A Genetic Programming Hyper-Heuristic for On-line VM Selection and Creation in Container-based Clouds

Boxiong Tan, Hui Ma, Yi Mei
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
Email:{Boxiong.Tan, Hui.Ma, Yi.Mei}@ecs.vuw.ac.nz

***Abstract*—Container technology has become a new trend in both software industry and in cloud computing. Containers support a fast development of web applications and their potential to reduce energy consumption in data centers. However, since containers are usually first deployed in virtual machines (VMs) and VMs are deployed in physical machines, the container allocation is a new two-level allocation problem. Current research overly simplifies the container allocation into one-level and uses simple rule-based approaches to solve the problem. This paper, first, provides a novel definition of the two-level container allocation problem. Then, we develop a hybrid approach of using genetic programming hyper-heuristics combined with human-designed rules to solve the problem. The simulation-based experiments show that the our hybrid approach is able to significantly reduce the energy consumption than the solely use human-designed rules.**

***Index Terms*—container allocation, server consolidation, Cloud resource allocation, genetic programming, hyper-heuristic**

I. Introduction

Along with the fast increasing in popularity of micro-services and server-less architecture [1], container-based clouds [2] have become a hot topic in cloud computing. Compared to Virtual Machine (VM)-based clouds, which runs a unique operating system and libraries for each application, container technology achieves a fast developing, testing, and deployment by sharing a host OS among applications.

In addtion, containers can be used to reduce energy consumption from data centers [2], however, the problem of container allocation is new and difficult to solve. To reduce energy consumption of data centers, containers are be allocated to fewer number of physical machines (PMs). Compared to VM-based clouds, container allocation is a new problem because of the two-level structure. In container-based clouds, containers are first deployed in VMs, then VMs are deployed in PMs while in VM-based clouds, VMs are directly deployed to PMs. Consequently, previous allocation methods in VM-based clouds cannot be directly applied in container-based clouds. In each level, the allocation can be seen as a vector bin packing problem [3] which is NP-complete.

Current research on container allocation has disadvantages on both problem definition and solutions. From the problem perspective, most research either simplifies the two-level problem to one-level [4], [5] or lacks critical factors such as affinity constraints in their problem definition [6]. The major drawback of these simplifications is that they can only be applied in narrow scenarios where all containers can be co-located. However, one of the most important feature of container is that they have various affinity constraints such as security level and Operating System requirements [7]. Hence, it is an urgent to have a generalized two-level problem definition of container allocation.

From the solution perspective, current works mostly apply AnyFit-based algorithms [8], [9] which have limited search space, instead, a reservation-based technique is more promising. AnyFit-based algorithms [10], such as BestFit and FirstFit , has been widely applied in on-line bin packing problems for their simplicity [11]. However, a major drawback of AnyFit-based algorithms is that these algorithms only select the existing bins unless no bin is available [12]. This strategy limits the search space, therefore, these algorithms can hardly find the optimal solution. To resolve this drawback, a reservation technique is proposed by Lee and et al [13]. They *Harmonic-Fit* algorithm have shown better performance in one-dimensional on-line bin packing than the AnyFit-based algorithms. However, *Harmonic-Fit* algorithm cannot be applied in container allocation problem because it can not decide when to create a VM and which type of VM is suitable.

Hence, this research, first, provides a novel problem definition of the two-level container allocation problem. This novel problem definition introduces various new key features of container allocation such as affinity constraints and VM overheads. Then, we focus on developing a hybrid approach of using both Genetic programming-based hyper-heuristic (GPHH) and human-designed rules to solve the two-level container allocation problem. Specifically, we use GPHH to automatically generate rules for allocating containers to VMs, and use human-designed rules for allocating VMs to PMs.

We propose a learning algorithm: a GPHH approach because, first, our previous work [14] have shown that GPHH is able to learn the workload patterns from historical data. Therefore, it can generate rules which adapt to the current workloads while human-designed rules may have poor performance facing various real-world workloads. Second, GPHH overcomes the design for complex allocation rules by training in an off-line fashion and applying in an on-line fashion.

The overall goal is to propose a hybrid approach of using GPHH and human-designed rules to reduce more energy consumption than solely use human-designed rules on the two-level container allocation problem. More specifically, we have the following objectives:

1) Introduce a novel problem definition for two-level container allocation;
2) Develop a hybrid approach of GPHH and human-designed rules;
3) Evaluate our proposed approach by comparing it with human-designed rules on benchmark datasets;

II. Related Work

A. Drawbacks on the Problem Definition

Previous works on container allocation problem mainly have two types of issue. First, some research simplifies the two-level allocation problem into one-level by allocating containers directly to PMs [15], [16]. The major drawback for this simplification is that their allocation approaches can only be used in a narrow range of scenarios. More generally, containers are allocated to VMs instead of PMs due to the affinity constraints of containers.

The affinity constraint is a critical issue in container-based clouds. This is because, compared to VMs, containers have more requirements such as distinct Operating Systems (OSs), software libraries, and different levels of security []. Since containers are naturally running in different environment, not all containers can be co-located. Virtual machines could provide a stronger security and performance isolation as well as OSs [17]. Therefore, to resolve the typical container allocation scenarios, a generalized model for container allocation should include two levels: containers to VMs and VMs to PMs.

The second type of issue in current research is that, although some research considers the two-level structure , they lack some critical factors such as VM overhead [6], [18] and affinity constraints [8]. VM overheads is generated by hypervisor which consumes a certain portion of resources (CPU and memory). Creating tiny VMs (e.g one VM for each container) leads to a large amount of VM overheads, on the other hand, creating large VMs may also lead to unused resources because not all containers can be co-located. This trade-off is the core issue in the container allocation problem. Some research has the awareness of overheads [19], but they did not provide much analysis.

Therefore, this research first provides a novel problem definition to the two-level container allocation problem. We break the two-level problem into four decision making procedures: VM creation and selection, PM creation and selection with the consideration of VM overheads and affinity constraints.

B. Existing allocation methods

Existing research [6], [8] on container allocation problem mostly rely on AnyFit-based algorithms [11] with human-designed rules for all four decision making procedures. AnyFit-based algorithms, such as BestFit and FirstFit, are greedy algorithms. A human-designed rule acts like a score function which assigns a score to a solution. For example, in one-dimensional bin packing, we may define a rule as a bin with fewer residual resources earns a higher score. Then, a BestFit algorithm iteratively evaluates all bins and selects the bin with the highest score to allocate the item. In multi-dimensional bin packing, however, it is not obvious how to combine multiple resources into a single value so that all dimensional resources can be effectively used.

Mann's [8] applied six rules (such as sub, sum, and $product$) for VM selection. For VM creation, they apply a *just-fit* approach which selects the smallest possible VM for each container. Piraghaj et al [6] apply *Least Full Host Selection* for VM selection and apply a *Largest* approach for VM creation. For PM selection, both research apply a FirstFit algorithm.

From the literature of VM allocation, *volume* rule is introduced by Wood et al. [9]. They choose target PMs based on the least $volume = \frac{1}{1-cpu} * \frac{1}{1-mem}$. *Energy-aware* BestFit [20] and *Least Full Host Selection* algorithm [6] essentially are the same algorithm because the *Energy-aware* approach uses the energy evaluate function (see in Eq.1) is proportional to the CPU usage of a PM.

In our previous work [14], we have shown the performance of sub, sum, and $random$ rules on three datasets. The experiments indicate that human-designed rules cannot adapt to different scenarios in the real world data centers.

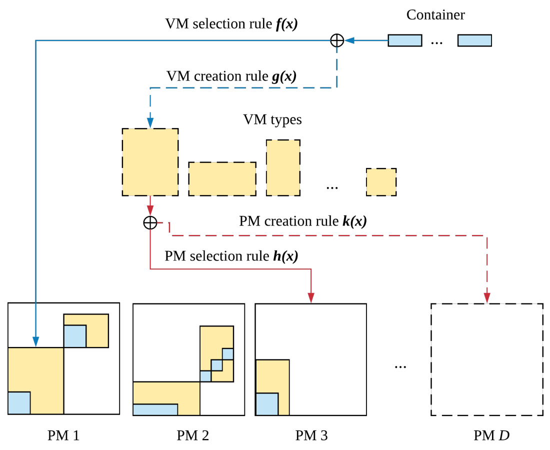
C. Genetic Programming

Genetic programming [21] is an evolutionary computation technique, inspired by biological evolution, to automatically find computer programs for solving a specific task. In a GP population, each individual represents a computer program. In each generation, these programs are evaluated by a predefined fitness function, which accesses the performance of each program. Then, individuals will go through several genetic operators such as selection, crossover, and mutation. A number of top individuals will survive to the next generation while others will be discarded.

GPHH has been successfully applied in a variety of problems. Xie et al [22], [23] propose a GPHH for storage location assignment problem. Liu et al [24] and Jacobsen-Grocott et al [25] use GP to design heuristics for arc routing problem. Mei solves a stochastic team orienteering problem with GPHH [26]. These literature show that GP has been successfully used for generating rules in various problems. Therefore, we also apply GPHH in our problem.

In summary, this work mainly focuses on a new container allocation model which addresses the four drawbacks in current research. In addition, instead of using human-designed allocation rules, we propose a Genetic Programming hyper-

Fig. 1: The container allocation procedure.



heuristic approach to automatically generate rule to solve the VM selection and creation problems.

III. Problem Description

In the two-level container allocation problem, containers arrive at data center during time duration of t_1 and t_2. The overall objective is to allocate containers to existing virtual machine (VMs) or creating new VMs, then allocate new VMs to physical machines (PMs), so that the accumulated energy consumption of all PMs is minimized.

The container allocation problem is separated into four decision making procedures: VM selection and creation, PM selection and creation (see Fig 1). VM selection chooses existing VMs to allocate containers when the chosen VM satisfies the requirement of the container. VM creation chooses a type of VM, which is predefined by cloud providers, then creates the VM to allocate the container. After a new VM is created, PM selection chooses existing PMs to allocate the VMs. Similar to VM creation, PM creation chooses a type of PM and allocate the new PM to the data center.

To evaluate a container allocation process, we apply accumulated energy used in our previous work [14]. The accumulated energy is calculated by aggregating the energy consumption of all activated PMs throughout the time duration $t_2 - t_1$. That is, whenever a container is allocated, we add the energy consumption of all PMs. The energy model of a PM (Eq.1) is widely used model proposed by Fan [27]. In their energy model, P^{idle} and P^{max} are the energy consumption when a PM is idle and fully used. $u^t_{cpu}(d)$ is the CPU utilization of a PM d at time t.

$$P^t_d = P^{idle}_d + (P^{max}_d - P^{idle}_d) \cdot u^t_{cpu}(d) \qquad (1)$$

The resource entities: containers, VMs and PMs have the following features. Each entity has two types of resources: CPU and memory. This work considers a data center with homogeneous PMs, which means all PMs have the same CPU and memory capacities, and heterogeous VMs, which means different types of VMs. Each type of VM is predefined with a tuple of resources. In addition to resource capacity, VM has a unique feature of VM overhead. VM overhead represents the resources consumed by a hypervisor. The overheads are also represented tuples of resources for each type of VM. Each VM runs a type of Operating System (OS). The type of OS is also predefined by a cloud provider. For containers, unlike Piraghaj's approach [6] which uses three types of containers for all applications, we consider one-on-one mapping between applications and containers. That is, we define the domain of containers' resource requirement between 1 to the capacity of PMs. Therefore, the definition of container is much flexible and more realistic [28]. Each container has an OS requirement, which means it can only be deployed in a VM which runs the same OS.

We consider three types of constraints in the model. First, similar to other models [6], the total resource requirement of containers cannot exceed the capacity of the target VM. The aggregate resource requirement of VMs cannot exceed the capacity of the target PM. Second, a container can only be deployed once. Third, we define an affinity constraint where containers can only be deployed in OS-compatible VMs.

IV. Methodology

This section describes our GPHH approach for on-line container allocation problem. We first describes the simulation model of VM selection and creation. Then, we describe two GPHH frameworks for VM selection and the combined VM selection and creation.

A. Simulation Model of VM Selection and Creation

This paper focuses on minimizing the energy consumption of VM selection and creation from on-line container allocation (discussed in Section II-B) instead of migration. Therefore, we create a stimulation to train and test algorithms. All the experiments in this paper are based on the simulation model, which has been used in our previous work [14]. Here are the simulation configurations.

1) Assume an infinite number of available VMs/PMs that can be used
2) Containers arrive uniformly between t_1 and t_2
3) Arrival containers must be allocated immediately
4) Overload threshold of VM/PM is 100%, this value does not affect the behavior of algorithms
5) No weights of containers, which means they are equally important
6) Five types of VMs (see Table I) and homogeneous PMs

In each simulation, we start with a randomly initialized data center. The randomly initialized data center contains a set of containers run on VMs/PMs. Then, a set of predefined containers arrive at the data center one by one. Since we simulates an on-line allocation, no information of the future container is exposed to the allocation algorithm. The allocation procedure is depicted in Fig 1.

The allocation procedure (see Fig 1) includes two decision steps: allocate containers to VMs (blues lines) and VMs to PMs (red lines). Each step includes two alternative choices. In containers to VMs level, we either select an existing VM

TABLE I: Configuration of VMs

VM size	CPU (MHz)	Memory (MB)
xSmall	825	250
Small	825	500
Medium	825	800
Large	1650	800
xLarge	3300	4000

according to a *VM selection rule* $f(x)$ or create a new VM from a list of VM types based on a *VM creation rule* $g(x)$. If a container is allocated to an existing VM, the allocation procedure ends. Otherwise, a new VM is created. In VMs to PMs level, we either select an existing PM based on *PM selection rule* $h(x)$ or create a new PM based on *PM creation rule* $k(x)$ to host the new VM. These rules act like a ranking function which assigns a value to each candidate VM/PM or VM types. Then, the container/VM will be allocated to the VM/PM with the highest value.

This work focuses on the containers to VMs level and employs human-designed rules in VMs to PMs level. In *PM selection*, we employ First Fit [12] which allocates the VM to the first PM with enough resources starts from the oldest PM. In *PM creation*, since we consider homogeneous PMs, the creation rule is straightforward – create the same size of PM when no existing PM is available.

After each container allocation, we will update the accumulated energy consumption by adding up the current energy of the data center. The simulation stops when all the predefined containers are allocated. Then, the performance measure is the accumulated energy consumption of all used PMs. Although this simulation is relatively simple, it still reflects the key issues of on-line container allocation such as two-level of on-line vector bin packing.

The data center initialization is designed for two reasons. First, in real-world data centers, allocating a set of containers to an empty data center rarely happens. In most of cases, containers are allocated to existing PMs instead of solely creating new PMs. Second, in order to test the robustness of allocation algorithms, we create distinct initial data center for each simulation. Therefore, we can test whether an allocation algorithm is sensitive to the initial state.

To reliably measure the effectiveness of evolved rules, a large number of simulation are usually needed [29] (e.g 30 to 50 simulation are usually needed). Therefore, in our experiments, we use two sets of simulation, each has 50, are used for training and testing. The training set is used to train GPHH and the test set is used to examine the performance of rules on unseen data. In this on-line allocation case, we cannot identify the best rule with the training performance because the risk of over-fitting.

B. Basic Framework of GP

Fig 2 shows the procedure of a basic GP framework. The framework is similar with other Evolutionary algorithms, except that GP uses a special representations of individuals (i.e. GP trees). The framework starts from randomly initializing

Fig. 2: The work flow of a basic GP

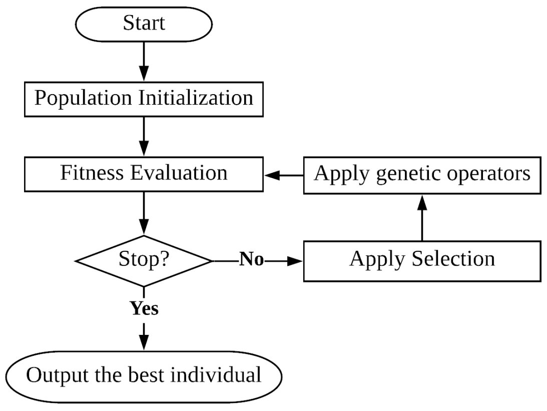
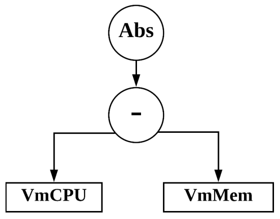


Fig. 3: The tree-based representation of the sub rule $|vmCPU - vmMem|$



a population of rules. Then, every rule will be evaluated using the simulation. The fitness of each rule is measured by applying the rule on the training set. If a rule earns the best fitness value (in our case, the minimum fitness value), then this rule is considered as the best rule in this generation. If the stopping condition is met (i.e., maximum generation in GP), GP will stop and output the best rule; otherwise, the population go through a selection method which selects rules from the population with a probability based on their fitness. These selected individuals then participate in the genetic operators. Genetic operators, such as *mutation*, *crossover*, and *reproduction*, have two purposes. First, *crossover* and *mutation* allow the individual rules to explore the search space by changing the structure of the rules. Second, *reproduction* retains the rules with good fitness values by bringing them to the next generation.

C. Representation

This paper uses the traditional GP tree to represent rules. An example of the sub rule $|vmCPU - vmMem|$ is given in Fig 3. For evolving rules with GP, the terminal set design has been demonstrated that including irrelevant features can deteriorate the performance [30]. In contrast, relevant features can lead to a significant improvement [31]. Table II shows the terminal set and function set used by GP to construct rules.

We design the attributes for VM selection according to Burke's research on one-dimensional bin packing problem [32]. For the combined of VM selection and creation, we design two extra attributes. *vmMemOverhead* and *vmCpuOverhead* returns the amount of CPU/memory overhead when the rule creates a new VM. If the evolved rule selects an existing

TABLE II: Terminal and Function sets of GP

Symbol	Description
	Attributes for VM selection
leftVmMem	VM memory - container memory requirement
leftVmCpu	VM CPU - container CPU requirement
coCpu	container CPU requirement
coMem	container memory requirement
	Extra Attributes for VM selection and creation
vmMemOverhead	memory overhead of VM
vmCpuOverhead	CPU overhead of VM
Function set	+,-,×, protected %

VM then the overhead is 0. The reason that we design these attributes is because the major difference between selecting an existing VM and creation is the extra overhead. For the function set, we use four basic arithmetic to construct rules (the protected division returns a value of 1 when division by 0).

D. Fitness Function

When evaluating each individual (Step 2 in Fig 2), the fitness function is defined as follows:

$$fitness = \frac{\sum_{k=1}^{|training\ instances|} \frac{AE_{normalized}}{N}}{|training\ instances|} \quad (2)$$

where AE is the accumulated energy of a training instance normalized by a benchmark rule. N is the number of containers in this training instance. With $\frac{AE_{normalized}}{N}$, we calculate the average AE of a training instance. Then, we average the AE of all training instances.

The normalization is shown as follows:

$$AE_{normalized} = \frac{AE}{AE_{benchmark\ rule}} \quad (3)$$

where we normalize the AE of the evolved rule with the AE of a benchmark rule. The reason that we use normalized AE is because different training instances may have major differences in AE. It is unfair to use the aggregation of AE of all training instances to compare algorithms.

For example, we applied two algorithms A and B on two training instances α and β. With algorithm A, we obtain $AE_\alpha = 35000$ and $AE_\beta = 2500$. With algorithm B, we obtain $AE_\alpha = 30000$ and $AE_\beta = 7500$. As we may see, although two algorithms obtain the same aggregation of $AE = 37500$, it is unfair to say A and B perform the same. Because, in α, the AE difference between A and B is only 16.7%, but in β, the difference is 200%. If we use a benchmark algorithm C ($AE_\alpha = 32500$, $AE_\beta = 5000$) to normalize A and B. For A, the aggregation of normalized AE is 35000/32500 + 2500/5000 = 1.58 and for B, the aggregation of normalized AE is 30000/32500 + 7500/5000 = 2.42. Since we aim at minimizing the energy, it is easy to see that algorithm A performs much better than B.

E. Framework for the Combined VM Selection and Creation

The framework for the combined VM selection and creation is designed for a generalized two-level container allocation shown in Algorithm 1. In this framework, we solve the selection and creation in a single step. In *line 3*, we append five empty VMs (one for each VM type) into the existing VM list. Consequently, the selection rule also evaluates the empty VMs and compares their values with the existing VMs.

This framework is inspired by the idea of reservation techniques for bin packing problem [12]. Reservation techniques, such as *Harmonic-Fit* algorithm [13] designed by Lee and et al, have shown better performance in one-dimensional on-line bin packing than the AnyFit-based algorithms. *Harmonic-Fit* divides items into k intervals and k corresponding types of bin. Each bin only hosts corresponding types of items. The major disadvantage of *Harmonic-Fit* is the waste of resource in those bins reserved for large items. The similarity between our problem and *Harmonic-Fit* is that, as we designed an OS affinity constraint, containers are naturally divided into categories based on their OS requirements. We allow the evolved rules to choose the appropriate types instead of fixing the types of VM so that it is much flexible than *Harmonic-Fit* algorithm.

We introduce two additional attributes *vmMemOverhead* and *vmCpuOverhead* (discussed in Section IV-C). The additional attributes help GPHH to distinguish the existing VMs and the empty VMs.

Algorithm 1: Framework for the Combined VM Selection and Creation

```
Input  : container, A list of VM types, A list of
         available VMs, A list of available PMs,
Output: The best VM
1  $BestVM = nil$;
2  $bestFitness = nil$;
3  Append the VM types to the VM list as empty
     VMs;
4  Filter the VMs without enough resources for this
     container;
5  while $VM_i$ in VMs do
6      $fitness$ =Rule($container, PM_i$);
7      if $fitness > bestFitness$ then
8          $bestFitness = fitness$;
9          $BestVM$ =$VM_i$;
10     end
11     $i = i + 1$;
12 end
13 return $BestVM$;
```

V. Design of Experiments

The aim of this study is to propose a GPHH approach to one-level and two-level of on-line container allocation problem. In the above section, we present the GPHH algorithm, representation, fitness function, and two problem

TABLE III: Test instances for the combined VM selection and creation

instances	number of OS	scenario	number of containers	initialization
instance 1	2	unbalanced scenario	Medium (200)	5 types of VM
instance 2	3	unbalanced scenario	Medium (200)	5 types of VM
instance 3	4	unbalanced scenario	Medium (200)	5 types of VM
instance 4	5	unbalanced scenario	Medium (200)	5 types of VM

TABLE IV: Frequency of OS requirements

number of OS	distribution (%)
2	95–5
3	50–30–20
4	62.5–17.5–15-5–4.5
5	17.9–45.4–23.6–10.5–2.6

frameworks. We conduct a set of experiments over the two problem frameworks. In the experiments of VM selection, we design two experiments. We first compares GPHH with five human-designed rules over three scenarios; Second, we examine the effect of GPHH over a larger sizes of containers. In the experiment of the combined VM selection and creation, we compare the GPHH approach with two human-designed rules.

All algorithms were implemented in Java version 8 and the experiments were conducted on an i7-4790 3.6 GHz with 8GB of RAM memory running Linux Arch 4.14.15-1.

A. Dataset

We design 6 instances for the experiments of VM selection (see Table **??**) and 4 instances for the experiments of combined VM selection and creation (see Table III). In the experiments of VM selection, we use the medium size of instances 1, 3, 5 in experiment one to test the performance of rules when facing the balanced and unbalanced data center environment. We use instances 2,4,6 to in experiment two to test the performance of evolved rules when allocating larger sizes of containers. In the experiments of combined VM selection and creation, we use instances 7$\sim$10 in experiment three to compare the performance of evolved rules and human-designed rules. Each of the above instance includes 100 simulation cases as mentioned in Section IV-A, we use 50 for training and 50 for testing. Each case includes a number of containers listed as Medium (200 containers) and Large (1000 containers).

We use both real-world and synthetic datasets to construct the Test instances for VM selection. We use real world workload trace (AuverGrid trace [33]) to construct the unbalanced scenario. Fig 4 shows the distribution of CPU and memory requirement of the containers in this dataset. We observe that the memory usage is on average three times as much as the CPU usage. Therefore, we use the real-world dataset to represent the **unbalanced resource requirement** in containers. In the meanwhile, we apply an unbalanced CPU and memory (3300, 4000) in the VM configurations to represent the **unbalanced resource capacity** in VMs. We generated the **balanced resource requirement** from exponential distribution with the rate $\lambda = 0.004$ in both CPU and memory. The reason for using this rate is because we want to have similar

TABLE V: Parameter Settings

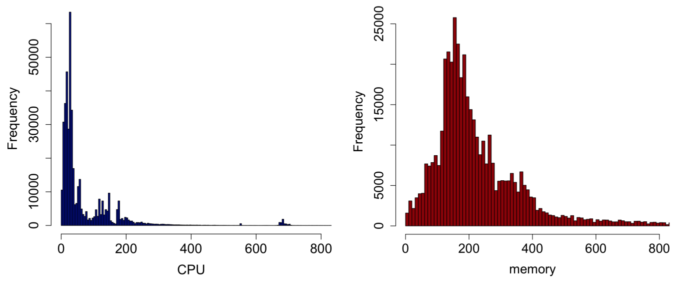
Parameter	Description
Initialization	ramped-half-and-half
Crossover/mutation/reproduction	80%/10%/10%
Maximum Depth	7
Number of generations	100
Population	1024
Selection	tournament selection (size = 7)

size of containers between the synthetic dataset and real-world dataset. From the empirical study, we found the average number of containers allocated to a PM is 15 containers in real-world dataset. With the $\lambda = 0.004$, it gives us a similar average number of containers allocated to a PM. We apply a balanced CPU and memory (3300, 3300) in the VM configurations to represent the **balanced resource capacity** in VMs.

In the experiments of the combined VM selection and creation, we design 4 test instances with various number of OS requirement to test the performance of the evolved rules. The frequency of OSs in each instances is shown in Table IV. We intend to simulate a real-world market share of OS [34].

We applied Wilcoxon signed rank test on all results between GPHH and human-designed rules. We show ten test instances because we observe a similar pattern throughout all test instances. Therefore, we choose ten representative instances to show the results.

Fig. 4: Resource usage frequency in the real world dataset



(a) CPU usage frequency **(b)** Memory usage frequency

B. Parameter Settings

Table V shows the parameters that we used in all experiments. Most parameters are the standard values in GPHH field [31]. The algorithm was implemented by ECJ [35].

C. Experiments on the Combined VM selection and Creation

This section designs an experiment to study the effect of applying GPHH on the combined VM selection and creation framework.

1) Comparison with two human-designed rules: This experiment has two intentions. First, we try to unveil drawbacks of human-designed rules and explain the reasons. Second, we would like to evolve rules to find better VM creation strategies.

To compare with the benchmark algorithms, we implement two algorithms proposed in [19] and [6]. The framework of two algorithms is essentially a VM selection method combined with a VM creation method. For VM selection, we implement the sub rule. For VM creation rules, [19] designs a *just-fit* rule. *just-fit* creates the smallest VM that can satisfy the resource requirement of the container. [6] designs a *largest* rule. *largest* rule first scans the PMs to find one that has enough resources for this container. Then, it creates the largest possible VM that can fit in this PM. The third step is to allocate the container into the VM. We use the result from *just-fit* to normalize other rules.

2) Result analysis of experiments in comparison with two human-designed rules: Fig 5 shows the accumulated energy comparison among three rules: evolved rules, *just-fit*, and *largest*. The evolved rules and *largest* have a great advantage than the *just-fit* rule. In comparison between evolved rules and *largest* rule, Table VI shows that evolved rules still dominate *largest* in all scenarios.

We examine the reason for the poor performance of human-designed rules by looking at the waste of resources in PMs. The waste resources in PMs can be divided into two parts – unused resources and resources Overhead. As an example, we plot the unused CPU and CPU overhead of three rules on a test instance (from instance 10) in Fig 6. We observe that the waste on unused CPU is almost five times more than the waste on overhead. It shows that the unused CPU is a much crucial factor than CPU overhead.

At the beginning (from 0 to 75 containers in Fig 6a), *just-fit* performs the best because it creates smallest possible VMs for containers to reduce the waste. However, the unused resource accumulated fast because they are too small to be used in the future. In the meanwhile, containers cannot be consolidated into these small VMs. This leads to the creation of a large number of small VMs. The VM overhead also increases fast because it is proportional to the number of VMs. Small VMs also prevent new VMs from consolidating in the same PM because the PMs has insufficient resources. Consequently, the number of PMs increases quickly. The average utilization of CPU and memory in PMs is around 15% only.

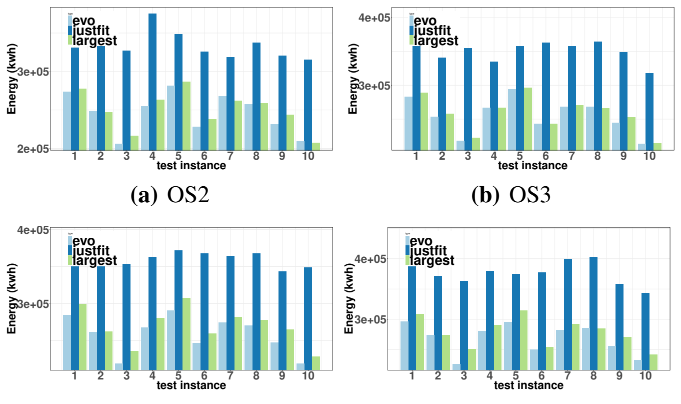
The strategy of the *largest* rule reduces the unused resources by creating the largest consecutive pieces of resources for the future containers. However, the *largest* rule neglects the effect of OS constraint. When the rule creates the largest possible VM for a container who requires an OS with a low frequency, it is unlikely that the VM will be filled in a long period of time because there is not many container who also requires this type of OS. Hence, the resources in this VM will be unused for a long time. With *largest* rule, the unused resource increases along with the number of OSs.

The evolved rules, on the other hand, create a VM neither from the smallest or the largest. Instead, it considers the distribution of CPU and memory of containers and the size of VMs' types. Therefore, evolved rules avoid creating many small VMs as well as creating large VMs. A detail explanation of the behavior of rules will be discussed in Section **??**.

VI. Analysis and Discussion

This section focuses on analyzing the structure of the combined rules. Most of the evolved rules are overly complex,

Fig. 5: Accumulated energy comparison (first 10 test instances) among three rules:evolved rules, *just-fit*, and *largest*



(a) OS2 **(b)** OS3 **(c)** OS4 **(d)** OS5

TABLE VI: Energy comparison between evolved rules and *largest* rule in four scenarios

	OS2	OS3	OS4	OS5
evo **vs.** *largest*	30-0-20	31-0-19	44-0-6	42-0-8

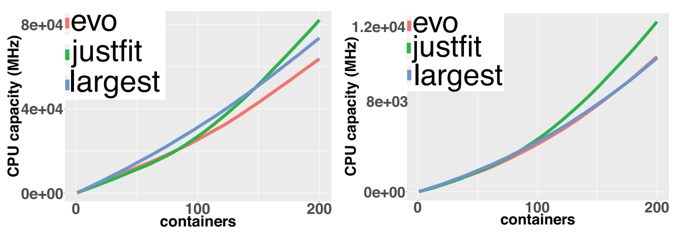
therefore, it is difficult to understand the insight. In order to study the behavior of rules, we select a relatively simple rule which evolves on instance 10. We manually simplify the rule as shown in Fig 7. This rule obtains a fitness value of 1210.47 on test set in compare with 1596.97 from the *just-fit* rule and 1303.47 from the *largest* rule.

We consider the behavior of the rule in VM selection and creation. In VM selection, the vmCpuOverhead is 0. Therefore, we may further simplify the rule as $coCpu \times leftVmMem^{11}$. Since in each allocation, the *coCpu* does not affect the behavior of the rule, the solely decision variable is the *leftVmMem*. This makes sense because, as we mentioned in Section V-A that the memory requirement of containers is roughly three times higher than the CPU requirement, the memory is the critical resource. Therefore, the rule only selects the VM with more memory.

On the other hand, in VM creation, the *vmMemOverhead* becomes a constant of 200 MB. As we normalize all the resources with the PM's capacity, we have the $vmMemOverhead = 0.05$. Then, we assume $coCpu = 0.02$ since this assumption does not affect the result. This rule can be simplified as $f = 0.02 \times leftVmMem^{11} - (1 + 0.05 \times leftVmMem^{13} \times leftVmCpu)$.

To visualize the rule, we plot it on a 3D landscape (see Fig 8) with both $leftVmMem$ and $leftVmCpu$ range from [0,1]. We observe that the value is better when $leftVmMem$ is closed to 1 and the $leftVmCpu$ is closed to 0. The meaning of the rule is to select a VM with large memory and small CPU. By examining the Table I, we found that the *Medium* type (825 MHz, 800 MB) has the highest value. Hence, during the container allocation, this rule will compare the existing VMs and the *Medium* type of VM to decide the allocation,

Fig. 6: unused CPU and CPU overhead comparison among evolved, *largest*, and *just-fit* rules



(a) unused CPU **(b)** CPU overhead

```
((coCpu - (vmMemOverhead * ((leftVmMem * leftVmCpu) * leftVmMem))) * ((((leftVmMem *
leftVmMem) * leftVmMem) * ((leftVmMem * leftVmMem) * leftVmMem)) * ((leftVmMem * leftVmMem)
* ((leftVmMem * leftVmMem) * leftVmMem)))) - ((leftVmMem * coMem) / (((coCpu - coCpu) /
vmCpuOverhead) / vmCpuOverhead))
```

Simplify

```
coCpu * leftVmMem^11- (1 + vmMemOverhead * leftVmMem^13 * leftVmCpu)
```

Fig. 7: Rule Simplification

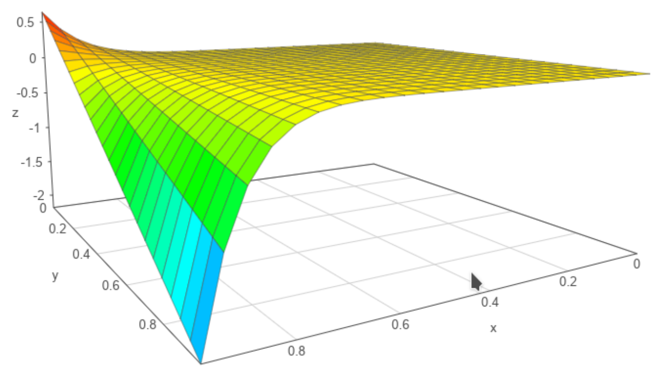


Fig. 8: This graph shows the landscape of the GP tree: $f = 0.02 \times leftVmMem^{11} - (1 + 0.05 \times leftVmMem^{13} \times leftVmCpu)$ where the x-axis is the leftVmMem and y-axis is the leftVmCpu for a candidate VM.

which is the reason that the rule performs better than the human-designed rules.

VII. Conclusion

The advent of container-based clouds brings not only the opportunity to improve the utilization of data centers but also the challenge of server consolidation. The current techniques of server consolidation in container-based clouds mostly applied human-designed rules in VM selection and creation. These rules often neglect the impact of the workload of applications and the size of VMs, therefore, they lead to a low utilization of resources.

In this paper, we have successfully developed a GPHH system on the VM selection and creation problem for online container allocation. The novelty of this system lies on the new problem model, two allocation frameworks, and the GP system with new fitness function and terminal set. The experiment results verify the effectiveness of our algorithms as compared to the state-of-the-art human-designed rules in the literature. Our algorithm outperforms other algorithms in terms of energy consumption. The analysis not only provides the reasons for the poor performance from the human-designed rules but also shows the insights from evolved rules.

References

[1] B. Familiar, *Microservices, IoT and Azure: leveraging DevOps and Microservice architecture to deliver SaaS solutions*. Apress, 2015.

[2] C. Pahl, "Containerization and the paas cloud," *IEEE Cloud Computing*, vol. 2, no. 3, pp. 24–31, 2015.

[3] W. Song, Z. Xiao, Q. Chen, and H. Luo, "Adaptive resource provisioning for the cloud using online bin packing," *IEEE Transactions on Computers*, vol. 63, no. 11, pp. 2647–2660, 2014.

[4] K. Kaur, T. Dhand, N. Kumar, and S. Zeadally, "Container-as-a-service at the edge: Trade-off between energy efficiency and service availability at fog nano data centers," *IEEE Wireless Communications*, vol. 24, no. 3, pp. 48–56, June 2017.

[5] P. K. Sundararajan, E. Feller, J. Forgeat, and O. J. Mengshoel, "A constrained genetic algorithm for rebalancing of services in cloud data centers," in *2015 IEEE 8th International Conference on Cloud Computing*, June 2015, pp. 653–660.

[6] S. F. Piraghaj, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, "A Framework and Algorithm for Energy Efficient Container Consolidation in Cloud Data Centers," *Proceedings - 2015 IEEE International Conference on Data Science and Data Intensive Systems*, pp. 368–375, 2015.

[7] W. Itani, A. Kayssi, and A. Chehab, "Privacy as a service: Privacy-aware data storage and processing in cloud computing architectures," in *Dependable, Autonomic and Secure Computing, 2009. DASC'09. Eighth IEEE International Conference on*. IEEE, 2009, pp. 711–716.

[8] Z. Á. Mann, "Interplay of virtual machine selection and virtual machine placement," in *Lecture Notes in Computer Science*. Cham: Springer, 2016, vol. 9846 LNCS, pp. 137–151.

[9] T. Wood, P. J. Shenoy, A. Venkataramani, and M. S. Yousif, "Sandpiper - Black-box and gray-box resource management for virtual machines." *Computer Networks*, vol. 53, no. 17, pp. 2923–2938, 2009.

[10] K. Maruyama, S. K. Chang, and D. T. Tang, "A general packing algorithm for multidimensional resource requirements," *International Journal of Computer & Information Sciences*, vol. 6, no. 2, pp. 131–149, Jun 1977. [Online]. Available: https://doi.org/10.1007/BF00999302

[11] D. S. Johnson, "Fast algorithms for bin packing," *Journal of Computer and System Sciences, Elsevier*, vol. 8, no. 3, pp. 272 – 314, 1974.

[12] E. G. Coffman Jr., J. Csirik, G. Galambos, S. Martello, and D. Vigo, *Bin Packing Approximation Algorithms: Survey and Classification*. New York, NY: Springer New York, 2013, pp. 455–531. [Online]. Available: https://doi.org/10.1007/978-1-4419-7997-1_35

[13] C. C. Lee and D. T. Lee, "A simple on-line bin-packing algorithm," *J. ACM*, vol. 32, no. 3, pp. 562–572, Jul. 1985. [Online]. Available: http://doi.acm.org/10.1145/3828.3833

[14] B. Tan, H. Ma, and Y. Mei, "A genetic programming hyper-heuristic approach for online resource allocation in container-based clouds."

[15] C. Guerrero, I. Lera, and C. Juiz, "Genetic algorithm for multi-objective optimization of container allocation in cloud architecture," *Journal of Grid Computing*, vol. 16, no. 1, pp. 113–135, Mar 2018. [Online]. Available: https://doi.org/10.1007/s10723-017-9419-x

[16] Y. Tao, X. Wang, X. Xu, and Y. Chen, "Dynamic resource allocation algorithm for container-based service computing," in *2017 IEEE 13th International Symposium on Autonomous Decentralized System (ISADS)*. IEEE, 2017, pp. 61–67.

[17] E. Reshetova, J. Karhunen, T. Nyman, and N. Asokan, "Security of os-level virtualization technologies," in *Nordic Conference on Secure IT Systems*. Springer, 2014, pp. 77–93.

[18] X. Guan, X. Wan, B.-Y. Choi, S. Song, and J. Zhu, "Application oriented dynamic resource allocation for data centers using docker containers," *IEEE Communications Letters*, vol. 21, no. 3, pp. 504–507, 2017.

[19] Z. A. Mann, "Resource optimization across the cloud stack," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 1, pp. 169–182, 2018.

[20] 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, special Section: Energy efficiency in large-scale distributed systems. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X11000689

[21] J. R. Koza, “Genetic programming as a means for programming computers by natural selection,” *Statistics and Computing*, vol. 4, no. 2, pp. 87–112, Jun 1994. [Online]. Available: https://doi.org/10.1007/BF00175355

[22] J. Xie, Y. Mei, A. T. Ernst, X. Li, and A. Song, “A genetic programming-based hyper-heuristic approach for storage location assignment problem,” in *IEEE Congress on Evolutionary Computation (CEC)*, Beijing, China, 2014, pp. 3000–3007.

[23] ——, “Scaling up solutions to storage location assignment problems by genetic programming,” in *Simulated Evolution and Learning*, G. Dick, W. N. Browne, P. Whigham, M. Zhang, L. T. Bui, H. Ishibuchi, Y. Jin, X. Li, Y. Shi, P. Singh, K. C. Tan, and K. Tang, Eds. Cham: Springer International Publishing, 2014, pp. 691–702.

[24] Y. Liu, Y. Mei, M. Zhang, and Z. Zhang, “Automated heuristic design using genetic programming hyper-heuristic for uncertain capacitated arc routing problem,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO ’17. New York, NY, USA: ACM, 2017, pp. 290–297.

[25] J. Jacobsen-Grocott, Y. Mei, G. Chen, and M. Zhang, “Evolving heuristics for dynamic vehicle routing with time windows using genetic programming,” in *2017 IEEE Congress on Evolutionary Computation (CEC)*, June 2017, pp. 1948–1955.

[26] Y. Mei and M. Zhang, “Genetic programming hyper-heuristic for stochastic team orienteering problem with time windows,” in *2018 IEEE Congress on Evolutionary Computation (CEC)*, July 2018, pp. 1–8.

[27] X. Fan, W.-D. Weber, and L. A. Barroso, “Power provisioning for a warehouse-sized computer,” *ACM SIGARCH Computer Architecture News*, vol. 35, no. June, p. 13, 2007.

[28] “Automatic vertical scaling,” https://docs.jelastic.com/automatic-vertical-scaling/, accessed: 2018-10-05.

[29] S. Nguyen, M. Zhang, and K. C. Tan, “Surrogate-assisted genetic programming with simplified models for automated design of dispatching rules,” *IEEE transactions on cybernetics*, vol. 47, no. 9, pp. 2951–2965, 2017.

[30] T. Hildebrandt, J. Heger, and B. Scholz-Reiter, “Towards improved dispatching rules for complex shop floor scenarios: A genetic programming approach,” in *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO ’10. New York, NY, USA: ACM, 2010, pp. 257–264. [Online]. Available: http://doi.acm.org/10.1145/1830483.1830530

[31] Y. Mei, M. Zhang, and S. Nyugen, “Feature selection in evolving job shop dispatching rules with genetic programming,” in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, ser. GECCO ’16. New York, NY, USA: ACM, 2016, pp. 365–372. [Online]. Available: http://doi.acm.org/10.1145/2908812.2908822

[32] E. K. Burke, M. R. Hyde, and G. Kendall, “Evolving bin packing heuristics with genetic programming,” in *Parallel Problem Solving from Nature*. Berlin, Heidelberg: Springer, 2006, pp. 860–869.

[33] S. Shen, V. v. Beek, and A. Iosup, “Statistical characterization of business-critical workloads hosted in cloud datacenters,” in *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2015, pp. 465–474.

[34] “Revenue share of global server operating system market in 2015, by operating system,” https://www.statista.com/statistics/639574/worldwide-server-operating-system-market-share/, accessed: 2018-10-05.

[35] S. et al, “A java-based evolutionary computation research system,” https://cs.gmu.edu/~eclab/projects/ecj//, accessed: 2018-10-05.