Optimization Techniques for Virtual Machine Placement and Migration

SATORU OHTA

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

Abstract: Virtualization is widely used due to its flexibility, scalability, and cost reduction. In virtualization, virtual machines (VM) should be placed optimally onto physical machines (PM) to reduce power consumption and avoid resource shortages. VM placement is an intractable combinatorial optimization problem. Moreover, optimal VM placement changes if the loads on VMs change over time. This means that load change necessitates VM migration among PMs. Since VM migration incurs network load, migration frequency must be small. Thus, both power consumption and the number of migrations should be minimized when determining VM placement. This research formulates the problem and examines algorithms that solve it. The examined algorithms include two metaheuristics, i.e., simulated annealing and tabu search methods. A method previously presented by the author was also tested for comparison. These methods were evaluated through computer simulation.

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

1 Introduction

Currently, virtualization [1] is widely used as the basis of cloud computing due to its multiple advantages, including high flexibility, scalability, security, and low cost [2]. Generally, in a virtualized environment, multiple virtual machines (VM) are hosted on a physical machine (PM). Shared computational resources are assigned to each VM by the host PM. However, the host PM's resources should not be consumed excessively for sufficient VM performance.

Assume that multiple VMs are hosted on multiple PMs and that the load varies among VMs. Each VM should be allocated sufficient resources to ensure satisfactory performance. The number of required PMs and power requirements depend on the placement of VMs among the PMs. Efficient placement of VMs among PMs is a combinatorial optimization problem that cannot be resolved easily. In some cases, VM placement optimization is equivalent to a bin-packing problem [3]. Thus, the problem is NP-hard.

Optimal VM placement will change when the loads on VMs vary over time. This requires migration of VMs among PMs. The live migration technique [4] enables VMs to be moved among PMs without interrupting services. However, VM migration incurs a load on the network. Thus, placement and migration relative to load changes must be determined so as to minimize power consumption and network load.

Methods to optimize VM placement and migration have been reported in the literature [5-8]. Reference [5] attempted to minimize several efficiency metrics. However, it is unclear whether the objective function used in [5] is practical. The optimization methods reported in [6, 7] minimize power consumption. However, they do not consider load incurred by migration. The method proposed in the author's previous work [8] optimizes VM placement considering both power consumption and network load incurred by migration. However, this method assumes the computational capability of each PM is identical. Optimization should consider the heterogeneity of computational capability because PMs may have different specifications in a real-world environment. In addition, a solution obtained by the author's previously proposed method [8] may not be sufficiently close to the strict optimum.

This study investigates VM placement and migration optimization assuming a time-varying load, multiple computational resources that affect performance, and heterogeneous PM specifications. The objective function considers power consumption and the network load incurred by migration. This study examines two metaheuristics, i.e., simulated annealing and tabu search methods. These methods are known to be effective for complex optimization problems, such as VM placement. The previous method [8], which has been modified for heterogeneous PM capability, is also tested. The algorithms are assessed through computer simulation,

and the results show that the simulated annealing and tabu search metaheuristics provide better solutions than the method of [8].

The remainder of this paper is organized as follows. Section 2 describes the problem. The examined algorithms are discussed in Section 3. Section 4 evaluates the algorithms through computer simulation. Related work is reviewed in Section 5, and conclusions are provided in Section 6.

2 Problem Description

Assume m VMs denoted by VM_1, VM_2,…, VM_m. Each VM is hosted by one of n PMs denoted by PM_1, PM_2,…, PM_n. The computational capability of the PMs may vary. Consider that the computational capability of a certain PM, for example PM_1, is the standard. Then, the computational capability of PM_j is η_j $(1 \le j \le n)$ times greater than that of the PM with standard capability.

The performance of a VM depends on the consumption of K computational resources, e.g., CPU, memory, and network I/O, 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 PM. Note that $u_{i,k}(t)$ is expressed as a percentage. If VM_i is hosted by PM_j, it consumes $u_{i,k}(t)/\eta_j$ % of resource k of PM_j. The load on VM_i is specified by resource consumptions $u_{i,1}(t)$,…, $u_{i,K}(t)$.

Let $U_{j,k}(t)$ denote the percentile consumption of resource k on PM_j at time t. Then, $U_{j,k}(t)$ is expressed as follows.

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

Resource consumption $U_{j,k}(t)$ should not be greater than that required to provide VMs with sufficient resources to achieve good performance. Thus, this study introduces a constant C, and the VMs are assigned resources such that the following constraint is satisfied.

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

Power consumption depends on the utilization of computational resources [9]. Also, PM_j can be turned off (i.e., no power is consumed) if it is not hosting any VMs. Here let $P_j(t)$ denote the power consumed by PM_j. Power $P_j(t)$ is expressed as follows:

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

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

The placement of VMs is expressed by a binary 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, … Then, the problem is to determine $x_{i,j}(t)$ at $t = t_0$, t_1, t_2, … such that $P_j(t)$ and the migration load are minimized and Eq. (2) is satisfied.

The network load incurred by migration is roughly determined by the memory size assigned to the VM [10]. Consequently, if memory size is identical for each VM, the network load is proportional to the number of migrated VMs. Here, binary variable $y_i(t)$ for VM_i and time t is used to estimate the number of migrating VMs. At time t_s (s = 1, 2,…), $y_i(t_s)$ is 1 if VM_i is being migrated. Otherwise, $y_i(t_s)$ is 0. If VM_i migrates to PM_j, $x_{i,j}(t_s)$ is 1 and differs from $x_{i,j}(t_{s-1})$. 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 migrated VMs. $v(t)$ is the sum of $y_i(t)$, which 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 $P_j(t)$. The objective function $f(\mathbf{x})$ is defined as the weighted sum of the power consumed by the system and the number of migrations:

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

where w $(w \ge 0)$ is the weight parameter. Then, the problem is to determine $\mathbf{x}$ that minimizes $f(\mathbf{x})$ for a given $u_{i,k}(t_s)$ and $x_{i,j}(t_{s-1})$ at time t_s $(s \ge 0)$. This is formulated as the following mixed integer programming (MIP) problem:

$$\text{minimize } f(\mathbf{x}) = \sum_{j=1}^{n} P_j(t_s) + w \cdot \sum_{i=1}^{m} y_i(t_s) \tag{8}$$

subject to

$$U_{j,k}(t_s) = \sum_{i=1}^{m} \frac{u_{i,k}(t_s) x_{i,j}(t_s)}{\eta_j} \le C, \quad 1 \le j \le n, 1 \le k \le K \quad (9)$$

$$\sum_{j=1}^{n} x_{i,j}(t_s) = 1, \quad 1 \le i \le m \quad (10)$$

$$y_i(t_s) \ge x_{i,j}(t_s) - x_{i,j}(t_{s-1}), 1 \le j \le n \quad (11)$$

$$P_j(t_s) = P_{j,0} \sum_{i=1}^{m} x_{i,j}(t_s) + \sum_{k=1}^{K} \frac{P_{j,k} U_{j,k}(t_s)}{100}, \quad 1 \le j \le n \quad (12)$$

where $s > 0$. When $s = 0$, the restriction of Eq. (11) is omitted and $y_i(t_0)$ is set to 0. This means that migrations do not need to be considered when the initial placement is determined at t_0.

In the above formulation, Eq. (9) is equivalent to Eqs. (1) and (2). Thus, the equation represents a restriction that avoids resource shortage. Eq. (10) requires all VMs to be placed on a PM. Eq. (11) is the same as Eq. (5) and determines the number of migrations. Eq. (12) is an alternate expression of Eq. (3), i.e., it is derived by rewriting Eq. (3) using $x_{i,j}(t_s)$. This equation determines the power consumption of a PM.

3 Algorithms

3.1 Greedy Method for Initial Placement

The two metaheuristics examined in this study require an initial solution, which is obtained using a greedy algorithm. For a given $u_{i,k}(t)$, the greedy algorithm 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** $V \neq \emptyset$ **do**
4. max:= $-\infty$;
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:

$$e_j = \frac{\eta_j{}^K \prod_{k=1}^{K} U^*_k}{P_j}, \quad (13)$$

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

3.2 Previous Method [8]

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

- Type 1: migration that avoids overload
- Type 2: migration that reduces power consumption by deploying VMs onto as few PMs as possible

The solution at t_0 is found by the *Greedy-Fit* algorithm. At t_1, t_2, …, the algorithm selects the source PM, destination PM, and VM to be moved by evaluating the efficiency metric for the migration. Specifically, in Type 1, the metric difference caused by migration is estimated for all possible combinations of an overloaded source PM, a destination PM with spare resource capacity, and a VM hosted by the source PM. Then, migration is performed for the combination that maximizes the metric difference. This is repeated until overload is eliminated from all PMs. In Type 2, the metric increase is estimated for all combinations of a source PM, a destination PM, and a VM hosted by the source PM. Again, the migration is executed such that the metric increase is maximized. This is repeated until no combination can increase the metric.

The efficiency metric employed in this study is slightly different from that used in [8] in order to assess heterogeneous PM performance. In other words, PM and VM selection is performed using the metrics in Eq. (13).

3.3 Simulated Annealing

Simulated 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 when the objective function increases. Here, let T denote the parameter that controls p. This parameter is called temperature. 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 described 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. **if** $q = Q_1$ or Q_2 **then** $T := T_1$;
5. **while** the system is not in equilibrium **do**
6. $\mathbf{x}^{next} :=$ neighborhood of $\mathbf{x}^{now}$;
7. **if** $f(\mathbf{x}^{next}) < f(\mathbf{x}^{best})$
8. **then** $\mathbf{x}^{best} := \mathbf{x}^{now} := \mathbf{x}^{next}$
9. **else** with probability p, $\mathbf{x}^{now} := \mathbf{x}^{next}$;
   end while
10. $T := \alpha T$;
    end for
11. Output $\mathbf{x}^{best}$ and $f(\mathbf{x}^{best})$;

In the algorithm, Q_1 and Q_2 are integers, where $Q_1 < Q_2 < Q$, parameter α is a real number, where $0 < \alpha < 1$, T_0 is a positive constant, and T_1 is a constant, where $T_1 < T_0$.

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

$$p = e^{-\Delta f / T} \tag{14}$$

The increase rate Δf is defined as follows.

$$\Delta f = \frac{f(\mathbf{x}^{next}) - f(\mathbf{x}^{now})}{f(\mathbf{x}^{now})} \tag{15}$$

Temperature T is first set to a large value T_0. The process is then repeated with decreasing T. Thus, the acceptance probability p also decreases, as implied by Eq. (14). As step 10 suggests, T is decreased gradually with the completion of steps 5-9. In other words, the system is cooled after it reaches equilibrium. A near-optimal solution is obtained when T becomes sufficiently low.

A neighborhood solution $\mathbf{x}^{next}$ is generated from $\mathbf{x}^{now}$ by randomly executing one of the following methods.

- *Method 1*: A VM is selected randomly. A PM is then selected randomly from the PMs 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 chosen randomly from $\mathbf{x}^{now}$. Then, $\mathbf{x}^{next}$ is created by exchanging the PMs for these VMs.
- *Method 3*: Two VMs hosted by different PMs are chosen randomly from $\mathbf{x}^{now}$. A PM that differs from the hosts is also selected randomly. Then, $\mathbf{x}^{next}$ is generated by reassigning the first VM to the PM that hosts the second VM and reassigning the second VM to the third PM.

In a simulation, the probabilities of selecting Methods 1, 2, and 3 were tuned to 0.5, 0.2, and 0.3, respectively. The state for each value of T was considered to be in equilibrium if the neighbor solution is accepted X times or unaccepted Y times. Initial temperature T_0 was set to 0.07 in the simulation. Parameters X, Y, α, and Q were set to $100mn$, $400mn$, 0.995, and 3000, respectively.

In the above algorithm, step 4 provides a "re-annealing" process. This process increases the temperature to T_1 after the system is cooled sufficiently. Then, the cooling process is resumed from temperature T_1. The behavior of the temperature is shown in Fig. 1.

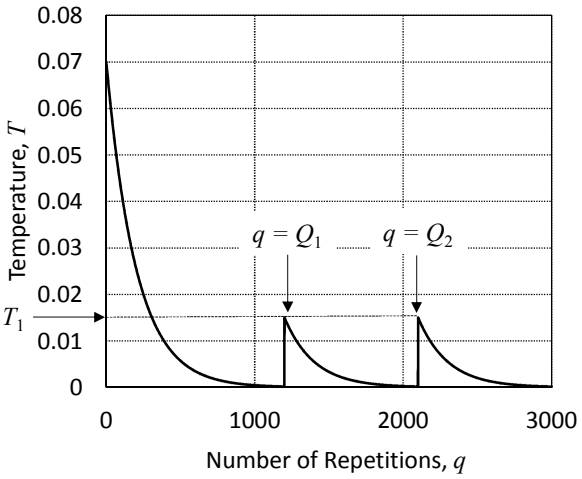

Fig. 1 Temperature behavior employed in the simulated annealing method

In the re-annealing process, parameters Q_1 and Q_2 determine when the temperature increases to T_1. Here, the values of Q_1, Q_2, and T_1 were tuned to 1200, 2100, and 0.015, respectively. The re-annealing process is considerably effective at improving solution goodness without increasing computational time.

3.4 Tabu Search

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

Algorithm *Tabu-Search*

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

In this study, steps 3 and 4 are executed by randomly selecting one of the following methods at equal probability.

- Method 1: Sets G_1 and G_2 are constructed by each pair of VM_i and PM_j that does not host VM_i in $\mathbf{x}^{now}$. For each pair, $\mathbf{x}$ is obtained by reassigning VM_i to PM_j. Solution $\mathbf{x}$ is added to G_2 if VM_i is listed in the tabu table or the frequency of assigning VM_i to PM_j exceeds a threshold. Otherwise, $\mathbf{x}$ is added to G_1.
- Method 2: A neighborhood is found by each pair of VMs hosted by different PMs in $\mathbf{x}^{now}$. For such a pair, $\mathbf{x}$ is obtained by changing the hosting PMs. Solution $\mathbf{x}$ is added to G_2 if the VMs are included in the tabu list. Otherwise, $\mathbf{x}$ is added to G_1.

The tabu table and frequency are updated after steps 3-7 are completed. The tabu table used in Method 1 lists the VMs recently used to create the new solution. Its size is denoted by S_1. The tabu table of Method 2 also lists the VMs affected in the recently accepted neighborhood solution. The size is denoted by S_2. In Method 1, let variable $F_{i,j}$ 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 is expressed as follows:

$$F_{1,i,j} < \left\lceil \frac{\beta R_1}{mn} \right\rceil \tag{16}$$

where β is a constant and $\lceil \bullet \rceil$ is the smallest integer not less than $\bullet$. The probability of randomly selecting a combination of VM_i and PM_j is $(mn)^{-1}$. Thus, Eq. (16) omits the combination of VM_i and PM_j from G_1 if it has been used β times more frequently for a new neighborhood than the average.

Parameters S_1, S_2, β, and R were tuned to 7, 8, 2.6, and 6×10^6, respectively, in the simulation.

4 Evaluation

The optimization algorithms were evaluated through a computer simulation, and the algorithms were executed for randomly generated problems. The solutions obtained by each algorithm were then compared.

The simulation model is specified as follows. The number of VMs, m, was 40 and the number of PMs, n, was 20. The number of computational resources was two, and constant C was set to 90. The performance parameter η_j and power coefficients $P_{j,k}$ for PM_j were set as shown 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 was provided at times t_0, t_1, …, t_{14}, and VM placement was determined for these times. The load on VM_i was specified by $u_{i,k}(t)$. The base value denoted by $\tilde{u}_{i,k}$ was used to determine $u_{i,k}(t)$. This base value is an integer randomly selected in the range [1, 50] for each pair of i and k with equal probability. Load $u_{i,k}(t)$ was then determined as summarized in Table 2.

Table 2 VM resource consumption

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

Thirty 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 generated problems. The programs were executed on a Linux (CentOS 7) PC with a Core i5 CPU and 16 GB RAM.

The generated problems were also solved by optimization software [13] for comparison. This was performed by formulating each problem as a MIP problem according to Eqs. (8)-(12) and feeding it to GAMS/CPLEX [14], which runs on the Microsoft Windows operating system.

Fig. 2 compares the objective function value obtained for each method. Here, the x axis is the weight parameter w and 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.

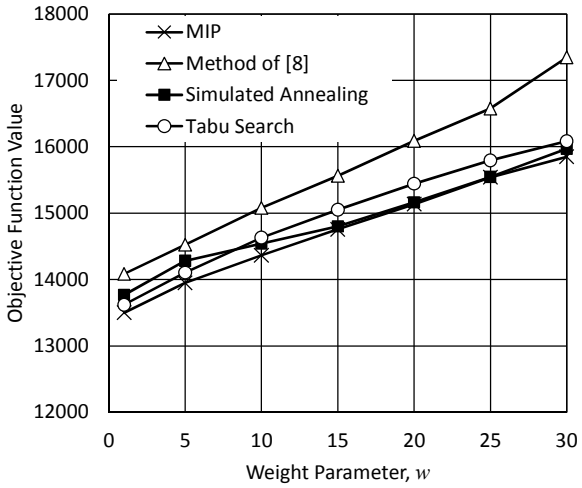

Fig. 2 Objective function values

Fig. 2 clearly shows that the simulated annealing and tabu search methods provide good solutions that are very close to those obtained by MIP. These metaheuristics are superior to the algorithm of [8] relative to solution goodness. The tabu search method provides better solutions when $w \leq 10$, whereas the simulated annealing method is superior when $w \geq 15$. Thus, it is inconclusive which metaheuristic is more advantageous. The best method should be determined by considering whether power or network load is more important.

Fig. 3 plots power consumption against the number of migrations for different w values. The power consumption and the number of migrations are the average of the sum for $t_0, t_1, \ldots, t_{14}$. As can be seen, the number of migrations is greater for the simulated annealing method to obtain smaller power consumption (i.e., a smaller w value). This method yields worse solutions than the MIP and tabu search methods for small w values due to this characteristic. In contrast, the simulated annealing method yields a solution that is very close to that of the MIP approach when w is large. However, the reason for this behavior remains unclear. Thus, further study is required to examine this problem.

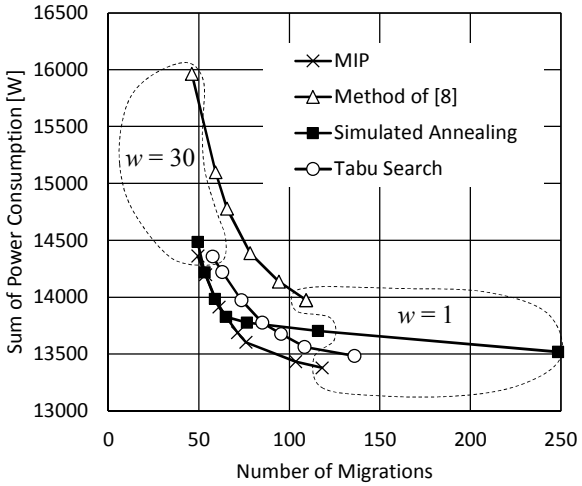

Fig.3 Relation between power consumption and number of migrations

Table 3 compares computational time. The computational time was measured using the Linux "`time`" command for the execution of each algorithm. The time is the average over 15 time periods for 30 problems. As can been seen, the computational time is much greater for the simulated annealing and tabu search methods than the method proposed in [8]. However, the time is acceptable when the placement interval is greater than several minutes. Note that the computational time for a single time period for the MIP approach is longer than three hours for some problems. Thus, the assessed metaheuristics are more advantageous and practical than the MIP approach relative to computational time.

Table 3 Average computational time to obtain a solution for a single time period

Method of [8]	Simulated annealing	Tabu search
0.000311 s	123.34 s	53.15 s

5 Related Work

VM placement and migration optimization has been reported in several studies [3, 5-8, 15-17]. Reference [3] explored multiple aspects of the problem, i.e., demand characteristics, a benefit evaluation of dynamic VM placement, demand forecasting, and the placement algorithm. The algorithm proposed in [3]

attempts to reduce the number of PMs and satisfy a given service level agreement.

The method proposed in [5] considered temperature, performance, and power efficiency metrics. That method determines initial and dynamic VM placement to maximize the utility function that combines these metrics. The considered computational resources include a CPU, I/O, and a 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 MIPS (million instructions per second). The objective of this optimization is to minimize only power consumption. Thus, the method does not consider the network load incurred by migration. Moreover, the method considers CPU utilization as the sole computational resource. However, other resources can influence performance.

Reference [7] applied the ant colony heuristic to the VM placement problem. Similarly, this method did not consider the network load incurred by migration.

The author's previous research [8] attempted to optimize both power consumption and migration load. However, the method assumes uniform computational capability for all PMs. In addition, the employed algorithm does not necessarily provide good solutions compared to the MIP approach.

Some studies have dealt with optimal VM placement from a quite different perspective. For example, the purpose of [15, 16] is to determine VM placement that minimizes data access latency. In these studies, the power consumption and load by migration were not considered even though they are practically important.

Reference [17] considered average power consumption and wastage balance between the CPU and memory. However, this may not be relevant because it is not always necessary to consume two resources equally to obtain good performance or less power consumption. In addition, in some cases, the performance of a system can depend on three or more resources, e.g., CPU, memory, and network I/O. It is unclear how the method in [17] treats such cases.

6 Conclusion

This study has investigated algorithms to optimize the placement and migration of VMs among PMs. The algorithms determine placement and migration to minimize cost assuming PMs have heterogeneous power consumption and computational performance. In this study, cost was defined by the weighted sum of power and the number of migrations. The examined algorithms included the method proposed in [8] and two metaheuristics, i.e., simulated annealing and tabu search methods. These methods were evaluated in a computer simulation. The simulation results demonstrate that the metaheuristics obtained better solutions than the previously proposed method.

Acknowledgment

The author would like to thank Takuma Iwami for parameter tuning of metaheuristics. The author would also like to thank Enago (www.enago.jp) for the English language review.

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 *Proc. 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. IEEE 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*, Vol.6, No.2, 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*, Vol.220, No.4598, 1983, pp. 671-680.
[12] F. Glover, "Tabu search: a tutorial," *Interfaces*, Vol.20, No.4, 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.
[15] M. Alicherry and T. V. Lakshman, "Optimizing data access latencies in cloud systems by intelligent virtual machine placement," in *Proc. IEEE INFOCOM 2013*, Turin, Italy, 2013, pp. 647-655.
[16] J. Kuo, H. Yang, and M. Tsai, "Optimal approximation algorithm of virtual machine placement for data latency minimization in cloud systems," in *Proc. INFOCOM 2014*, Toronto, ON, Canada, 2014, pp.1303-1311.
[17] Y. Gao, H. Guan, Z. Qi, Y. Hou, and L. Liu, "A multi-objective ant colony system algorithm for virtual machine placement in cloud computing," Journal of Computer and Systems Science, Vol.79, No.8, 2013, pp. 1230-1242.