Energy-Aware Dynamic Resource Allocation in Container-based Clouds via Cooperative Coevolution Genetic Programming

Chen Wang[1], Hui Ma[2], Gang Chen[2], Victoria Huang[1], Yongbo Yu[2], and Kameron Christopher[1]

[1] HPC and Data Science Department, National Institute of Water and Atmospheric Research, New Zealand
{chen.wang, victoria.huang, kameron.christopher}@niwa.co.nz
[2] School of Engineering and Computer Science,Victoria University of Wellington, New Zealand {hui.ma, aaron.chen, yuyong1}@ecs.vuw.ac.nz

Abstract. As a scalable and lightweight infrastructure technology, containers are quickly gaining popularity in cloud data centers. However, dynamic Resource-Allocation in Container-based clouds (RAC) is challenging due to two interdependent allocation sub-problems, allocating dynamic arriving containers to appropriate Virtual Machines (VMs) and allocating VMs to multiple Physical Machines (PMs). Most of existing research works assume homogeneous PMs and rely on simple and manually designed heuristics such as Best Fit and First Fit, which can only capture limited information, affecting their effectiveness of reducing energy consumption in data centers. In this work, we propose a novel hybrid Cooperative Coevolution Genetic Programming (CCGP) hyper-heuristic approach to automatically generate heuristics that are effective in solving the dynamic RAC problem. Different from existing works, our approach hybridizes Best Fit to automatically designed heuristics to coherently solve the two interdependent sub-problems. Moreover, we introduce a new energy model that accurately captures the energy consumption in a more realistic setting than that in the literature, e.g., real-world workload patterns and heterogeneous PMs. The experiment results show that our approach can significantly reduce energy consumption, in comparison to two state-of-the-art methods.

Keywords: container-based clouds · container allocation · energy efficiency · genetic programming · hyper-heuristic.

1 Introduction

Cloud computing has become a pillar of today's software industry by providing on-demand computing resources [4]. A fundamental technology that powers cloud computing is virtualization, e.g., Virtual Machine (VM)-based clouds which enable multiple VMs with different Operating Systems (OS) to share one Physical Machine (PM). However, VM deployment is often considered *heavy-weight* in terms of memory, CPU and start-up time since each VM is essentially a standalone machine with its own OS and a virtual copy of hardware. A recent trend in cloud computing is container-based cloud, which enables multiple applications to be deployed and share resources of the same OS on a VM [4]. Compared with VMs, containers are known for *lightweight* which utilize less resources and can easily scale up.

The widespread use of containers poses new research challenges. One essential issue is the Resource Allocation problem in Container-based clouds (RAC). Different from traditional VM-based cloud which only requires one-level resource allocation (i.e., VM instances are directly allocated to PM instances), RAC involves a two-level decision making process [12]. At the first level, a container needs to be assigned to a VM which involves decisions on VM selection and VM creation when existing VMs may not have enough resources to host the container. The second level allocation is similar to VM-based cloud, which deploys VMs to existing or newly created PMs with varied capacities.

RAC is usually modelled as a vector bin-packing problem [14,15,18,20] which is NP-hard. Thus, existing works usually simplified it by decoupling RAC into two independent single-level optimization without taking consideration of the inter-dependencies between container-VM and VM-PM allocations [10,17,20,26]. Moreover, RAC has been further simplified by considering only homogeneous PMs [20, 22] where the searching space is significantly reduced. As a result, their proposed algorithms may not be suitable for a cloud data center where PMs are equipped with different capacities [25]. Further, existing works [20, 22] often assume that application deployment requests arrive to a data center with a simple workload pattern, e.g., one container arrives at the data center every five minutes. However, the number and the time of containers arriving at a data center are changing dynamically [19].

The goal of this paper is to propose an effective approach for energy-aware dynamic resource allocation in container-based clouds with heterogeneous PMs. To achieve this goal, we first propose a problem formulation on the dynamic RAC with the goal of minimizing the overall energy consumption. Different from existing works [10, 20, 22, 25, 26], our problem formulation adopts a new energy model and jointly considers heterogeneous VMs and PMs and dynamic container arrivals. Meanwhile, a new Hybrid cooperative co-evolution genetic programming (CCGP) approach is proposed to learn allocation rules hybridized with Best Fit to simultaneously address both container-VM and VM-PM allocations. To evaluate the performance of the hybrid approach in real-world settings, extensive experiments have been conducted on real-world workload patterns. Experiment results show that our proposed hybrid approach can significantly outperform widely-used heuristics and a state-of-the-art algorithm CCGP in reducing the energy consumption.

2 Related Work

Resource allocation in clouds has been widely investigated in the literature [1,5,7,8,10,11,17,20,22,23,26]. However, most studies [1,5,7,8,11,23] only focused on one-level resource allocation where either VM-PM [1,5,7,8,14,23] or container-VM allocation [11] is tackled, neglecting the inter-dependencies between VM-PM and container-VM allocations.

Several recent research works [2,3,10,15,17,20,22,25,26] have studied the problem of two-level resource allocation in container-based clouds as a static problem and proposed meta-heuristic approaches to solve the problem. For example, [20] minimized the energy consumption by maximizing the PM utilization to reduce the number of active PMs. A Two-stage Multi-type Particle Swarm Optimization (TMPSO) algorithm was proposed which combines greedy and heuristic optimization. Similarly, [10] minimized the number of active PMs and instantiated VMs as well as maximized resource utilization. They proposed an Ant Colony Optimization based on Best-Fit (ACO-BF) algorithm where enough VMs are pre-created and assigned to PMs using Best-fit and containers are subsequently allocated to VMs using ACO. However, the above mentioned meta-heuristic approaches consider the resource allocation problem as a static problem where all the requests arrive at a data center at the same time. To handle the dynamic resource allocation problem, a CCGP approach is developed in [22] which simultaneously learns allocation rules for both levels.

From the problem perspective, existing works do not fully capture important features of the problem, e.g., the energy consumption is either not directly optimized or is only measured at a certain time point rather than over a time period [10,20,26]. Evidence has been shown in [22] that even with the same temporal energy consumption, the accumulated energy consumption can be significantly different. Moreover, existing works also tend to simplify the problem by considering homogeneous PMs and simple workload patterns [2,20,22], which cannot be applied in a real-world cloud data center with different types of PMs with different capacities [25].

From the solution perspective, RAC is a challenging two-level optimization problem with a large search space. Existing studies [15,18,20] usually simplified the problem as two independent one-level optimization without taking into account their inter-dependencies [10,17,20,26]. Moreover, a majority solutions relied on human-designed heuristics, e.g., greedy [16,20] and best-fit [10,26]. Their performance heavily relies on the workload patterns [21]. Although different searching algorithms, e.g., ACO, GP, and PSO, have been proposed [10,20–22], GP has shown its promise in effectively handling dynamic resource allocation problems [21,22].

3 Problem Formulation

Dynamic RAC is the process of dynamically allocating a set of arriving containers to VMs and subsequently mapping VMs to PMs according to various constraints.

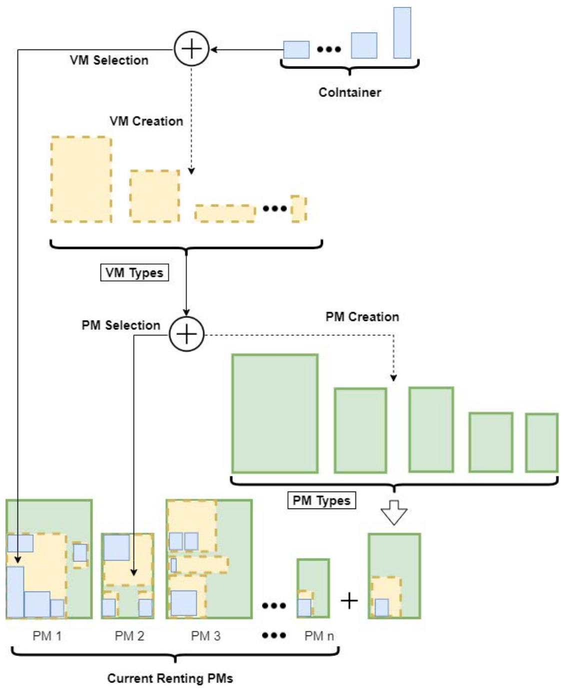


Fig. 1: An overview of dynamic RAC

In the remaining of this paper, we use RAC to refer dynamic RAC. As shown in Fig. 1, RAC follows a decision making process from *VM selection*, *VM creation*, *PM selection*, to *PM creation*. VM selection allocates a newly arrived container to a VM chosen from existing VMs. VM creation can be invoked if no existing VMs are preferred for hosting the container. VM creation creates a VM with a selected VM type, which is used for container allocation. When a new VM is created, it is allocated to an exiting PM via PM selection or a new PM via PM creation.

RAC considers a set of $|\mathcal{C}_t|$ containers $\mathcal{C}_t = \{c_1, \ldots, c_{|\mathcal{C}_t|}\}$ arriving at time t. Each container c_i is defined with three attributes, i.e., the CPU $\zeta_{c_i}^{cpu}$, memory $\zeta_{c_i}^{men}$, and OS $\zeta_{c_i}^{os}$ for running container c_i. In general, $\zeta_{c_i}^{os}$ is selected from a given set of $|\Phi|$ OS types $\Phi = \{\psi_1 \ldots, \psi_{|\Phi|}\}$, i.e., $\zeta_{c_i}^{os} \in \Phi$.

When a new container arrives, it needs to be allocated to a VM instance (or VM in short). A VM is defined with an OS $\Omega_j^{vos} \in \Phi$ and a VM type τ_j selected from a set of $|\Gamma|$ VM types provided by cloud providers where $\Gamma = \{\tau_1, \ldots, \tau_{|\Gamma|}\}$. Each VM type τ_j is associated with its CPU capacity $\Omega_{\tau_j}^{vcpu}$, memory capacity $\Omega_{\tau_j}^{vmen}$, CPU overhead $\pi_{\tau_j}^{vcpu}$ and memory overhead $\pi_{\tau_j}^{vmen}$ for creating a VM.

When a new VM is created, it needs to be allocated to a PM. A PM is associated with a PM type $\top_{p_k}$ from a set of $|\Upsilon|$ PM types, $\Upsilon = \{\top_1, \ldots, \top_{|\Upsilon|}\}$. Each PM type $\top_{p_k}$ is defined with a tuple, i.e., $\top_{p_k} = (\Omega_{p_k}^{pcpu}, \Omega_{p_k}^{pmen}, \pi_{p_k}^{pcpu}, \pi_{p_k}^{pmen})$, capturing its CPU and memory capacities, as well as the CPU and memory overhead for creating a new PM.

To calculate the energy consumption for a given PM p at time t, we define a non-linear energy model Eq. (1) based on [6], which can capture energy consumption more accurately than the linear models [22, 25].

$$E_{p,t} = E_p^{idle} + (E_p^{full} - E_p^{idle})(2\mu_{p,t}^{cpu} - (\mu_{p,t}^{cpu})^{1.4}) \tag{1}$$

where E_p^{idle} and E_p^{full} represent the energy consumption of the PM p per time unit when it is idle and fully loaded, respectively. Given a number of $|L_t|$ VM instances used at time t, $\mu_{p,t}^{cpu}$ is the CPU utilization level at time t, which can be calculated as

$$\mu_{p,t}^{cpu} = \frac{\sum_{l=1}^{|L_t|}(\sum_{j=1}^{|\mathit{\Gamma}|} \pi_{\tau_j}^{vcpu}\mathfrak{a}_{lj} + \sum_{i=1}^{|\mathcal{C}_t|} \zeta_{c_i}^{cpu}\mathfrak{c}_{il})\mathfrak{b}_{lp}}{\sum_{k=1}^{|\mathit{\Upsilon}|}(\pi_{\top_{p_k}}^{pcpu} + \mathit{\Omega}_{p_k}^{pcpu})\mathfrak{x}_{pk}}$$

where $\mathfrak{a}_{lj}$, $\mathfrak{b}_{lp}$, $\mathfrak{c}_{il}$, and $\mathfrak{x}_{pk}$ are binary variables. For example, $\mathfrak{a}_{lj} = 1$ if the l^{th} running VM instance belongs to type τ_j. $\mathfrak{c}_{il} = 1$ if the i^{th} container is allocated to the l^{th} VM. $\mathfrak{b}_{lp} = 1$ if the l^{th} VM instance is allocated to the p^{th} PM instance. $\mathfrak{x}_{pk} = 1$ if the p^{th} created PM is type of $\top_{p_k}$.

RAC aims to find the mapping of containers to VMs (i.e., $\{\mathfrak{a}_{lj}\}$ and $\{\mathfrak{c}_{il}\}$) and VMs to PMs (i.e., $\{\mathfrak{b}_{lp}\}$ and $\{\mathfrak{x}_{pk}\}$) so that the accumulated energy consumption over the allocation period T can be minimized:

$$J = \int_0^T \left(\sum_{p=1}^{|P_t|} E_{p,t} \right) \mathrm{d}t \tag{2}$$

where $|P_t|$ a step function that gives the number of PM used at any time t. In line with Eq. (2), RAC is subject to the following constraints:

– A container c_i can only be allocated to one VM instance, i.e., $\sum_{l=1}^{|L_t|} \mathfrak{c}_{il} = 1$.
– A VM can only be allocated to one PM instance, i.e., $\sum_{p=1}^{|P_t|} \mathfrak{b}_{lp} = 1$.
– Each VM and PM instance must have a specific type, i.e., $\sum_{j=1}^{|\mathit{\Gamma}|} \mathfrak{a}_{lj} = 1$ and $\sum_{k=1}^{|\mathit{\Upsilon}|} \mathfrak{x}_{pk} = 1$
– OS compatibility: A container c can only be allocated to a VM instance v if they have the same OS, i.e., $\zeta_{c_i}^{os}\mathfrak{c}_{il} = \mathit{\Omega}_l^{vos}$.
– VM capacity constraint: For a VM instance, the total CPU and memory requirements of all allocated containers must satisfy its capacity, i.e., $\sum_i \zeta_{c_i}^{cpu}\mathfrak{c}_{il} \le \sum_{j=1}^{|\mathit{\Gamma}|} \mathit{\Omega}_{\tau_j}^{vcpu}\mathfrak{a}_{lj}$ and $\sum_i \zeta_{c_i}^{men}\mathfrak{c}_{il} \le \sum_{j=1}^{|\mathit{\Gamma}|} \mathit{\Omega}_{\tau_j}^{vmen}\mathfrak{a}_{lj}$.
– PM capacity constraint: For a PM instance, the total CPU and memory requirements of all allocated VMs must not exceed its capacity, i.e., $\sum_{l=1}^{|L_t|}(\sum_{j=1}^{|\mathit{\Gamma}|} \mathit{\Omega}_{\tau_j}^{vcpu}\mathfrak{a}_{lj})\mathfrak{b}_{lp} \le \sum_{k=1}^{|\mathit{\Upsilon}|} \mathit{\Omega}_{p_k}^{pcpu}\mathfrak{x}_{pk}$ and $\sum_{l=1}^{|L_t|}(\sum_{j=1}^{|\mathit{\Gamma}|} \mathit{\Omega}_{\tau_j}^{vmem}\mathfrak{a}_{lj})\mathfrak{b}_{lp} \le \mathit{\Omega}_{p_k}^{pmem}\mathfrak{x}_{pk}$.

We summarize the symbols used in this section in Table 1 to facilitate reading.

Table 1: Notations for problem model

Symbol	Meaning
$\mathcal{C}_t$	Set of containers arriving at time t.
c_i	A container of index i, and $c_i \in \mathcal{C}_t$
$\zeta_{c_i}^{cpu}$, $\zeta_{c_i}^{men}$	CPU and memory requirements for container c_i
$\zeta_{c_i}^{os}$	A OS type of container c_i
τ_j	A VM type of index j
$\mathit{\Gamma}$	Set of different VM types provided by cloud providers, and $\tau_j \in \mathit{\Gamma}$
$\mathit{\Omega}_j^{vos}$	A OS type of VM τ_j
$\mathit{\Phi}$	Set of different OS types, and $\zeta_{c_i}^{os} \in \mathit{\Phi}$, and $\mathit{\Omega}_j^{vos} \in \mathit{\Phi}$
$\mathit{\Omega}_{\tau_j}^{vcpu}$, $\mathit{\Omega}_{\tau_j}^{vmen}$	CPU capacity and memory capacity of a VM type τ_j.
$\pi_{\tau_j}^{vcpu}$, $\pi_{\tau_j}^{vmen}$	CPU overhead and memory overhead of a VM type τ_j.
$\top_{p_k}$	A PM type of index k
$\mathit{\Upsilon}$	Set of different PM types provided by cloud providers, and $\top_{p_k} \in \mathit{\Upsilon}$
$\mathit{\Omega}_{p_k}^{pcpu}$, $\mathit{\Omega}_{p_k}^{pmen}$	CPU capacity and memory capacity of a PM type $\top_{p_k}$
$\pi_{p_k}^{pcpu}$, $\pi_{p_k}^{pmen}$	CPU and memory overhead of a PM type $\top_{p_k}$
$E_{p,t}$	Energy consumption of a given PM p at time t
E_p^{idle}, E_p^{full}	Energy consumption of a given PM p per time unit when it is idle and fully loaded
$\lvert L_t \rvert$	Number of VM instances used at time t
$\mu_{p,t}^{cpu}$	CPU utilization level of a given PM p at time t
$\mathfrak{a}_{lj}$	$\mathfrak{a}_{lj} = 1$ if the l^{th} running VM instance belongs to type τ_j, else $\mathfrak{a}_{lj} = 0$
$\mathfrak{b}_{lp}$	$\mathfrak{b}_{lp} = 1$ if the l^{th} VM instance is allocated to the p^{th} PM instance, else $\mathfrak{b}_{lp} = 0$
$\mathfrak{c}_{il}$	$\mathfrak{c}_{il} = 1$ if the i^{th} container is allocated to the l^{th} VM, else $\mathfrak{c}_{il} = 0$
$\mathfrak{x}_{pk}$	$\mathfrak{x}_{pk} = 1$ if the p^{th} created PM is type of $\top_{p_k}$, else $\mathfrak{x}_{pk} = 0$
T	Allocation period
J	Accumulated energy consumption over the allocation period T
$\lvert P_t \rvert$	A step function that gives the number of PM used at any time t

4 A Hybrid Approach for Dynamic RAC

As presented in Sec. 3, the allocation decision includes four components, VM selection, VM creation, PM selection, and PM creation. The proposed hybrid-heuristic approach, aims to automatically generate two rules as solutions for these four decision components. The two rules can be used for four decisions because creating new VM/PM is treated as a special case of VM/PM selection. In particular, one rule r^v is trained for VM selection and creation, and the other hybrid rule, i.e., r^p with the Best Fit, is for PM selection and creation, respectively. A rule is used as a priority function to indicate the preference of selecting a given candidate allocation decision.

Our hybrid-heuristic approach follows the CCGP framework from [22] (shown in Fig. 2) by first randomly initializing two sub-populations. Each sub-population contains candidates for one rule in the form of a tree (see details in Sec. 4.1). The rules are evolved cooperatively by the genetic operators, e.g. crossover and mutation. Based on historical data, a set of training instances are processed and used for fitness evaluation. To evaluate a rule, a pair of rule is formed from each population. The performance of each pair of rule is measured by its fitness value which will be discussed in Sec. 4.2. Based on the fitness values, the best performing rules within each sub-population are selected to generate the next generation. This process repeats until a predefined stopping condition, e.g., maximum generations, is met.

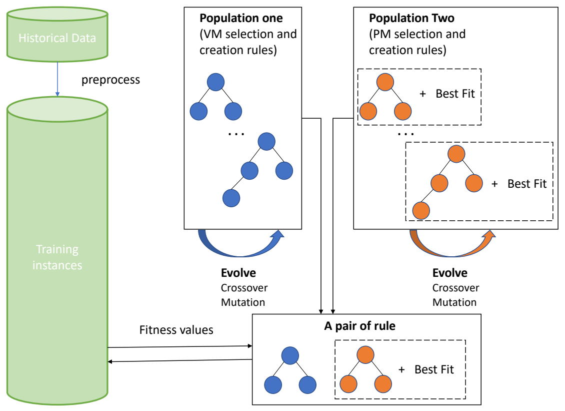


Fig. 2: An overview of the training process of CCGP

4.1 Representation, Terminal and Function Set

We adopt a tree representation for two allocation rules due to its promise in interpretability, expressiveness, and flexibility [22]. To capture features of dynamic arriving container as well as properties of VMs and PMs, a set of terminal nodes and functional nodes are shown in Table 2. As shown in Fig. 3, a simple example of an allocation rule is constructed by a set of terminal nodes, i.e., LeftVMMem, ConCPU and LeftVMCPU, and function/intermediate nodes, i.e., + and /.

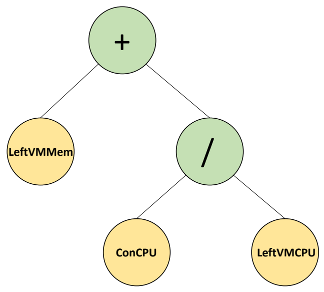


Fig. 3: A simple example of allocation rule for VM selection and creation

Table 2: Terminal and function sets used in our tree-based rules

Terminal Set (VM Selection and Creation Features)	**Description**
ConMem	Memory requirement of a container
ConCPU	CPU requirement of a container
LeftVMMem	Remaining memory of a VM
LeftVMCPU	Remaining CPU of a VM
VMMemOverhead	Memory overhead of a VM
VMCPUOverhead	CPU overhead of a VM
Terminal Set (PM Selection and Creation Features)	**Description**
VMMem	Memory capacity of a VM
VMCPU	CPU capacity of a VM
LeftPMMem	Remaining Memory of a PM
LeftPMCPU	Remaining CPU of a PM
PMMem	Memory capacity of a PM
PMCPU	CPU capacity of a PM
PMCore	Core requirement of a PM
Function Set (PM/VM Selection and Creation Features)	**Description**
Add, Substract, Multiply	Arithmetic operations (+, -, ×)
Protected Division	Protected division (/) that returns 1 if the denominator is 0.

4.2 Fitness Evaluation

Algorithm 1 shows the process of fitness evaluation on a pair of rules, i.e, VM creation and selection rule r^v, PM creation and selection rule r^p. For every training instance, we start with a data center initialization process, where a group of containers are randomly allocated to suitable VMs, and these VMs are randomly allocated to suitable PMs. Note that the same randomness is used for the data center initialization process in every generation through the use of the same random seed. By doing this, it can ensure the fitness values of every pair of rules in one generation are comparable. Afterwards, we allocated containers one by one based on their arrival time. Note that multiple containers can arrive at the same time t. The arrival time t is also utilized for triggering the process of updating total energy consumption using Eq. (1). For every container, we first filter out running VMs with different OSs from its OS. The remaining running VMs combined with new VMs (one for each VM type) are merged as a candidate VM list, on which VM selection or creation operation is performed based on the calculated priority value using r^v. Whenever a new VM is created, a PM type is selected and PM of the selected type is created using r^p, where the trained heuristic part in r^p is used for the selection of a running PM while Best Fit part in r^p is used for the selection of a new PM instance (one for each PM type) for a PM creation. After allocation all containers in the training instance, we return the total energy consumption J as the fitness value of the pair of rules.

Algorithm 1: Fitness evaluation

```
Input: Containers ($c_t$), a pair of rules ($r^v$ and $r^p$)
Output: Accumulated energy consumption ($J$)
for each training instance do
    $J = 0$;
    Initialize data center;
    for each $C_t$ over a period of time do
        for each $c_t$ in $C_t$ do
            Update $J$;
            Filter out the $VMs$ that require different OS;
            $vm$ = VMSelectionCreation($c_t$, $r^v$);
            allocateVM(c,$vm$);
            if $vm$ is Newly Created then
                addVM($vm$, the running VM list);
                $pm$ = PMSelectionCreation($vm$, $r^p$);
                if $pm$ is Newly Created then
                    addPM($pm$, the running PM list);
                end
                allocatePM($vm$, $pm$);
            end
        end
    end
end
Return energy consumption;
```

5 Experiment

To evaluate the proposed approach, we conducted an empirical experiment using real-world datasets in this section. Notably, we compare our approach with a popular baseline used in the recent works [2, 9, 13, 22], i.e., Sub&JustFit/FF rule [15], and a GPHH approach [22] that is recently proposed to solve a similar RAC problem. The problem investigated in this paper is more challenging because it considers real-world workload patterns (instead of using synthetic workload studied in [22]) and heterogeneous PM types. We modify Sub&JustFit/FF rule and GPHH approach [22] to cope with the selection on heterogeneous PM types. In particular, a Best-Fit (BF) rule is added to the Sub&JustFit/FF rule for new PM creation. Meanwhile, GPHH approach [22] is adapted to heterogeneous PM types by considering different PM type options when training the second level allocation rules. In this section, we will use Sub&JustFit/FF&BF and Evo/Evo to represent rules generated by these two modified methods, and use Evo/Hybrid-Evo to represent our hybrid-heuristic approach.

5.1 New Dataset

We designed six scenarios, as summarized in Table 3, for our experimental analysis. In particular, we consider distinct numbers of OS types in the experiment.

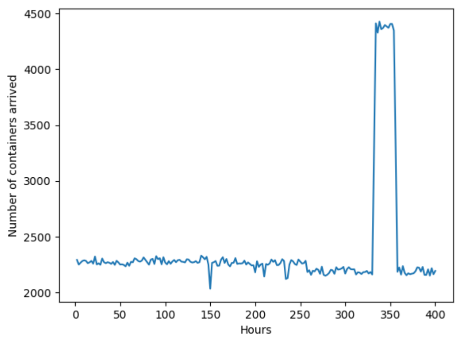


Fig. 4: An example of real-world workload pattern

The proportions of the OS types are determined using the recent market shares of OS reported in [24]. In Table 4, we simulate three OS scenarios where the number of OS increases from 3 to 5. Furthermore, as shown in Tables 5 and 6, we uses popular VM types and PM types settings from Amazon EC2 VM instances and PM reported in [22] as examples in this paper. Note that, we convert CPU Cores of VM to MHz using $VM_{Mhz} = PM_{MHz}/PM_{Core} * VM_{Core}$. Using this formula, we actually create 45 different VM types in Mhz.

To create workload patterns that consider dynamic number and time arrival for containers, we use Bitbrains data [19], i.e., time-serial CPU and memory usage records of applications, to generate 400 instances for each scenario. Each instance is made of two hour records of containers arrived at the data center. This can ensure the number of containers to be large enough for an algorithm to reach a stable status. Note that, we exclude containers that cannot be fulfilled with the largest VM types in Table 5, and the remaining containers are augmented with OS requirements based on the OS distribution in Table 4. We use 100 instances for data center initialization, 200 instances for rule training, and 100 instances for rule testing.

Table 3: Scenarios

Scenarios	Number of OSs	VM types	Container workload patterns
S-1	3	Amazon EC2 VM types	BitBrains Pattern 1
S-2	3	Amazon EC2 VM types	BitBrains Pattern 2
S-3	4	Amazon EC2 VM types	BitBrains Pattern 1
S-4	4	Amazon EC2 VM types	BitBrains Pattern 2
S-5	5	Amazon EC2 VM types	BitBrains Pattern 1
S-6	5	Amazon EC2 VM types	BitBrains Pattern 2

Table 4: OS Distribution

Number of OS	OS distribution (%)
3	50, 30, and 20
4	62.5, 17.5, 15, and 5
5	17.9, 45.4, 23.6, 10.5, and 2.6

Table 5: VM types

VM	CPU ($Core$)	Memory (GB)
c5.large	2	4
c5.xlarge	4	8
c5.2xlarge	8	16
c5.4xlarge	16	32
c5.9xlarge	36	72
c5a.8xlarge	32	64
c5a.12xlarge	48	96
c5a.16xlarge	64	128

Note that, different from the dataset used in the very recent work [22], which assumes containers arrive one by one uniformly and a fixed number of contains (i.e., 2500 in [22]) in every instance, our dataset does not make such unrealistic assumptions and consider a varied number of containers can come at any time. See an example of workload pattern in Fig. 4. The new benchmark datasets and the implementation of all approaches in this paper have been made freely available online. [3]

5.2 Parameter Settings

We strictly follow the setting reported in the recent work [22] except the number of generations. The number of generation increases to 200 for the training on challenging new dataset. Other settings remain the same, e.g., number of population is two, each population size is 512, elite size is 3, tournament selection size is 7, crossover rate is 0.8, mutation rate is 0.1, reproduction rate is 0.1, crossover max depth and mutate max depth are 7.

5.3 Performance Comparison

Table 7 shows the comparison of accumulated energy consumption among the three competing rules, i.e., Sub&JustFit/FF&BF, Evo/Evo, and Evo/Hybrid-Evo. Our results show that the Evo/Hybrid-Evo rule learned in all scenarios

[3] Online resource allocation benchmarks for container-based clouds and the source code of all discussed approaches are available from https://github.com/chenwangnida/RAC.

Table 6: PM types

PM (CPU)	CPU (MHz)	Memory (MB)	P_{Idle} (Wh)	P_{Max} (Wh)	Cores
E5-2630	27600	128000	99.6	325	12
E5430	22640	16000	166	265	8
E5507	20264	8000	67	218	8
E5-2620	24000	3200	70	300	12
E5645	28800	16000	63.1	200	12
E5-2650	32000	24000	52.9	215	12
E5-2651	21600	32000	57.5	178	16
E5-2670	41600	24000	54.1	243	12
E5540	10000	72000	151	312	16
E5560	22400	128000	133	288	4
E5-2665	24000	256000	117	314	8
XS5650	31992	64000	80.1	258	12

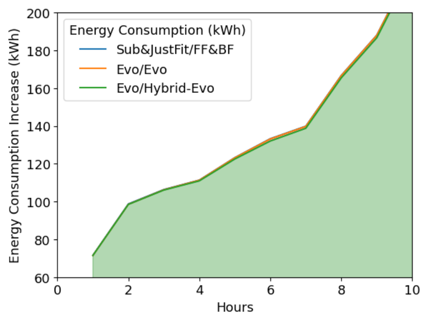
achieve the lowest accumulated energy consumption. Note that the performance of Evo/Evo does not match the reported results in [22], where Evo/Evo outperforms Sub&JustFit/FF&BF rule. Evo/Evo does not work well under real-world workload pattern settings. This might due to that the big searching space using our dataset.This finding also indicates that Evo/Evo may not be suitable for a cloud data center where different PMs types are considered.

Table 7: The best and mean accumulated energy consumption of three competing rules tested over 100 hours (Note: the lower the value the better)

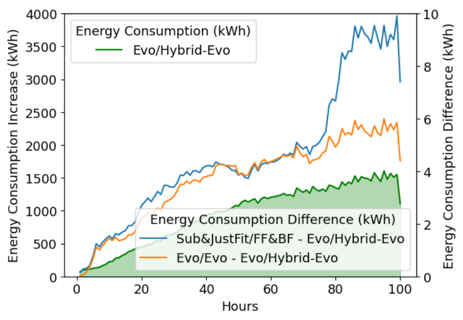
Scenarios	Sub&JustFit/FF&BF	Evo/Evo (best value)	Evo/Hybrid-Evo (best value)	Evo/Evo (mean value)	Evo/Hybrid-Evo (mean value)
S-1	95959	95868	**95496**	125994 ± 22734	**95550 ± 43**
S-2	85296	85128	**84850**	114775 ± 27378	**85049 ± 138**
S-3	95503	95399	**95061**	121537 ± 21237	**95165 ± 137**
S-4	85615	85486	**85181**	116209 ± 24509	**85366 ± 98**
S-5	95442	95289	**94961**	120951 ± 19164	**95018 ± 65**
S-6	85316	85185	**84944**	116336 ± 23293	**85069 ± 60**

To further analyze the differences in energy consumption during the allocation process, we use the deterministic sub&JustFit/FF&BF rule and the two best evolved rules found from Evo/Evo rule, and Evo/Hybrid-Evo in scenario S-1 in Table 3 as an example.

Fig. 5 (a) shows that all the rules have comparable energy consumption in the first 10 hours. This is because new PMs are barely created, and containers are mainly allocated to the existing running PMs. As time goes by, new PMs must be created to satisfy the resource needs of newly arriving containers. In particular, Evo/Hybrid-Evo rule is expected to create new PMs that result in more opportunities in allocating more containers than the sub&JustFit/FF&BF rule, Evo/Evo rules in the future. Fig. 5 (b) shows the average increase in energy consumption over the first 100 hours using Evo/Hybrid-Evo rule and the aver-



(a) The first 10 hours



(b) Total 100 hours

Fig. 5: Energy consumption during the allocation process for Scenario S-1

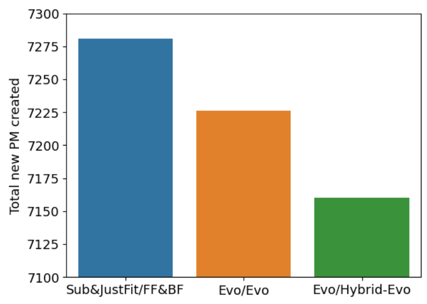


Fig. 6: Total new PMs created

age increase differences between Evo/Hybrid-Evo and other two rules. We can see that the average increase in energy consumption using Evo/Hybrid-Evo rule is much slower than the sub&JustFit/FF&BF and Evo/Evo rules. Meanwhile, sub&JustFit/FF&BF rule is the worst performer in reducing energy consumption.

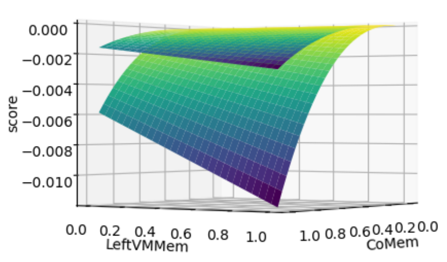
Fig. 6. shows the total number of new PMs created in 100 hours using the three rules. The Evo/Hybrid-Evo rule creates the significantly less number of PMs than the other two rules after 100 hours. This observation indicates that Evo/Hybrid-Evo rule creates less new PMs in the long run, which could lead to better energy consumption.

5.4 Further Analysis

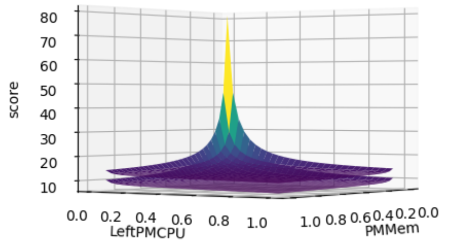
In this section, we use a simple example to explain the best pair of rule learned from Evo/Hybrid-Evo in 30 runs. The best pair of rule includes a VM selection and creation rule r^v and a PM selection rule, i.e., a trained heuristic part in r^p. These two rules are presented in Eq. (3) and Eq. (4), respectively.

$$score_{r^v} = (((((\text{VMMemOverhead} - \text{CoMem}) - \text{LeftVMMem}) * \text{VMCPUOverhead}) * \text{VMCPUOverhead}) * \text{CoMem}) * \text{CoMem} \tag{3}$$

$$score_{r^p} = \text{PMCore} - ((\text{VMMem}/((\text{VMMem}/\text{PMMem})/\text{LeftPMCPU})) - ((\text{VMMem}/\text{PMMem})/\text{LeftPMCPU})) \tag{4}$$



(a) Upper plane τ_1 and lower plane τ_2



(b) Upper plane PM E5540 and Lower plane PM E5-2665

Fig. 7: The 3D contour plots of the best pair of rules learned from Evo/Hybrid-Evo in 30 runs

Let's consider an example of a cloud provider, who provides two types of 8-core VMs (e.g., τ_1 and τ_2) and two types of PMs (e.g., 8-core PM E5-2665 and 16-core PM E5540). For VM type τ_1, its CPU capacity $\Omega^{vcpu}_{\tau_1}$ is equal to 18400 and its memory capacity $\Omega^{vmen}_{\tau_1}$ is equal to 21000. For VM type τ_1, its CPU capacity $\Omega^{vcpu}_{\tau_2}$ is equal to 5000, and memory capacity $\Omega^{vmen}_{\tau_2}$ is equal to 16000. Meanwhile, the overhead on CPU and Memory are 10% of CPU capacity and 200 MB, respectively.

Taking the above numbers into Eq. (3), we can plot $score_{r^v}$, LeftVMMem, and ConMem in a 3D contour figure, shown in Fig 7 (a) over two planes. In Fig 7 (a), the upper plane and lower plane are corresponds to VM types τ_1 and τ_2, respectively. This figure indicates that for allocating any incoming container to VMs, VM instance selection or creation is determined by the priority score $score_{r^v}$. Such score is determined not only by VM types, but also by $ConMem$ and $LeftVMMem$. Meanwhile, this score is more sensitive to $ConMem$ than $LeftVMMem$. Moreover, τ_1 is often preferable, compared to τ_2.

In Fig. 7 (b), we assume a VM of type τ_1 is created, and it waits to be allocated to two running PMs instances (e.g., 8-core PM E5-2665 and 16-core PM E5540). Based on this assumption and the PM selection rule in r^p, i.e., Eq. (4), we can plot another 3D contour figure for explaining decisions on PM selection, which is determined by $LeftPMCPU$, $PMMem$, and two planes in Fig. 7 (b). The upper plane and lower plane are corresponds to PM types E5540 and E5-2665, respectively. Similar to Fig. 7 (a), this plot indicates that how the PM selection is affected by $LeftPMCPU$, $PMMem$, and PM types.

6 Conclusions

In this paper we proposed a novel hybrid CCGP hybrid-heuristic approach, named Evo/Hybrid-Evo, to solve a new challenging dynamic RAC problem. We formulated a formal energy consumption model for this problem that that considers heterogeneous PM types. We also evaluated its performance against other promising approaches in a more realistic experimental settings. Our experiment results show that the Evo/Hybrid-Evo achieves significantly lower accumulated energy consumption than the human-designed rule (sub&JustFit/FF&BF rule,) and a state-of-the-art approach (Evo/Evo).

References

1. Abohamama, A.S., Hamouda, E.: A hybrid energy–aware virtual machine placement algorithm for cloud environments. Expert Systems with Applications **150**, 113306 (2020)
2. Akindele, T., Tan, B., Mei, Y., Ma, H.: Hybrid grouping genetic algorithm for large-scale two-level resource allocation of containers in the cloud. In: Australasian Joint Conference on Artificial Intelligence. pp. 519–530. Springer (2022)
3. Al-Moalmi, A., Luo, J., Salah, A., Li, K., Yin, L.: A whale optimization system for energy-efficient container placement in data centers. Expert Systems with Applications **164**, 113719 (2021)
4. Bhardwaj, A., Krishna, C.R.: Virtualization in cloud computing: Moving from hypervisor to containerization—a survey. Arabian Journal for Science and Engineering **46**(9), 8585–8601 (2021)
5. Bhattacherjee, S., Das, R., Khatua, S., Roy, S.: Energy-efficient migration techniques for cloud environment: a step toward green computing. The Journal of Supercomputing **76**(7), 5192–5220 (2020)
6. Dayarathna, M., Wen, Y., Fan, R.: Data center energy consumption modeling: A survey. IEEE Communications Surveys & Tutorials **18**, 732–794 (2015)
7. Ding, W., Luo, F., Han, L., Gu, C., Lu, H., Fuentes, J.: Adaptive virtual machine consolidation framework based on performance-to-power ratio in cloud data centers. Future Generation Computer Systems **111**, 254–270 (2020)
8. Gharehpasha, S., Masdari, M., Jafarian, A.: Virtual machine placement in cloud data centers using a hybrid multi-verse optimization algorithm. Artificial Intelligence Review **54**(3), 2221–2257 (2021)
9. Guo, M., Guan, Q., Chen, W., Ji, F., Peng, Z.: Delay-optimal scheduling of vms in a Queueing cloud computing system with heterogeneous workloads. IEEE Transactions on Services Computing **15**(1), 110–123 (2022)

10. Hussein, M.K., Mousa, M.H., Alqarni, M.A.: A placement architecture for a container as a service (CaaS) in a cloud environment. Journal of Cloud Computing **8**(1), 1–15 (2019)
11. Kaewkasi, C., Chuenmuneewong, K.: Improvement of container scheduling for docker using ant colony optimization. In: 2017 9th international conference on knowledge and smart technology (KST). pp. 254–259. IEEE (2017)
12. Kanso, A., Youssef, A.: Serverless: beyond the cloud. In: Proceedings of the 2nd International Workshop on Serverless Computing. pp. 6–10 (2017)
13. Li, F., Tan, W.J., Cai, W.: A wholistic optimization of containerized workflow scheduling and deployment in the cloud–edge environment. Simulation Modelling Practice and Theory **118**, 102521 (2022)
14. Long, S., Wen, W., Li, Z., Li, K., Yu, R., Zhu, J.: A global cost-aware container scheduling strategy in cloud data centers. IEEE Transactions on Parallel and Distributed Systems **33**(11), 2752–2766 (2021)
15. Mann, Z.Á.: Interplay of virtual machine selection and virtual machine placement. In: European Conference on Service-Oriented and Cloud Computing. pp. 137–151. Springer (2016)
16. Nardelli, M., Hochreiner, C., Schulte, S.: Elastic provisioning of virtual machines for container deployment. In: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion. pp. 5–10 (2017)
17. Piraghaj, S.F., Dastjerdi, A.V., Calheiros, R.N., Buyya, R.: Efficient virtual machine sizing for hosting containers as a service (services 2015). In: 2015 IEEE World Congress on Services. pp. 31–38. IEEE (2015)
18. Piraghaj, S.F., Dastjerdi, A.V., Calheiros, R.N., Buyya, R.: A framework and algorithm for energy efficient container consolidation in cloud data centers. In: 2015 IEEE International Conference on Data Science and Data Intensive Systems. pp. 368–375. IEEE (2015)
19. Shen, S., Van Beek, V., Iosup, A.: Statistical characterization of business-critical workloads hosted in cloud datacenters. In: 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. pp. 465–474. IEEE (2015)
20. Shi, T., Ma, H., Chen, G.: Energy-aware container consolidation based on PSO in cloud data centers. In: IEEE CEC 2018. pp. 1–8
21. Tan, B., Ma, H., Mei, Y.: A genetic programming hyper-heuristic approach for online resource allocation in container-based clouds. In: Australasian Joint Conference on Artificial Intelligence. pp. 146–152. Springer (2018)
22. Tan, B., Ma, H., Mei, Y., Zhang, M.: A cooperative coevolution genetic programming hyper-heuristics approach for on-line resource allocation in container-based clouds. IEEE Transactions on Cloud Computing **10**, 1500–1514 (2022)
23. Tarahomi, M., Izadi, M., Ghobaei-Arani, M.: An efficient power-aware vm allocation mechanism in cloud data centers: a micro genetic-based approach. Cluster Computing **24**(2), 919–934 (2021)
24. Taylor, P.: Global market share held by operating systems for desktop PCs, from January 2013 to December 2021. Tech. rep. (2022), https://www.statista.com/statistics/218089/global-market-share-of-windows-7
25. Zhang, C., Wang, Y., Wu, H., Guo, H.: An energy-aware host resource management framework for two-tier virtualized cloud data centers. IEEE Access **9**, 3526–3544 (2020)
26. Zhang, R., Zhong, A.m., Dong, B., Tian, F., Li, R., et al.: Container-vm-pm architecture: A novel architecture for docker container placement. In: International Conference on Cloud Computing. pp. 128–140. Springer (2018)