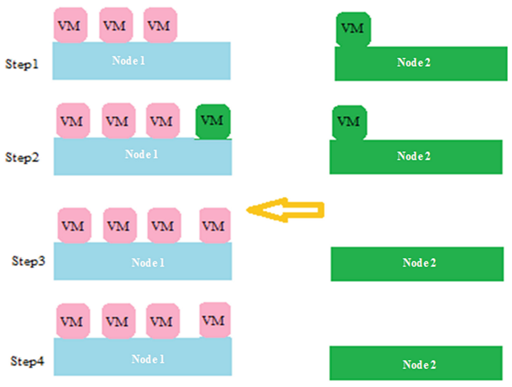


Fig. 1. An example of migrating VM [6].

Figure 1 shows the process of migrating VM. Initially, there are three VMs on a physical server Node 1 and one VM on a physical server Node 2. The migration process allows VM to run on Node 2 switched on Node 1.

The nature of the migration of a VM from one server to another server is that it transfers the whole system run-time state, including CPU state, memory data, and local disk storage, of the VM.

VM Migration methods are divided into two types [7]: Live migration and non-live migration. The status of the VM loses and user can notice the service interruption in cold migration. VM keeps running while migrating and does not lose its status. User doesn't feel any interruption in service during live migration. In live migration process, the state of a VM to migrate is transferred. The state consists of its memory contents and local file system. Local file system does not need to be transferred. First, VM is suspended, then its state is transferred, and lastly, VM is resumed at destination host. This migration method is applied to cloud computing environments.

Anton Beloglazov *et al.* [2] proposed migration techniques and VM consolidation for ultimate energy in a data center. The algorithms [2] were proposed: a general algorithm of migration decision-making and an algorithm to select VMs to migrate. Ts EpoMofolo *et al.* [8] proposed a prediction method based on the allocation of resources using VM migration. The authors proposed an algorithm to select the destination server. The goal of this algorithm is ensuring the servers to operate at an acceptable level to maximize available capacity of the system. Anton Beloglazov *et al.* [3] proposed two algorithms, the first: selecting the VM to migrate based on the criteria of the least of migration times; the second: Selecting the destination server to migrate with a goal to optimize the energy consumption of the system. Christopher Clark *et al.* [4] focus on the study of how to make "live migration", in [4] study Pre-copy migration method and its optimal solution. In addition, several other studies also study VMs migration on the CC environment [1, 5, 10, 11], the general characteristics of these researchs are aimed at achieving optimal use of energy of the system, also fully exploiting the processing power of the system.

In the CC environment, the implementation of live migration aimed at optimal system performance as well as operating costs while ensuring QoS. The migration process is divided into two phases: deciding phase migration and implementing phase migration. Deciding stage migration includes determining which conditions will be needed for the migration of VM, which VM needs migrating, and where to imgrate. In the second stage, this stage applied different techniques to the movement of the VM state is working to the destination server.

3 Proposed Algorithm

In this section, we analyze migration algorithm of selecting VM that requires minimum times of migration [3]. Then, we propose an improved algorithm with the goal of improving VM migration selection, so that it could have number of times at least and the remaining available CPU is minimal.

Algorithm [3] has the advantage that the number of times of migration is at least, leading to less occurrence of SLA violations while also minimizes consuming CPU performance as well as network bandwidth for migration process. Besides, the list of VMs is sorted before performing VM selection to migrate, in case of migrating only one VM, then the remaining available CPU after migration is smallest. That exploits thoroughly the resources.

However, in case you need to migrate more than one VM, the algorithm will not get the maximum exploitation of system resources. That is becaused during implemention process, if the algorithm does not find a VM to satisfy the condition in line 9 in Fig. 2, the algorithm will select a VM with the highest CPU usage to migrate, which will obviously ensure the selecting number of VMs at least leading to smallest of migration times, but the availability of the CPU of server after migration of VMs found was not the smallest. For example, assuming that a host has many VMs with CPU utilizations were 1, 3, 4, 5, 7 and it has the maximum CPU utilization threshold UT = 16. Thus, this server is overloaded, and has to migrate one or more VMs, so that total CPU utilization of the VMs migration is always greater than 8 to put the server operating within allowed threshold. If we apply above algorithm to choose the VM for migration, then two VMs with CPU utilization were 3 and 7 will be selected. The CPU utilization of server after migrating two selected VMs is $3 + 7 - 8 = 2$. But, we easily see that two VMs have CPU utilization 4, 5 as well as that make server not overloaded. Wherea, the CPU utilization after migrating two VMs is $4 + 5 - 8 = 1$. If these two VMs are selected, it will ensure miminum number of times, but still fully utilize system capacity.

From above limitation, we can improve the algorithm in [3] by Anton *et al.* so that the number of VMs, which are migrated, that is at least. At the same time achieving the purpose of maximizing the performance of the server. Meaning, the CPU utilization of

```
1  Input: hostList;    Output: MigrationList
2  foreach h in hostList do
3    vmList ← h.getVmlist()
4    vmList.sortDecreasingUtilization()
5    hUtil ← h.getUtil()
6    bestFitUtil ← MAX
7    while hUtil > THRESH_UP do
8      foreach vm in vmList do
9        if vm.getUtil() > hostUtil – THRESH_UP then
10         t ← vm.getUtil() – hUtil + THRESH_UP
11       if t < bestFitUtil then
12         bestFitUtil ← t
13         bestFitVm ← vm
14       else
15         if bestFitUtil = MAX then
16         bestFitVm ← vm
17       break;
18     hUtil ← hUtil – bestFitVm.getUtil()
19     migrationList.add(bestFitVm)
20     vmList.remove(bestFitVm)
21   if (hUtil < THRESH_LOWER) then
22     migrationList.add(bestFitVm)
23     vmList.remove(bestFitVm)
24 return migrationList
```

Fig. 2. Pseudo-code for the Minimization of Migration (MM) Algorithm [3]

the server after performing the migration will reach the threshold of the most overloaded. We suggest the improvements selecting VM in case you need to migrate multiple VMs. In the data center, there is a list of overloaded servers, each server has a list of VMs *vmList* that has to migrate the VM (*migrationList*) in oder to put server on an acceptable level within thresholds, CPU utilization of the server is *hostUtil* which are exceeded on the *UT, diffValue* is a real number representing the difference between current CPU utilization of the server and above threshold. Select the minimum number of VMs for migration, the total CPU utilization of the VMs, which are migrated, *vmsMigrateUtil* is larger than *diffValue* (bringing server utilization to be within threshold), while (*bestFitValue* = *vmsMigrateUtil* – *diffValue*) is minimal to maximize exploitation of system resources.

The proposed algorithm for multi VMs immigration are: Determine number of migrating VMs: *numbersVmMigrate.*

After determined number of VMs that going to be migrated, list out the migration plans based on VM's location in vmlist of the host. With each plans the algorithm will calculate *bestFitValue* and decide migration plan based on the value of *bestFitValue,* the plan with smallest *bestFitValue* will be chosen.

Listing out migration plans is actually matter of listing element *numbersVmMigrate*=*k* in array *listVm.size*() =*n.* This paper uses backtracking algorithm to accomplish. Optimal migration plan is determined through attempts. The result is represented by a vector *XOPT[]* including *numbersVmMigrate* elements. Also, each migration plan is represented by a vector *result* that consists of *numbersVmMigrate* element *result* = (x_1, x_2, …, x_k).

At each step i:

Built up elements x_1, …, x_{i-1}

Creating components x_i by evaluating all the possibilities that x_i can be.

If there is an ability j that is appropriate to x_i, then x_i is determined according to j. If *i* = *numbersVmMigrate*, we identified a complete migration plan. Check if this migration plan is better than previous plan, then we proceed an update, if it is not better there wil be no update required. However, if *i* is smaller than *numbersVmMigrate* proceed step *i+1* to determine x_i*+1.*

Backtracking algorithms can be written in the form of pseudo-code as follows:

Procedure In line 6 of Fig. 3 is used to update the migration plan. To clarify how the listing of this approach work, we describe an example that illustrate to select two elements of sequence {7, 5, 4, 3, 1} - the sequence has been shown in the example above (Fig. 4).

```
1 void Try(1, result,totalVms, numberVmMigrate, vmList) {
2   for (int j =result[i-1]+1; j<= totalVms –numberVmMigrate +i, j++)
3     {result[i] = j
4     if (i==k)
5     Process(result, numberVmMigrate, vmList)
6     else Try(i+1, result, n,k)
7   }
8 }
```

Fig. 3. Pseudo-code for the Backtracking Algorithm

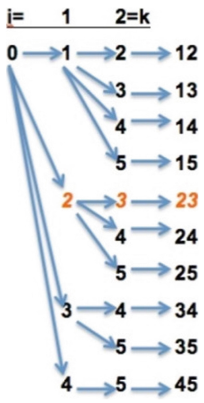


Fig. 4. Illustration of selected VM migration.

Using the backtracking technique to list out all of avaibility that are selected with two elements. Each result is the position of the two numbers that taken out from sequence. In each plan, the process will be executed to filter out the plan that is most fit to requirement. The improved algorithm is described in Fig. 5.

4 Experiment

In this section, we conducted experiments on heterogeneous cloud computing environment which has a data center with features such as Table 1. The algorithms are implemented through simulation package CloudSim based tool [9]. Java language is used for develop and implement the algorithms. Experiments will be conducted to compare the improved algorithm with the algorithm in [3] based on the following scenario:

Pair 1: Select migrated VM that has smallest times of migration according to the original algorithm and Select lowest energy consumption server

Pair 2: Select migrated VM that has smallest times of migration according to improved algorithm and Select lowest energy consumption server (Table 2)

In the simulation environment with a Broker's request of 2001 VMs, as follows (Table 3):

These VMs require same number of hard-drive and bandwidth. Assuming the VMs are running the game services, online services with parameters of cloudlet: 2500 Mips, outputFile = inputfile = 300 bytes.

The over-load and under-load threshold used in simulation programs are 70 % and 20 % respectively. Simulation time in scenarios is 1 day = 60 * 24 = 1440 min. The time for the system considering the migration is 350 s.

The simulation results are shown in Figs. 6 and 7: the blue line is the result of applying the original algorithm which has not been improved, the red line is the result of improved algorithm.

```
Input: hostList;     Output: MigrationList
foreach h in hostList do
  vmList ← h.getVmlist()
  vmList.sortDecreasingUtilization()
  hUtil ← h.getUtil()
  bestFitUtil ← MAX
  numberVmMigrate =0 // Number of VM to migration
  tongUtil =0 //Ulti is number of VM as needed migrating multiple VM
  while hUtil > THRESH_UP do
    foreach vm in vmList do
      bestFitVm = Null
      if vm.getUtil() > hostUtil – THRESH_UP then {
        t ← vm.getUtil() – hUtil + THRESH_UP
      if t < bestFitUtil then {
        bestFitUtil ← t
        bestFitVm ← vm
      }}
      else {
        if bestFitUtil = MAX then {
        numberVmMigrate++
        tongUtil = tongUtil+vm.getUtil()
        if (tongUtil> hostUtil-THRESH_UP) then
          break;
      } else
        break;
      }
    if (bestFitVm !=NULL) then {
      hUtil ← hUtil – bestFitVm.getUtil()
      migrationList.add(bestFitVm)
      vmList.remove(bestFitVm)
    }
    if (numberVmMigrate > 1) then {
      int totalVms = vmList.size()
      XOPT = new int[numberVmMigrate + 1]
      int result[] = new int[numberVmMigrate + 1]
      XOPT[i] =0
      result[0] =0
      Try(1, result, totalVms, nuberVmMigrate, vmList)
      Get result from XOPT[]
      }
      if (hUtil < THRESH_LOWER) then
        migrationList.add(bestFitVm)
        vmList.remove(bestFitVm)
return migrationList
```

Fig. 5. Pseudo-code for the Improved MM Algorithm

Thus, Figs. 6 and 7 show that the improved algorithm is better than original algorithm in multiple evaluation criteria.

In theory, improved algorithm enables maximum server resource utilization, namely the CPU usage after migrating VMs get close to the upper limit, leading to higher Utilization index (Fig. 7, The Utilization is represented in red line). Sometimes, the utilization level of two algorithms are relatively balanced. This happens when the migration of just one VM server can provide acceptable threshold.

Table 1. Detail of the data center

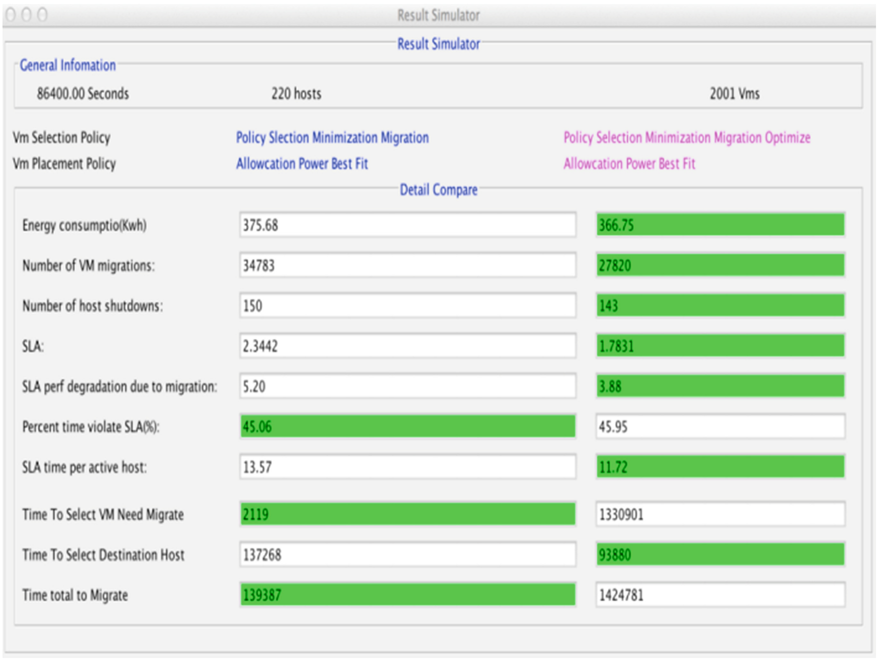
OS	Linux
Architecture	X86
VMM	Xen
TimeZone	10
VM Migration	Enable
Cost per Second ($)	3
Cost per RAM ($)	0.05
Cost per Storage ($)	0.001
Cost per Bandwidth ($)	0

Table 2. The table in the simulation server

Type of server	HP Proliant G4
Number	220
Host Mips	1860
Number of PE	2
Ram	4 Gb
Bandwidth	10 Gbps
HDD	100000 Tbs

Table 3. Detail of VMs in simulations

Name of VM	Mips	Ram	Policy	Number lượng
low	500	500	Time Shared	2001


Result Simulator

Result Simulator

General Infomation

86400.00 Seconds | 220 hosts | 2001 Vms

Vm Selection Policy	Policy Slection Minimization Migration	Policy Selection Minimization Migration Optimize
Vm Placement Policy	Allowcation Power Best Fit	Allowcation Power Best Fit

Detail Compare

Energy consumptio(Kwh)	375.68	🟩 366.75
Number of VM migrations:	34783	🟩 27820
Number of host shutdowns:	150	🟩 143
SLA:	2.3442	🟩 1.7831
SLA perf degradation due to migration:	5.20	🟩 3.88
Percent time violate SLA(%):	🟩 45.06	45.95
SLA time per active host:	13.57	🟩 11.72
Time To Select VM Need Migrate	🟩 2119	1330901
Time To Select Destination Host	137268	🟩 93880
Time total to Migrate	🟩 139387	1424781

Fig. 6. Result implemented in the form (Color figure online)

In terms of energy consumption, improved algorithms also give better results that are shown in Fig. 6. In the improved algorithm, the number of active servers maintained at a lower level that are as shown in Fig. 7, therefore energy consumption of servers and cooling systems will be less.

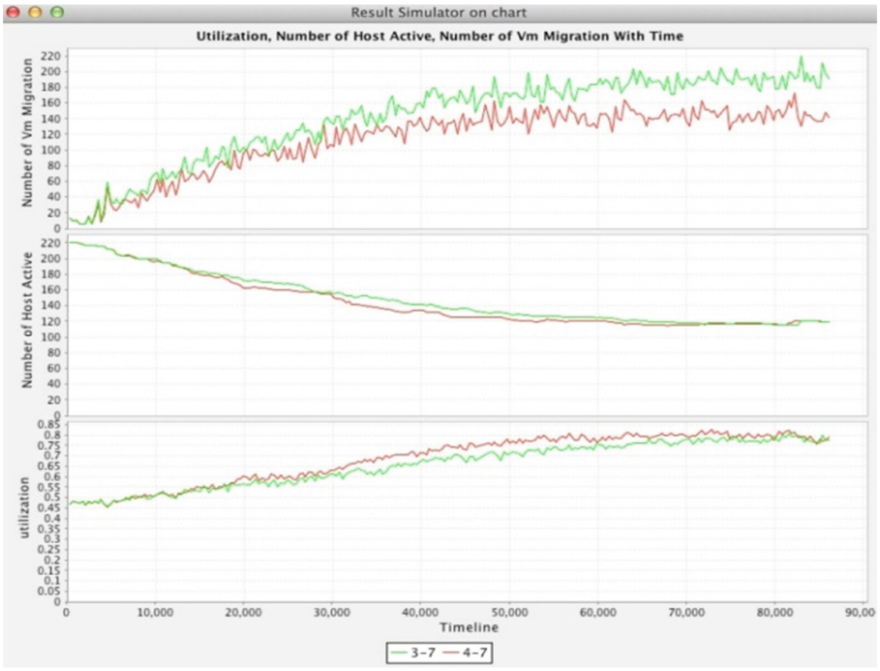


Fig. 7. Performance comparison results in graph (Color figure online)

Figure 6 shows the system maintenance operations in CPU consumption close to the upper limit, due to improved algorithm requires less migrations, it helps reducing the rate SLA violations. On the other hand, improved algorithms select the VM which has a lower CPU consumption, so the migration time of the VM will be less than the migration time of VMs of original algorithm. This reduces the violations to committed service, the service provider reduces the cost of fines for violations committed service triggers and the time required to select the destination server drops (Fig. 6).

5 Conclusion

VMs migration in the cloud is the problem which has drawn great concern from many interested researchers. There are many VM migration algorithms that have been proposed. The effectiveness of these techniques helped solve many problems on CC, such as load balancing, system maintenance *etc.* to enhance performance using cloud computing as well as quality of customer service.

This paper proposed an efficient VM migration decision-making algorithm on CC to solve the above problems. However, due to the fact that the new method was tested on simulated cloud environment using CloudSim tool, this method should be qualified on real environment in the future to clearly see its performance.

References

1. Ashima, A., Shangruff, R.: Live migration of virtual machines in cloud. Int. J. Scient. Res. Publ. **2**(6) (2012). ISSN 2250-3153
2. Anton, B., Rajkumar, B.: Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual Machines in Cloud Data Centers. Published online in Wiley InterScience (2012). doi:10.1002/cpe.1867
3. Anton, B., Jemal, A., Rajkumar, B.: Energy-aware resource allocation heuristics for efficient management of datacenter for cloud computing (2011). doi:10.1016/j.future.2011.04.017
4. Christopher, C., Keir, F., Steven, H., Jacob, G.H., Eric, J., Christian, L., Ian, P., Andrew, W.: Live migration of virtual machines. In: Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation, NSDI 2005, vol. 2, pp. 273–286 (2005)
5. Michael, R.H., Kartik, G.: Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In: Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, pp. 51–60 (2009). ISBN: 978-1-60558-375-4
6. Kakhi, K.R., Getzi, J.L.P.: Live virtual machine migration techniques-a survery. Int. J. Scient. Res. Publ. **1**(7) (2012). ISSN 2278-0181
7. Michael, R., Michael, H.: A new process migration algorithm (1996). ISBN-13: 978-1-86451-041-6
8. Ts`epoMofolo, Suchithra, R.: Heuristic based resource allowcation using virtual machine migration: a cloud computing perspective (2013). ISSN (online) 2319-183X, (Print) 2319-1821
9. Rodrigo, N.C., Rajiv R., Anton, B., Cesar, A.F.D.R., Rajkumar, B.: Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw.: Pract. Exp. **41**(1), 23–50 (2011). ISSN: 0038-0644, Wiley Press, New York, USA
10. Richa, S., Nidhi, P., Hiteshi, D.: Power Aware Live Migration For Data Centers in Cloud using Dynamic Threshold (2012). ISSN 2229-6093
11. Xiaoying, W., Xiaojing, L., Lihua, F., Xuhan, J.: A decentralized virtual machine migration approach of data centers for cloud computing. Article ID 878542, 10 (2013)
12. Divya, K., Emmanuel, S.P., Ramesh, C.J.: Live Virtual Machine Migration Techniques: Survey and Research Challenges (2012). 978-1-4673-4529-3/12/$31.00⊕2012 IEEE