Heuristics for Optimal Placement and Migration of Virtual Machines

SATORU OHTA

Department of Information Systems Engineering,
Toyama Prefectural University,
5180 Kurokawa, Imizu-shi, Toyama 939-0398
JAPAN
ohta@pu-toyama.ac.jp

Abstract: Virtualization is widely used owing to its advantages, such as flexibility, scalability, and cost reduction. One important advantage is the decrease in power consumption, which is obtained by concentrating virtual machines (VMs) into a fewer physical machines (PMs). This is done by optimally placing VMs to their hosts. This placement problem is an intractable combinatorial optimization problem. The optimal placement will also change if the load on the VMs changes with time. This change necessitates the migrations of VMs among PMs. The number of executed migrations should be small because migrations offer load on the network. Thus, both power consumption and number of migrations should be minimized. This research examines algorithms that solve this optimization problem. The examined algorithms include two metaheuristics: simulated annealing and tabu search methods. The method previously presented by the author is also tested for comparison. These methods are evaluated through a computer simulation wherein problems are randomly generated.

Key-Words: virtualization; optimization; metaheuristic; algorithm; cloud computing; tabu search

1 Introduction

Currently, virtualization [1] is widely used as the basis of cloud computing owing to its multiple advantages, including high flexibility, scalability, security, and low cost [2]. Multiple virtual machines (VMs) are generally hosted on a physical machine (PM) in a virtualized environment. The computational resources assigned to each VM are provided by the host PM and shared among VMs. The resources of the host PM should not be excessively consumed to run a VM with good performance.

Let us assume that multiple VMs are hosted by multiple PMs and that the load is varied among VMs. Each VM should be placed to its host PM to obtain a satisfactory performance and avoid the excessive consumption of PM resources. The required number or electric power consumption of PMs then depends on the placement of VMs among PMs. The problem of efficiently placing VMs over PMs is a combinatorial optimization problem, which cannot be easily resolved. In a special case, the VM placement optimization becomes a bin-packing problem [3]. Thus, the problem is NP-hard.

The optimal VM placement will change when the load on VMs varies. This necessitates the migrations of VMs among PMs. The live migration technique [4] enables VMs to be moved among PMs without stopping services. However, as migrations offer load on the network, the placement and migration against load changes must be determined by minimizing not the power consumption as well as the network load generated by migrations.

The optimization of the VM placement and migration was reported in [5-8]. Reference [5] aims at minimizing the combination of several efficiency metrics. However, it is unclear whether the objective function used in [5] is practical. The optimization reported in [6, 7] minimizes the power consumption; however, it does not consider the load due to migrations. Meanwhile, the method proposed in [8] optimizes the placement considering both the power consumption and migrations. However, the method assumes that the computational capability of each PM is identical. The optimization should be executed considering the heterogeneity of computational capability because the specification of usable PMs may not be uniform in a real world.

This study investigates the optimization of VM placement and migrations assuming time-varying load, multiple computational resources affecting performance, and heterogeneous PM specifications. The objective function is defined to consider the power consumption and load offered by migrations. As for the algorithm, the study examines two metaheuristics: the simulated annealing and tabu search methods. The method of [8], which has been modified for a heterogeneous PM performance, is also tested. The algorithms are assessed through a computer simulation. The result shows that two metaheuristics provides better solutions than the method of [8].

The paper is organized as follows: Section 2 describes the problem to be tackled in this paper. The examined algorithms are explored in Section 3. Section 4 evaluates these algorithms through a computer simulation. Related work is briefly reviewed in Section 5. Section 6 concludes the study.

2 Problem Description

Suppose that m VMs, which are denoted by VM_1, $VM_2,\dots, VM_m$, are operated. Each VM is hosted by one of the n PMs, which are denoted by $PM_1, PM_2,\dots, PM_n$. The computational capability may be varied among these PMs. Let us assume that one of the PMs has a standard computational capability. Consider that the computational capability of PM_j is η_j $(1 \le j \le n)$ times larger than that of the PM with a standard capability.

The performance of the VMs depends on the consumption of K computational resources indexed as 1, 2, …, K. Let $u_{i,k}(t)$ $(1 \le i \le m,\ 1 \le k \le K)$ denote the consumption of resource k at time t assuming that VM_i runs on the standard capability of PM. The value $u_{i,k}(t)$ is expressed in percent. The resource k of PM_j is consumed by $u_{i,k}(t) / \eta_j$ % by VM_i if VM_i is hosted by PM_j. The load on VM_i is specified by $u_{i,1}(t),\dots, u_{i,K}(t)$.

Let $U_{j,k}(t)$ denote the percentile consumption of resource k on PM_j at time t. Clearly, from the above definition:

$$U_{j,k}(t) = \sum_{i \in \{i \mid \mathrm{VM}_i \text{ is assigned to } \mathrm{PM}_j\}} \frac{u_{i,k}(t)}{\eta_j} \tag{1}$$

The resource consumption $U_{j,k}(t)$ should not be too large to provide a sufficient amount of resources to VMs and achieve a good performance. Thus, this study introduces a constant C, and the VMs are assigned satisfying the following restriction.

$$U_{j,k}(t) \le C \tag{2}$$

The electric power consumption depends on the utilization of computational resources [9]. PM_j can be turned off if no VMs are hosted on PM_j; thus, the power turns to 0. Let $P_j(t)$ denote the electric power consumed by PM_j. $P_j(t)$ is defined as follows to express the abovementioned characteristic:

$$P_j(t) = \begin{cases} 0, & \mathrm{PM}_j \text{ is turned off} \\ P_{j,0} + \sum_{k=1}^{K} \frac{P_{j,k} U_{j,k}(t)}{100}, & \text{otherwise} \end{cases} \tag{3}$$

where $P_{j,0}$ is the portion not affected by the load, and $P_{j,1},\dots, P_{j,K}$ are the coefficients showing how the consumptions of resources 1,…, K affect the power.

The placement of the VMs is expressed by a 0–1 variable $x_{i,j}(t)$ defined as follows:

$$x_{i,j}(t) = \begin{cases} 1, & \text{if } \mathrm{VM}_i \text{ is placed to } \mathrm{PM}_j \text{ at } t \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

Assume that the VM load is given at a discrete time t_0, t_1, t_2, … The problem then is to determine $x_{i,j}(t)$ at $t = t_0, t_1, t_2, \dots$ so as to minimize P_j and migration load as well as satisfy Eq. (2).

The network load offered by migrations is roughly determined by the memory size assigned to the moved VM [10]. The network load is proportional to the number of moved VMs if the memory size is identical for every VM. Let us introduce the 0–1 variable $y_i(t)$ for VM_i and time t to estimate this number. At time t_s (s = 1, 2,…), $y_i(t_s)$ is 1 if VM_i migrates; otherwise, $y_i(t_s)$ is 0. $x_{i,j}(t_s)$ turns to 1 and differs from $x_{i,j}(t_{s-1})$ if VM_i migrates to PM_j. Therefore, $y_i(t_s)$ for $s > 0$ is expressed as follows:

$$y_i(t_s) \ge x_{i,j}(t_s) - x_{i,j}(t_{s-1}),\ 1 \le j \le n \tag{5}$$

At t_0, $y_i(t_0)$ is 0 because no previous placement exists. Let $v(t)$ denote the number of the moved VMs. $v(t)$ is the sum of $y_i(t)$ and is expressed as follows:

$$v(t) = \sum_{i=1}^{m} y_i(t) \tag{6}$$

Let $\mathbf{x}$ denote the vector of decision variables, including $x_{i,j}(t)$, $y_i(t)$, $v(t)$, $U_{j,k}(t)$, and $P_j(t)$. Let us define the objective function $f(t, \mathbf{x})$ as the weighted sum of power consumed by the system and the number of migrations.

$$f(t, \mathbf{x}) = \sum_{j=1}^{n} P_j(t) + w \cdot v(t) \tag{7}$$

where w $(w \ge 0)$ is the weight parameter. The problem is to determine $\mathbf{x}$ that minimizes $f(t_s, \mathbf{x})$ for a given $u_{i,k}(t_s)$ and $x_{i,j}(t_{s-1})$ at time t_s $(s \ge 0)$.

3 Algorithms

3.1 Greedy Method for Initial Placement

The two metaheuristics examined in this study requires an initial solution. We employ a greedy algorithm to obtain an initial solution. For a given

$u_{i,k}(t)$, the method assigns VMs to their hosts as follows:

Algorithm *greedy-fit*
1. $U_{j,k}(t)$:= 0 for all (j, k);
2. V:= set of all VMs.
3. **while** V ≠ Ø **do**
4. max:=–∞;
5. **for each** pair of VM_i in V and PM_j **do**
6. $U^*_k := U_{j,k}(t) + u_{i,k}(t)/\eta_j$ for all k;
7. Compute P_j for resource consumption U^*_k;
8. Compute efficiency metric e_j;
9. **if** $e_j > max$ **then**
10. $max := e_j$;
11. $VM_{best} := VM_i$;
12. $PM_{best} := PM_j$;
    end if
    end for
13. Assign VM_{best} to PM_{best};
14. V:= V – {VM_{best}};
    end while

The efficiency metric e_j is defined for PM_j as follows:

$$e_j = \frac{\eta_j^K \prod_{k=1}^{K} U_k^*}{P_j}, \qquad (8)$$

where U^*_k is the tentative resource utilization obtained assuming that VM_i is assigned to PM_j. With this metric, the priority is higher for the placement that achieves a low power consumption, higher resource utilization, and higher performance. Thus, a good solution is expected.

3.2 Method of Reference [8]

The first examined method is a modified version of the algorithm described in [8]. The method calculates the placement at time t_s by modifying t_{s-1} through two types of migrations:

- Type 1: migrations for overload avoidance, and
- Type 2: migrations for decreasing electric power consumption by integrating VMs into as few PMs as possible.

The solution at t_0 is found by the *greedy-fit* algorithm. The algorithm chooses the source PM, destination PM, and VM to be moved by evaluating the efficiency metric for the migration. The efficiency metric employed in this study is made slightly different from that used in [8] to assess the heterogeneous PM performance. In other words, the PM and VM selection is performed using the metrics of Eq. (8).

3.3 Simulated Annealing

Simulating annealing [11] is a powerful metaheuristic for solving complex optimization problems. This method repeatedly updates a solution by searching a neighborhood of the current solution. In the update process, the neighborhood solution is accepted as a new solution if the objective function decreases. The neighborhood solution is accepted with some probability p even for the increase of the objective function. Let T denote the parameter that controls p. Moreover, let $\mathbf{x}^{now}$ and $\mathbf{x}^{best}$ be the decision variables of the current solution and the best discovered solution, respectively. The method is then written as follows:

Algorithm *simulated-annealing*
1. $\mathbf{x}^{best} := \mathbf{x}^{now} :=$ the output of *greedy-fit*;
2. $T := T_0$;
3. **for** q:= 1 to Q **do**
4. **while** the system is not in equilibrium **do**
5. $\mathbf{x}^{next} :=$ neighborhood of $\mathbf{x}^{now}$;
6. **if** $f(t, \mathbf{x}^{next}) < f(t, \mathbf{x}^{best})$
7. **then** $\mathbf{x}^{best} := \mathbf{x}^{now} := \mathbf{x}^{next}$
8. **else** with probability p, $\mathbf{x}^{now} := \mathbf{x}^{next}$;
   end while
9. $T := \alpha T$;
   end for
10. Output $\mathbf{x}^{best}$ and $f(t, \mathbf{x}^{best})$;

The method enables the solution to escape from a local minimum by allowing the degradation of the tentative solution. The acceptance probability p for the objective function increase is defined by the increase rate of the objective function Δf and temperature T.

$$p = e^{-\Delta f / T} \qquad (6)$$

The increase rate Δf is defined as follows:

$$\Delta f = \frac{f(t, \mathbf{x}^{next}) - f(t, \mathbf{x}^{now})}{f(t, \mathbf{x}^{now})} \qquad (7)$$

Temperature T is first set to a large value T_0. The process is then repeated with a decreasing T. Thus, the acceptance probability p also decreases as implied by Eq. (6). A near-optimal solution is obtained when T becomes sufficiently low.

$\mathbf{x}^{next}$ is generated herein from $\mathbf{x}^{now}$ by randomly executing one of the following methods:

- *Method 1*: A VM is randomly selected. A PM is then randomly selected from the PMs, which are not hosting the selected VM in $\mathbf{x}^{now}$. Subsequently, $\mathbf{x}^{next}$ is created by reassigning the VM to the selected PM.

- *Method 2*: Two VMs hosted by different PMs are randomly chosen from $\mathbf{x}^{now}$. $\mathbf{x}^{next}$ is then created by exchanging the PMs for these VMs.
- *Method 3*: Two VMs hosted by different PMs are randomly chosen from $\mathbf{x}^{now}$. A PM that differs from the hosts of these VMs is also randomly selected. $\mathbf{x}^{next}$ is then generated by reassigning the first VM to the PM that hosts the second VM and reassigning the second VM to the third PM.

The probabilities of selecting methods 1, 2, and 3 were tuned to 0.7, 0.1, and 0.2, respectively, through a simulation. The state for each value of T is judged to be in equilibrium if the neighbor solution is accepted by X times or unaccepted by Y times. T_0 was set to 0.07 in the computer simulation. Parameters X, Y, α, and Q were set to $100mn$, $400mn$, 0.998, and 3000, respectively.

3.3 Tabu Search

Tabu search [12] is another powerful metaheuristic used for optimization problems. This method repeatedly updates a solution by searching for a neighborhood of the current solution. The update is performed according to a rule, which is determined to effectively search for the solution space. That is, recently examined variable changes are recorded in the "tabu" list and avoided. The frequency of a variable change is also considered, and a less frequent change has a higher priority. Even for a change that does not satisfy these rules, the neighborhood solution is accepted if it improves the solution. Thus, the algorithm is written as follows:

Algorithm *tabu-search*

1. $\mathbf{x}^{best}$:= $\mathbf{x}^{now}$:= the output of *greedy-fit*;
2. **for** r:= 1 to R **do**
3. G_1:= {$\mathbf{x}$ | neighborhood of $\mathbf{x}^{now}$ and $\mathbf{x}$ satisfies the rule};
4. G_2:= {$\mathbf{x}$ | neighborhood of $\mathbf{x}^{now}$ and $\mathbf{x}$ does not satisfy the rule};
5. $\mathbf{x}^{now}$:= $\mathbf{x}$ that minimizes $f(t, \mathbf{x})$ for $\mathbf{x}$ in G_1;
6. **if** $f(t, \mathbf{x}) < f(t, \mathbf{x}^{best})$ and $f(t, \mathbf{x}) < f(t, \mathbf{x}^{now})$ for some $\mathbf{x}$ in G_2 **then** $\mathbf{x}^{now}$:= $\mathbf{x}$;
7. **if** $f(t, \mathbf{x}^{now}) < f(t, \mathbf{x}^{best})$ **then** $\mathbf{x}^{best}$:= $\mathbf{x}^{now}$;

   end for
8. Output $\mathbf{x}^{best}$ and $f(t, \mathbf{x}^{best})$;

In this study, steps 4 and 5 are executed by randomly selecting one of the following methods with equal probability:

- Method 1: Sets G_1 and G_2 are constructed by every pair of VM$_i$ and PM$_j$ that does not host VM$_i$ in $\mathbf{x}^{now}$. $\mathbf{x}$ is obtained for each pair by reassigning VM$_i$ to PM$_j$. $\mathbf{x}$ is added to G_2 if VM$_i$ is listed in the tabu table or the frequency of assigning VM$_i$ to PM$_j$ excesses a threshold; otherwise, $\mathbf{x}$ is added to G_1.
- Method 2: A neighborhood is found by every pair of two VMs hosted by different PMs in $\mathbf{x}^{now}$. $\mathbf{x}$ is obtained for such a pair by changing the hosting PMs. $\mathbf{x}$ is add to G_2 if the change from $\mathbf{x}^{now}$ to $\mathbf{x}$ is included in the tabu list; otherwise, $\mathbf{x}$ is added to G_1.

The tabu table and frequency for the accepted neighbor solution are updated. The tabu table used in Method 1 lists the VMs recently used in creating the new solution. Its size is denoted by S_1. The table of Method 2 also lists the VMs affected in the recently accepted neighbor solution. The size is denoted by S_2. In Method 1, let $F_{1,i,j}$ and R_1 denote the frequency of reassigning VM$_i$ to PM$_j$ and the frequency of executing the method, respectively. The frequency criteria for creating G_1 as follows:

$$F_{1,i,j} < \left\lceil \frac{\beta R_1}{mn} \right\rceil \quad (10)$$

where β is a constant, and $\lceil \bullet \rceil$ is the smallest integer that is not less than $\bullet$.

Parameters S_1, S_2, β, and R were set to 7, 8, 2.8, and 6×10^6, respectively, in the computer simulation.

4 Evaluation

The optimization algorithms were evaluated through a computer simulation. The algorithms were executed for randomly generated problems. The obtained solutions were then compared among the algorithms.

The simulation model is specified as follows: the number of VMs, m, was 40, while that of PMs, n, was 20. The number of computational resources was 2. The constant C was 90. The performance parameter η_j and the electrical power coefficients $P_{j,k}$ for PM$_j$ was set as summarized in Table 1.

Table 1 PM$_j$ parameters.

Range of j	η_j	$P_{j,0}$	$P_{j,1}$	$P_{j,2}$
$1 \le j \le 5$	1.0	80.0	40.0	10.0
$6 \le j \le 10$	1.5	120.0	60.0	20.0
$11 \le j \le 20$	1.0	120.0	60.0	20.0

The load on the VMs is provided at times $t_0, t_1, \ldots, t_{14}$. The placement is determined when the load is provided. The load on VM$_i$ is specified by $u_{i,k}(t)$. The base value denoted by $\tilde{u}_{i,k}$ was randomly selected from integers in [1, 50] for every pair of i and k with equal probability to determine $u_{i,k}(t)$. $u_{i,k}(t)$ was then determined as summarized in Table 2.

Table 2 Resource consumption by VM.

VMs	$u_{i,k}(t)$
$VM_1, \ldots, VM_{20}$	Randomly selected integer from $[1, \tilde{u}_{i,k}]$
$VM_{21}, \ldots, VM_{40}$	$\tilde{u}_{i,k}$, for $t_5, t_6, \ldots, t_9$ $\tilde{u}_{i,k} / 2$, otherwise

A total of 30 problems were generated by changing the random seed for $\tilde{u}_{i,k}$ and $u_{i,k}(t)$. The algorithms described in Section 3 were programmed in the C language and executed for the problems. The programs were executed on a Linux (CentOS 7) PC that held Core i5 CPU and 16GB RAM.

The optimization was also formulated into a mixed-integer programming problem for comparison and solved by an optimization software [13]. The formulation is similar to the one presented in [8], except for using parameter η_j. GAMS/CPLEX [14], which runs on MS Windows PC, was used as the optimization software. This approach is referred to as the MIP.

Fig.1 compares the objective function value obtained for each method. The x axis is the weight parameter, w, while the y axis is the objective function value. The value is the average of the sum for $t_0, t_1, \ldots, t_{14}$ over 30 problems.

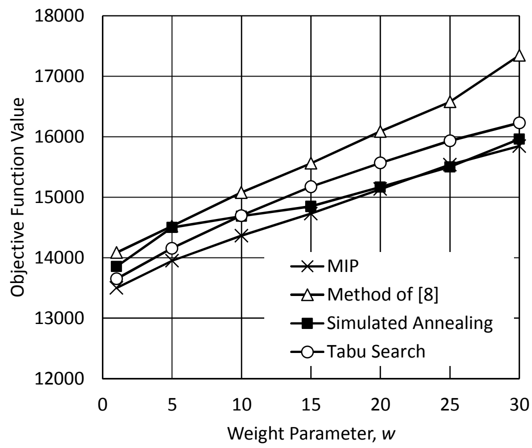

Fig.1 Objective function values obtained by the algorithms.

Fig.1 clearly shows that the simulated annealing and tabu search methods provide good solutions, which are very close to those obtained by the mixed integer programming. These metaheuristics are superior over the algorithm of [8] in solution goodness. The tabu search method provides better solutions for $w \leq 10$, whereas the simulated annealing method is superior for $w \geq 15$. Thus, it is inconclusive which of these two metaheuristics is more advantageous. The best method should be determined considering which of the power and network load is more important.

Fig.2 shows a plot for the power consumption against the number of migrations for different w values. The figure clearly shows that the number of migrations is larger for the simulated annealing method to obtain smaller power consumption (i.e., a smaller value of w). The method yields worse solutions than the MIP or tabu search methods for small values of w because of this characteristic. By contrast, the simulated annealing method yields a solution that is very close to that of the MIP approach if w is large. The reason for this behavior of the simulated annealing method is unclear. A further study is needed to discuss this problem.

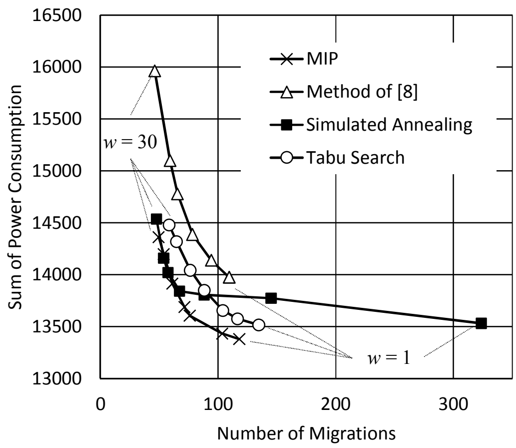

Fig.2 Relation between power consumption and the number of migrations.

Table 3 compares the computational time. The time is the average over 15 time periods of 30 problems. The computational time is much larger for the simulated annealing and tabu search methods than the method of [8]. However, the time will be acceptable if the placement interval is longer than several minutes. The computational time for the one-time period for the MIP approach becomes larger than 3 h for some problems. Thus, the assessed heuristics are more advantageous and practical than the MIP approach in computational time.

Table 3 Computational time for one time period.

Method of [8]	Simulated annealing	Tabu search
0.000311 s	100.60 s	52.82 s

5 Related Work

The optimization of the VM placement and migration was reported in several studies [3, 5-8, 14]. Reference

[3] explored multiple aspects of the problem: demand characteristics, benefit evaluation of the dynamic VM placement, demand forecasting, and placement algorithm. The algorithm of [3] aimed to reduce the number of PMs as well as satisfy the service level agreement.

The method of [5] considered three efficiency metrics associated with temperature, performance, and electrical power. The method decides the initial and dynamic VM placement to maximize the utility that combined these metrics. The considered computational resources include a CPU, I/O, and network. However, the validity of their utility definition is unclear.

Reference [6] presented a VM placement optimization method assuming heterogeneous power consumption and computational capability represented by the MIPS. The objective of optimization is to minimize power consumption. Thus, the method does not consider the network load offered by migration. Moreover, the method only considers CPU utilization as the resource that affects the performance.

Reference [7] applied the ant colony heuristic to the VM placement problem. This method also did not consider the network load offered by migration.

The author's previous research [8] aimed at optimizing both the power consumption and the migration load. However, the method assumes a uniform computational ability for PMs. In addition, the algorithm does not necessarily provide good solutions compared with the MIP approach.

6 Conclusion

This study investigated algorithms to optimize the placement and migrations of VMs over PMs. The algorithms decided on the placement and migration to minimize the cost, assuming the heterogeneous power consumption and computational performance for PMs. The cost was defined by the weighted sum of power and the number of migrations. The examined algorithms included the method of [8] and two metaheuristics: simulated annealing and tabu search methods. These methods were evaluated through a computer simulation. The results showed that the metaheuristics yielded a better solution than the method of [8].

References:

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauery, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *in Proc. SOSP'03*, Bolton Landing, New York, USA, 2003, pp. 164-177.

[2] J. Sahoo, S. Mohapatra, and R. Lath, "Virtualization: a survey on concepts, taxonomy and associated security issues," in ICCNT 2010, Bangkok, Thailand, 2010, pp. 222-226.

[3] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing SLA violations," *in Proc. IM'07*, Munich, Germany, 2007, pp. 119-128.

[4] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," *in Proc. USENIX NSDI'05*, Boston, MA, USA, 2005, pp. 273-286.

[5] J. Xu and J. A. B. Fortes, "A multi-objective approach to virtual machine management in datacenters," *in Proc. ICAC'11*, Karlsruhe, Germany, 2011, pp. 225-234.

[6] D. G. D. Lago, E. R. M. Madeira, and L. F. Bittencourt, "Power-aware virtual machine scheduling on clouds using active cooling control and DVFS," *in Proc. MGC2011*, Lisbon, Portugal, 2011.

[7] S. R. M. Amarante, F. M. Roberto, and A. R. Cardos, "Using the multiple knapsack problem to model the problem of virtual machine allocation in cloud computing," *in Proc. CIT 2013*, Sidney, Australia, 2013, pp. 476-483.

[8] S. Ohta, "Strict and heuristic optimization of virtual machine placement and migration," *in Proc. WSEAS CEA'15*, Dubai, UAE, 2015, pp. 42-51.

[9] S. Ohta, "Obtaining the knowledge of a server performance from non-intrusively measurable metrics," *International Journal of Engineering and Technology Innovation*, 6, 2, Apr. 2016, pp. 135-151.

[10] T. Tanabe and S. Ohta, "Experimental evaluation of network load caused by live migration," in *Proc. 2015 Joint Conference of Hokuriku Chapters of Electrical Societies*, Nonoichi, Japan, 2015, E-31 (in Japanese).

[11] S. Kirkpatrick, C. D. Gellat, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, 220, 4598, May 1983, pp. 671-680.

[12] F. Glover, "Tabu search: a tutorial," *Interfaces*, 20, 4, Jul. 1990, pp.74-94.

[13] J. J. More and S. J. Wright, *Optimization Software Guide*, Philadelphia: SIAM, 1993.

[14] GAMS, https://www.gams.com, 2017.