Proceedings of the 42nd Chinese Control Conference
July 24-26, 2023, Tianjin, China

Generating Heuristic Policies from Optimization in Large Scale Cloud Computing VM scheduling

Yuexian Zhang[1], Jianchen Hu[1], Xunhang Sun[1], Qiaozhu Zhai[1], Lei Zhu[2], Li Su[2], Wenli Zhou[2], Fangzhu Ming[1], Xiaoyu Cao[1], and Feng Gao[1]

1. School of Automation Science and Engineering, Xi'an Jiaotong University, Xi'an, Shaanxi 710049, China

2. Huawei Cloud Huawei Technologies Co., Ltd. Xi'an, Shaanxi 710049, China

Abstract: The VM placement problem has emerged as a critical challenge in cloud resource scheduling. This type of problem, often formulated as a vector bin packing problem, is known to be NP-hard. For practical large-scale problems, the optimization-based algorithm fails to promptly accommodate on-demand user requests, while the heuristic algorithms face the scalability issues. To tackle the online VM placement problem, this paper shows a VM placement model considering NUMA architecture and presents an algorithm that converts optimal fine-grained solutions into coarse-grained placement policies so that the on-line implementation is simply the heuristic placement policies. The placement policies, which are generated from the offline optimal solutions of the past few time-steps, are refreshed (time- or event-triggered) every few steps. Our experiments demonstrate that the algorithm we proposed can balance the quality of the solution with execution time compared to BestFit and FirstFit in large scale cloud computing backgrounds.

Key Words: VM placement, Cloud computing, Multi-NUMA, Optimization, Heuristic

1 Introduction

Cloud computing technology has seen remarkable advancements in recent years. Amazon EC2, Google Cloud, Microsoft Azure, Huawei ECS and other large public cloud providers have promoted the development of elastic cloud computing architecture, whose ”pay-as-you-go” business model has met the high-performance computing needs of many customers. With the elastic cloud architecture, customers can access on-demand virtual machine(VM) resource configurations from service providers, such as Huawei Cloud's c6s.large.2 with 2U vCPU and 4GB memory, to run their applications anytime[1]. The cloud provider maps virtual resources to physical resources on the datacenter's hosts through a process called VM placement, which is initiated upon receipt of a customer's request.

The VM placement problem is a key challenge in cloud resource scheduling. The dynamic nature of the ”pay-as-you-go” business model, with customers having the flexibility to create and delete VMs at any time, makes it difficult to predict the future demand. Fault tolerance is crucial for prediction-based VM placement algorithms to account for inaccurate predictions. Additionally, cloud providers must prioritize user experience in the on-demand cloud market and make quick decisions on VM placement to promptly respond to customers' requests. Improving the speed and efficiency of cloud resource scheduling is two priorities for the cloud providers. However, the growing demand for data storage and expanding datacenters have resulted in a larger physical machine(PM) resource pool, causing previously effective algorithms to struggle with satisfying in online scheduling demands.

Cloud providers face the challenge of maximizing the utilization of physical resources in the resource pool through effective algorithms. An inefficient VM placement policy can influence the capacity of the datacenters. Currently, there are several popular algorithms that address this issue.

The optimization-based algorithms often formulate the VM placement problem as a vector bin packing problem, which is a type of integer programming problem. In [2], a stochastic integer program algorithm has been proposed to host VMs in a multiple cloud provider environment under future demand and price uncertainty. In [3], the authors formulate the problem as a Min-Max optimization based on binary integer programming and perform some pre-processing before solving the optimization problem. These optimization-based algorithms can provide efficient resource allocation and accommodate new constraints, making them more flexible than other methods. However, as an NP-hard problem, this model presents difficulties in finding an optimal solution for large-scale problems within a short time period, thus limiting its application in real-time. The execution time also increases with additional constraints, making the algorithm less practical for online use, though it performs well in offline scenarios.

Many meta-heuristic algorithms have also been used to deal with the VM placement problems, such as simulated annealing algorithm[4], genetic algorithm[5], colony optimization[6], and biogeography-based optimization[7]. Another alternative approach is to use reinforcement learning algorithm. In [8], an algorithm called SchedRL with a delta reward scheme and an episodic guided sampling strategy has been proposed. However, these algorithms typically require significant computational resources and are prone to becoming trapped in local optima, leading to less reliable results.

As a result, most online scheduling algorithms are heuristics, such as BestFit and FirstFit, which are popular for their speed and user experience[9]. These greedy algorithms sort PMs in the resource pool according to specific criteria and place VMs starting from the first PM in the sorted list. However, these approaches do not consider batches of user requests and only make placement decisions individually. On

This work was supported by the National Natural Science Foundation of China under Grant 62103319, and by the National Key Research and Development Program of China under Grant 2022YFA1004600, and also by a grant from Huawei Cloud.

the other hand, as the problem size grows, there are more and more constraints to consider for VM placement problem[10]. For example, the updating of PM architecture introduces NUMA constraint and the goal of placing VM groups that communicate frequently on the same PM introduces affinity constraint. It can be expected that as the heterogeneity of resources and applications increases, users or administrators of the system will require more complex constraints. However, the poor scalability of the heuristic algorithm makes it difficult to consider these constraints.

The analysis demonstrates that the optimization-based methods are better suited for offline scheduling, while the heuristic-based methods are better for online scheduling. This study aims to integrate the advantages of both methods by using optimization-based methods to generate heuristic placement policies. We model the VM resource allocation process, then use the model's computational results as the optimal allocation solution for the current resource pool and forecasted user requests. Heuristic policies are generated through clustering to guide the online scheduler in making fast VM placement decisions. In real-world datacenters, simple and feasible scheduling policies are preferred due to the complexity of scaling. This study explores how to generate a practical, coarse-grained heuristic policy from time-consuming exact algorithms for real-world cloud resource scheduling, offering inspiration for other scheduling areas.

The rest of the paper is organized as follows: Section 2 outlines the basic concept and framework of the algorithm process. Section 3 details the optimization model and its fine-grained optimal solutions. Section 4 provides a generic method for transforming the fine-grained solutions into coarse-grained heuristic policies and applying them to VM placement. The results are presented in Section 5, and conclusions and future work are discussed in Section 6.

2 Framework of Our Algorithm

2.1 NUMA-aware VM Placement

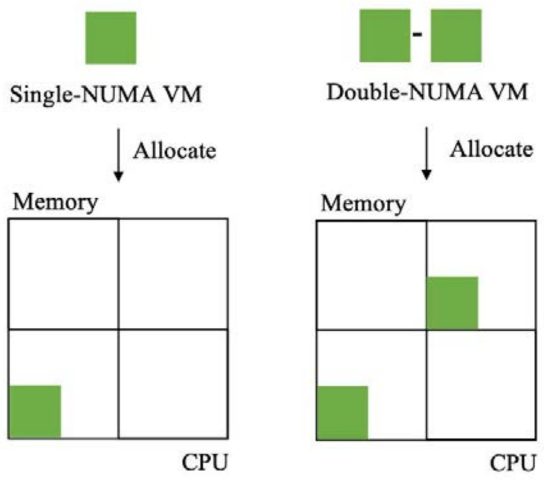


Fig. 1: An illustration of the NUMA-aware VM placement

As cloud technology advances, multicore systems have become the cornerstone of cloud computing infrastructures. Cloud providers opt for Non-Uniform Memory Access (NUMA) architecture, which is the latest architecture in new multicore processors, to offer large VMs. However, lacking NUMA awareness can result in poor VM placement performance and practical incompatibilities. It's crucial to develop an approach that incorporates NUMA awareness into the VM placement model to enhance the efficiency of the placement process.

The NUMA system features multicore processors and multiple memory nodes in a single host[11], but for simplicity, this article only considers a host with two NUMA nodes. As depicted in Fig. 1, VMs and PMs can be represented as two-dimensional quadrilaterals for both CPU and memory resources. The VM is depicted as a green square with its length indicating the CPU resource and its width indicating the memory resource. The black square with no fill represents a PM with its length and width indicating its CPU and memory capacity, respectively. Small-sized VMs are typically deployed with single-NUMA architecture and can be assigned to either NUMA node 1 or 2 in the PM. For large-sized VMs, the resources in a single NUMA node may not be sufficient, so they are usually deployed with double-NUMA architecture, which is divided into two parts and assigned to both NUMA nodes in the PM. The introduction of NUMA architecture makes the traditional bin packing problem model inapplicable and creates a new combinatorial optimization problem. In Section 3, this problem is re-modeled as a multiple-choice vector bin packing problem (MVBPP) and more specific information about NUMA architecture's model is illustrated in this section.

2.2 Value-aware Objective Funtion

The optimization objective of the VM placement problem greatly impacts its solution. A widely used objective is to minimize the number of active PMs by allocating VMs to as few PMs as possible, allowing the remaining PMs to be shut down or hibernated to reduce power consumption. Some researchers utilize VM migration techniques to achieve this goal[12], but due to the limitations of live migration, it cannot be the primary scheduling strategy in real-world datacenters. Hence, only initial VM placement is considered in this study. For datacenters, cloud service providers prioritize resource pool utilization over power consumption, as the cost of electricity is relatively minor compared to the cost of physical resources. To meet user demand efficiently, cloud providers may choose to keep all PMs on. Thus, optimizing the utilization of the resource pool to maximize its benefits is a suitable objective.

Therefore, we aimed to find a way to measure the future serviceability of PM resources. This paper considers a cloud computing system with a predetermined set of VM types identified by distinct resource combinations, such as CPU and memory, sold as commodities by the cloud provider. For example, a VM that provides 2U of vCPU and 4GB of memory is a 2U4G VM. A user selects the appropriate VM type for their requirements and budget and submits the requests to the cloud platform, which provides multiple PMs to fulfill the user's resource needs. The scheduler determines which PMs to assign the VMs to, similar to the classical bin packing problem, and the user can execute their jobs once allocation is complete. The VM placement problem entails loading all the items (VMs) into bins (PMs) based on optimization objectives. From a financial perspective, larger-sized VMs are more valuable, for example, a 4U4G VM is priced higher than two 2U2G VMs due to its ability to occupy the same space as two 2U2G VMs, but not vice versa. In brief, the PM's service potential can be evaluated based on the highest value VM combination it can host. This indicates the maximum future profitability for the cloud provider. To take advantage of this, cloud providers reserve space for larger and higher-value VMs. Our VM placement algorithm takes into account the value of each VM type, which is determined based on their pricing, and prioritizes the allocation of space for the most profitable VMs. The objective function for this approach is presented in Section 3.

2.3 Framework of Our Algorithm

The VM placement problem can be divided into two scenarios: offline and online. The offline scenario involves periodic placement and is typically solved using optimization-based algorithms with predefined schedule intervals, and can be used to evaluate the theoretical upper bound of the online policy. However, these solutions are time-consuming and not suitable for real-time user requests. In contrast, the online scenario involves event-driven scheduling and requires the use of heuristic algorithms for real-time resource allocation. To achieve a balance between solution quality and execution time, we propose a prediction-based placement scheme that combines the advantages of both scenarios. Inspired by [13], the scheme consists of two parts: an offline planner that provides a coarse-grained placement policy and an online scheduler that schedules real-time VM requests based on the policy.

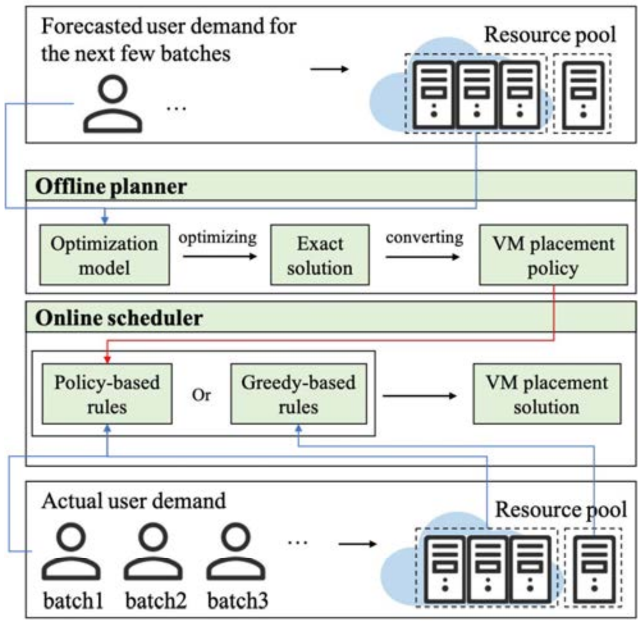


Fig. 2: Framework of Our Algorithm

The overall framework of our algorithm is shown in Fig. 2 and the timeline is shown in Fig. 3. Firstly, the future VM request batches are forecasted along with their respective quantities. The PM resource pool is then divided into two groups: one for policy-based rules and the other for greedy-based rules. As shown by the blue arrow at the top of the figure, the forecasted requests and current PM resource usage in the policy-based resource pool are sent to the offline planner. The optimization model in the offline planner produces fine-grained solution results, which are then converted into coarse-grained placement policies(shown by the red arrow). These policies represent the optimal placement within each PM, subject to resource and demand constraints. When actual VM demands arrive in batches, the online scheduler assigns the target PM for placement based on the placement policies. In case of unsuccessful placement, the VM requests are placed according to the greedy-based rules.

3 Optimization Problem Formulation

In this section, we describe the mathematical model of VM placement problem that considers user demand constraints, resource constraints, and NUMA architecture as well as value-driven objective function.

3.1 Resource Pool Model

We consider a cloud platform with a large number of homogeneous PMs. The resource pool for policy-based VM placement can be defined as: $RP = \{PM_1, PM_2, \ldots, PM_k, \ldots, PM_K\}$ where $k \in [1, K]$ and K is the total number of these PMs.

Each PM comprises CPU and memory. In order to introduce NUMA-structure into the model, we use a four dimensional vector to describe each PM_k as $PM_k = (CPU_k^1, RAM_k^1, CPU_k^2, RAM_k^2)$, where CPU_k^1 represents the remaining CPU resource for PM_k in NUMA node 1, and RAM_k^1 represents the remaining memory resource for PM_k in NUMA node 1. Likewise, the second two dimensions represent the same things in NUMA node 2. Here we mainly take two dimensions of resources in to account. But due to the form of our model, new kind of resources are easy to add.

3.2 User Request Model

Our cloud platform predefines several VM types for users, which comprise different configurations of CPU and memory. User demand arrives in batches and a batch can be represented as $B_w = \left(V^s, R_w^s, V^d, R_w^d\right)$, where:

- w represents the ID of the forecasted VM demand batch, where $w \in [1, W]$, and W is the total number of VM demand batches submitted by users in the forecasted time period.
- V^s and V^d represent single-NUMA and double-NUMA VM types that a user can apply for respectively. $V^s = \left\{V_1^{s1}, V_1^{s2}, ..., V_i^{s1}, V_i^{s2} ..., V_N^{s1}, V_N^{s2}\right\}$ where $i \in [1, N]$, represents N different single-NUMA VM types in two NUMA nodes. To better match the form of PM, we also use a four-dimensional vector to describe each single-NUMA VM type. $V_i^{s1} = (CPU_i^1, RAM_i^1, 0, 0)$ and $V_i^{s2} = (0, 0, CPU_i^2, RAM_i^2)$ are two different patterns with the same resource configurations for single-NUMA VM type i.
  $V^d = \left\{V_1^d, \ldots, V_j^d, \ldots, V_M^d\right\}$ where $j \in [1, M]$ represents M different double-NUMA VM types and $V_j^d = (CPU_j^1, RAM_j^1, CPU_j^2, RAM_j^2)$ shows the resource configuration it offers.

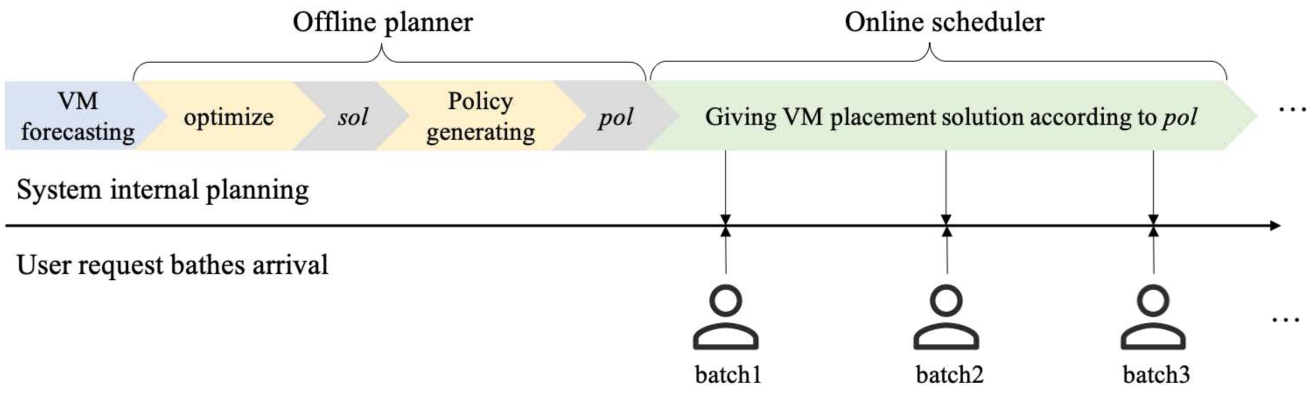


Fig. 3: Timeline of Our Algorithm

- $R_w^s = \{n_{w1}, \dots, n_{wN}\}$ is the number of requests for each type of single-NUMA VM in the wth batch of user demand. Similarly $R_w^d = \{m_{w1}, \dots, m_{wM}\}$ indicates the number of requests for each type of double-NUMA VM in the wth batch of user demand.

3.3 Problem Formulation

Under the condition that all VM demand arriving in the forecasted time period is known, the optimal solution for their placement can be solved with the following model:

$$\max_{x_{jk}^1, x_{jk}^2, y_{jk}} \quad \sum_{k=1}^{K} \left[\sum_{i=1}^{N} \omega_i \left(x_{ik}^1 + x_{ik}^2 \right) + \sum_{j=1}^{M} \delta_j y_{jk} \right] \quad (1)$$

$$\sum_{k=1}^{K} \left(x_{ik}^1 + x_{ik}^2 \right) \geq \sum_{w=1}^{W} n_{wi}, \quad i = 1, \cdots, N \quad (2)$$

$$\sum_{k=1}^{K} y_{jk} \geq \sum_{w=1}^{W} m_{wj}, \quad j = 1, \cdots, M \quad (3)$$

$$\sum_{i=1}^{N} \left(V_i^{s1} x_{ik}^1 + V_i^{s2} x_{ik}^2 \right) + \sum_{j=1}^{M} V_j^d y_{jk} \leq PM_k$$
$$k = 1, \cdots, K \quad (4)$$

$$x_{ik}^1, x_{ik}^2, y_{jk} \in \mathbb{N},$$
$$i = 1, \cdots, N \quad j = 1, \cdots, M, \quad k = 1, \cdots, K \quad (5)$$

The objective function in formula (1) is to maximize the total value of the remaining resources. Decision variable x_{ik}^1 and x_{ik}^2 denotes the number of single-NUMA VM type i allocated to the first and the second NUMA node of host PM_k. Decision variable y_{jk} denotes the number of double-NUMA VM type j allocated to host PM_k. ω_i is the value of single-NUMA VM type i and δ_j is the value of double-NUMA VM type j.

Constraint in formula (2) and (3) ensures that all batches of user demand arriving in the current time period are met. Here multiple VMs can be placed on the same PM. Constraint in formula (4) ensures the allocation of VMs in the host PM_k must not exceed the resource capacity offered by the host PM_k. Constraint in formula (5) indicates that decision variables take a value from a set of non-negative integer number.

4 Algorithm Design

This section outlines the conversion of fine-grained optimization results into coarse-grained heuristic policies. The exact optimization algorithm's model is highly extensible and can be updated with new constraints, but its solution time is too long to meet the demands of online scheduling. Using the heuristic policies generated by the algorithm proposed in this section not only meets time requirements but also results in efficient placement.

4.1 Forecasting

Cloud providers typically offer two payment options: the reservation plan and the on-demand plan. The reservation plan provides resources at a lower cost, but only for a predetermined period in the future. The on-demand plan allows for more flexible resource acquisition, without long-term commitments, at a potentially higher cost. The existence of reservation plans has improved the accuracy of forecasting VM batches in the future, and there are many forecasting methods in the literature for on-demand plans. [14] presents a cross-correlation prediction approach based on machine learning that predicts resource demands of multiple resources of VMs running in a cloud infrastructure. [15] demonstrates that a time-dependent hidden Markov model with an autoregressive observation process replicates the properties of the CPU consumption data. In our algorithm, the order of arrivals is not crucial, as long as the basic number of each VM type in the predetermined time period can be accurately predicted.

4.2 Placement Policy Generating

The optimization-based algorithm provides optimal placement solutions in each PM, which can be translated into a coarse-grained placement policy through clustering. This study aims to develop a type-oriented VM placement algorithm (as shown in Section 3) and clusters different VM types into groups based on similarity, specifically the size ratio of CPU and memory. All VM types with equal size ratio are considered one group, with N+M VM types clustered into Q groups.

Fig. 4 illustrates the process of clustering as we forecast a future batch of VM requests, which consists of five VM types, 4U2G, 2U4G, 2U4G, 2U1G, and 1U2G, listed from left to right. The VMs are grouped into two cate-

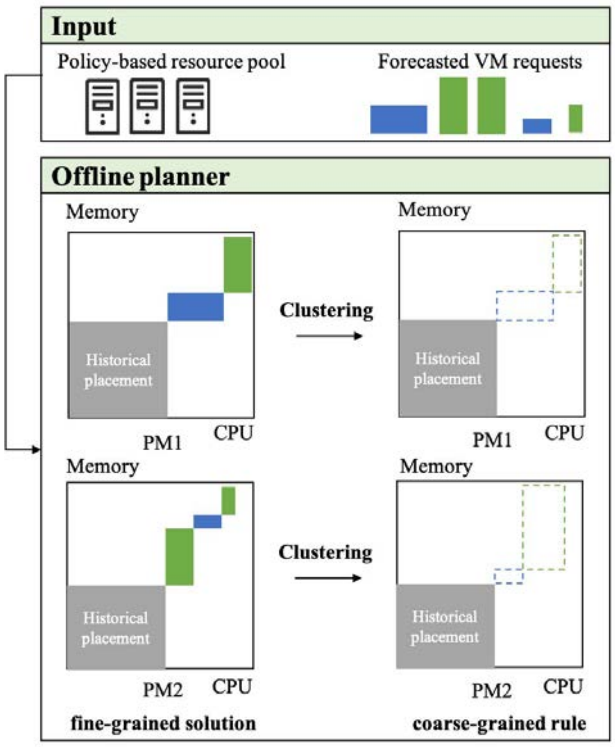


Fig. 4: An illustration of clustering process

gories based on their size ratios, depicted by blue for 2:1 and green for 1:2 in the figure. The current PM state and the forecasted VMs are calculated using the model described in Section 3 to produce a fine-grained solution, shown on the left of the figure. The solution in each PM, represented by the optimal value of the decision variable in Section 3 as $sol_k = \left\{x_{1k}^{1,*}, \cdots, x_{Nk}^{1,*}, x_{1k}^{2,*}, \cdots, x_{Nk}^{2,*}, y_{1k}^{*}, \cdots, y_{Mk}^{*}\right\}$, is then aggregated by VM type groups.

In this example, PM2 aggregates one 2U4G and one 1U2G VM, leaving 3U6G of space for a 1:2 ratio group. Instead of directly using the optimization model's results, the placement policy improves the algorithm's fault tolerance by aggregating the VM types based on their size ratio. In case the prediction is inaccurate, it would still work if the size of each VM group could be accurately predicted, such as placing a 3U6G VM request in PM2, even if it doesn't appear in the prediction. The problem's simplicity is further enhanced as the VM group sizes have the same ratio, allowing for the efficient use of heuristics for one-dimensional bin-packing problem, such as BestFit. The specific algorithm flow is depicted in Algorithm 1.

4.3 Online scheduling

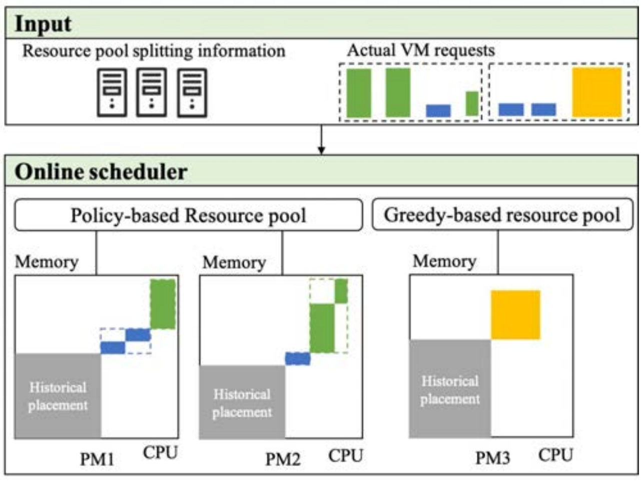
Our online VM placement algorithm works in two phases:

- *Policy-based placement phase*: each arriving VM batch is placed guided by policy-based rules.
- *Greedy-based placement phase*: VM demands that could not be placed in the first phase is placed by greedy-based rules.

In Fig. 5, two batches of forecasted VM requests have already been considered and a placement policy has been generated by the offline planner. When the actual VM requests arrive in batches, they are sorted by CPU size and placed

Algorithm 1 Placement policy generating algorithm

```
Input: Fine-grained solution $sol$, Number of VM groups $Q$, Number of PMs $K$ , Information of VM type $V^s$and$V^d$
Output: policy $pol$
 1: Initialize $pol$=zeros$(K, Q)$
 2: for $k = 1, \cdots, K$ do
 3:     for $i = 1, \cdots, N$ do
 4:         determine VMGroup $q$ to which VMtype of $i$ belongs
 5:         if single-NUMA VMtype $i \in$ VMGroup $q$ then
 6:             $pol[k][q] += (x_{ik}^{1,*} V_i^{s1} + x_{ik}^{2,*} V_i^{s2})$;
 7:             break;
 8:     for $j = 1, \cdots, M$ do
 9:         determine VMGroup $q$ to which VMtype of $j$ belongs
10:         if double-NUMA VMtype $j \in$ VMGroup $q$ then
11:             $pol[k][q] += y_{jk}^{*} V_j^{d}$;
12:             break;
13: return $pol$
```


Input
Resource pool splitting information
Actual VM requests
Online scheduler
Policy-based Resource pool
Greedy-based resource pool
Memory
Historical placement
PM1
CPU
PM2
PM3
Fig. 5: An illustration of online scheduling

one by one. The first batch consists of 2U4G, 2U4G, 2U1G, and 1U2G VMs. To place each VM, we find the PM with the least space reserved for its corresponding VM group in the policy-based resource pool, starting with the 2U4G VM placed in PM1. The offline planner first reduces the two-dimensional problem to a one-dimensional problem and then uses a heuristic algorithm for placement. If a VM that was not predicted, such as the 4U4G in the next batch, cannot be accommodated by a policy-based PM, the greedy-based placement phase is initiated. The 4U4G is placed in the greedy-based resource pool using the BestFit method, based on CPU size. The greedy-based placement phase is essentially a one-dimensional bin-packing algorithm, but not as efficient as the policy-based placement phase. In the policy-based placement phase, our policy divides the space for each PM and tries to reserve space for VM types not included in the forecast, particularly high-value VM types, while satisfying the predicted demand constraints with the model from section 3. Large cloud providers usually have ample resources and can complete placement in the first stage. However, if the policy reserved space differs significantly from actual VM arrivals, a second placement phase is required. The overall process of the online scheduling can be summarized in Algorithm 2.

Algorithm 2 Online scheduling algorithm

```
Input: policy $pol$, Number of VM groups $Q$, Newly arrived VM request $Req$, Information of VM type $V^s$and$V^d$, PM list for policy-based phase $PM$
1:  all requests in descending order according to their CPU size
2:  $R = Req.length$
3:  for $r = 1, \cdots, R$ do
4:      determine VMGroup $q$ to which VMtype of $Req[r]$ belongs
5:      if $Req[r]$ is double-NUMA then
6:          if $\exists$ a PM$k$ with minimum CPU capacity for VM group $q$ in $pol$ to be allocated successfully then
7:              allocate $Req[r]$ to PM$k$;
8:              $pol[k][q] -= V_j^d$
9:          else
10:             allocate $Req[r]$ by BestFit
11:     else
12:         if $\exists$ a PM$k_1$ with minimum CPU capacity for VM group $q$ in $pol$ to be allocated successfully in Numa node 1 and a PM$k_2$ in NUMA node 2 then
13:             if CPU capacity of PM$k_1$ is smaller then
14:                 allocate $Req[r]$ to PM$k_1$
15:                 $pol[k][q] -= V_i^{s1}$
16:             else
17:                 allocate $Req[r]$ to PM$k_2$
18:                 $pol[k][q] -= V_i^{s2}$
19:         else
20:             allocate $Req[r]$ by BestFit
```

5 Experiment

In this Section, we present results for our algorithm's performance and corresponding computational time compared to other algorithms. The integer optimization model in the offline planner is solved via branch and bound algorithm. All algorithms are tested by Matlab on a computer equipped with Intel(R)Core(TM) i7-10710U CPU @1.10GHz and 1.61GHz, 16GB RAM, running the Windows 10 system.

5.1 Experiment Setment

Workload and environment. For this experiment, actual data from a large-scale cloud platform was gathered. Since our method is intended for large-scale cloud VM scheduling, the user request data in the current real cloud environment was inadequate. Consequently, we extracted 9000 VM data from the actual data, with the number of VMs in each batch determined based on experimental requirements. As outlined in Table 1, this study involved 18 VM types, including 16 single-NUMA VM types and 2 double-NUMA VM types, classified into 5 VM groups based on sizeratio and NUMA status. The negative value in the table means that this type of VM is not favored and cloud providers want to consider as few as possible.

Additionally, 20,000 PM data were selected for testing, all with the same specifications but varying resource consumption.

Baseline algorithms. In the following experiments, we compare our algorithm with three baseline methods.

- *FirstFit*: Select the PM with the smallest index that satisfies the resource demand of the VM request.
- *BestFit*: Select the PM with the smallest remaining resources that satisfies the resource demand of the VM request.
- *Optimal*: Optimal placement solution for each batch of VMs calculated by integer programming.

Table 1: Description of VM types considered in the experiment

VM type	NUMA	Proportion	Value
1	single	6.07%	1
2	single	40.44%	1.4
3	single	6.96%	4
4	single	4.04%	11.3
5	single	0.56%	-32
6	single	0.94%	2
7	single	1.63%	5.6
8	single	0.79%	16
9	single	0.98%	-45.2
10	single	1.73%	12
11	single	2.28%	33.9
12	single	1.16%	96
13	single	0.39%	176.3
14	single	1.30%	271.5
15	single	0.12%	498.8
16	single	17.72%	768
17	double	11.61%	1410.9
18	double	1.28%	2172.2

5.2 Results and analysis

Performance. In the upcoming experiments, we assess the efficacy of the various algorithms utilizing the service capability values proposed in Section 3. The 9000 VM data is split into 30 batches, with each batch comprising 300 VM data. Our approach typically predicts 5 batches of data in advance with 100% prediction accuracy. Figure 6 shows the experimental outcomes. After placing 30 batches, the resource pool's overall service capacity has decreased, necessitating resource consumption to meet current user requests, resulting in a reduction in the remaining resources' serviceability. However, as depicted in the figure, the FirstFit approach results in a greater reduction than the BestFit method. Our approach is mostly consistent with the service capacity degradation caused by the optimal placement scheme, with a 9.69% less degradation than the FirstFit approach and an 8.04% less degradation than the BestFit approach.

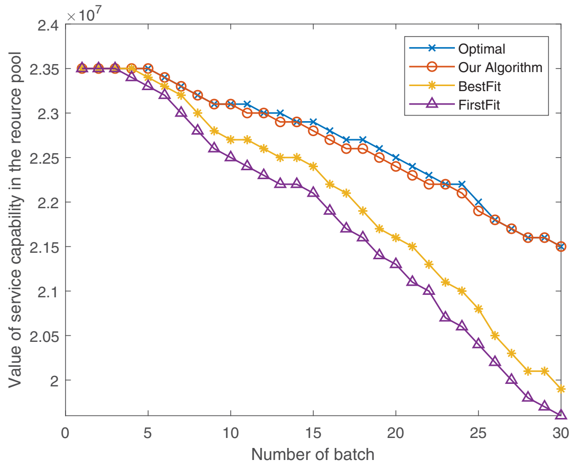


Fig. 6: Performance of each algorithm

Computational time. In addition to algorithm effectiveness, computational time is also a crucial aspect to measure online algorithms. Our tests demonstrate this in two ways. Firstly, we show the relationship between computational time and the number of VMs in each batch in Table 2. Here, we placed a total of 10 batches of VMs, consisting of 100, 200, 300, 400, and 500 VMs each. There are 10,000 PMs in the resource pool, with the first 9,000 PMs for the policy-based placement phase and the remaining 1,000 PMs for the Greedy-based placement phase. The optimal algorithm takes the longest time, which is unsuitable for an online algorithm. Our online method's placement time in online scheduler aligns with the heuristic algorithms BestFit and FirstFit, and adjusting the number of predicted batches with the computation time of the offline planner results in better placement. Secondly, we show the relationship between computational time and the number of PMs in the resource pool in Table 3. The resource pool consists of 2,500, 5,000, 10,000, 15,000, and 20,000 PMs, with the first 90% for the policy-based placement phase and the remaining 10% for the Greedy-based placement phase. As the PM pool size increases, the computational time of all algorithms except FirstFit increases significantly. The FirstFit method automatically picks the first PM as the placement target, so changes in the size of the resource pool do not affect its computational time. The optimal algorithm's computational time gradually becomes unacceptable, while our algorithm's computational time can be kept within an acceptable range.

Table 2: Computational time comparison with varying number of VMs in each batch

Number of VMs in each batch	Computational time(s)				
	FirstFit	BestFit	our algorithm		Optimal
			offline	online	
100	0.015	0.336	149.77	0.111	8063.9
200	0.009	0.635	2343.5	0.764	13713
300	0.014	0.975	4005.7	0.449	13622
400	0.017	1.259	2371.7	0.647	12319
500	0.023	1.603	2379.3	0.898	11403

Table 3: Computational time comparison with varying number of PMs in resource pool

Number of PMs in resource pool	Computational time(s)				
	FirstFit	BestFit	our algorithm		Optimal
			offline	online	
2500	0.016	0.274	70.875	0.122	437.26
5000	0.014	0.574	247.14	0.338	875.48
10000	0.014	0.975	4005.7	0.449	13622
15000	0.016	1.388	5659	0.658	36807
20000	0.015	1.964	9529.7	1.227	53448

Impact of prediction accuracy. Our algorithm's placement effect is impacted by the accuracy of the prediction since it needs to consider future batches of VMs. To examine the relationship between the prediction accuracy and the placement effect, we prepared 30 batches of VM requests, with each batch comprising 300 VM data. Also, there are 10000 PMs with the first 90% for the policy-based placement phase and the remaining 10% for the Greedy-based placement phase. Our approach typically predicts 5 batches of data in advance with 50% and 100% prediction accuracy respectively. After placing 30 batches of VMs, Figure 7-10 shows the experimental outcomes. The orange part represents the CPU consumption before placement, and the blue part represents the increased CPU consumption after placement. To clearly represent the placement effect, the PMs in both resource pools are sorted from highest to lowest CPU resource consumption, and empty PMs with no resource consumption are hidden. The results indicate that with a prediction accuracy of 50%, the greedy-based resource pool consumes more resources, implying that some VMs were placed in the greedy-based phase. Conversely, with a prediction accuracy of 100%, the resource consumption in the greedy-based resource pool remains the same as before placement, indicating that with higher prediction accuracy, VMs are mainly placed in the policy-based phase, which is consistent with the results obtained from the optimization method.

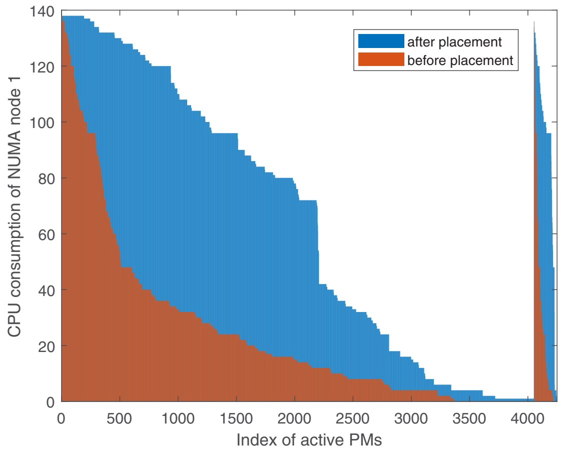


Fig. 7: CPU consumption of NUMA node 1 after 30 batches placement with 50% prediction accuracy

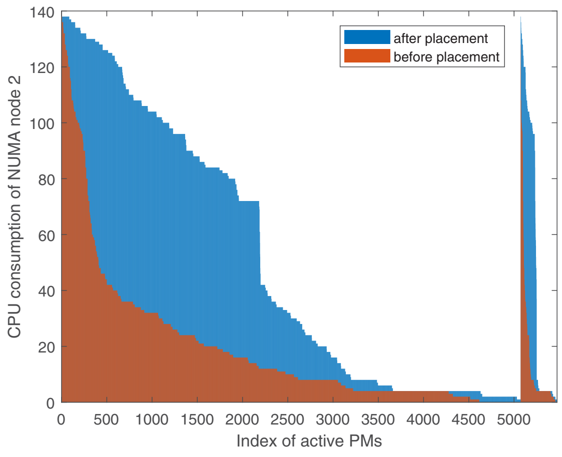


Fig. 8: CPU consumption of NUMA node 2 after 30 batches placement with 50% prediction accuracy

6 Conclusion

In this paper, we demonstrate the challenges of VM placement in large cloud platforms and propose an algorithm to

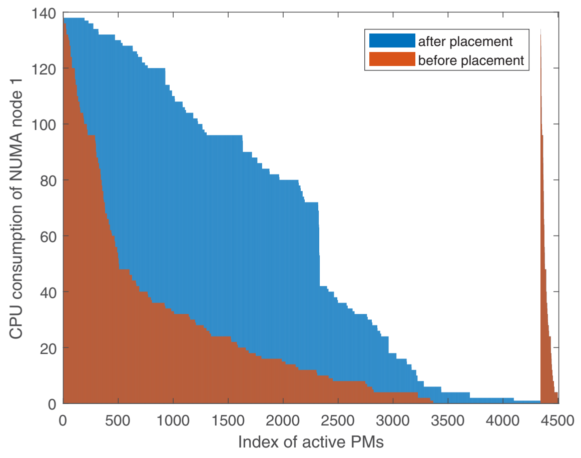


Fig. 9: CPU consumption of NUMA node 1 after 30 batches placement with 100% prediction accuracy

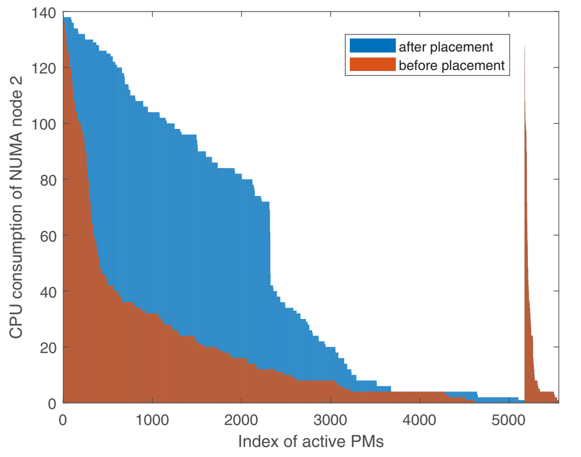


Fig. 10: CPU consumption of NUMA node 2 after 30 batches placement with 100% prediction accuracy

address them by converting fine-grained optimizations into coarse-grained heuristics. Our approach models the VM placement as an MCBPP problem, considers NUMA structure, and maximizes service capability value of the resource pool. In experiments, our algorithm performs comparably to common heuristics in placement time and approaches optimization algorithms in placement efficiency.

References

[1] Shifeng Shang, Bo Wang, Jinlei Jiang, Yongwei Wu, and Weimin Zheng. An intelligent capacity planning model for cloud market. *J. Internet Serv. Inf. Secur.*, 1(1):37–45, 2011.

[2] Sivadon Chaisiri, Bu-Sung Lee, and Dusit Niyato. Optimal virtual machine placement across multiple cloud providers. In *2009 IEEE Asia-Pacific Services Computing Conference (APSCC)*, pages 103–110. IEEE, 2009.

[3] Wubin Li, Johan Tordsson, and Erik Elmroth. Virtual machine placement for predictable and time-constrained peak loads. In *International Workshop on Grid Economics and Business Models*, pages 120–134. Springer, 2012.

[4] Antonio Marotta and Stefano Avallone. A simulated annealing based approach for power efficient virtual machines consolidation. In *2015 IEEE 8th international conference on cloud computing*, pages 445–452. IEEE, 2015.

[5] Dabiah Alboaneen, Hugo Tianfield, Yan Zhang, and Bernardi Pranggono. A metaheuristic method for joint task scheduling and virtual machine placement in cloud data centers. *Future Generation Computer Systems*, 115:201–212, 2021.

[6] Y. Q. Gao, H. B. Guan, Z. W. Qi, Y. Hou, and L. Liu. A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. *Journal of Computer and System Sciences*, 79(8):1230–1242, 2013.

[7] Rui Li, Qinghua Zheng, Xiuqi Li, and Jie Wu. A novel multi-objective optimization scheme for rebalancing virtual machine placement. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pages 710–717. IEEE, 2016.

[8] Junjie Sheng, Yiqiu Hu, Wenli Zhou, Lei Zhu, Bo Jin, Jun Wang, and Xiangfeng Wang. Learning to schedule multi-numa virtual machines via reinforcement learning. *Pattern Recognition*, 121:108254, 2022.

[9] Shuo Fang, Renuga Kanagavelu, Bu-Sung Lee, Chuan Heng Foh, and Khin Mi Mi Aung. Power-efficient virtual machine placement and migration in data centers. In *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, pages 1408–1413. IEEE, 2013.

[10] Seontae Kim and Young-ri Choi. Effects of vm placement constraints in heterogeneous virtual clusters. In *2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS* W)*, pages 30–36. IEEE, 2018.

[11] Yuxia Cheng, Wenzhi Chen, Zonghui Wang, and Xinjie Yu. Performance-monitoring-based traffic-aware virtual machine deployment on numa systems. *IEEE Systems Journal*, 11(2):973–982, 2015.

[12] Hongjian Li, Guofeng Zhu, Chengyuan Cui, Hong Tang, Yusheng Dou, and Chen He. Energy-efficient migration and consolidation algorithm of virtual machines in data centers for cloud computing. *Computing*, 98(3):303–317, 2016.

[13] Jiyuan Shi, Junzhou Luo, Fang Dong, Jiahui Jin, and Jun Shen. Fast multi-resource allocation with patterns in large scale cloud data center. *Journal of computational science*, 26:389–401, 2018.

[14] Dorian Minarolli and Bernd Freisleben. Cross-correlation prediction of resource demand for virtual machine resource allocation in clouds. In *2014 Sixth International Conference on Computational Intelligence, Communication Systems and Networks*, pages 119–124. IEEE, 2014.

[15] Hugo Lewi Hammer, Anis Yazidi, and Kyrre Begnum. An inhomogeneous hidden markov model for efficient virtual machine placement in cloud computing environments. *Journal of Forecasting*, 36(4):407–420, 2017.