Deadlock-free migration for virtual machine consolidation using Chicken Swarm Optimization algorithm

Tian, F. , Zhang, R. , Lewandowski, J. , Chao, K-M. , Li, L. and Dong, B.

Original citation:
Tian, F. , Zhang, R. , Lewandowski, J. , Chao, K-M. , Li, L. and Dong, B. (2016) Deadlock-free migration for virtual machine consolidation using Chicken Swarm Optimization algorithm. Journal of Intelligent & Fuzzy Systems, volume In press. DOI: 10.3233/JIFS-169136

http://dx.doi.org/10.3233/JIFS-169136

IOS Press

The final publication is available at IOS Press through http://dx.doi.org/10.3233/JIFS-169136

Deadlock-free migration for Virtual Machine Consolidation using Chicken Swarm Optimization Algorithm

Feng Tian [a,b,*], Rong Zhang[a,b], Jacek Lewandowski[c,d], Kuo-Ming Chao[c], Longzhuang Li [e] and Bo Dong [a]
[a] *The MoE Key Lab for INNS, Xi'an Jiaotong University, Xi'an 710049, P.R. China*
[b] *Systems Engineering Institute, Xi'an Jiaotong University, Xi'an 710049, P.R. China*
[c] *School of Computing, Electronics and Mathematics, Coventry University, UK*
[d] *Department of Genetics, Wroclaw University of Environmental and Life Sciences, Poland*
[e] *Department of Computer Science and Technology, Texas A&M University-Corpus Christi, TX, USA*

Abstract. Consolidation of services is one of the key problems in cloud data centers. It consists of two separate but related issues: Virtual machine (VM) placement and VM migration problems. In this paper, a VM consolidation scheme is proposed that turns the virtual machine consolidation (VMC) problem into a vector packing optimization problem based on deadlock-free migration (DFM) to minimize the energy consumptions. To solve this NP-hard and computationally infeasible for large data centers problem, a novel algorithm named Chicken Swarm Optimization based on deadlock-free migration (DFM-CSO) algorithm is proposed. The DFM-CSO algorithm is characterized by the 'one-step look-ahead with n-VMs migration in parallel (OSLA-NVMIP)' method, which carries out the VM migration validation and the rearrangement of target physical host, as well as records the migration order for each solution placement, so that VM transfer can be completed according to the migration sequence. The experimental results, for both real and synthetic datasets, show that the proposed algorithm with higher convergence rate is favourable in comparison with the other deadlock-free migration algorithms.

Keywords: VM consolidation, VM placement, Deadlock-free migration, Chicken Swarm Optimization

1. Introduction

The energy consumed by a data center can be broadly categorized into two categories: energy used by IT equipment such as servers, networks, storage, etc., and energy usage by infrastructure facilities such as cooling and power conditioning systems. The energy consumption of IT equipments accounts for about half of the total energy consumption, of which nearly 40% is consumed by servers. One of the most important reasons for energy inefficiency in data centers is too much idle time when servers run at a low load [1]. One of the main techniques to improve the energy-efficiency of servers in data center is called the virtual machine consolidation, which focus on application workloads consolidation on a smaller amount of physical hosts (PHs). The research shows that the cost of running idle servers with no tasks assigned accounts for over 50% of the peak power consumption [2-3]. Therefore, consolidation of virtual machines and shutting down idle servers are an effective energy-saving strategy.

Power modeling is an active area of research, studying both linear and nonlinear correlations between the system utilization such as VM placement or migration and power consumption [4-5]. While most of the virtual machine consolidation problems focus on VM placement optimization, which is the mapping of virtual machines to physical hosts, yet little research concerns how the initial VM placement can be transformed into the final placement and what the migration sequence is. In this paper we cover both VM placement and migration problems which can help to reduce data center's energy consumption through efficient VM management.

*Corresponding author. E-mail: fengtian@mail.xjtu.edu.cn.

2. Background

VM migration is a technology which has attracted considerable interest from data center researchers in recent years. It allows a virtual machine to migrate from one physical host called source PH to another physical host called target PH. The VM migration to the new placement takes place after the workload optimization. When there are *m* such virtual machines that need to be migrated to the new placements, the new VM placement has *m* migration tasks. If there are insufficient resources on the target physical host for a VM that needs to be migrated, then the migration of the VM is called '*infeasible migration*', otherwise, the migration of the VM is called '*transferable migration'* or '*transferable*'. If there is at least one infeasible migration task among the *m* migration tasks, then the migration of the whole VM cannot be successfully completed, and the new placement is called '*infeasible placement*'.

In practice, deadlock may occur during virtual machine migrations, which transform the initial placement into the new placement solution. There are four conditions for the deadlock occurrence: mutual exclusion, hold while waiting, no preemption and circular wait [6]. These conditions on both direct and indirect deadlock examples illustrated in Figure 1 and 2 below are further discussed. Note that VM *i*-M notation, denotes that the *i*-th VM needs to take up M units of host's CPU.

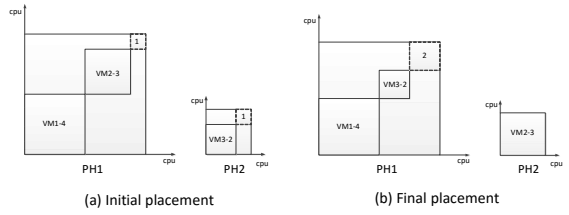

Fig. 1. Direct deadlock

In the first example of direct deadlock, also known as *infeasible migration*, let's assume that the data center is made of two physical hosts: PH1 and PH2. As shown in Fig. 1, the PH1 has 8 CPUs and PH2 has 3 CPUs available. In this case, when the initial VM needs to be transformed into the final placement, there are 2 migration tasks to do. The first task is the migration of VM2 requiring 3 CPUs from its initial placement on PH1 to the new placement on PH2.The second task in turn is the migration of VM3 requiring 2 CPUs from PM2 to PM1. In this case each VM migration requires the other VM to release its resources, what results in the deadlock, if and only if there exit two physical hosts. Although resources requirement in the final placement will not exceed the PM's maximum resources, the migration will not be completed without resources from other servers.

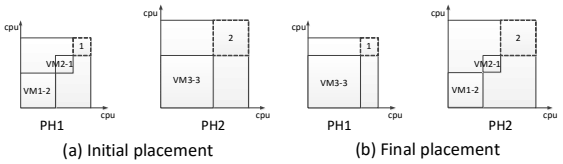

Fig.2. Indirect deadlock.

In the second example of indirect deadlock, improper migration order leads to the deadlock. As shown in Fig. 2, assume that the data center is made of two physical hosts: PH1 and PH2 and that PH1 has 4 CPUs and PH2 has 5 available. In this case there are 2 migration tasks that need to be done, and if the migration order is: VM1-2→PH2, VM3-3→PH1, and VM2-1→PH2, then the migration can be completed. However if the order is different and the first migration is VM2-1→PH2, then there are only 2 available CPUs left in PH1, and this PH cannot longer meet the requirements for the VM3 migration. Meanwhile, similar problem exists on PH2 where only two CPUs are available, and also PH2 cannot meet the requirement for the VM2 migration to that host. Consequently a circular wait is formed according to the migration order VM3-3→PH1 and VM1-2→PH2, that results in deadlock which does not allow the VM migration to complete.

Although it is rare to break first three conditions of deadlock's occurrence, breaking the fourth condition is relatively common. When these happen, the VM migration may lead to deadlock and in consequence may require redundant servers to be added in order to resolve the deadlock problem. Considering constrained resources and heavily loaded data centers, it is almost impossible to include additional physical servers, especially for some private clouds composed of a really small scale of physical servers.

This paper helps to solve this problem by proposing the novel algorithm named Chicken Swarm Optimization based on deadlock-free migration (DFM-CSO). It aims to find an optimal virtual machine placement and the migration sequence, which will not require redundant servers to mitigate the deadlock problem. The DFM-CSO algorithm is characterized by the *OSLA-NVMIP* method, which carries out the VM migration validation and the rearrangement of target physical hosts, as well as records the migration order for each solution placement,

so that VM transfer can be completed according to the migration sequence. It can help to obtain an optimal placement and a specific migration order which ensures that the optimal placement is transferable. Moreover, the *OSLA-NVMIP* method takes the idea of parallel priority to reduce the migration time. The experimental results, for both real and synthetic datasets, show that the proposed algorithm with higher convergence rate is favorable in comparison with the other deadlock-free migration algorithms. This paper is organized as follows. Section 3 focuses on problem formulation and presents the proposed methods. Section 4 introduces the DFM-SCO algorithm and the *OSLA-NVMIP* deadlock avoidance strategy. Section 5 presents the experiments and discusses the results. The paper is concluded in Section 6.

3. Problem formulation

The study in this paper is presented under one assumption that servers share the same hard disks pool, bandwidth, CPUs, and memory taken as the computing resources. Furthermore, redundant servers in data center are not allowed. Such presented VM consolidation problem is described as a vector packing problem and uses minimization of the energy consumption of the placement as the objective function.

3.1. Power consumption model

One of the most popular power consumption models is linearly proportional to the CPU utilization [7-9]. However, with rapid development of computer hardware technologies, the prediction performance of linear model is not accurate enough. Literature review [10] shows that the cubic polynomial power consumption model is significantly better than the linear model.

Let's assume there are M servers and N VMs in a data center. The power consumption, according to polynomial model, for the *i*-th server can be defined as :

$$C(i) = C_i^{idle} + \alpha_i \times U_i^{cpu} + \beta_i \times (U_i^{cpu})^2 + \gamma_i \times (U_i^{cpu})^3 \quad (1)$$

, with the total power consumption model defined as:

$$Minimize: C = \sum_{i=1}^{M} x_i C(i) \quad (2)$$

,where,

C_i^{idle} represents the power consumed when the *i*-th server is in idle state.

a_i, b_i, g_i are three regression coefficients, which describe the *i*-th server's power consumption.

U_i^{cpu}, U_i^{mem} represent the CPU-utilization and memory-utilization of *i*-th server, respectively.

The constraints conditions for such defined model are as follows:

$$x_i = \begin{cases} 0, & i\text{-}th\ server\ idel \\ 1, & other \end{cases} \quad (3)$$

$$0 \le U_i^{mem} \le 1 \quad (4)$$

$$0 \le U_i^{cpu} \le 1, \qquad i = 1, 2, \cdots, M. \quad (5)$$

The x_i in Eq. (1), is used to describe whether the *i*-th server is shutdown or not. Eq. (4) and Eq. (5) constrain the physical machine resource occupancy upper limitation of VMs' memory and CPU respectively. Under such defined constraints the goal of this study is to minimize the energy function C presented in equation Eq. (2). For this purpose the experiments were conducted on IBM 3850 X5 severs located in the data center at the Distance Learning College of the Xi'an Jiaotong University in China, which provides educational courses for over 69,000 students. For the purpose of this experiment the performance data was collected by Veeam Monitor [11] every 2 hours from 10/2/2014 to 1/2/2015 [12].

4. Methods

To solve the NP-hard and computationally infeasible for large data centers problem of VM migration, a novel algorithm named Chicken Swarm Optimization based on deadlock-free migration (DFM-CSO) is proposed. In this section, the main steps of the DFM-CSO algorithm will be introduced, and several key optimization strategies will be discussed.

4.1. Introduction to the framework of DFM-CSO algorithm

DFM-CSO is an optimization algorithm which adds deadlock avoidance strategy named *OSLA-*

NVMIP to the CSO algorithm. CSO was first proposed by Meng et al. [13] in 2014, as an swarm intelligence algorithm. It is a stochastic optimization algorithm which imitates the behavior of a group of chickens searching for food. This algorithm classifies 'chickens' into three categories, namely: rooster, hen and chick according to their fitness level. In this model each type of 'chicken' carries out different searching strategy and the chicken swarm updates itself after several generations. What characterizes this algorithm is its ability to avoid local optima and quickly find the global optimal value, when solving the optimization problem. The *OSLA-NVMIP* deadlock avoidance strategy means that, in each step of transferring VMs, all *transferable* migration of the VMs are moved into target PHs in parallel, while the transferability of each solution placement is verified and modified according to whether there exist sufficient resources on the target physical host for each VM that needs to be migrated, which ensures that every solution placement can be transferred. In principle if one solution placement is unable to be transferred, then the target PHs is rearranged until the placement becomes transferable. This strategy will be discussed in detail further in this section.

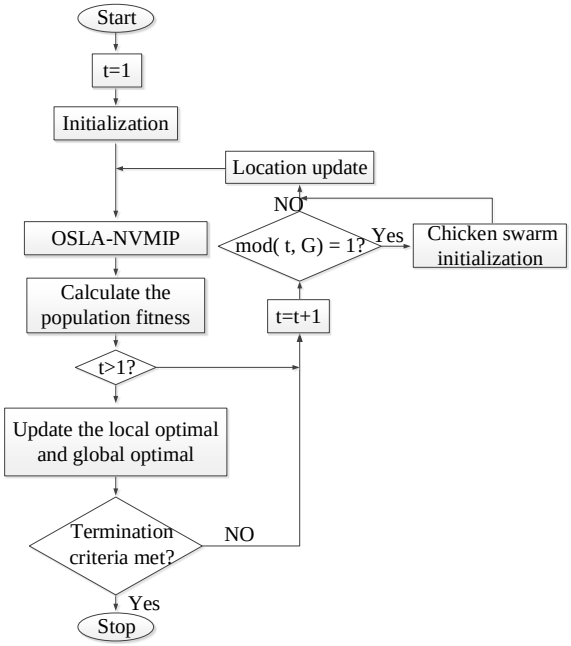

Fig. 3. Flowchart diagram of DFM-CSO algorithm.

Outlined in Figure 3 is the flowchart of the DFM-CSO algorithm, proposed in this paper, which is made of the following eleven main steps:

Step 1: Set t=1.
Step 2: Initialization. Initialize servers and virtual machines, create placements, and set the swarm population size and other parameters. Each 'chicken' in the pool is encoded to represent a placement.
Step 3: OSLA-NVMIP. "one-step look-ahead with n-VMs migration in parallel" method carries out the VM migration validation and the rearrangement of target PH, as well as records the migration order for each solution placement.
Step 4: Calculate the population fitness. Calculate the fitness for each placement.
Step 5: if *t* is greater than 0, then go to Step 10, otherwise go to Step 6.
Step 6: *t*=*t*+1.
Step 7: Detect the judgement conditions. If the conditions are met then go to Step 8, else, go to Step 9.
Step 8: 'Chicken' swarm initialization. Classify 'chickens' into three categories according to their fitness.
Step 9: Location update. Update the location of different 'chicken' groups according to predefined location model and encode them to represent their placement. Go to Step 3.
Step 10: Update the local optimal and global optimal values.
Step 11: Detect the termination conditions. If the termination conditions are met then exit the loop, else, go to Step 6.

4.2. Core models and strategy

4.2.1 Swarm location update model

The rooster's location update model is defined as follows:

$$chrom_{i,j}^{generations+1} = chrom_{i,j}^{generations} * (1 + N(0, \sigma^2))$$

$$\sigma^2 = \begin{cases} 1, & if \quad f_i \le f_k \\ \exp\left(\frac{f_k - f_i}{|f_i| + \varepsilon}\right), & otherwise \end{cases}$$

$$k \in [1, CN], k \ne i,$$

where CN is the number of chicken swarms , $chrom_i$ depicts the position of the *i*-th chicken; $chrom_{i,j}^{\text{generations}}$ is the *j*-th element of the *i*-th chrom at time step *generations.* $N(0, s^2)$ is the normal distribution with means 0 and standard deviation s^2. e

is the smallest constant used to avoid zero-division-error. k is a rooster's index randomly selected from the roosters group, f_i is the fitness value of the corresponding $chrom_i$.

In turn, the hen's location update formula is defined as follows:

$$chrom_{i,j}^{generations+1} = chrom_{i,j}^{generations} + p1 * Rand * (chrom_{c1,j}^{generations} - chrom_{i,j}^{generations}) + p2 * Rand * (chrom_{c2,j}^{generations} - chrom_{i,j}^{generations}),$$

$$p1 = \exp(- | \frac{f_i - f_{c1}}{f_i + e} |),$$

$$p2 = \exp(- | f_{c2} - f_i |),$$

$c1 \neq c2$.

Where *Rand* is a uniform random number from [0,1]. $c1 \in [1,\cdots,CN]$ is the rooster's index, which is the *i*-th hen's group-mate, while $c2 \in [1,\cdots,CN]$ is an index of the chicken (rooster or hen), which is randomly chosen from the swarm. *p1* is a influence factor that the chrom is affected by the rooster, which is the hen's group-mate, while *p2* is the influence factor that the chrom is affected by other hens and roosters.

Similar to nature, where chicks move around hens to forage for food, the CSO model has its 'chicks' which move around 'hens' to search for optima. This feature is defined as:

$$chrom_{i,j}^{generations+1} = chrom_{i,j}^{generations} + L * (chrom_{m,j}^{generations} - chrom_{i,j}^{generations})$$

,where $chrom_{m,j}^{\text{generations}}$ stands for the position of the *i*-th hen($m \in [1, CN]$) and L ($L \in (0,2)$) is a parameter, which ensures that chick follows its hen to search for an optima. The parameter L value for each chick is randomly chosen between 0 and 2.

4.2.2 OSLA-NVMIP

It is very difficult to estimate placement transferability without the virtual machine migration sequence. To estimate the given placement weather it is transferable or not from initial placement is not a trivial task. Moreover, it is the NP-hard problem to search the VM migration sequence knowing only the initial and the given placement. Since tracing whether a solution can be transferred or not requires the VM migration sequence, the VM migration sequence becomes the key to transferability detection.

Xing et al. [14] adopted "one-step look-ahead" method to solve the deadlock problem in flexible manufacturing system. The idea behind this method is that if one step forward enters the unsafe state, then the method returns this deadlock path and takes other path instead. In turn, Sarker and Tang [15] proposed an algorithm, which is similar to one-step look-ahead with n-VM migration in parallel method to deal with migration deadlock problem. This paper adopts the *OSLA-NVMIP* strategy, which can rearrange the target PHs for n-number of VMs which are failing to be successfully migrated.

The framework of *OSLA-NVMIP* strategy proposed below, takes the length of the vector as the amount of VM's, and each vector component value is the corresponding physical host number assigned to each VM. For example, chrom=[4 2 2] represents that No.1 VM is placed in the No.4 PH, and that No.2 VM and No.3 VM are placed in the No. 2 PH.

The main steps of '*OSLA-NVMIP*' strategy are as follows:

Framework of '*OSLA-NVMIP*' strategy:

Step 1: Find out all VMs which need to be migrated.

Step 2:For every PH, find all transferable VMs, and record them (see Algorithm 1 for more details). Then immigrate these VMs into the corresponding PHs.

Step 3: Detect if the termination condition is met. That is, judge whether the number of VMs need to be migrated before the transferring in Step 2 is equal to the number of VMs need to be migrated after the transferring in Step 2. If these two numbers are zero, then stop this procedure; If the two values are equal but not zero, then go to Step 4; otherwise, return to Step 2.

Step 4: Calculate idle virtual machines.

Step 5: If there still exist VMs that need to be migrated, then continue to Step 6, else stop the procedure.

Step 6: Rearrange the target PH for the VM which needs to be migrated. The target PH is selected from the list of currently used PHs. Calculate available resources and the number of idle PMs after each migration.

Step 7: Detect whether the termination condition is met or not. Compare the number of migrated VMs with the number of VMs which needed migration in Step 1. If the two values are

equal, stop the procedure; otherwise, reset the solution to the initial state and set the migration sequence to null.

For all the VMs that are identified in Step 2, the migration process to different target PHs can be done in parallel. In this way, the migration time can be shortened due to multi-VM migration within one step. The pseudo code for Step 2 is showed in Algorithm 1 listing below.

Algorithm 1 **The pseudo codes for Step 2**

```
1 for every physical host PH i  do
2     find all the VMs that need to be migrated to PH i as
      a set, named vmposition1
3    if length(vmposition1)>0   then
4          Cc_cost=0;
5          Cm_cost=0;
6          for j = 1 to length(vmposition1)   do
7              Cc_cost= Cc_cost +
                       VM.Cc(vmposition1(j));
8              Cm_cost= Cm_cost +
                       VM.Cm(vmposition1(j));
9              If (Cc_cost <= PMuseable.Cc(i)) ∧
                  (Cm_cost <= PMuseable.Cm(i)) then
10                migratenum=migratenum+1;
11            migrationsequence(migratenum)=
                              vmposition1(j);
              else
12              Cc_cost= Cc_cost –
                       VM.Cc(vmposition1(j));
13              Cm_cost= Cm_cost –
                       VM.Cm(vmposition1(j));
14            end if
15        end for
16    end if
18 end for
```

In the above listing, Cc_cost is an occupancy rate of the sum of CPU utilization of all transferable VMs that migrated to a specific PH in a step; and Cm_cost is an occupancy rate of memory utilization of all transferable VMs that migrated to the specific PH in a step. *VM.Cc* is an occupancy rate of CPU utilization of single transferable VM that migrated to a specific PH; and VM.Cm is an occupancy rate of memory utilization of single transferable VMs that migrated to the specific PH. PMuseable.Cc(i) and PMuseable.Cm(i) are the percentage of the residual CPU and Memory capacity of the *i*-th PH

5. Experiment and analysis

Performance of the proposed DFM-CSO and other four improved migration algorithms: DFM-PSO, DFM-GA, DFM-IGA(improved DFM-GA algorithm) and DFM-BBO/DE, were compared and evaluated in experiments on both real and synthetic datasets. Synthetic VM instances have been generated using method proposed by Gao et al. [16]. In turn, for real dataset generation 10 types of Amazon EC2 [17] instances have been used. In the experiment scenario, described below, the initial placement is what the placement state (location, CPU, Memory, etc.) of all VMs considered are in a moment. To simulate this, Matlab software has been used. The results obtained on both datasets show that the proposed algorithm with higher convergence rate is favourable in comparison with the other improved deadlock-free migration algorithms. Note that, after introducing the OSLA-NVMIP deadlock avoidance strategy into PSO [18], GA [18], IGA [19] and BBD/DE [10], we implemented and obtained four improved algorithms, DFM-PSO, DFM-GA, DFM-IGA and DFM-BBO/DE.

Ref. [20] and [21] had given the convergence proof of the PSO algorithm, which shows that the original PSO is neither with local convergence nor with global convergence. So, the same thing happens in the convergence of DFM-PSO. It is proved by means of homogenous finite Markov chain analysis that a generic GA will never converge to the global optimum regardless of the initialization, crosser, operator and objective function. However, variants of canonical GA's that always maintain the best solution in the population, either before or after selection, are shown to converge to the global optimum [22]. As the same theory, both of the GA and IGA in this paper adopt the method which maintain the best solution after selection, so the DFM-GA and DFM-IGA which proposed in this paper are with global convergence under the deadlock avoiding strategy. Ref. [23] gives the convergence proof of the BBO algorithm based on the assumption that the iteration time tends to be infinite. So, the BBO/DE have the same convergence property under the deadlock avoiding strategy. Ref. [13] indicates that, for the CSO, the appropriate choose of parameter G is problem-based. If the value of G is very big, it's not conducive for the algorithm to converge to the global optimal quickly. While if the value of G is very small, the algorithm may trap into local optimal [13]. This principle also works on DFM-CSO.

5.1 Scenario

There are 4 PHs and 8 VMs with the same configuration. Initial VM placement is VM_inital = [1, 1, 2, 2, 3, 3, 4, 4], where each VM's CPU and memory occupancy rate demands are: VM.Cc = [1/4, 2/4, 1/4, 2/4, 1/4, 2/4, 1/4, 2/4] and VM.Cm = [1/10, 2/10, 1/10, 2/10, 1/10, 2/10, 1/10, 2/10] respectively. Since memory utilization demand of each VM is relatively low and the required resources of PH are adequate, hence this scenario can be regarded as single-resource case. Figure.4 shows the CPU resources of the initial placement.

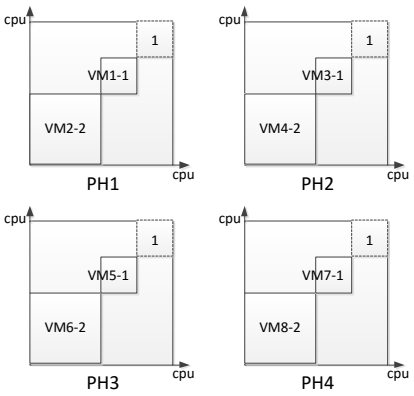

Fig.4. Scenario VM's initial placement

There is a variety of candidate final placements with the same minimum power consumption, because the method presented in this paper considers the final placement energy consumption as the only cost function that needs to be optimized without looking for example at the number of VMs to be migrated. The optimal solution calculated by DFM-CSO algorithm is shown below:

- Optimal placement: bestchrom=[2, 3, 3, 2, 3, 1, 2, 1] ;
- VM migration sequence: VM_Migratesequence=[1, 3, 6, 7, 2, 8] ;
- Parallel nodes: parallernode=[2, 4, 5, 6].

The *i*-th element of parallelnode denotes the aggregated number of VM migrations until the *i*-th step. There are 4 parallel migrations according to the results:

The first parallel migration: VM1-1→PH2;
VM3-1→PH3;
The second parallel migration: VM6-2→PH1;
VM7-1→PH2;
The third parallel migration: VM2-1→PH3;
The fourth parallel migration: VM8-2→PH1.

The results indicate that according to the migration sequence, all migrations are deadlock-free. Thanks to parallel migrations, the time spent for these migrations is shorter than the time required to migrate each VM separately. The optimal placement best chrom shows that no VM is placed to PH4, therefore PH4 will be idle after migrations and the number of physical hosts with workload is three. This will save the energy consumption.

5.2 Synthetic dataset

The method used to generate Synthetic instances is showed in Algorithm 2.

Algorithm 2 Generation of Synthetic Instances

1 for i=1 to n **do**
2 $Cc_i = 2*rand(\overline{Cc})$;
3 $Cm_i = rand(\overline{Cm})$;
4 $r = rand(1)$;
5 if $(r < P \wedge Cc_i \geq \overline{Cc}) \vee (r \geq P \wedge Cc_i < \overline{Cc})$ **then**
6 $Cm_i = Cm_i + \overline{Cm}$;
7 end if
8 end for

, where $\overline{Cc}$ and $\overline{Cm}$ are parameters used to control the utilization of CPU and memory respectively. P is corresponding to the correlations between CPU and Memory utilization. The algorithm 2 is introduced from Ref. [16] into this paper.

5.3 Real dataset

As outlined in Table 2, the real dataset has been made of 10 general purpose T2 and C3 instances from Amazon EC2 [17].

TABLE 2
Instance Types from Amazon EC2

Instance Type	vCPU	Memory (GiB)	Physical Processor
t2.nano	1	3	Intel Xeon family
t2.micro	1	6	Intel Xeon family
t2.small	1	12	Intel Xeon family
t2.medium	2	24	Intel Xeon family
t2.large	2	36	Intel Xeon family
c3.large	2	3.75	Intel Xeon E5-2680 v2
c3.xlarge	4	7.5	Intel Xeon E5-2680 v2
c3.2xlarge	8	15	Intel Xeon E5-2680 v2
c3.4xlarge	16	30	Intel Xeon E5-2680 v2
c3.8xlarge	32	60	Intel Xeon E5-2680 v2

5.4 Synthetic datasetResult and analysis

Several scenarios are used to compare the performance of the DFM-CSO algorithm with that of DFM-PSO, DFM-GA, DFM-IGA and DFM-BBO/DE. For a fair comparison, all of the common parameters of these methods are set to be the same. We set the population size as 50 and the maximum number of generations as 500 and 100 physical hosts as servers. The related parameter values of these algorithms are showed in TABLE 3.

TABLE 3
The related parameter values

Algorithm	**Paramerters**
DFM-PSO	*c1=c2=1.49445,w=0.729*
DFM-GA	*pmutation=0.3,pcrossover=0.7*
DFM-IGA	*pmutation=0.3,pcrossover=0.7*
DFM-MBBO	*Pmutation=0.2,I=1,E=1, F=0.6,pcrossover=0.2*
DFM-CSO	CN_r*=0.2*CN,*CN_h*=0.6*CN,* CN_c*=CN-*CN_r*-*CN_h*,* CN_m*=0.1**CN_c*,G=3,* $L \in [0.5,0.9]$

5.5 Experment result based on synthetic dataset

Scenario 2 sets parameters $\overline{Cc} = \overline{Cm} = 0.25$, $P = -0.072$ to general 200 VMs synthetic instances, and generate initial placement randomly.

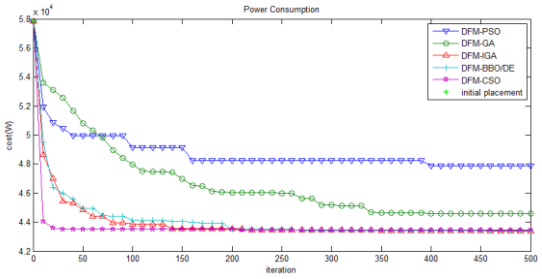

Fig.5. Comparison of DFM-CSO with four algorithms on Synthetic dataset

The experimental results shown in Fig.5 and TABLE 4, show that the proposed algorithm is characterized by the highest convergence rate in comparison with other four migration algorithms with the solution being close to the optimum after about 25 iterations. The algorithms which optimum solution is closest to the one obtained with the DFM-CSO are the DFM-IGA and DFM-BBO/DE. The convergence rates of these two algorithms are very similar. The other two algorithms performed significantly worst in terms of the optimal solution accuracy as well as the convergence rate.

TABLE 4
Comparison of DFM-CSO with four algorithms on Synthetic dataset

Algorithms	Cost(W)	Idle servers	Save Cost
Initial placement	57846	10	0%
DFM-PSO	47896	32	17.2%
DFM-GA	44574	38	22.9%
DFM-IGA	**43387**	**41**	**25%**
DFM-BBO/DE	43417	**41**	24.9%
DFM-CSO	43470	**41**	24.9%

The DFM-GA algorithm characterized by slow search rate in the earlier stages of operation has been improved after the number of iteration. DFM-PSP algorithm characterized by the general slow search rate was prone to trap into local optima. When looking at the idle physical servers consolidation solutions obtained with different algorithms, the DFM-PSO and DFM-GA increased from 10 idle servers in the initial placement up to 32 and 38 idle servers, respectively, after 500 consolidation iterations. The DFM-IGA, DFM-BBO/DE and DFM-CSO all reached up to 41 idle physical servers.

Comparing the energy consumption of the initial placement with the energy consumption of the optimum placement of various methods, we observed that the optimum placement obtained by DFM-CSO algorithm saves 24.9% energy and beats DFM-PSO and DFM-BBO/DE. The proposed algorithm, DFM-CSO, has an outstanding feature that the speed of approaching optimum placement is faster than others, seen in Fig. 5.

TABLE 5
Comparison of DFM-CSO with three algorithms on Synthetic dataset

Algorithms	Average Cost(W)	Standard deviation(W)
DFM-PSO	47953	513.165
DFM-IGA	**43211**	196.060
DFM-BBO/DE	43552	135.100
DFM-CSO	43389	**131.464**

Using the same parameter values and running the DFM-PSO, DFM-IGA, DFM-BBO/DE and DFM-CSO for 10 times, respectively. The results are shown in TABLE 5. The DFM-CSO has the minimal standard deviation as 131.464 compare with other three algorithms, which mean that DFM-CSO has batter stability than others. The convergence rate of DFM-CSO is outstanding, because its results approached the optimum placement after 25 iterations. Moreover, the experiment has been carried out 15 times when set the number of iterations as 30 and set G as 2, 3, 4, 5, 6, 7, 8, 9, 10, respectively, in scenario 2. The experiment results are shown in Figure 6.

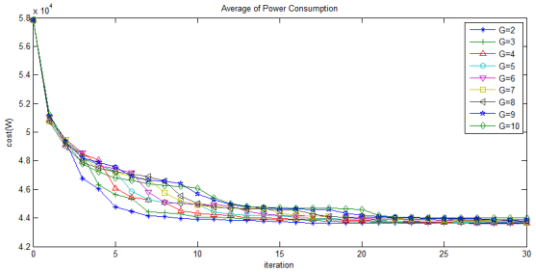

Fig.6. Comparison of DFM-CSO with different G

In Figure 6, we can observe that, with the decrease of the value of G, the convergence rate of DFM-PSO becomes fast, while the optimum values that the algorithm obtained are very close. This can conclude that, with the influence of location update strategy, the faster the speed of chicken grows and smaller the value of G is, the faster the convergence rate of the proposed algorithm is.

5.6 Experment result based on Real-world dataset

5.6.1 Real data scenario 1

In the first real data scenario, 200 virtual machines were generated with 5 types of C3 instances and their initial placement allocations were random. There were 100 PHs with the same specification and each PH had 40 CPUs and 128 GB of memory. The experiment results are shown in Figure 7 and TABLE 6.

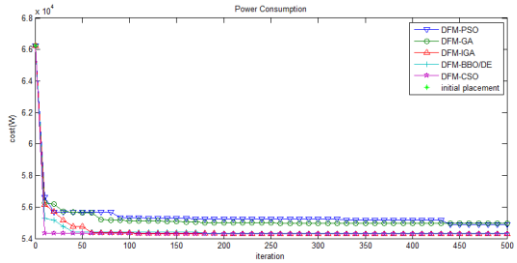

Fig.7. Comparison of DFM-CSO with four algorithms on real dataset of C3 instances

TABLE 6
Comparison of DFM-CSO with four algorithms on real dataset of C3 instances

Algorithms	Cost(W)	Idle servers	Save Cost
Initial placement	66228	4	0%
DFM-PSO	54878	29	17.1%
DFM-GA	54967	28	17.0%
DFM-IGA	54284	30	18.0%
DFM-BBO/DE	54324	30	17.97%
DFM-SCO	54306	30	18.0%

Comparing Figure 5 with Figure 7 we can notice certain similarity between these two graphs. We can also note that the convergence rate is smaller when the virtual machine resources are larger.

5.6.2 Real data scenario 2

In the second real data scenario, the total of 200 virtual machine instances were generated and randomly initialized with 5 types of T2 instances. There were 100 physical hosts with the same specification and each physical host had 40 CPUs and 128 GB of memory.

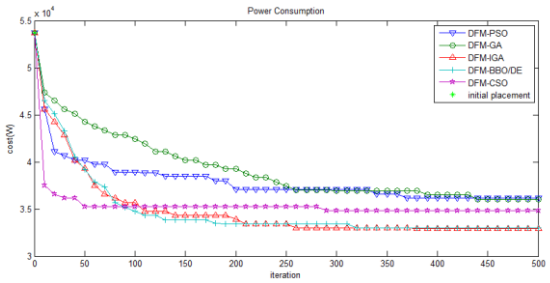

Fig.8. Comparison of DFM-CSO with four algorithms on real dataset made of T2 instances.

Comparison of Figure 7 and Figure 8, reveals that the convergence rate of DFM-CSO algorithm is relatively high compared with DFM-IGA and DFM-BBO/DE, but much less effective than the other two.

The resource requirement of virtual machines in scene 3 is much less than in scene 2, and DFM-CSO, which is more applicable to virtual machines that request more resources, may not have obvious energy-saving effect for correction of all the possible solutions. Besides, it also shows that the virtual machine which have more resources leads to iteration of convergence reducing relatively.

6. Conclusions

This paper presents a new algorithm for virtual machine consolidation based on the Chicken Swarm Optimization model. The experimental results, for both real and synthetic datasets, indicated that the proposed algorithm with higher convergence rate is favourable in comparison with other deadlock-free migration algorithms.

Future work on this algorithm will focus on server's load balance on heterogeneous server infrastructures and the placement migration ability, combining new strategies [24-25].

7. Acknowledgement

This research was partially supported by the National Natural Science Foundation of China under Grant Nos. 91118005, 91218301, 91018011, 61472315 and 61502379, MoE Innovative Research Team in University under Grant No. IRT13035, Innovation Project of Shaanxi Province Key lab (2013SZS05-p01) and by Project of China Knowledge Centre for Engineering Science and Technology.

References

[1] F Farahnakian , P Liljeberg , J Plosila, Energy-Efficient Virtual Machines Consolidation in Cloud Data Centers using Reinforcement Learning, *Parallel, Distributed, & Network-based Processing*, 2014:500-507.

[2] G. CHEN, et al, Energy-aware server provisioning and load dispatching for connection-intensive internet services, *Usenix Symposium on Networked Systems Design & Implementation*, 2008: 337-350.

[3] Khosravi A, Garg S K, and Buyya R, Energy and carbon-efficient placement of virtual machines in distributed cloud data centers, *International Conference on Parallel Processing*, Aug. 2013:317-328.

[4] Mohammad Masdari, Sayyid Shahab Nabavi, Vafa Ahmadi, An over view of virtual machine placement schemes in cloud computing, *Journal of Network & Computer Applications*, 2016, 66(C):106-127.

[5] Sandeep Kaur, Prof. Vaibhav Pandey, A Survey of Virtual Machine Migration Techniques in Cloud Computing, *Computer Engineering and Intelligent Systems*, 2015, 28-34.

[6] ZA Banaszak, BH Krogh, Deadlock Avoidance in Flexible Manufacturing Systems with Concurrently Competing Process Flows, *IEEE Transactions on Robotics & Automation*, 1990, 6(6):724-734.

[7] W. Tian, G. Lu, C. Jing, Y. Zhong, J. Hu, X. Dong. Method and device for implementing load balance of data center resources, *US Patent8*,510,747 (Aug. 13 2013).

[8] S. Srikantaiah, A. Kansal, F. Zhao, Energy aware consolidation for cloud computing, *Cluster Computing*, 2008, 12(1):10-15

[9] X. Fan,W.-D.Weber, L. A. Barroso, Power provisioning for a warehouse-sized computer, *Acm Sigarch Computer Architecture News*, 2007, 35(2):13-23

[10] Qinghua Zheng, Jia Li, et al. Multi-objective Optimization Algorithm based on BBO for Virtual Machine Consolidation Problem, *IEEE International Conference on Parallel & Distributed Systems*,2015:414-421.

[11] C.-H. Lien, Y.-W. Bai, M.-B. Lin, Estimation by software for the power consumption of streaming-media servers, Instrumentation and Measurement, *IEEE Transactions on Instrumentation & Measurement*, 2007, 56(5):1859-1870.

[12] Q Zheng, R Li, X Li, N Shah, J Zhang, et al. Virtual Machine Consolidated Placement Based on Multi-Objective Biogeography-Based Optimization, *Future Generation Computer Systems*, 2016, 54(C): 95-122

[13] Xianbing Meng, Yu Liu, Xiaozhi Gao, Hengzhen Zhang, A New Bio-inspired Algorithm: Chicken Swarm Optimization, *Hefei: Springer International Publishing*, 2014: 86–94.

[14] Xing, K. Y., Zhou, M. C., Liu, H. X., & Tian, F. (2009). Optimal Petri net based polynomial-complexity deadlock avoidance policies for automated manufacturing systems. *IEEE Transactions on Systems Man & Cybernetics Part A Systems & Humans*, 2009, 39(1):188-199.

[15] TK Sarker, M Tang, Performance-driven Live Migration of Multiple Virtual Machines in Datacenters, *IEEE International Conference on Granular Computing*, 2013,8151:253-258.

[16] Y Gao, H Guan, Z Qi, Y Hou, L Liu, A multi-objective ant colony system algorithm for virtual machine placement in cloud computing, *Journal of Computer & System Sciences*. 2013, 79(8):1230-1242.

[17] http://aws.amazon.com/ec2/instance-types/?nc1=h_ls.

[18] F Gao, MATLAB Super Learning Manual for Intelligent Algorithm, *Posts & Telecom Press*, 2014

[19] SN Sivanandam, SN Deepa, Introduction to genetic algorithms, *MIT Press*, 1998, 33(3):293--315

[20] Frans van den Bergh, A P Engelbrecht. A New Locally Convergent Particle Swarm Optimize. *IEEE International Conference on Systems, Man & Cybernetics*, 2002, 3(3):94-99.

[21] FVD Bergh,AP Engelbrecht,A study of particle swarm optimization particle trajectories , *Information Sciences*, 2006, 176(8):937-971

[22] G Rudolph, Convergence analysis of canonical genetic algorithms, *IEEE Transactions on Neural Networks*, 1994, 5(1):96-101

[23] D Simon, A probabilistic analysis of a simplified biogeography-based optimization algorithm , *Evolutionary Computation*, 2011, 19(2):167-188

[24] R Yousefian, S Aboutorabi, V Rafe, A greedy algorithm versus metaheuristic solutions to deadlock detection in Graph Transformation Systems, in: *Journal of Intelligent and Fuzzy Systems*, 31(1) April 2016.

[25] KW Huang, JL Chen, CS Yang, CW Tsai, PSGO: Particle swarm gravitation optimization algorithm, *Journal of Intelligent & Fuzzy Systems*, 2015, 28(6):2655-2665.