A Genetic Programming Hyper-heuristic Approach for online Resource Allocation in Container-based Clouds

Boxiong Tan, Hui Ma, Yi Mei

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

Abstract. The popularity of container-based clouds is its ability to deploy and run applications without launching an entire virtual machine (VM) for each application. Such a container supports a fast deployment of applications, which brings the potential to reduce the energy consumption of data centers. With the goal of energy reduction, it is more difficult to optimize the allocation of containers than traditional VM-based clouds because of the finer granularity of resources. Existing methods have shown poor performance in dealing with both balanced and unbalanced resources. In this paper, we first compare three human-design heuristics and show they cannot handle balanced and unbalanced resources scenarios well. We propose a learning-based algorithm: genetic programming hyper-heuristic (GPHH) to automatically generate a suitable heuristic for allocating containers in an online fashion. The results show that the proposed GPHH managed to evolve better heuristics than the human-designed ones in terms of energy consumption in a range of cloud scenarios.

Keywords: cloud computing, resource allocation, energy consumption, genetic programming, hyper-heuristic

1 Introduction

A container-based cloud [1] is a promising new technology for both software and cloud computing industries. Containers are beneficial for cloud providers because they can potentially reduce the energy of data centers [2]. Energy reduction is achieved by deploying more applications in fewer physical machines (PMs).

Although container-based clouds have better energy efficiency, the complexity for allocating both containers and VMs is much higher than solely managing VMs. In a container-based cloud, a typical PM may host multiple VMs with different operating systems. Each VM hosts multiple containers. This box-inside-box structure forces us to break down the container allocation process into two levels: containers to VMs and VMs to PMs.

To address the high complexity, this work considers a simplified structure and focuses on the challenge of the container allocation problem. Many cloud providers (e.g. Amazon) skip VM level and deploy containers directly to PMs. Moreover, we consider an online container allocation in which the request come in real time, and the information of each request (e.g. CPU and memory demand) is unknown until the request arrives.

Existing approaches [2] apply AnyFit-based algorithms with human-designed greedy rules such as *sub* and *sum* (detailed discussed in Section 2). We argue that the goal for resource allocation in clouds is to minimize the accumulated energy consumption instead of the cutting-point energy. It is critical to consider the order of creating new PMs when allocating containers [3]. Therefore, the container allocation problem can be treated as a scheduling task. Existing human-designed rules, therefore, may not be suitable for the scheduling task.

To address the drawbacks of human-design rules and the high design difficulty, we propose a learning algorithm: Genetic programming-based hyper-heuristic (GPHH) to automatically design scheduling rules using the information of a data center. To apply GPHH in the container allocation problem, we need to develop new terminal sets and fitness function.

In this paper, our contributions are:

- We compare the widely used human-designed greedy rules: *sub*, *sub*, and *random* in container allocation. This comparison provides an important insights. Therefore, we need to develop a learning algorithm to automatically generate rules to adapt all scenarios.
- We develop a GPHH for generating rules for online container allocation.

2 Background

Problem description: The container allocation problem can be described as, for a given set of t containers, each of which arrives at a time i, $0 <= i <= t$, the overall objective of container allocation is to allocate containers to PMs so that during the period of time of allocation, the accumulated energy consumption of PMs $E = \sum_{i=1}^{t} \sum_{j=1}^{p} P_j \cdot [u_{cpu}(j) > 0]$ are minimized.

Assuming that we have t containers to be allocated into p Physical Machines (PMs). Each container i has a CPU demand A_i and a memory occupation M_i. Each PM $j \in \{1, \cdots, p\}$ has a CPU capacity PA_j and a memory capacity PM_j. A PM can host multiple containers. We consider all PMs have the same size of CPU capacity and memory.

Energy consumption P_j is the energy consumption of a PM j. $[u_{cpu}(j) > 0]$ returns 1 if $u_{cpu}(j) > 0$ (PM j is active), and 0 otherwise.

P_j is determined by a widely used energy model $P_j = \alpha \cdot P^{max} + (1 - \alpha) \cdot P^{max} \cdot u_{cpu}(j)$ [4]. The CPU and memory utilization of PM j are denoted as $u_{cpu}(j) = \sum_{i=1}^{t}(x_j^i \cdot A_i)$ and $u_{mem}(j) = \sum_{i=1}^{t}(x_j^i \cdot M_i)$. Where x_j^i is a binary value (e.g. 0 and 1) denoting whether a container i is allocated on a PM j.

Constraints: A container can be allocated on a PM if and only if the PM j has enough resources required by the container.The other constraint is that each container can only be deployed once.

Human-designed rules for online container allocation: AnyFit algorithms [2] are greedy-based algorithms and use human-designed rules for evaluating an allocation.Mann's [2] applied six rules (such as *sub*, *sum*, and *product*) for container allocation. However, the different effects of these rules have not been shown. Therefore, it motivates us to explore the effectiveness of the most used rules – *sum* and *sub* – in solving the problem of container allocation problem.

Sum is the most commonly used rule in multi-dimensional bin packing. It can be represented as $resourceA+resourceB$ in the two-dimensional case. Resources A and B are the residual resources of a chosen bin after the item has been allocated. The smaller the function result, the better the candidate bin. This heuristic tries to minimize the residual resources in all dimensions. It is based on a simple assumption that less residual resource results in fewer number of used bins.

Sub is designed to maintain the balance in a bin. It can be represented as $|resourceA - resourceB|$. Similar to *sub*, we prefer a smaller function value. *sub* rule tries to minimize the difference between the two resources. With the assumption of balanced resource allocation can lead to fewer bins.

In summary, the performance of these simple rules has not been well studied. Because of their simplicity, we believe they cannot fully capture the complex behavior of diverse resource requirements and temporal effect. Therefore, these two reasons motivate us to investigate a learning method: GPHH using the information from a data center to generate rules.

3 GPHH for online Container Allocation

This section describes the training, testing process, GPHH function and terminal nodes, and fitness function.

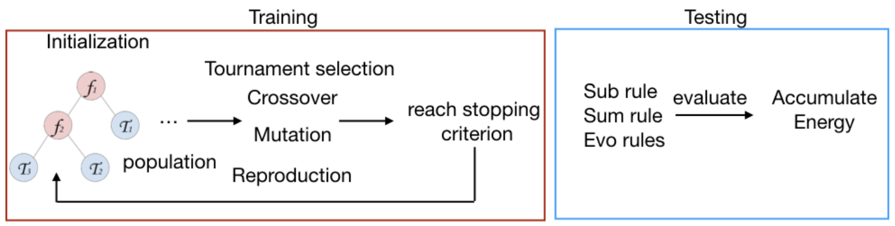

Fig. 1: The overview of GPHH training and testing process

Training and testing: both training and testing processes rely on the simulation of container allocation for evaluating the quality of rules (see Fig. 1). The simulation includes two parts: data center initialization and container allocation. Data center initialization randomly initializes a data center with PMs and containers. Without initialiazation, allocation algorithms perform similarily. Container allocation process allocate containers to the PM according to a BestFit-based algorithm (see Algorithm 1). The rule decides the goodness of a candidate PM.

For training, we follow a standard GP framework (details in [5]), which includes initialization, evaluation, crossover, mutation, and reproduction, to generate allocation rules. For testing, we run GP 30 times to generate 30 rules with different random seeds. Each evolved rule is applied on the test instances. The accumulated energy of each test instance is then normalized with the benchmark *sub* rule with equation $normalized\ E_{evolve} = \frac{E_{evolve}}{E_{sub}}$. Then, for each instance, we calculate the average normalized accumulated energy from 30 runs. Lastly, we applied the paired Wilcoxon test to calculate the statistic significance between the evolved rules and the benchmark rules (*sub*, *sum*, and *random*).

Algorithm 1: BestFit framework for the evolved rules

```
   Input  : container, A list of available PMs, features of the data center
   Output: The best PM
1  $BestPM = nil$;
2  $bestFitness = nil$;
3  while $PM_i$ in $PMs$ do
4      $fitness$ =Rule($container, PM_i, features$);
5      if $fitness > bestFitness$ then
6          $bestFitness = fitness$;
7          $BestPM = PM_i$;
8      end
9      $i = i + 1$;
10 end
11 return $BestPM$;
```

Function, Terminal Sets, and Fitness function: Our function set includes $\{+, -, \times\}$ and the protected $\div$ which returns 1 when divided by 0. Terminal set includes four features. The CPU and memory requirement of a container and the residual CPU and memory from a PM. Residual CPU and memory are calculated as the current PM's resources subtract the resource requirement of the container.

We calculate the fitness function $fitness\ function = \frac{\sum_{k=1}^{training\ instances} \frac{E}{t}}{training\ instances}$. The fitness value represents the average increase in energy consumption after allocating a container. Therefore, it is free from the bias of the randomize initial data center.

4 Experiments

We design three scenarios (see Table 1) for two objectives: testing whether existing rules can work well in container allocation and evaluate the proposed GPHH.

Experiment settings: Each of the three scenarios includes 100 of test instances. They are splitted equally into training and testing set. Each test instance consists of 200 containers to be allocated. The scenario with unbalanced containers and PMs is the most common scenario in the real world because of the diverse applications. We use a real-world dataset (AuverGrid trace) to generate test instances of unbalanced container scenario.

Table 1

scenarios	dataset	PM size (CPU[MHz], Memory[MB])
unbalanced containers and PMs	real-world dataset	(3300, 4000)
balanced containers and unbalanced PMs	synthetic dataset	(3300, 3300)
balanced containers and PMs	synthetic dataset	(3300, 3300)

For container generation, we randomly choose pairs of CPU and memory from the dataset with both resource requirement less than or equal to the maximum capacity. Balanced containers are generated from an exponential distribution with the rate $\lambda = 0.004$ in both CPU and memory. For initialization of a data center, we randomly generate 4 to 8 running PMs. Each VM will host at least one container. In addition, we use the corresponding dataset as the test cases for generating the initial containers in PMs.

To compare the performance between *sub* and *sum*, we add a *random* rule. The *random* rule chooses a random available PM instead of the best one. We intend to compare *sub* and *sum* with the *random* rule as a baseline. Hence, we can identify which rule performs badly in which scenario.

For GPHH, we use the population size of 1024. The number of generation is 100. For crossover, mutation, and reproduction, we use 0.8, 0.1, and 0.1 respectively. We use tournament selection and the size of the tournament is 7.

Experiment results: In all scenarios, the evolved rules show significant advantages than other rules (Table 2 to 4). Table 2 shows the Win-Draw-Loss of the unbalanced containers and PMs dataset among four algorithms. *sub* rule is significantly worse than all the other rules. Evolved rules dominate *sub* and *random*, and is better than *sum* with a small margin.

In balanced containers and unbalanced PMs scenario(Table 3), there is no statistic difference between *sub*, *sum* and *random* rules. In balanced containers and PMs (Table 4), evolved rules dominate other rules. Both *sub* and *sum* are significantly better than the *random* rule.

Table 2: real world scenarios.

	evo	sub	sum	random
evo	Nah	**49-0-1**	**30-0-20**	**45-0-5**
sub	1-0-49	Nah	2-1-47	5-2-43
sum	20-0-30	47-1-2	Nah	40-4-6
random	5-0-45	43-2-5	6-4-40	Nah

Table 3: unbalanced PMs

	evo	sub	sum	random
evo	Nah	**44-0-6**	**41-0-9**	**43-0-7**
sub	6-0-44	Nah	20-1-29	30-0-20
sum	9-0-41	29-1-20	Nah	33-1-16
random	7-0-43	20-0-30	16-1-33	Nah

To explain the goodness of evolved rules, Figure 4 shows the energy consumption of the data center while allocating 200 containers with four rules. The initial energy consumption are the same because of the same initialized data center. With the allocation processing, *random* rule (yellow) creates a new PM which incurs the sudden increase of energy while other rules can still allocate containers to existing PMs. Similarly, *sub* and *sum* create new PMs earlier than the evolved rule. Although, in most cases, all four rules create the same number of PMs (not in this case), evolved rules always allocate more containers to the existing PMs.

	evo	sub	sum	random
evo	Nah	**43-0-7**	**38-0-12**	**46-0-4**
sub	7-0-43	Nah	23-0-27	32-0-18
sum	12-0-38	27-0-23	Nah	35-0-15
random	4-0-46	18-0-32	15-0-35	Nah

Table 4: balanced containers and PMs

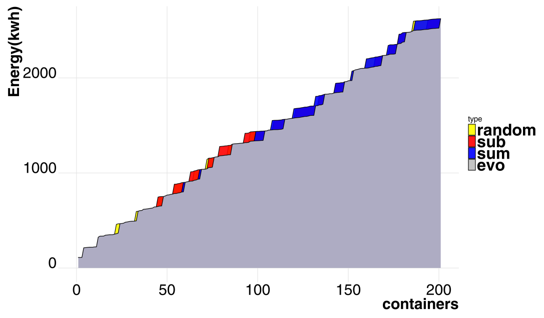

Fig. 2: The energy consumption of allocating 200 containers with four algorithms (from realworld dataset, run 17, test case 27)

5 Conclusion

In this paper, we first show that exising rules for container allocation do not perform well in dealing with real-world resource requirement and PM. Second, we develop a new GPHH approach for container allocation to automatically generate rules using the information of data centers.

Experiments show that the evolved rules perform significantly better the human-designed rules in all scenarios. The evolved rules from GPHH are useful for automatically generating rules for various scenarios in data centers. In future work, we will investigate the container allocation in a general architecture where multiple VMs are allocated in PMs.

References

1. D. Bernstein, “Containers and cloud: From LXC to docker to kubernetes,” *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.
2. 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.
3. M. D. Cauwer, D. Mehta, and B. O’Sullivan, “The temporal bin packing problem: An application to workload management in data centres,” in *28th International Conference on Tools with Artificial Intelligence (ICTAI)*, 2016, pp. 157–164.
4. 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.
5. E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, “Hyper-heuristics: a survey of the state of the art,” *Journal of the Operational Research Society, Springer*, vol. 64, no. 12, pp. 1695–1724, 2013.