# A Hybrid Genetic Programming Hyper-Heuristic Approach for Online Two-level Resource Allocation 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 the software industry and cloud computing. Containers support the fast development of web applications and they have the potential to reduce energy consumption in data centers. Containers are usually first allocated to virtual machines (VMs) and VMs are allocated to physical machines. The container allocation is a challenging task which involves a two-level allocation problem. Current research overly simplifies the container allocation into a one-level allocation problem and uses simple rule-based approaches to solve the problem. As a result, the resource is not allocated efficiently which leads to high energy consumption. This paper provides a novel definition of the two-level container allocation problem. Then, we develop a hybrid approach using genetic programming hyper-heuristics combined with human-designed rules to solve the problem. The experiments show that our hybrid approach is able to significantly reduce energy consumption than solely using human-designed rules.**

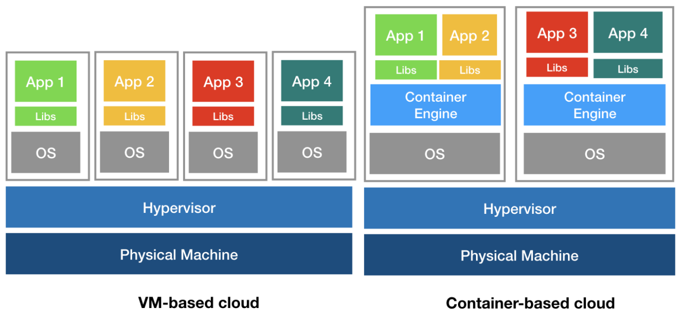
***Index Terms*—container-based clouds, resource allocation, genetic programming, evolutionary computation, hyper-heuristic**

## I. Introduction

Along with the increase 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 (OS) and libraries for each application, container technology achieves a fast developing, testing, and deployment by sharing a host OS among applications.

In addition, using containers can reduce energy consumption from data centers [2]. However, the problem of *online container allocation* (container allocation in short) is new and challenging to solve due to its *NP-completeness*. To reduce energy consumption, containers need to be allocated to minimum number of physical machines (PMs) so that idle PMs can be turned off. Compared to VM-based clouds, container allocation is a new problem because of the two-level structure (see Fig 1) where containers are first allocated to VMs, then VMs are allocated to PMs. On the other hand, in VM-based clouds, VMs are allocated to PMs. Existing resource allocation methods in VM-based clouds cannot be directly applied in container-based clouds. The online container allocation problem is *NP-complete* because, at each level, the allocation can be considered as a vector bin packing problem [3].

**Fig. 1:** Container-based cloud vs. VM-based cloud



Current research on container allocation has deficiencies on both problem definition and solutions. From the problem perspective, most studies either simplify the two-level problem to one-level [4], [5] or neglect critical factors such as affinity constraints in their problem definition [6]. The major drawback of these simplifications is that they can only be applied to limited scenarios where all containers can be co-located. However, one of the most important features of the container is that they have various affinity constraints, e.g. only containers that require the same OS can be allocated to the same VM [7]. Hence, the container allocation problem should be solved with the consideration of affinity constraints.

From the solution perspective, current works [8], [9] mostly apply AnyFit-based algorithms [10] which consider only the existing bins, therefore, have a limited search space. Instead, a reservation-based technique [11] is more promising. A reservation technique, which considers not only existing bins but empty bins, has been developed for one-dimensional online bin packing problem. However, these reservation techniques, such as *Harmonic-Fit* [12], cannot be applied to solve the container allocation problem directly because the resources are multi-dimensional (i.e CPU and memory) and the bins are heterogeneous (e.g. various types of VMs). Therefore, efficient and effective methods are needed for the online container allocation problem.

Hence, this research contributes to a novel problem definition and proposes a novel reservation-based algorithm. This novel definition separates the problem into four decision-making procedures: VM selection and creation, PM selection and creation. New features as affinity constraints and VM overheads are also first introduced in this work. Then, we focus on developing a hybrid approach using 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 deciding VM selection and creation simultaneously and use human-designed rules for allocating VMs to PMs.

We propose a learning algorithm: a GPHH approach for automatically generating rules for VM selection and creation simultaneously. Our previous work [13] developed a GPHH for VM selection solely. It has been shown that GPHH is able to use multiple predefined characteristics and the rules generated by GPHH outperform human-designed rules. Therefore, this work develops a GPHH-based reservation algorithm. The rules evolved by GPHH can be used to allocate containers to either an existing VM or a new VM with a selected type.

The overall goal of this paper is to propose a reservation-based algorithm which uses GPHH and human-designed rules to reduce energy in the two-level container allocation problem. More specifically, we have the following objectives:

1) To introduce a novel problem definition for two-level container allocation;
2) To develop a hybrid approach GPHH and human-designed rules;
3) To evaluate our proposed approach on benchmark datasets;

## II. Related Work

### A. Drawbacks on the Problem Definition

The *online container allocation* problem allocates containers to PMs immediately when containers arrive with the aim of achieving the lowest energy consumption.

Previous works on the container allocation problem have two issues. First, some research simplifies the two-level allocation problem into a one-level allocation problem which allocates containers directly to PMs [14]. The major drawback of the simplification is that their allocation approaches can only be used in a narrow range of scenarios where all containers can be co-located. However, containers should be allocated to VMs instead of PMs to satisfy the affinity constraints of containers.

The affinity constraint [15] defines which containers can be co-located and this is a critical issue in container-based clouds. Compared to VMs, containers have more requirements such as distinct OSs, software libraries, and different levels of security [16]. Therefore, containers are naturally running in a different environment and not all containers can be co-located. VMs could provide a strong security and performance isolation as well as OSs [16]. Hence, a generalized model for container allocation should include two levels: containers to VMs and VMs to PMs.

The second issue in current research is that, although some research considers container allocation as a two-level problem, they neglect some critical factors such as VM overheads [6], [17]. VM overheads are generated by a VM hypervisor which consumes resources (CPU and memory) on the host PM. Creating tiny VMs (e.g. one VM for each container) leads to swarms of VMs, consequently generating 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.

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. Additionally, we consider the VM overheads and affinity constraints.

### B. Existing allocation methods

Existing research [6], [8] on the container allocation problem mostly apply AnyFit-based algorithms for selection and creation. AnyFit-based algorithms [18], such as BestFit and FirstFit, have a major drawback that they only select the existing bins (e.g. VMs and PMs) until none is available [19]. This strategy limits the search space of the allocation solutions. Therefore, they can hardly find the optimal solution.

To overcome this drawback, the reservation technique is promising. Lee and et al. [12] propose a *Harmonic-Fit* algorithm for the one-dimensional online bin packing problem. This algorithm defines multiple categories of items and reserves bins for each category. Each item is only allocated to bins reserved for a certain category. *Harmonic-Fit* has shown better performance than the AnyFit-based algorithms [12]. However, *Harmonic-Fit* algorithm cannot be simply applied to solve the container allocation problem because it neither tackles the multi-dimensional problem nor solves the heterogeneous bins problem (e.g. multiple types of VM).

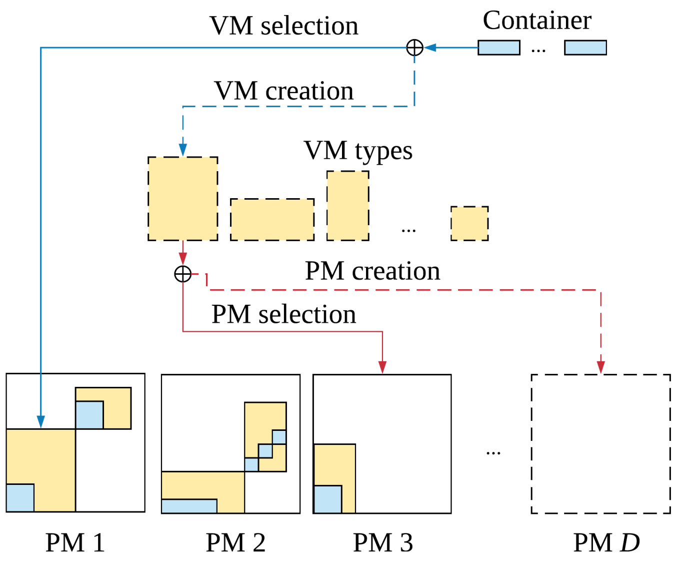
Hence, this research aims to develop a reservation technique to tackle the multi-dimensional allocation problem with heterogeneous bins. For multi-dimensional resources, our previous work [13] has successfully applied GPHH for evolving VM selection rules. These evolved rules transfer multi-dimensional resources into a score for the BestFit algorithm to select the best VM. The experiments have shown the evolved rules can reduce energy consumption significantly compared with human-designed rules.

The success of applying GPHH motivated us to develop a reservation technique with GPHH evolved rules to select existing VMs and create VMs from multiple types. Next section provides a brief background of GPHH.

### C. Genetic Programming

Genetic programming [20] is an evolutionary computation technique, to automatically find computer programs for solving a specific task. In a GP population, each individual represents a computer program. In each generation, a fitness function evaluates these programs. Then, the individuals will go through several genetic operators such as selection, crossover, and

**Fig. 2:** The online container allocation procedure.



mutation. Elitism keeps a number of top individuals (in terms of fitness values) to the next generation while others are discarded.

GPHH has been successfully applied to a variety of problems. In the field of job shop scheduling (JSS) problems, GP outperforms neural network and the linear combination techniques [21]. For Dynamic JSS, GPHH has been demonstrated to be a powerful hyper-heuristic [22]. GP has also been used for a number of variants of bin packing problems such as 1-D bin packing [23] and 2-D strip packing [24]. In these cases, GP has superior performance to the BestFit rule. For real-world applications, Xie et al. [25] propose a GPHH for the storage location assignment problem. Liu et al. [26] and Jacobsen-Grocott et al. [27] use GP to design heuristics for the vehicle routing problem. Mei et al. solve the stochastic team orienteering problem with GPHH [28]. These works show that GP has been successfully used for generating rules in various online problems. Therefore, we also apply GPHH in our problem.

## III. Problem Description

Assuming there is a set of containers arriving at data center during time period between $t_1$ and $t_2$. The overall objective of online container allocation is to allocate containers to existing VMs or new created VMs, then allocates the VMs to PMs, so that the accumulated energy consumption of all PMs is minimized.

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 heterogeneous VMs with different types. Each type of VM is predefined with a tuple of resources (e.g. Small VM [825 MHz, 800 MB]). In addition to the resource capacity, each VM has overheads. VM overheads represent the resources consumed by a hypervisor. The overheads are also represented by tuples of resources for each type of VM. Each VM can install and run a type of OS. The supporting types of OS are 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-to-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 the container is much more general and realistic [29]. Each container has an OS requirement, which means it can only be allocated to a VM which runs the same OS.

We model the online container allocation problem as four decision-making procedures: VM selection and creation (blue lines), PM selection and creation (red lines) (see Fig 2). VM selection chooses existing VMs to allocate containers. Alternatively, VM creation chooses a type of VM, which is predefined by cloud providers, then creates the VM to allocate the container. The OS type of the VM is decided by the first container's OS requirement. After a new VM is created, PM selection chooses existing PMs to allocate the VMs. Similar to VM creation, PM creation creates a type of PM and allocates the VM into it.

To evaluate a container allocation, we consider the accumulated energy $AE$ which has been used in our previous work [13]. The accumulated energy (see Eq. 1) is calculated by aggregating the energy consumption $P_d^t$ of all activated PMs throughout the time period $[t_1, t_2]$. That is, we add the energy consumption of all PMs at every time interval $t_i$. The energy model of a PM (Eq.2) is a widely used model proposed by Fan [30]. In their energy model, $P^{idle}$ and $P^{max}$ are the energy consumption when a PM is idle and fully used. $u_{cpu}^t(d)$ is the CPU utilization of a PM $d$ at time $t$. The objective of container allocation is to minimize the overall energy consumption, i.e., $Min(AE)$.

$$AE = \int_{t=t_1}^{t=t_2} \sum_{d=1}^{D} P_d^t \tag{1}$$

$$P_d^t = P_d^{idle} + (P_d^{max} - P_d^{idle}) \cdot u_{cpu}^t(d) \tag{2}$$

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

## IV. Methodology

This section describes our hybrid approach that uses GPHH to evolve VM creation and selection rules combined with human-designed rules for PM selection. Since we assume PMs are homogeneous, PM creation is straightforward. We first describe the simulation model of the problem. Then, we describe the GPHH for VM selection and creation.

**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 |

## A. Simulation Model

We create a simulation to train and test the allocation rules for the online container allocation. All the experiments in this paper are based on the simulation model, which has been used in our previous work [13]. Below 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) Arrived containers must be allocated immediately;
4) Overload threshold of VM/PM is 100% of resource utilization;
5) No weights of containers, which means they are equally important;
6) Five types of VMs (see Table I) and homogeneous PMs (all PMs have the same initial resources);

Each simulation has two steps, data center initialization and container allocation. In the first step, the data center initialization randomly initializes a data center with a set of containers running on a set of VMs and PMs. In the second step, a set of containers arrives at the data center one by one.

The data center initialization is designed to simulate a real-world scenario in which there are PMs with VMs and containers. Random initialization can also help to test the robustness of the algorithm. The robustness indicates whether an algorithm is sensitive to the initial state.

This work focuses on using GPHH to generate rules for VM selection and creation and employs human-designed rules in PM selection and creation. **In *PM selection*, we employ First Fit** [19] which allocates the VM to the PM with enough resources (if multiple PMs have enough resources, select the first one). In *PM creation*, since we consider homogeneous PMs, the creation rule creates a 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 containers are allocated. Then, the performance measure is the accumulated energy consumption of all used PMs.

To reliably measure the effectiveness of evolved rules, a large number of simulations is usually needed [31] (e.g. 30 to 50 simulations). Therefore, in our experiments, we use two sets of simulations, each has 50 simulations for training and testing. The training set is used to train the allocation rules and the test set is used to examine the performance of trained rules on unseen data.

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

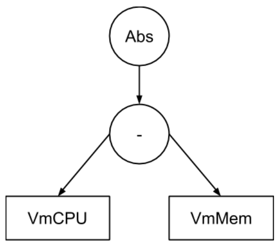


**TABLE II:** Terminal and Function sets of GP

| Symbol | Description |
|---|---|
| Attributes for Container Allocation | |
| **leftVmMem** | remaining memory of a VM |
| **leftVmCpu** | remaining CPU of a VM |
| **coCpu** | container CPU |
| **coMem** | container memory |
| **vmMemOverhead** | memory overhead of a VM |
| **vmCpuOverhead** | CPU overhead of a VM |
| **Function set** | +,-,×, protected % |

## B. Representation and Terminal set

We use trees to represent rules of online container allocation. An example tree of the manually designed $sub$ rule (see [13]) $|vmCPU - vmMem|$ is shown in Fig 3. In contrast, relevant features can lead to a significant improvement [32]. Table II shows the terminal set and function set used by GP to construct rules.

We design six features for the terminal set. Our previous work [13] designs four features for VM selection. *leftVmMem* and *leftVMCpu* are the remaining resources of a VM after the container is allocated in. *coCpu* and *coMem* are the resource requirement of the container. This work introduces two extra features. *vmMemOverhead* and *vmCpuOverhead* are the amount of CPU and memory overheads when a new VM is created. If the evolved rule selects an existing VM then the overhead is 0. For the function set, we use four basic arithmetic operators to construct rules (the protected division returns a value of 1 when divided by 0).

## C. Fitness Function

To evaluate the quality of an individual rule, we design a fitness function as follows:

$$fitness = \frac{\sum_{k=1}^{|simulations|} \frac{\widetilde{AE}}{N}}{|simulations|} \tag{3}$$

where $\widetilde{AE}$ is the accumulated energy consumption of a simulation normalized by a benchmark rule (see Eq.1). $N$ is the number of containers of a simulation. With $\frac{\widetilde{AE}}{N}$, we calculate the average accumulated energy consumption of a simulation. Then, we average the $\widetilde{AE}$ of all simulations.

The $\widetilde{AE}$ is normalized as follows:

$$\widetilde{AE} = \frac{AE}{AE_b} \tag{4}$$

where we normalize the $AE$ of the evolved rule with the $AE_b$ of a benchmark rule. The reason for using the normalized $AE$ is that different simulations may have differences in $AE$. It is unfair to use the aggregation of $AE$ of all simulations to compare the algorithms.

For example, we applied two algorithms A and B on two allocation task $\alpha$ and $\beta$. 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 value of $AE = 37500$, it is unfair to say A and B perform the same. Because, at $\alpha$, the $AE$ difference between A and B is only 16.7%, but in $\beta$, 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 consumption, it is easy to see that algorithm A performs much better than B.

### *D. Algorithm*

Algorithm 1 shows a GPHH for the online container allocation. The algorithm follows a standard GP framework with an embedded simulation as the evaluation process (from *line 3* to *line 23*). VM selection and creation (see Algorithm 2) decides whether a container is allocated to an existing VM or a new VM. In addition, it also decides which type of VM to create.

At the beginning, lists of running VMs and PMs, VM types are provided. In *line 3*, we temporarily append empty VMs (one for each VM type) into the existing available VM list. Consequently, the VM selection and creation rule evaluates the empty VMs as well as the existing VMs. *Line 4* filters the VMs without enough resources or OS-incompatible. From *line 5* to 12, the VM creation and selection rule evaluates all the candidate VMs and assign them a score. *Line 13* removes the unused empty VMs from the list of existing VMs. The algorithm ends and returns the VM with the highest score.

**Algorithm 1:** GPHH for online container allocation

```
Input  : A set of simulations,
Output: The best VM selection and creation rule
1  Initialize a population of rules $R$;
2  while stopping criteria is not met do
3      for each rule $r$ in $R$ do
4          for each simulation in the training simulations do
5              $AE = 0$;
6              Initialize the data center;
7              for each container do
8                  vm = vmSelectionCreation(container, $r$);
9                  allocate(container, vm);
10                 if vm is new then
11                     add(vm, the list of VMs);
12                     pm = pmSelection(vm);
13                     if pm is none then
14                         pm = create(pm);
15                         add(pm, the list of PMs);
16                     end
17                     allocate(vm, pm);
18                 end
19                 $AE$ += calculateEnergy(the list of PMs);
20             end
21         end
22         calculate $\widetilde{AE}$;
23         fit = fitness($\widetilde{AE}$);
24     end
25     TournamentSelection;
26     Crossover;
27     Mutation;
28     Reproduction;
29 end
30 return the best rule;
```

## V. Design of Experiment

This experiment compares our hybrid approach of with human-designed rules to demonstrate the effectiveness of the proposed hybrid algorithm. Then, we provide further analysis of the evolved 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. Human-design rules*

We consider two benchmark rules *sub&Just-fit/FF* proposed in [33] and *LFHS&Largest/FF* [6] as they are the state-of-the-art rules. [33] applies a BestFit with $sub$ rule (see Section IV-B). For the VM creation rule, [33] designs a *Just-fit* rule. *Just-fit* creates the smallest VM that can satisfy the resource requirement of the container. [6] applies a *Least Full Host Selection (LFHS)* for VM selection and a *Largest* rule for VM creation. *LFHS* selects the VM with the lowest CPU utilization. *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.

For allocating VMs to PMs, we apply First Fit in both human-designed rules and our evolved rules.

### *B. Dataset*

We designed 4 instances (see Table III) with an increasing number of OSs as the affinity constraint. The frequency of OSs is to simulate a real-world market share of OS [34]. Each of the above instances includes 100 simulation cases where 50 for training and 50 for testing. Each case includes 200 containers to be allocated.

We use a real-world dataset (AuverGrid trace [35]) to construct the resource requirement of containers. Fig 4 shows the distribution of CPU and memory requirement of the

**Algorithm 2:** Algorithm for vmSelectionCreation

```
Input : A rule, A container, A list of VM types, A
        list of VMs, A list of PMs,
Output: The best VM
1  $BestVM = nil$;
2  $bestScore = nil$;
3  **Append the list of VM types to the VM list as
     empty VMs**;
4  Select the VMs with enough resources and
     OS-compatible for this container from the list of
     VMs ;
5  while $VM_i$ in the list of available VMs do
6      $score$ =rule($container, VM_i$);
7      if $score > bestScore$ then
8          $bestScore = score$;
9          $BestVM = VM_i$;
10     end
11     $i = i + 1$;
12 end
13 **Remove the unused empty VMs from the list of
     VMs**;
14 return $BestVM$;
```

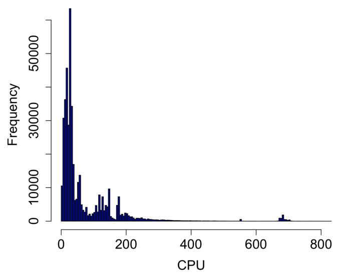
**TABLE III:** Test instances

| instances | number of OS | OS distribution (%) |
|---|---|---|
| instance 1 | 2 | 95–5 |
| instance 2 | 3 | 50–30–20 |
| instance 3 | 4 | 62.5–17.5–15-5–4.5 |
| instance 4 | 5 | 17.9–45.4–23.6–10.5–2.6 |

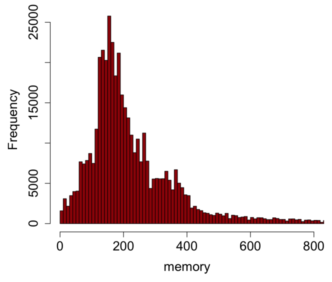
containers in this dataset. We observe that the memory usage is on average three times as much as the CPU usage.

We define the size of a PM as (3300, 4000) in CPU and memory. The configuration of VMs is shown in Table I. All results have been tested with Wilcoxon signed rank test between the rules from our hybrid algorithm and the existing rules.

**Fig. 4:** Resource usage frequency histogram of the real world dataset



**(a)** CPU usage frequency



**(b)** Memory usage frequency

**TABLE IV:** 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) |

## C. Parameter Settings

Table IV shows the parameters that we used in all experiments. Most parameters are the standard values in GPHH field [32]. The algorithm was implemented using ECJ [36].

## D. Result analysis

Fig 5 shows the accumulated energy consumption comparison among three rules: *evo/FF* (short for evolved rules/First Fit), *sub&Just-fit/FF*, and *LFHS&Largest/FF* for the first 10 simulations (out of 50). Table V shows the mean and standard deviation of the energy consumption of 50 simulations for 4 OSs among three rules. Table VI shows the win-draw-loss of the *evo/FF* and *LFHS&Largest/FF* over 50 simulations.

The results clearly show that the *evo/FF* and *LFHS&Largest/FF* have a great advantage over the *sub&Just-fit/FF* rule. In comparison with *evo/FF* and *LFHS&Largest/FF* rule, the *evo/FF* dominates *LFHS&Largest/FF* in all scenarios.

To understand the reason for the poor performance of human-designed rules, we analyze the waste of resources in PMs. The waste resources in PMs consists of two parts – unused resources and the overhead. Unused resources are the idle resources in a PM. The overhead is the aggregation of all VM overheads in a PM.

Fig 6 shows the increases of unused CPU and CPU overhead of *evo/FF*, *LFHS&Largest/FF*, and *sub&Just-fit/FF* from instance 10 throughout the 200 container allocations. At the beginning (from 0 to 75 containers in Fig 6a), *sub&Just-fit/FF* performs the best because it creates the smallest possible VMs for containers. In this period, it wastes the least of resources. However, the unused resources accumulated fast because the created VMs have few resources to accommodate other containers in the future. 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 *LFHS&Largest/FF* rule reduces the unused resources by creating the largest VMs for future containers. However, the *Largest* rule neglects the effect of OS constraint. When the rule creates the largest possible VM for a container which requires an OS with a low frequency (e.g. 2.6%), it is unlikely that the VM will be filled in for a long period of time because there are not many containers who also require this type of OS. Hence, the resources in this

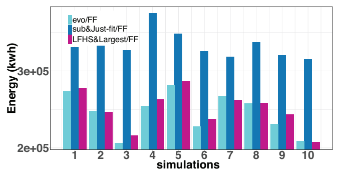
**TABLE V:** Mean and standard deviation of the energy consumption (kwh) of 50 simulations for 4 OSs among *sub&Just-fit/FF*, *LFHS&Largest/FF*, *evo/FF*.

| | *sub&Just-fit/FF* | *LFHS&Largest/FF* | *evo/FF* |
|---|---|---|---|
| OS2 | 319395.5 ± 29329.7 | 260695.5 ± 29140.3 | **242837.4 ± 24500.35** |
| OS3 | 338685.5 ± 31475.3 | 269360.1 ± 28839.94 | **253285.7 ± 24730.37** |
| OS4 | 345279 ± 27997.99 | 281830.6 ± 29274.85 | **256319.3 ± 22998.9** |
| OS5 | 363917.4 ± 31868.45 | 291642.2 ± 27662.37 | **265346.4 ± 24252.62** |

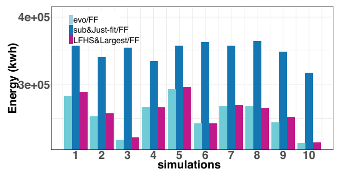
VM will be unused for a long time. With *Largest* rule, the unused resource increases along with the number of OSs.

The *evo/FF*, on the other hand, creates a VM neither the smallest or the largest. Instead, it considers the distribution of CPU and memory of containers and the size of different VM types. Therefore, the evolved rules can avoid creating many small VMs as well as creating large VMs. A detailed explanation of the behavior of rules will be discussed in Section VI.

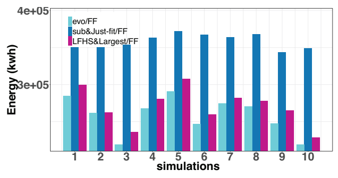
**Fig. 5:** Accumulated energy comparison (first 10 simulations) among three rules:*evo/FF*, *sub&Just-fit/FF*, and *LFHS&Largest/FF*



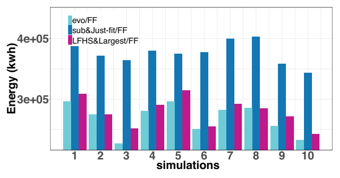
**(a)** OS2



**(b)** OS3



**(c)** OS4

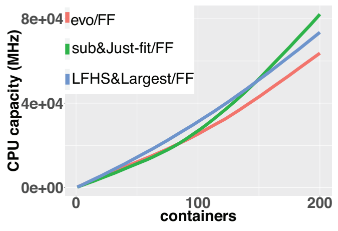


**(d)** OS5

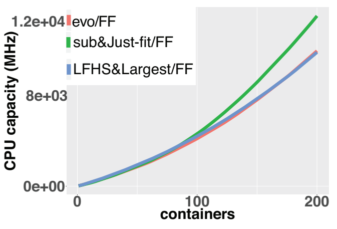
**TABLE VI:** Comparison between *evo/FF* and *LFHS&Largest/FF* rule in four scenarios

| | OS2 | OS3 | OS4 | OS5 |
|---|---|---|---|---|
| *evo/FF* **vs.** *LFHS&Largest/FF* | 30-0-20 | 31-0-19 | 44-0-6 | 42-0-8 |

**Fig. 6:** Comparison among *evo/FF*, *LFHS&Largest/FF*, and *sub&Just-fit/FF* in unused CPU and CPU overhead



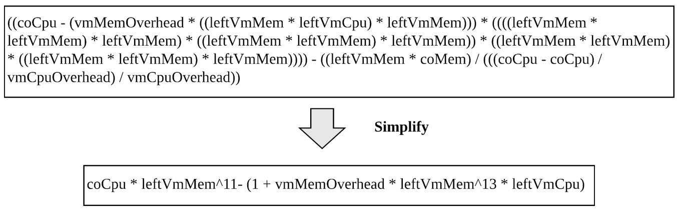
**(a)** unused CPU



**(b)** CPU overhead

## VI. Analysis and Discussion

This section provides an insight of the evolved rules generated by GPHH to understand the reason of their good performance shown above. In order to study the behavior of rules, we select a relatively simple rule which evolves on simulation 10. We manually simplify the rule as shown in Fig 7. This rule obtains a fitness value (with Eq.3) of 1210.47 on test in compared with 1596.97 of the *sub&Just-fit/FF* rule and 1303.47 of the *LFHS&Largest/FF* rule.



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

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

**Fig. 7:** Rule Simplification

We study the behavior of the rule for VM selection and creation separately. For VM selection, the evolved rule selects a PM solely based on PM's memory. We may simplify the rule as $coCpu \times leftVmMem^{11}$ because no new VM is created the vmCpuOverhead is 0. Since each allocation, the *coCpu* does not affect the behavior of the rule, the sole decision variable is the *leftVmMem*. This makes sense because, as we mentioned in Section V-B, 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, for VM creation, the evolved rule selects the VM type based on *leftVmCpu* and *leftVmMem*. We may further simplify the rule as follows. Since a new VM is created, the *vmMemOverhead* is 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 $f = 0.02 \times leftVmMem^{11} - (1 + 0.05 \times leftVmMem^{13} \times leftVmCpu)$.

We visualize the rule (see Fig 8) with both $leftVmMem$ and $leftVmCpu$ within a range of [0,1]. We observe that the score is better when the rule selects a VM with large memory and small CPU. From Table I, we found that the *Medium* type (825 MHz, 800 MB) has the highest value. Hence, during the

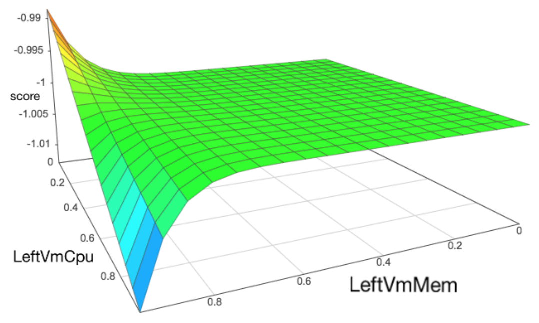


**Fig. 8:** This graph shows the landscape of the GP tree: $f = 0.02 \times leftVmMem^{11} - (1 + 0.05 \times leftVmMem^{13} \times leftVmCpu)$ where the x-axis is the leftVmMem and y-axis is the leftVmCpu for a candidate VM.

container allocation, this rule will compare the existing VMs and the *Medium* type of VM to decide the allocation, which is the reason that the rule performs better than the human-designed rules.

In conclusion, GPHH considers multiple features and generates rules to select and create VMs. These generated rules select VMs and types of VMs based on the features of the current environment that impacts the overall energy, therefore, they outperform the human-designed rules which only consider limited features.

## VII. Conclusion

This paper proposes a novel problem model for the two-level container allocation problem. We develop a hybrid approach GPHH to generate rules for VM selection and creation, and uses FirstFit for PM selection. The experiment shows that our hybrid approach outperforms existing rules for all test cases. In the future, we will apply a cooperative coevolution GPHH to evolve rules for both levels in the container allocation problem.

## 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, 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 *IEEE 8th International Conference on Cloud Computing*, Toronto, Canada, 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," *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 *Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing*. Chengdu, China: IEEE, 2009, pp. 711–716.

[8] Z. Á. Mann, "Interplay of virtual machine selection and virtual machine placement," Cham, Switzerland, pp. 137–151, 2016.

[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.

[11] E. G. Coffman, M. R. Garey, and D. S. Johnson, "Approximation algorithms for bin-packing—an updated survey," in *Algorithm design for computer system design*. Springer, 1984, pp. 49–106.

[12] C. C. Lee and D. T. Lee, "A simple on-line bin-packing algorithm," *Journal of the ACM*, vol. 32, no. 3, pp. 562–572, Jul. 1985.

[13] B. Tan, H. Ma, and Y. Mei, "A genetic programming hyper-heuristic approach for online resource allocation in container-based clouds," in *Australasian Joint Conference on Artificial Intelligence*. Cham: Springer International Publishing, 2018, pp. 146–152.

[14] 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, 2018.

[15] S. Sudevalayam and P. Kulkarni, "Affinity-aware modeling of cpu usage for provisioning virtualized applications," in *IEEE International Conference on Cloud Computing (CLOUD)*. Washington, USA: IEEE, 2011, pp. 139–146.

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

[17] 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.

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

[19] 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.

[20] 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.

[21] J. Branke, T. Hildebrandt, and B. Scholz-Reiter, "Hyper-heuristic evolution of dispatching rules: A comparison of rule representations," *Evolutionary Computation*, vol. 23, no. 2, pp. 249–277, Jun. 2015.

[22] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 5, pp. 621–639, Oct 2013.

[23] 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.

[24] E. K. Burke, M. Hyde, G. Kendall, and J. Woodward, "A genetic programming hyper-heuristic approach for evolving 2-d strip packing heuristics," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 6, pp. 942–958, Dec 2010.

[25] 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.

[26] 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. Berlin, Germany: ACM, 2017, pp. 290–297.

[27] J. Jacobsen-Grocott, Y. Mei, G. Chen, and M. Zhang, "Evolving heuristics for dynamic vehicle routing with time windows using genetic programming," in *IEEE Congress on Evolutionary Computation (CEC)*, Donostia - San Sebastián, Spain, 2017.

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

[29] "Automatic vertical scaling," https://docs.jelastic.com/automatic-vertical-scaling/, accessed: 2018-10-05.

[30] 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.

[31] 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.

[32] 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*, ser. GECCO '16. Denver, Colorado, USA: ACM, 2016, pp. 365–372.

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

[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. 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*, Shenzhen, China, 2015, pp. 465–474.

[36] L. Sean, "A java-based evolutionary computation research system," https://cs.gmu.edu/~eclab/projects/ecj//, accessed: 2018-10-05.