A NSGA-II-based Approach for Service Resource Allocation in Cloud

Boxiong Tan, Hui Ma, Yi Mei
Victoria University of Wellington
Wellington, New Zealand
Email: {Boxiong.tan, Hui.Ma, Yi.Mei}@ecs.vuw.ac.nz

***Abstract*—Web service and Cloud computing have significantly reformed the software industry. The need for web service allocation in the cloud environment is increasing dramatically. In order to reduce the cost for service providers as well as improve the utilization of cloud resource for cloud providers, this paper formulates the web service resource allocation in cloud environment problem as a two-level multi-objective bin packing problem. It proposes a NSGA-II-based algorithm with specifically designed genetic operators. We are compared with two varieties of the algorithm. The results show that the proposed algorithm can provide reasonably good results with low violation rate.**

I. Introduction

Service Oriented Architecture (SOA) and Cloud computing have significantly reformed the software industry. SOA provides a decentralized application architecture which allows software composition and reuse in a large, global scale. Meanwhile, Cloud computing provides a scalable, reliable, and flexible infrastructure to web services.

As the dramatic increase of web services and cloud facilities, the management of resources has become a critical issue. In recent years, as the power bill has become the largest fraction of the operating cost of Cloud facility [1], to reduce power consumption has become a paramount concern for Cloud service providers. In order to achieve that, a common approach is to re-allocate web services to a minimum number of physical machines (PMs) [2]. Therefore, idle computing servers are turned down or put into save mode. This optimization process, often called *consolidation* involves with two levels of delivery mode, Software as a service (SaaS) and Infrastructure as a service (IaaS). Because of the complexity, consolidation tasks for IaaS and SaaS are often considered as separated tasks with different objectives. For SaaS, the challenges concentrate on satisfying the Service Level Agreements (SLAs) with unpredictable requests using a minimum amount of resources. Whereas, for IaaS, the challenges are the VM migrations and energy conservation.

There are extensive algorithms proposed for SaaS and IaaS levels of resource allocation [3], [4]. Ref. [5] proposes a heuristic algorithm for service consolidation in a set of servers with minimizing costs while avoiding the overload of server and satisfying end-to-end response time constraints.

Ref. [6] proposes two algorithms for energy efficient scheduling of VMs in Cloud, including an exact VM allocation algorithm which is an extended Bin-Packing approach, and a migration algorithm based on integer linear programming.

However, as the two levels of resource allocation are interact with each other, we believe they cannot be separated. They should be considered as one global optimization with multi-objectives from the perspectives of both service providers and cloud providers. Therefore, in this paper, we first propose a model for solving *service resource allocation in Cloud (SRAC)*. Secondly, we propose a NSGA-II-based multi-objective algorithm with specifically designed operators to solve the problem. The two objectives are:

1) propose a model for solving IaaS and SaaS resource allocation together
2) propose a NSGA-II-based algorithm to solve SRAC.

The rest of the paper is organized as follows. Section II discusses the traditional approaches for IaaS and SaaS and the power model for VM allocation. It will also introduce related works of evolutionary multi-objective optimization techniques. Section III describes the definition of the SRAC problem. Section IV introduces the representation and genetic operators for SRAC problem. Section V illustrates the experiment design, results and discussions. Section VI draws a conclusion and discusses the future work.

II. Background

A. Traditional approaches

Ref. [7] proposes a single-objective genetic algorithm to solve placement of service (SaaS) on physical machines. Their major contributions are three-fold. Firstly, they consider web services as a workflow and optimize the makespan of a workflow. Secondly, they design a representation to the problem. Thirdly, they do not only consider computing nodes, but storage nodes as well.

Ref. [8] develops a *Resource-Allocation-Throughput (RAT)* model for web service allocation. The *RAT model* mainly defines several important variables for an atomic service which represents a software component. Based on this model, firstly, an atomic service's throughput equals its coming rate if the resources of the allocated VM are not exhausted. Secondly, increasing the coming rate will also increase an atomic service's throughput until the allocated resource is exhausted. Thirdly, when the resource is exhausted, the throughput will not increase as request increasing. At this time, the virtual machine reaches its capacity.

Anton Beloglazov et al. [9] propose two algorithms for VM allocation. The first one is a bin-packing algorithm, called

978-1-5090-4601-0/17/$31.00

Modified Best Fit decreasing (MBFD) which is used when a new VM allocation request arrives. The second algorithm, named Minimization of Migration, is used to adjust the current VMs allocation according to the CPU utilization of a physical machine. Their experiments have shown that these methods lead to a substantial reduction of energy consumption in Cloud data centers.

B. Power Model

Shekhar's research [10] is one of the earliest in energy aware consolidation for cloud computing. They conduct experiments of independent applications running in physical machines. They explain that CPU utilization and disk utilization are the key factors affecting the energy consumption. They also find that only consolidating services into the minimum number of physical machines does not necessarily achieve energy saving, because the service performance degradation leads to a longer execution time, which increases the energy consumption.

Bohra [11] develops an energy model to profile the power of a VM. They monitor the sub-components of a VM which includes: CPU, cache, disk, and DRAM and propose a linear model (Eq 1). Total power consumption is a linear combination of the power consumption of CPU, cache, DRAM and disk. The parameters α and β are determined based on the observations of machine running CPU and IO intensive jobs.

$$P_{(total)} = \alpha P_{\{CPU,cache\}} + \beta P_{\{DRAM,disk\}} \tag{1}$$

Although this model can achieve an average of 93% of accuracy, it is hard to be employed in solving SRAC problem, for the lack of data.

Anton Beloglazov et al. [9] propose a comprehensive energy model for energy-aware resource allocation problem (Eq 2). P_{max} is the maximum power consumption when a virtual machine is fully utilized; k is the fraction of power consumed by the idle server (i.e. 70%); and u is the CPU utilization. This linear relationship between power consumption and CPU utilization is also observed by [12], [13].

$$P(u) = k \cdot P_{max} + (1-k) \cdot P_{max} \cdot u \tag{2}$$

C. Multi-objective Evolutionary Optimization

A multi-objective optimization problem consists of multiple objective functions to be optimized. A multi-objective optimization problem can be stated as follows:

$$\min \quad \vec{f}(\vec{x}) = (f_1(\vec{x}), \dots, f_m(\vec{x})), \tag{3}$$
$$s.t. \quad \vec{x} \in \Omega. \tag{4}$$

where Ω stands for the feasible region of $\vec{x}$.

Multi-objective Evolutionary Optimization Algorithm (MOEA) are ideal for solving multi-objective optimization problems [14], because MOEAs work with a population of solutions. With an emphasis on moving towards the true Pareto-optimal region, a MOEA algorithm can be used to find multiple Pareto-optimal solutions in one single simulation run [15]. Therefore, this project would employ MOEA approaches. This is also the first time to employ MOEAs technique for SRAC problem.

III. Problem Description

We consider the problem as a multi-objective problem with two potentially conflicting objectives, minimizing the overall cost of web services and minimizing the overall energy consumption of the used physical machines.

To solve the SRAC problem, we model an atomic service as its request and requests' coming rate, also known as frequency.

The request of an atomic service is modeled as two critical resources: CPU time $A = \{A_1, A_i, \dots, A_t\}$ and memory consumption $M = \{M_1, M_i, \dots, M_t\}$, for each request consumes a A_i amount of CPU time and M_i amount of memory. The coming rate is denoted as $R = \{R_1, R_i, \dots, R_t\}$. In real world scenario, the size and the number of a request are both variant which are unpredictable, therefore, this is one of the major challenges in Cloud resource allocation. In this paper, we use fixed coming rate extracted from a real world dataset to represent real world service requests.

The cloud data center has a number of available physical machines which are modeled as CPU time $PA = \{PA_1, PA_j, \dots, PA_p\}$ and memory $PM = \{PM_1, PM_j, \dots, PM_p\}$. PA_j denotes the CPU capacity of a physical machine and PM_j denotes the size of memory. A physical machine can be partitioned or virtualized into a set of virtual machines; each virtual machine has its CPU time $VA = \{VA_1, VA_n, \dots, VA_v\}$ and memory $VM = \{VM_1, VM_n, \dots, VM_v\}$.

The decision variable of service allocation is defined as X_n^i. X_n^i is a binary value (e.g. 0 and 1) denoting whether a service i is allocated on a virtual machine n. The decision variable of virtual machine allocation is defined as Y_j^n. Y_j^n is also binary denoting whether a VM n is allocated on a physical machine j.

In this work, we consider homogeneous physical machine which means physical machines have the same size of CPU time and memory. The utilization of a CPU of a virtual machine is denoted as $U = \{U_1, U_n, \dots, U_v\}$. The utilization can be calculated by Eq.5.

$$U_k = \begin{cases} \frac{\sum_{i=1}^{t} R_i \cdot A_i \cdot X_n^i}{VA_n}, \text{If } \sum_{i=1}^{t} R_i \cdot A_i < 1 \\ 1 \qquad\qquad\qquad\quad , \text{otherwise} \end{cases} \tag{5}$$

The cost of a type of virtual machine is denoted as $C = \{C_1, C_n \dots, C_v\}$.

In order to satisfy the performance requirement, Service providers often define Service Level Agreements (SLAs) to ensure the service quality. In this work, we define throughput as a SLA measurement [16]. Throughput denotes the number of requests that a service could successfully process in a period of time. According to *RAT* model, the throughput is equal to the number of requests when the allocated resource is sufficient. Therefore, if a VM reaches its utilization limitation, it means that the services have been allocated exceedingly. Therefore, all services in that VM suffer from performance degradation.

Then we define two objective functions as the total energy consumption and the total cost of virtual machines:

minimize

$$Energy = \sum_{j=1}^{p}(k \cdot V_{max} + (1-k) \cdot V_{max} \cdot \sum_{n=1}^{v} U_n \cdot Y_j^n) \tag{6}$$

$$Cost = \sum_{j=1}^{p} \sum_{n=1}^{v} C_n \cdot Y_j^n \tag{7}$$

1) Hard constraint: A virtual machine can be allocated on a physical machine if and only if the physical machine has enough available capacity on every resource.

$$\begin{aligned} \sum_{n=1}^{v} VM_n \cdot Y_j^n \leq PM_j \\ \sum_{n=1}^{v} VA_n \cdot Y_j^n \leq PA_j \end{aligned} \tag{8}$$

2) Soft constraint: A service can be allocated on a virtual machine even if the virtual machine does not have enough available capacity on every resource, but the allocated services will suffer from a quality degradation.

$$\sum_{i=1}^{t} M_i \cdot R_i \cdot X_i^n \leq VM_n \tag{9}$$

IV. METHODS

As we have discussed, Multi-objective Evolutionary Algorithms are good at solving multi-objective problems and NSGA-II [17] has shown his effective and efficiency. NSGA-II is a well-known MOEA that has been widely used in many real-world optimization problems. In this paper we also adopt NSGA-II to solve the SRAC problem. We first propose a representation and then present a NSGA-II based algorithm with novel genetic operators.

A. Chromosome Representation

SRAC is a two-level bin-packing problem, in the first level, bins represent physical machines and items represent virtual machines. Whereas, in the second level, a virtual machine acts like a bin and web services are items. Therefore, we design the representation in two hierarchies, virtual machine level and physical machine level.

Figure 1 shows an example individual which contains seven service allocations. Each allocation of a service is represented as a pair where the index of each pair represents the number of web service. The first number indicates the type of virtual machine that the service is allocated in. The second number denotes the number of virtual machine. For example, in Figure 1, service #1 and service #2 are both allocated in the virtual machine #1 while service #1 and service #5 are allocated to different virtual machines sharing the same type. The first hierarchy shows the virtual machine in which a service is allocated by defining VM type and number. Note that, the VM type and number are correlated once they are initialized. With this feature, the search procedure is narrowed down in the range of existing VMs which largely shrinks the search space. The second hierarchy shows the relationship between a physical machine and its virtual machines, which are implicit. The physical machine is dynamically determined according to the virtual machines allocated on it. For example, in Figure 1, the virtual machines are sequentially packed into physical machines. The boundaries of PMs are calculated by adding up the resources of VMs until one of the resources researches the capacity of a PM. At the moment, no more VMs can be packed into the PM, then the boundary is determined. The reason we designed this heuristic is because a physical machine is always fully used before launching another. Therefore, VM consolidation is inherently achieved.

Clearly, specifically designed operators are needed to manipulate chromosomes. Therefore, based on this representation, we further developed initialization, mutation, constraint handling and selection method.

B. Initialization

Algorithm 1 Initialization

Inputs:
VM CPU Time VA and memory VM,
Service CPU Time A and memory M
consolidation factor c
Outputs: A population of allocation of services

for Each service t **do**
    Find its most suitable VM Type
    Randomly generate a VM type $vmType$ which is equal or better than its most suitable type
    if There are existing VMs with $vmType$ **then**
        randomly generate a number u
        if $u <$ consolidation factor **then**
            randomly choose one existing VM with $vmType$ to allocate
        else
            launch a new VM with $vmType$
        end if
    else
        Create a new VM with its most suitable VM type
    end if
end for

The initialization (see Alg 1) is designed to generate a diverse population. In the first step, for each service, it is able to find the most suitable VM type which is just capable of running the service based on its resource requirements. In the second step, based on the suitable VM type, a stronger type is randomly generated. If there exists a VM with that type, the service is either deployed in the existing VM or launch a new VM. We design a consolidation factor c which is a real number manually selected from 0 to 1 to control this selection. If a random number u is smaller than c, the service is consolidated in an existing VM.

This design could adjust the consolidation, therefore, controls the utilization of VM.

C. Mutation

The design principle for mutation operator is to enable individuals exploring the entire feasible search space. Therefore, a good mutation operator has two significant features, the

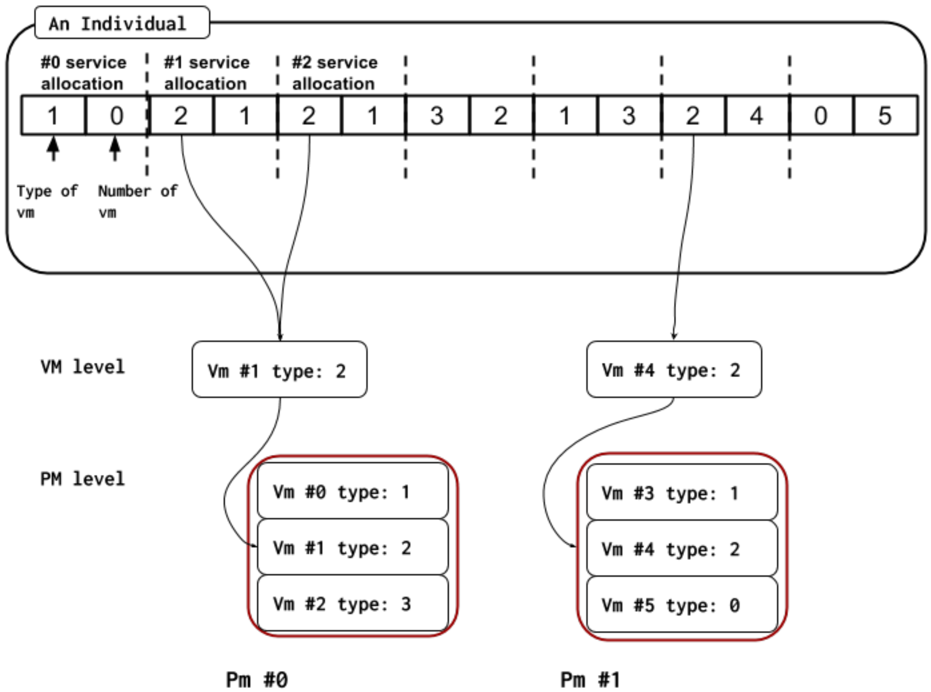


Fig. 1. An example chromosome representation

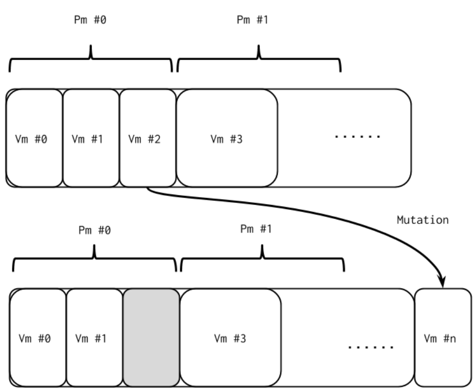


Fig. 2. An example mutation without insertion that causes a lower resource utilization

exploration ability and the its ability to keep an individual within the feasible regions. In order to achieve these two goals, firstly, we generate a random virtual machine type which has a greater capacity than the service needs. It ensures the feasible of solutions as well as exploration capability. Then, we consider whether a service is consolidated with the consolidation factor c.

The consolidation is conducted with a roulette wheel method which assigns fitness value to each VM according to the reciprocal of its current utilization. The higher the utilization, the lower the fitness value it is assigned. Therefore, a lower utilization VM has a greater probability to be chosen. At last, if a new VM is launched, it will not be placed at the end of VM lists. Instead, it will be placed at a random position among the VMs. The reason is illustrated in Figure 2. In the example, VM #2 is mutated into a new type and be placed at the end of the VM list. However, because of the size of VM #3 is too large for PM #0, the hollow in PM #0 will never be filled. This problem can be solved with the random insertion method.

Algorithm 2 Mutation

Inputs:
An individual VM CPU Time VA and memory VM,
Service CPU Time A and memory M
consolidation factor c
Outputs: A mutated individual

for Each service **do**
  Randomly generate a number u
  if $u <$ mutation rate **then**
    find the most suitable VM Type for this service
    Randomly generate a number k
    if $k <$ consolidation factor **then**
      calculate the utilization of used VMs
      assign each VM with a fitness value of 1 / utilization and generate a roulette wheel according to their fitness values
      Randomly generate a number p, select the VM according to p
      Allocate the service
    else
      launch a new VM with the most suitable VM Type
      insert the new VM in a randomly choose position
    end if
  end if
end for

D. Violation control method

A modified violation ranking is proposed to deal with the soft constraint, for the hard constraint is automatically eliminated by the chromosome representation. We define a violation number as the number of services which are allocated in the degraded VMs. That is, if there are excessive services allocated in a VM, then all the services are suffered from a degraded in performance. The violation number is used in the selection procedure, where the individuals with less violations are always preferred.

E. Selection

Our design uses the binary tournament selection with a constrained-domination principle. A constrained-domination principle is defined as following. A solution I is considered constraint-dominate a solution J, if any of the following condition is true:

1) Solution I is feasible, solution is not,
2) Both solutions are infeasible, I has smaller overall violations,
3) Both solutions are feasible, solution I dominates solution J.

An individual with no or less violation is always selected. This method has been proved effective in the original NSGA-II paper [17].

F. Fitness Function

The cost fitness (Eq.7) is determined by the type of VMs at which web service are allocated. The energy fitness is shown in Eq.6, the utilizations (Eq.5) of VM are firstly converted into the utilizations of PM according to the proportion of VMs and PMs CPU capacity.

G. Algorithm

The main difference between our approach and the original NSGA-II is that our approach has no crossover operator.

That is, a random switch of chromosome would completely destroy the order of VMs, hence, no useful information will be preserved. Therefore, we only apply mutation as the exploration method. Then, the algorithm becomes a parallel optimization without much interaction between its offspring, which is often addressed as Evolutionary Strategy [18].

V. Experiment

A. Dataset and Problem Design

This project is based on both real-world datasets *WS-Dream* [19] and simulated datasets [20]. The *WS-Dream* contains web service related datasets including network latency and service frequency (request coming rate). In this project, we mainly use the service frequency matrix. For the cost model, we only consider the rental of virtual machines with fixed fees (monthly rent). The configurations of VMs are shown in Table II, the CPU time and memory were selected manually and cost were selected proportional to their CPU capacity. The maximum PM's CPU and memory are set to 3000 and 8000

Algorithm 3 NSGA-II for SRAC

Inputs:
VM CPU Time VA and memory VM,
PM CPU Time PA and memory PM,
Service CPU Time A and memory M
consolidation factor c
Outputs: A Non-dominated Set of solutions

Initialize a population P
while Termination Condition is not meet **do**
    for Each individual **do**
        Evaluate the fitness values
        Calculate the violation
    end for
    non-Dominated Sorting of P
    calculate crowding distance
    while child number is less than population size **do**
        Selection
        Mutation
        add the child in a new population U
    end while
    Combine P and U { for elitism}
    Evaluate the combined P and U
    Non-dominated sorting and crowding distance for combined population
    Include the top popSize ranking individuals to the next generation
end while

TABLE I
PROBLEM SETTINGS

Problem	1	2	3	4	5	6
Number of services	20	40	60	80	100	200

respectively. The energy consumption is set to 220W according to [20].

We designed six problems shown in Table I, listed with increasing size and difficulty, which are used as representative samples of SRAC problem.

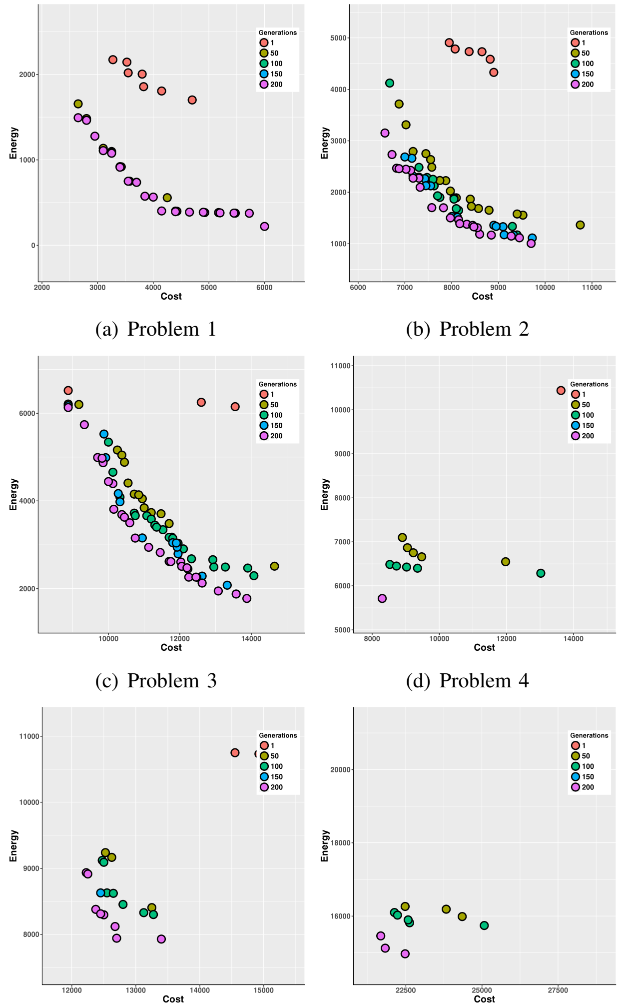
Selection Method with violation Control vs. without violation control

We conducted two comparison experiments. For the first experiment, we make a comparison between NSGA-II with violation control and NSGA-II without violation control. In second experiment, two mutation operators are compared. The first is the roulette wheel mutation, the second is the mutation with greedy algorithm. The mutation with greedy algorithm is a variant of roulette wheel mutation. The only difference is that instead of selecting a VM to consolidate with fitness values, it always selects the VM with the lowest CPU utilization. Therefore, it is a greedy method embedded in the mutation.

The experiments were conducted on a personal laptop with 2.3GHz CPU and 8.0 GB RAM. For each approach, 30 in-

TABLE II
VM CONFIGURATIONS

VM Type	CPU Time	Memory	Cost
1	250	500	25
2	500	1000	50
3	1500	2500	150
4	3000	4000	300

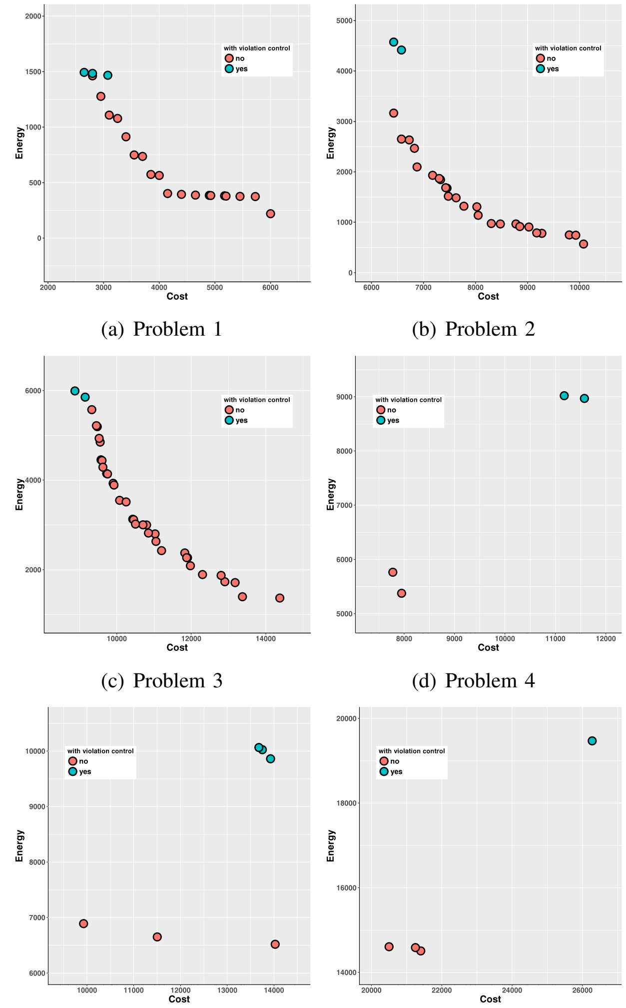


(a) Problem 1 (b) Problem 2

(c) Problem 3 (d) Problem 4

(e) Problem 5 (f) Problem 6

Fig. 3. Non-dominated solutions evolve along with the generation



(a) Problem 1 (b) Problem 2

(c) Problem 3 (d) Problem 4

(e) Problem 5 (f) Problem 6

Fig. 4. non-dominated solutions comparison between selection with violation control and without violation control

dependent runs are performed for each problem with constant population size 100. The maximum number of iteration is 200. k equals 0.7. We set mutation rate and consolidation factor to 0.9 and 0.01.

B. Results

As we conducted the experiment for 30 runs, we first obtain an average non-dominated set over 30 runs by collecting the results from a specific generation from all 30 runs, and then apply a non-dominated sorting over them.

Firstly, we show the non-dominated solutions evolve along with the evolution process in Figure 3. These results come from selection method without violation control. As it illustrated, different colors represent different generations from 0th to 200th. For problem 1, because the problem size is small, the algorithm converged before 100 generations. Therefore, the non-dominated set from the 100th and 150th generations are overlapping with results from the 200th generation. For problem 2 and problem 3, it clearly shows the improvement of fitness values. For problem 4 onwards, the algorithm can only obtain a few solutions as the problem size is large, it is difficult to find solutions.

Then, the non-dominated sets of the last generation from two selection methods are compared in Figure 4. There are much fewer results are obtained from the violation control method throughout all cases. For the first three problems, the non-dominated set from the violation control method has similar quality as the no violation control method. From problem 4 onwards, the results from selection with violation control are much worse in terms of fitness values. However, most of the results from non-violation control selection have a high violation rate. That is, the method without violation control is stuck in the infeasible regions and provide high-violation rate solutions.

From figure 5, we can observe the violation rate between two methods. It proves violation control has a great ability to prevent the individual from searching the infeasible region. On the other hand, without violation control, although, the algorithm can provide more solutions with better fitness values,

TABLE III
COMPARISON BETWEEN TWO MUTATION METHODS

Problem	roulette wheel mutation		Greedy mutation	
	cost fitness	energy fitness	cost fitness	energy fitness
1	2664.6 ± 66.4	1652.42 ± 18.2	2661.7 ± 56.9	1653.2 ± 18.2
2	6501.1 ± 130.2	4614.0 ± 110.7	6495.37 ± 110.7	4132.5 ± 80.4
3	8939.2 ± 118.5	6140.7 ± 204.0	9020.5 ± 204.0	5739.6 ± 148.6
4	11633.7 ± 301.1	9301.9 ± 254.0	12900.6 ± 243.0	9376.3 ± 120.9
5	14102.0 ± 231.7	10164.8 ± 238.9	14789.2 ± 238.8	9876.3 ± 120.9
6	27194.3 ± 243.0	19914.4 ± 307.5	27654.2 ± 307.5	19187.1 ± 176.6

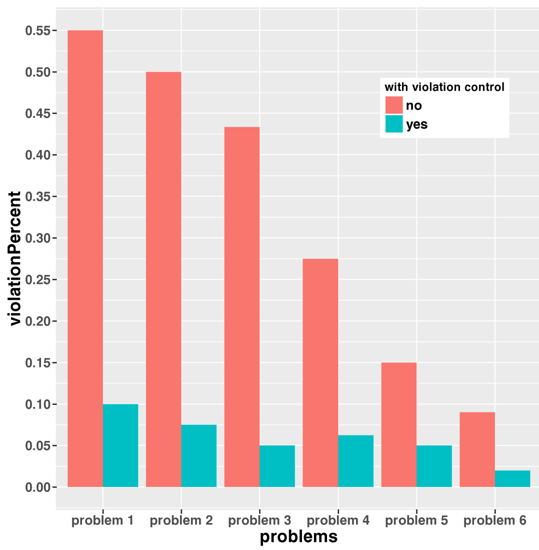


Fig. 5. Violation Percentage comparison between selection with violation control and without violation control

most of them have a high violation rate over 10% which are not very useful in reality.

As we mentioned in previous section, the mutation rate and consolidation factor are set differently for the two methods. For the method with violation control, the mutation rate is set to 0.9 and the consolidation factor c is set to 0.01, this is because the feasible region is narrow and scattered. In order to avoid stucking in the local optima, a large mutation rate can help escape local optima. For the factor c, a larger percentage would easily lead the algorithm to infeasible regions. Therefore, it is set to a small number.

Mutation with roulette wheel vs. Mutation with greedy algorithm

Table III shows the fitness value comparison between mutation methods. According to statistics significant test, there is little difference between methods. The possible reason is the consolidation factor is set to 0.01. In each mutation iteration, there is only 1% probability that a service will be consolidated in an existed VM, therefore, the influence between different consolidation strategies is trivial.

VI. CONCLUSION

In this paper, we first propose a multi-objective formulation of a two levels of bin packing problem, web service resource allocation in Cloud. It solves the resource allocation in IaaS and SaaS at the same time. Two objectives, minimizing the cost from service providers' perspective and minimizing the energy consumption from cloud provider's objective are achieved. Secondly, we propose a NSGA-II based algorithm with specific designed genetic operators to solve the problem. The results are compared with different variances of the algorithm. The results show our approach can solve the very complicate optimization problem.

With current work as a baseline, in future work, we could improve the quality of solutions as well as provide better violation control mechanisms.

REFERENCES

[1] R. E. Brown, E. R. Masanet, B. Nordman, W. F. Tschudi, A. Shehabi, J. Stanley, J. G. Koomey, D. A. Sartor, and P. T. Chan, "Report to congress on server and data center energy efficiency: Public law 109-431," 06/2008 2008.

[2] S. Desai, S. Bahadure, F. Kazi, and N. Singh, "Article: Multi-objective constrained optimization using discrete mechanics and NSGA-II approach," *International Journal of Computer Applications*, vol. 57, no. 20, pp. 14–20, 2012.

[3] S. Mazumdar and M. Pranzo, "Power efficient server consolidation for cloud data center," *Future Generation Computer Systems*, vol. 70, pp. 4 – 16, 2017.

[4] D. J. Dubois and G. Casale, "Autonomic provisioning and application mapping on spot cloud resources," in *Cloud and Autonomic Computing (ICCAC), 2015 International Conference on*. IEEE, 2015, pp. 57–68.

[5] J. Anselmi, E. Amaldi, and P. Cremonesi, "Service consolidation with end-to-end response time constraints," in *2008 34th Euromicro Conference Software Engineering and Advanced Applications*, Sept 2008, pp. 345–352.

[6] C. Ghribi, M. Hadji, and D. Zeghlache, "Energy efficient vm scheduling for cloud data centers: Exact allocation and migration algorithms," in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*. IEEE, 2013, pp. 671–678.

[7] Z. I. M. Yusoh and M. Tang, "A penalty-based genetic algorithm for the composite saas placement problem in the cloud," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*. IEEE, 2010, pp. 1–8.

[8] S. S. Yau and H. G. An, "Adaptive resource allocation for service-based systems," in *Proceedings of the First Asia-Pacific Symposium on Internetware*. ACM, 2009, p. 3.

[9] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future generation computer systems*, vol. 28, no. 5, pp. 755–768, 2012.

[10] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing," in *Proceedings of the 2008 conference on Power aware computing and systems*, vol. 10. San Diego, California, 2008, pp. 1–5.

[11] A. E. H. Bohra and V. Chaudhary, “Vmeter: Power modelling for virtualized clouds,” in *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*. Ieee, 2010, pp. 1–8.

[12] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, “No power struggles: Coordinated multi-level power management for the data center,” in *ACM SIGARCH Computer Architecture News*, vol. 36, no. 1. ACM, 2008, pp. 48–59.

[13] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang, “Power and performance management of virtualized computing environments via lookahead control,” *Cluster computing*, vol. 12, no. 1, pp. 1–15, 2009.

[14] S. Desai, S. Bahadure, F. Kazi, and N. Singh, “Article: Multi-objective constrained optimization using discrete mechanics and NSGA-II approach,” *International Journal of Computer Applications*, vol. 57, no. 20, pp. 14–20, 2012.

[15] D. Kanagarajan, R. Karthikeyan, K. Palanikumar, and J. Davim, “Optimization of electrical discharge machining characteristics of wc/co composites using non-dominated sorting genetic algorithm (NSGA-II),” *The International Journal of Advanced Manufacturing Technology*, vol. 36, no. 11-12, pp. 1124–1132, 2008.

[16] A. Paschke and E. Schnappinger-Gerull, “A categorization scheme for sla metrics.” *Service Oriented Electronic Commerce*, vol. 80, no. 25-40, p. 14, 2006.

[17] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: Nsga-II,” *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, Apr 2002.

[18] K. Y. Lee and F. F. Yang, “Optimal reactive power planning using evolutionary algorithms: A comparative study for evolutionary programming, evolutionary strategy, genetic algorithm, and linear programming,” *IEEE Transactions on power systems*, vol. 13, no. 1, pp. 101–108, 1998.

[19] Y. Zhang, Z. Zheng, and M. Lyu, “Exploring latent features for memory-based QoS prediction in cloud computing,” in *Reliable Distributed Systems (SRDS), 2011 30th IEEE Symposium on*, 2011, pp. 1–10.

[20] D. Borgetto, H. Casanova, G. Da Costa, and J.-M. Pierson, “Energy-aware service allocation,” *Future Generation Computer Systems*, vol. 28, no. 5, pp. 769–779, 2012.