# FreeVM: A Server Release Algorithm in DataCenter Network

Shiyan Zhang[†], ★Yuchao Zhang[†], Xiangyang Gong[†], Ran Wang[†],
[†]Beijing University of Posts and Telecommunications

***Abstract*—With the development of 5G access technology and the corresponding explosive growth of user requests, service providers have to activate more and more physical machines (P-M) in cloud datacenters. This simple expansion of PMs results in not only low utilization of servers but also high maintenance cost. Existing researches that try to release unnecessary servers by migrating VMs always face a severe challenge-the large searching space for the optimal VM placement solution. In this paper, we prose a two-stage variable neighborhood searching (STVNS) algorithm, named FreeVM, which can significantly reduce the number of occupied servers. FreeVM works harmoniously with all kinds of VM placement schemes under various scenarios. We conducted an extensive series of experiments using real traces, and the results show that FreeVM can release at least 15% physical machines compared with existing solutions.**

***Index Terms*—VM placement, energy conservation, resource management, data center**

## I. Introduction

In recent years, with the rapid development of virtualization techniques, Internet of Things (IoT) and AI driven applications, various cloud service providers have gradually established their own cloud-based data centers (DCs) [1]. Hence, modern cloud data centers need to tackle efficiently the increasing demand for computing resources while at the same time addressing the energy efficiency challenge. It is obviously not enough to expand the scale of the data center to cope with the surge in resource demand, and a large number of PMs in the power-on state will also bring huge energy consumption [2]. Only through efficient and reasonable management of the incoming user requests to improve the resource utilization of the PM, while reducing the number of active PMs as far as possible, can effectively deal with the rapid growth of user requests.

In green cloud computing, power-aware VM placement solutions are very important to make the data center more efficient and reduce the power consumption of the data center. First, one needs to consider the power consumed by various server components (CPU, memory, etc.) and the overhead of PM in different states. Then, the cost of VM migration needs to be considered when vacating the PM. Then, the basic challenge is to map each incoming VM to the most appropriate PM, thus minimizing the total power consumption of all active PMs and achieving maximum resource utilization per PM. Optimal configuration of VMs is one of the mainstream methods for efficient utilization of data center hardware and is considered to be at the core of cloud computing. In addition, the release of VMs in different life cycles leads to fragmentation on the PM, even if we carefully orchestrate and place different combinations of requests on the same PM, the resources of the PM can be fully utilized. However, as the life cycle of some of these requests ends, the resources on the PM are released. If reallocation is not performed, then the PM will generate fragments because the ratio of the remaining resources does not match the ratio of the request. Therefore, PM resource fragmentation is another reason for the low utilization of PM resources in the data center.

The VM placement approaches [3] can be static or dynamic. Static placement scheme [4] can pre-dispatch allocation strategies, but reallocation cannot be recalculated over time. Dynamic placement scheme [5] allows to adjust the placement policy according to the load of the system. Greed-based virtual machine placement algorithm implements load balancing [6], but introduces a large amount of computational overhead. Recently, the widely popular kubernetes is an open-source platform introduced by Google in 2014 to manage microservices-based applications. However, kubernetes is not well-suited for deploying containers in a geo-distributed computing environment and dealing with the dynamism of application workload and computing resources [7].

In this paper, we address the resource management issues in DCs, which aims to maximize resource utilization and take off the low utilization PM for minimizing the energy consumption. The first is to quickly place continuously arriving requests on PMs with matching attributes and available, and then use two-stage variable neighborhood searching (STVNS) algorithm to migrate all resources on the "cold" PMs to "hot" PMs, improving overall PM utilization. Finally, by predicting the resource demand in the next cycle to determine how many vacated PMs can be shut down, and finally achieve the purpose of limiting energy consumption. The results of the experiment show that the STVNS outperforms the existing solution, saving approximately 10% of the number of PMs.

The remainder of this paper is organized as follows. First, we start with summarizing the related existing studies in Section II. Then, we give the optimization objectives and

★Corresponding Author: Yuchao Zhang, yczhang@bupt.edu.cn.

The work was supported in part by the National Natural Science Foundation of China (NSFC) Youth Science Foundation under Grant 61802024, Key Special Projects of Beijing Natural Science Foundation under Grant M21030, the Fundamental Research Funds for The Central Universities under Grant 2482020RC36, and the National Key R&D Program of China under Grant 2019YFB1802603.

constraints of FreeVM in Section III. Section IV describes the detailed design of FreeVM. In Section V, extensive experiments are performed and results are analyzed. Finally, Section VI concludes the paper.

## II. Related Work

As the popularity of virtual network functions (NFV) and container grows, many research works have investigated in the cloud environment, which includes VM placement/allocation and energy-efficient.

**VM placement/allocation.** VMP is mapping VMs to appropriate PMs. The process of mapping the VMs to the PMs is called the VMP problem and is known to be NP-hard. Multi-Objective Genetic Algorithm (MOGA) [8] is proposed to solve the problem of interdependent VMs placement while considering the trade-off between reliable resource allocation and Qos. To achieve a balance between the communication overhead and overall throughput, [9] proposed a solution called FreeContainer, which uses a new two-phase algorithm to redistribute containers among hosts. Furthermore, FreeContainer does not require hardware modifications and has been extensively evaluated in real environment. Lv, Liang et al. [10] designed an efficient communication-aware worst fit decreasing algorithm to place a set of new containers into data centers, and further explored the conflict between container communication and resource utilization in a data center.

**Energy-efficient.** Despite the successes achieved in recent years in data center energy efficiency, the growth of data center electricity consumption after 2020 is uncertain. Therefore, cost savings and effective resource management strategies associated with cloud data center energy efficiency become critical. A joint computing, data transmission and migration energy cost (JCDME) model is proposed in [11], where the overall energy efficiency is improved by optimizing the virtual elements (VE) allocation in a way that introduces weighting parameters. [12] designed a QoS-based virtual machine integration framework that minimized Service Level Agreement violations while reducing energy consumption. [13] presented a phase-wise optimization method based on the ant colony meta-heuristic algorithm.

In summary, the main drawback of all the above studies is that they do not take into account VM requests and PMs attribute characteristics. In this paper, we build a mathematical model by converting the energy consumption problem into overhead billing, and we design FreeVM using the STVNS algorithm to save energy by quickly placing VM requests and further vacating underutilized PMs.

## III. Problem statement and formulation

In this section, we discuss how to allocate N VM requests to M PMs by analyzing the overall costs under specified constraints. To maximize resource utilization and minimize energy consumption in the data center, we face the trade-off between benefits (vacated machines) and overhead (VMs migration and multi-resource load balance). Table I summarizes the default notation used in the problem definition.

TABLE I
Notation Definitions

| Notation | Meaning |
|---|---|
| $P$ | The set of physical machines |
| $V$ | The set of VM requests |
| $P_{on}$ | The set of enabled PMs |
| $P(v)$ | PMs allocated for resourcerequest v |
| $P^{'}(v)$ | PMs reallocated for resourcerequest v |
| $D_v^r$ | The demand of resourcerequest v for resource r |

### A. Objective

There are two aspects when quantifying the total overhead of any distributed status, i.e., there are the PM cost and Qos. where Qos includes migration cost and balancing cost.

**PM Cost.** When a batch requests arrive, all PMs have On/Off status. A PM in On state consumes power, here, we convert time to billing. When the PM is On, it charges $C_{on}$ per minute; PM in Off state is not billed. The PM cost is then expressed as:

$$Pcost = C_{on} * |P_{on}| \tag{1}$$

**Migrate Cost.** Real-time migration of virtual machines makes it possible to transfer a VM from the source node to the destination node with minimal downtime and hang-up time. Although dynamic migration is transparent to end users, it may result in some performance degradation of applications running in the VM. To quantify migration costs, we have:

$$\begin{gathered} Mcost = C_M * \sum_{r\in R}\sum_{v\in V} D_v^r * f(v) \\ f(v) = \begin{cases} 1, & P(v) \neq P^{'}(v) \\ 0, & P(v) = P^{'}(v) \end{cases} \end{gathered} \tag{2}$$

**Balance Cost.** If the PM has no available resources for future VM allocations or upcoming requests, so the residual amount of multiple resources should be balanced. Balance_Ratio [9] represents the target ratio of resource $r_i$ and resource $r_j$. In addition, the Balance_Ratio dynamically varies per cycle.

$$\begin{gathered} Bcost = \sum_{(r_i,r_j),\forall r_i\neq r_j} cost(r_i,r_j) \\ cost(r_i,r_j) = \sum_{p_k\in P} |\, (r_i + vmr_i - (r_j + vmr_j) * \\ Balance\_Ratio)\,| - |r_i - r_j * Balance\_Ratio| \\ Balance\_Ratio = t(r_i,r_j) \end{gathered} \tag{3}$$

According to the above definitions, the objective function to be minimized can be defined as the weighted sum of all the costs, i.e.,

$$Cost = w_P * Pcost + w_M * Mcost + w_B * Bcost \tag{4}$$

### B. Constraints

To minimize the above cost, VMs should be placed or reassigned to the most appropriate PM, but this process should satisfy some strict constraints.

A resource request is equivalent to a VM request, and each resource request needs to be placed on an appropriate PM. In order to ensure that the VM re-distribution process should not be interrupted, some constraints need to be satisfied during VM migration.

Constraint 1: (Capacity) In each PM, resource usage cannot exceed its capacity.

Each resource usually has a capacity constraint. For the resources (cpu, memory, etc.) of each PM, the total amount of resources allocated to all the VMs cannot exceed the resource capacity of the PM. Assume that a VM request includes two resource types (CPU, memory). Let the sets of VMs and PMs be denoted by V and P, respectively. Without loss of generality, let $V = \{v_1, v_2, ..., v_N\}$ and $P = \{p_1, p_2, ..., p_M\}$. For each requested VM v, let $\alpha_v$ be the number of CPUs required and let $\beta_v$ be the memory requirement (in GiB). For each PM p, let $C_p$ be the number of CPUs it can support, $M_p$ be the amount of memory (in GiB). In addition, each $v \in V$ and each $p \in P$, let be $x_{vp}$ the binary assignment variable, which takes the value 1 if VM v is assigned to PM p and 0 otherwise. The following constraints are required:

$$\sum_{p \in P} x_{vp} = 1, \forall v \in V \tag{5}$$

$$\sum_{v \in V} \alpha_v x_{vp} \leq C_p, \forall p \in P \tag{6}$$

$$\sum_{v \in V} \beta_v x_{vp} \leq M_p, \forall p \in P \tag{7}$$

Among them, (5) ensures that every VM is assigned to exactly one PM. (6) and (7) are the constraints on the resource capacity of the number of CPUs and the total memory size of each PM p, respectively.

Constraint 2: (Transient) During the migration process, the vm instance will not be destroyed before it is established on the new PM.

The reassignment of a VM is achieved through a live migration, which means that the VM is transferred to the final machine while keeping it running on the original one. Such resources (e.g., CPU and memory) are needed on both machines (initial and final machines) during a live migration, as the processes use the resources on both machines during the reassignment.

Constraint 3: (Spread) Some services need to assign their VMs to different DCs, for example for security reasons.

A specific function in a high performance application is usually implemented on multiple VMs to support concurrent operations. For example, the user ID of a resource request involves two types of vip users and retail users. One user ID of a retail user corresponds to one VM request, and one user ID of a vip user corresponds to multiple VM requests. As these VMs are very sensitive to resource requests with the same user ID. Therefore, they cannot be placed on the same PM, otherwise, other resources such as memory and I/O will be wasted seriously.

Constraint 4: (Trait) The resourcerequest for the specified traits can only be placed on the PM that the traits match.

## IV. FreeVM

In this section, we present the design of FreeVM. The goal of FreeVM is to find a VM redistribution solution that minimizes the overall cost (Equation (4)). FreeVM is consisted by two stages, first placement (FP) and variable neighborhood searching (VNS), collectively known as two-stage variable neighborhood searching (STVNS). FP entails simple and fast placement of VMs, along with placing as many user requests as possible; VNS is aimed to compress the number of running PMs, which is mainly to vacate the "cold" PMs and process the VM on the "hot" PMs, so as to improve the resource utilization rate of the PMs.

*1) First placement:* In addition to basic information such as the request number and user ID, each resource request also specifies some qualitative attributes (trait, such as the operating system is Windows/Linux) and quantitative attributes (resource, such as the demand for CPU) of the target PM. Accordingly, each PM also has clear qualitative attributes, fixed PM specifications, current state (On/Off), and current amount of remaining resources. The FP algorithm needs to find the PM that matches the qualitative and quantitative attributes of the resource request and place the resource request on the selected PM. For a certain VM request, there are usually multiple PMs that meet the requirements. At the same time, for a certain PM, there are usually multiple VM requests competing for use. Therefore, we need to consider how to better place these VM requests so that the placement is completed as quickly as possible, and we can avoid the use of low-utilization PMs as much as possible during first placement, so as to reduce the workload of VNS.

In order to place customer requests (especially those from vip customers) on the PM as quickly and as much as possible. First, the vip and retail users are sorted in positive order according to the size of their requests, and then the PMs are sorted from hot to cold according to the lowest utilization rate of one of the many resources. Finally, the sequenced requests are placed one by one on the hottest PMs that match them to ensure that more requests are placed at a given PM resource level. The above description is a scheme of FP. The one-stage process of FreeVM designed by us can be compatible with various VMP algorithms. The target of first placement is to optimize a batch of alternative PMs, thus reducing the search space for VNS.

*2) Variable neighborhood searching:* After first placement is completed, the resource utilization of the entire cluster is first evaluated. When the average resource utilization of the PMs exceeds a certain threshold, the VNS algorithm is triggered to adjust the migration of the already placed VMs on the PMs, freeing up the low-utilization PMs to reduce energy consumption, while making the remaining resources on the PMs more balanced to improve the overall resource utilization of the cluster.

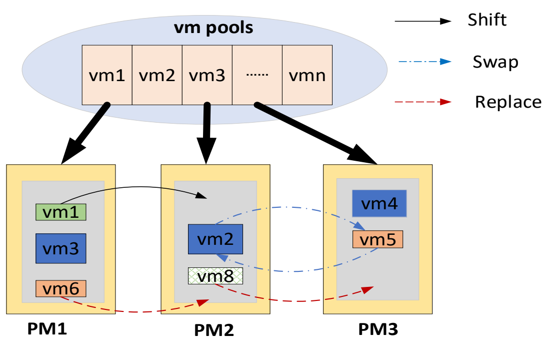


Fig. 1. The instance of scheduling with n vms

According to the instance of scheduling shown in Figure 1, the PM set associated with three objective PMs are selected. The process of reallocation of VMs is to migrate as many VMs as possible from the ”cold” PMs to the ”hot” PMs, which frees up the ”cold” PMs and thus saves energy. Our proposed VNS algorithm is used for VMs migration, and the VNS search process involves three schemes: shift, swap, and replace. As shown in Figure 1, shift means reallocating a VM from one PM to another; swap means swapping two VMs on different PMs; replace means attempting to move a zombie VM8 from PM2 to PM3 and then reallocating VM6 from PM1 to PM2.

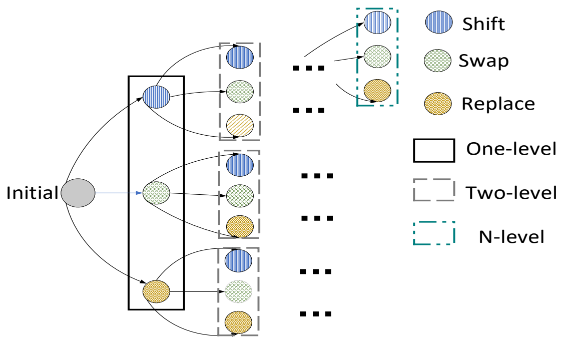


Fig. 2. The process of variable neighborhood search

The VNS is based on a two-level search algorithm that attempts to empty low utilization PMs one by one from ”cold” to ”hot”, with all VMs on every PM moving out of the source PM at the minimum cost. When the total overhead of emptying all VMs is less than the total gain, the emptying operation is performed, otherwise the emptying is abandoned. The one-level search is designed to handle cold PMs, i.e., migrate VMs on a ”cold” PM to a ”hot” PM. However, the secondary search is to further compress the hot PMs. Therefore, VNS is designed to continue the second-level search on top of the first-level search and eventually find the optimal solution inside the two-level search neighborhood.

VNS is an improved local search algorithm, as shown in Figure 2, which first defines N levels of neighborhood for the initial solution; then uses the neighborhood structure (e.g., one-level) to search until a local optimal solution is found. The

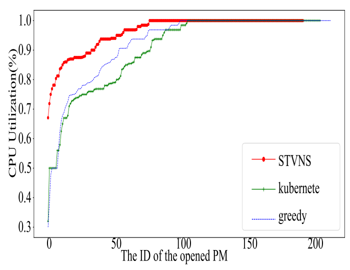
**algorithm 1** VNS Algorithm

```
Input: First-Placement scheme s
Output: Best found solution
 1: $best \leftarrow s$
 2: for each neighborhood $N_i(s)$ of solution s do
 3:     $s' \leftarrow$ randomly choose a new solution in $N_i(s)$
 4:     for each neighborhood $N_j(s')$ of solution $s'$ do
 5:         $s'' = FindBest(N_j(s'))$
 6:         if $Cost(s'') < Cost(s')$ then
 7:             $s' = s''$
 8:             break
 9:         else
10:             $j = j + 1$
11:         end if
12:     end for
13:     if $Cost(s') < Cost(s)$ then
14:         $best \leftarrow s'$
15:     end if
16: end for
17: return best
```

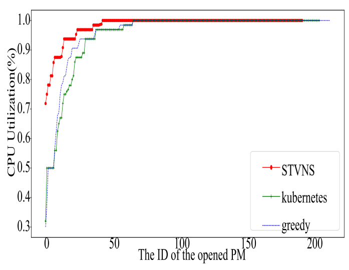
variable neighborhood search algorithm we designed consists of a two-level search to find the optimal solution in three branches, and then take the optimal one of them, each branch is a two-level search process. There are three move methods that can be used for neighborhood searching, which are shift, swap, and replace. The first level of the two-level search algorithm is a random move of the current VMs starting with the current placement scheme, which explores all three move methods. The second level of search makes another move based on the results of the first level of search, which is not random, but identifies the solution that has the minimum cost of each possible movement scheme and that makes the total cost of the two-level search smaller than the cost of the first level of search. Once such a solution is found in the current move, it is returned directly as the optimal solution for this branch, without any further consideration of the remaining moves.

The pseudo-code of two-level is shown in Algorithm 1. Line 2,3 represent a random selection of placement scheme $s'$ in one of the neighbors of the current placement state s (e.g., shift). From line 4,5 indicates doing a secondary search on top of the primary search. Find the local optimal solution $s''$ in a kind of neighborhood of $s'$ (e.g., shift) for that neighborhood. Where lines 6-11 explore the optimal solution, and if $s''$ is better than $s'$, then $s'$ is returned as the optimal solution of this branch; otherwise the other neighborhoods of $s'$ are searched for a solution better than $s'$. Lines 13-15 show that if there is no better solution than $s'$, then $s'$ is returned as the optimal solution of this branch.

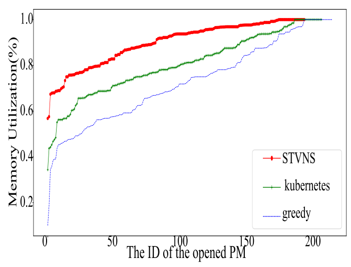
*3) PM start-stop prediction:* At the end of the VNS algorithm, a number of PMs that were initially underutilized may be vacated, and the start-stop policy makes a decision about whether to deactivate some of these PMs or to start more of them.



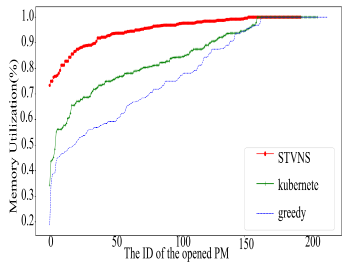
(a) $\lambda = 100$;Proportion1(Scene 1)



(b) $\lambda = 100$;Proportion2(Scene 2)

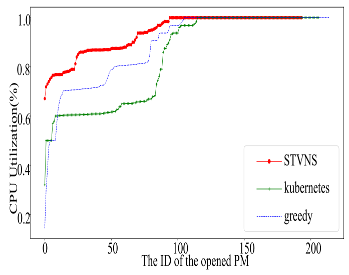


(c) $\lambda = 100$;Proportion1(Scene 1)

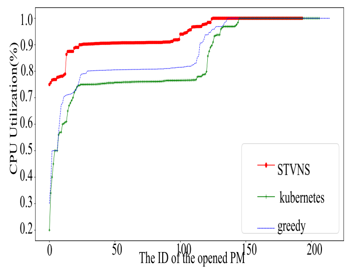


(d) $\lambda = 100$;Proportion2(Scene 2)

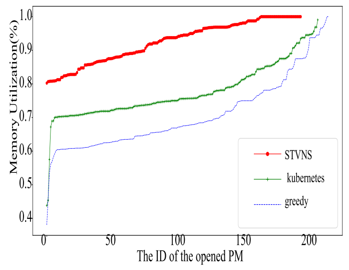
Fig. 3. The comparison of resource utilizations of different algorithms ($\lambda = 100$)



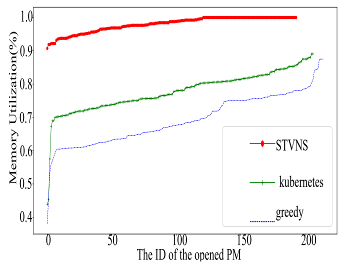
(a) $\lambda = 200$;Proportion1(Scene 3)



(b) $\lambda = 200$;Proportion2(Scene 4)



(c) $\lambda = 200$;Proportion1(Scene 3)



(d) $\lambda = 200$;Proportion2(Scene 4)

Fig. 4. The comparison of resource utilizations of different algorithms ($\lambda = 200$)

First, the VNS algorithm generates a list of the vacated start-up PMs. The start-stop policy then uses a custom Sliding-k algorithm [14] to predict how much resource $R_l$ will be released on the PM at the end of the current cycle and how much of the customer's resourcerequest $R_q$ will be reached in the next cycle. Assuming that the total resource of the current remaining PMs is $R_m$. If $(R_m + R_l)$ is greater than $R_q$, it indicates that the current machine resource can satisfy the request in the next cycle. Some of the PMs in the list are selected for shutdown, and the total resources of these PMs should be less than $(R_q - R_m - R_l)$. Once $(R_m + R_l)$ is less than $R_q$, it indicates that the current machine resources are not available for the next cycle, that it is not possible to shut down some of the PMs from the vacated PMs, and that some of the PMs need to be turned on from the deactivated PMs to satisfy the request of the next cycle.

## V. Experimentation and Results

This section describes our experimentation. Since FreeVM is essentially an algorithm for allocation, our testbed is built in python for efficiency. The testbed is associated with a mysql database in order to visualize the placement and migration of the VMs and to facilitate access to the data. At the same time, FreeVM has a series of external interfaces that can be interfaced with any data center cluster system.

We have conducted extensive experiments with variant parameter settings to evaluate FreeVM from different aspects: 1. the advantages of FreeVM; 2. the comparison results between FreeVM and the other algorithms; 3. the algorithm under different parameter settings.

### *A. Experimental Setup*

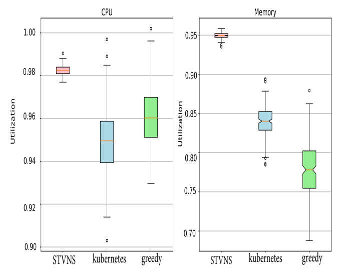
FreeVM was implemented in IDC with 1000 PMs. then, the scenarios with different granularity of user requests were evaluated experimentally. A batch of requests arrives every 5 minutes, the number of requests satisfies a Poisson distribution with mean $\lambda$, and the duration of the requests satisfies a Poisson distribution with a mean of 5 days. Where each request and PM contains both qualitative and quantitative attributes, and the qualitative attribute is the trait attribute. We consider two attributes (trait1: A/a, trait2: B/b), which need to be matched by qualitative attributes for resource allocation; quantitative attributes are specifications, and we consider two resources (CPU: 32/64/128 cores, Memory: 128/256 G). As shown below, Tables II and III represent the PM specification and the VM specification, respectively.

TABLE II
PM SPECIFICATION

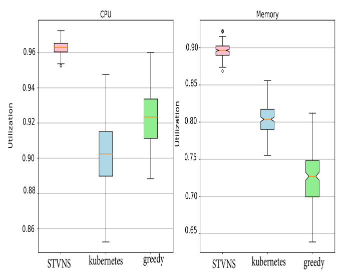
| **ID** | **CPU** | **Memory** | **Ratio** |
|---|---|---|---|
| 1 | 32 | 28 | 0.1 |
| 2 | 64 | 128 | 0.4 |
| 2 | 64 | 256 | 0.4 |
| 4 | 128 | 256 | 0.2 |

### *B. Results Analysis*

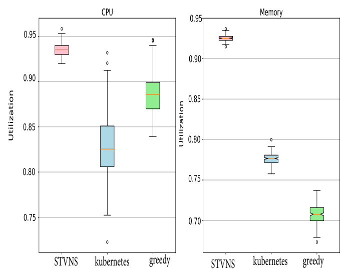
By simulating the arrival of user requests for 3 days, Figure 3 and Figure 4 show the CPU of the enabled PM and the resource utilization of memory. When $\lambda = 100$, as illustrated in Figure 3, taking CPU as an example, there are about 165 enabled PMs whose CPU utilizations exceed 5% under the



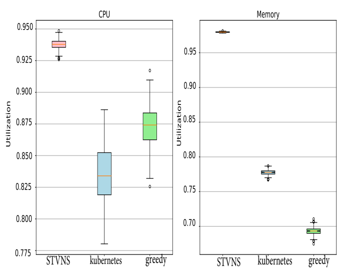
(a) $\lambda = 100$;Proportion1(Scene 1)



(b) $\lambda = 100$;Proportion2(Scene 2)



(c) $\lambda = 200$;Proportion1(Scene 3)



(d) $\lambda = 200$;Proportion2(Scene 4)

Fig. 5. Comparing the balance of different algorithms

TABLE III
VIRTUAL MACHINE SPECIFICATIONS

| CPU | Proportion1 | Proportion2 |
|---|---|---|
| 2 | 0.1 | 0.4 |
| 4 | 0.2 | 0.3 |
| 8 | 0.4 | 0.2 |
| 2 | 0.2 | 0.07 |
| 32 | 0.1 | 0.03 |

greedy algorithm and 13% under the kubernetes algorithm. In addition, we find that Scenario 2 converges faster than Scenario 1, due to the fact that Scenario 2 is a fine-grained user request that makes full use of the PM fragment space. When analyzing the convergence speed of the memory, STVNS is 14.84% faster than kubernetes and 9.7% faster than greedy. Comparing Figure 3 and Figure 4 at $\lambda = 200$, we see a significant increase in memory utilization due to the more intensive user requests arriving at $\lambda = 200$ and the ability to leverage the PM's free resources. Figure 5 reflects the balance of resource utilization. For example, in scenario 1, STVNS is 7.5% better than kubernetes and 15% better than greedy in analyzing the balance of CPU and memory utilization.

TABLE IV
COMPARISON OF THE NUMBER OF PMS OPENED

| Cycle | STVNS | kubernetes | greedy |
|---|---|---|---|
| 200 | 75 | 85 | 80 |
| 400 | 103 | 106 | 110 |
| 600 | 139 | 147 | 161 |
| 800 | 142 | 159 | 174 |
| 1000 | 195 | 207 | 215 |

Table IV compares the mean values of the number of PMs started under different algorithms. As the lifecycle of the request ends, reallocation takes advantage and vacates PMs. From this, we can infer that the STVNS algorithm is better at saving energy as the time increments.

## VI. CONCLUTION

In meeting the challenge of surging demand for cloud services, it is clear that simply scaling up the data center is not cost-effective, so this paper proposes a two-stage approach to PMs resource management in the data center, which ultimately enables rapid distribution of cyclically arriving customer requests through first placement, secondary scheduling, and application of start/stop strategies. By redistributing VMs, fragmentation in PMs is reduced, PM resource utilization is improved, and ultimately energy consumption is minimized as much as possible. In addition, STVNS algorithms can be used as a complement to other container placement methods as an optimized component of data center resource management.

## REFERENCES

[1] Zhang, Yuchao, et al. "BDS: a centralized near-optimal overlay network for inter-datacenter data replication." Proceedings of the Thirteenth EuroSys Conference. 2018.
[2] Sarwesh, P., N. Shekar V. Shet, and K. Chandrasekaran. "Effective integration of reliable routing mechanism and energy efficient node placement technique for low power IoT networks." International Journal of Grid and High Performance Computing (IJGHPC) 9.4 (2017): 16-35.
[3] Masdari, Mohammad, Sayyid Shahab Nabavi, and Vafa Ahmadi. "An overview of virtual machine placement schemes in cloud computing." Journal of Network and Computer Applications 66 (2016): 106-127.
[4] Wolke, Andreas, et al. "More than bin packing: Dynamic resource allocation strategies in cloud data centers." Information Systems 52 (2015): 83-95.
[5] Mosa, Abdelkhalik, and Norman W. Paton. "Optimizing virtual machine placement for energy and SLA in clouds using utility functions." Journal of Cloud Computing 5.1 (2016): 17.
[6] Kanagavelu, Renuga, et al. "Virtual machine placement with two-path traffic routing for reduced congestion in data center networks." Computer Communications 53 (2014): 1-12.
[7] Rossi, Fabiana, et al. "Geo-distributed efficient deployment of containers with Kubernetes." Computer Communications (2020).
[8] Alam, ABM Bodrul, et al. "Multi-Objective Interdependent VM Placement Model based on Cloud Reliability Evaluation." ICC 2020-2020 IEEE International Conference on Communications (ICC). IEEE, 2020.
[9] Zhang, Yuchao, et al. "A communication-aware container re-distribution approach for high performance VNFs." 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). IEEE, 2017.
[10] Lv, Liang, et al. "Communication-aware container placement and reassignment in large-scale internet data centers." IEEE Journal on Selected Areas in Communications 37.3 (2019): 540-555.
[11] Canali, Claudia, et al. "Joint minimization of the energy costs from computing, data transmission, and migrations in cloud data centers." IEEE Transactions on Green Communications and Networking 2.2 (2018): 580-595.
[12] Khattar, Nagma, Jaiteg Singh, and Jagpreet Sidhu. "An Energy Efficient and Adaptive Threshold VM Consolidation Framework for Cloud Environment." Wireless Personal Communications (2020): 1-19.
[13] Ch, Sudhakar, and T. Ramesh. "Energy efficient VM scheduling and routing in multi-tenant cloud data center." Sustainable Computing: Informatics and Systems 22 (2019): 139-151.
[14] Ran Wang, Yuchao Zhang, Wendong Wang. An Inter-DC Scheduling System based on Dynamic Traffic Prediction. China Information and Communication Conference (CICC) 2019. 29 Nov. - 1 Dec. 2019. Chengdu China. (in Chinese)