Prepartition: Load Balancing Approach for Virtual Machine Reservations in a Cloud Data Center

Wenhong Tian[a], Minxian Xu[b,*], Wenxia Guo[a], Kui Wu[c], Rajkumar Buyya[a,d]

[a] *School of Computer Science and Software Engineering University of Electronic Science and Technology of China (UESTC) Email: tian_wenhong@uestc.edu.cn*

[b] *Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China.*

[c] *Department of Computer Science, University of Victoria, Canada.*

[d] *CLOUDS Lab, School of Computing and Information Systems, The University of Melbourne, Australia; R. Buyya is also affiliated with UESTC as a Visiting Professor*

Abstract

Load balancing is critical for the efficient operation of cloud data centers. With virtualization, reactive (post) migration of virtual machines after allocation is the traditional way for load balance and traffic consolidation. Reactive migration, however, has difficulty in reaching predefined load balance objectives and may cause services interruption and instability. In view of this, we propose a new approach, called Prepartition, for load balancing. It partitions a VM request into a few sub-requests sequentially with start-times, end-times and capacity demands, and treats each sub-request as a regular VM request. In this way, it can proactively set a process-time bound for each VM request on each physical machine and makes the scheduler get ready before VM migration to achieve the predefined load balance goal. Simulation with real-world trace and synthetic data shows that Prepartition for offline (PrepartitionOff) scheduling has $10\% - 20\%$ better performance than the well-known load balancing algorithms with regard to several metrics, including average utilization, imbalance degree, makespan and Capacity_makespan. We also apply Prepartition to online load balance and compare it with existing online scheduling algorithms. Evaluation results show that our proposed approach can also achieve better performance than exiting online algorithms.

*Corresponding author

Keywords: Cloud Computing, Physical Machines, Virtual Machines, Reservation Model, Load Balancing, Prepartition

1. Introduction

Cloud data centers have become the foundation for modern IT services, ranging from general-purpose web services to many critical applications such as online banking and health systems. The service operator of a cloud data center is always faced with a difficult tradeoff between high performance and low operational cost [1]. On the one hand, to maintain high-quality services, a data center is usually over-engineered to be capable of handling peak workload. Such up-bound configuration can bring high expenses on maintenance and energy as well as low utilization to data centers [2]. On the other hand, to reduce cost, the data center needs to increase server utilization and shut down idle servers [3]. The key tuning knob in making the above tradeoff is datacenter load balancing.

Due to the importance of data center load balancing, tremendous research and development have been devoted to this domain in the past decades [4]. Yet, load balancing for cloud data centers is still one of the prominent challenges that need more attention. The difficulty is compounded by several issues such as virtual machine (VM) migration, service availability, algorithm complexity, and resource utilization. The complexity in cloud data center load balancing has actually fostered a new industry dedicating to offer load balance services [5].

Ignoring the subtle differences in detailed implementation of load balancing, let us first have a high-level view on how cloud data centers perform resource scheduling and load balancing. The process is illustrated in Fig. 1, which includes the following major steps:

1. User's request initialization: user initiates a VM request through the Internet.
2. Finding suitable resources: based on the user's feature (such as geographic location, quantity and quality requirements), the scheduling center submits the VM request to an appropriate data center, in which the management program submits the request to a scheduling domain. In the scheduling domain, a scheduling algorithm is performed and resource is allocated to the request.

3. Sending feedbacks (e.g., whether or not the request has been satisfied) to users.
4. Scheduling tasks: determine when a VM should run on which physical machine (PM).
5. Updating/Optimization: the scheduling center executes optimization in the back-end and makes decisions (e.g., VM migration) for load balancing.

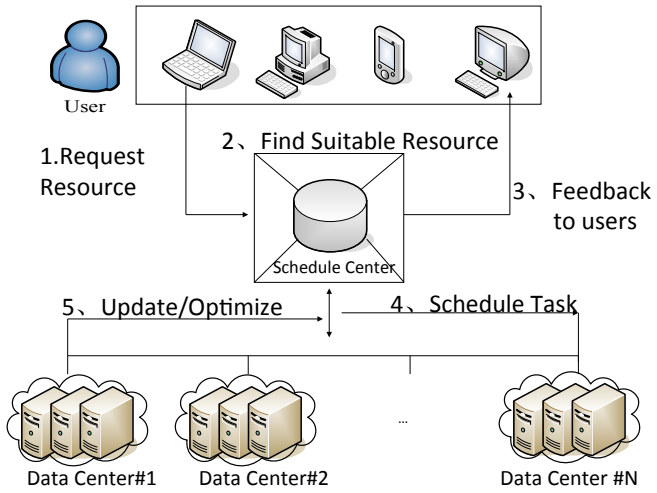


Figure 1: A high-level view on resource scheduling/load balancing in Cloud data centers.

In the above process, most existing work on load balancing is reactive, i.e., performing load balancing with VM migration when unbalancing or other exceptional things happen *after* VM deployment. Reactive migration of VMs is one of the practical methods for load balancing and traffic consolidation such as in VMWare. Nevertheless, it is well known that reactive VM migration has difficulty to reach predefined load balance objectives and may cause services interruption and instability. Our observation is that if load balancing is considered as one of the key criteria *before* VM allocation, we should not only reduce the frequency of (post) VM migration (thus less service interruption), but also reach better balanced VM allocation among different PMs.

Motivated by the above observation, we propose a new load balancing approach called Prepartition. By combining interval scheduling and lifecycles characteristics of both VMs and PMs, Prepartition handles the problem of

load balancing from a different angle. Starkly different with previous methods, it handles the VM load balance in a more proactive way.

The novelty of Prepartition is that it proactively sets a process-time bound (as per Capacity_makespan defined in Section 2) by *pre-partitioning* each VM request and therefore helps the scheduler get ready before VM migration to achieve the predefined load balancing goal. Pre-partitioning here means that a VM request may be partitioned into a few sub-requests sequentially with start-times, end-times and capacity demands, where the scheduler treats each sub-request as a regular VM request and may allocate the sub-requests to different PMs[1]. In this way, the scheduler can prepare in advance, without waiting for the VM migration signals as in traditional VM allocation/migration schemes.

To the best of our knowledge, we are the first to introduce the concept of pre-partitioning VM requests to achieve better load balancing performance in cloud data centers. The key contributions of this paper include:

- Proposing a modeling approach to schedule VM reservation with sharing capacity by combining interval scheduling and lifecycles characteristics of both VMs and PMs.
- Designing novel Prepartition algorithms for both offline and online scheduling which can prepare migration in advance and set process time bound for each VM on a PM.
- Deriving computational complexity and quality analysis for both offline and online Prepartition.
- Carrying out performance evaluation in terms of average utilization, imbalance degree, makespan, time costs as well as Capacity_makespan (a metric to represent loads, more details will be given in Section II) by simulating different algorithms using trace-driven and synthetic data.

The rest of this paper is organized as follows: Section 2 introduces problem formulation. Section 3 presents Prepartition in details for both offline and

[1]Note that in practice we need to copy data and running state information from a VM (corresponding to a sub-request) to another VM (corresponding to the next sub-request), i.e., the operations for VM migration. But since the scheduler knows information of all sub-requests, it can prepare early so that the VM state/data transition can be finished smoothly. The implementation detail is beyond the focus of this paper.

online algorithms. Performance evaluations of different scheduling algorithms are shown in Section 4. Section 6 discusses the related work on load balancing in cloud data centers. Finally, we conclude the paper in Section 6.

2. Problem description and formulation

VMs Reservation is considered here and the VM allocations are modeled by a modified interval scheduling problem (MISP) with fixed processing time. Details on traditional interval scheduling problems with fixed processing time were introduced in [6] and references therein. In the followings, a general formulation of the modified interval-scheduling problem is introduced and evaluated against some known algorithms.

2.1. Assumptions

The key assumptions are:

1) The time is given in slotted windows; all data is given deterministically. The total time period [0, T] is partitioned into equal length (sl_0), and the total number of slots is then t=T/sl_0. The start time s_i and finish time f_i are integer numbers of one slot. Then the interval of a demand can be expressed in slot fashion with (start-time, finish-time). For instance, if sl_0=5 minutes, an interval (3, 10) represents that it starts at the 3rd time slot and finish at the 10th time slot. The duration of this demand is (10-3)$\times$5=35 minutes.

2) For all VM reservations, there are no precedence constraints other than those implied by the start-time and finish-time.

3) For each request, the required capacity is a positive real number between (0,1]. Notice that the capacity of a single physical machine is normalized to be 1 and the required capacity of a VM can be 1/8, 1/4 or 1/2 or other portions of the total capacity of a PM. This is consistent with applications in Amazon EC2 [7] and [8].

2.2. Key Definitions

A few basic definitions are given as follows:

Definition 1. *Traditional interval scheduling problem (TISP) with fixed processing time*: A set of demands {1, 2,. . ., n} where the i-th demand refers

to an interval of time starting at s_i and finishing at f_i, each one requires a capacity of 100%, i.e. utilizing the full capacity of a server during that interval.

Definition 2. *Interval scheduling with capacity sharing (ISWCS)*: Difference from TISP, ISWCS can share the capacities among demands if the total capacity of all demands allocated on the single server at any time is still not fully utilized.

Definition 3. *Sharing compatible intervals for ISWCS, for short, SCI-ISWCS*: A subset of intervals with a total requested capacity below the total capacity of a PM at any time can share the capacity of a PM. Compared against ISWCS, the requests in SCI-ISWCS can be modelled as the ones with lifecycles, which can be represented as sharing the subset of intervals.

In the existing literature, *makespan*, i.e., the maximum total load (processing time) on any machine, is applied to measure load balance. Therefore, the makespan is the total length of the schedule.

In this paper, we aim to solve the problem based on ISWCS manner, and apply a new metric Capacity_makespan.

Definition 4. *The Capacity_makespan of a PM i*: In any allocation of VM requests to PMs, let $A(i)$ represent the set of VM demands allocated to PM_i. With this allocation, PM_i will have total load as the sum of the product of each requested capacity and its duration, called Capacity_makespan, abbreviated as CM, as follows:

$$CM_i = \sum_{j \in A(i)} d_j t_j \tag{1}$$

in which d_j is the capacity demand (some portion of total capacity) of VM_j from a PM where the capacity can be CPU or Memory or storage in this paper, it can be simplified as a capacity based on Assumption 3), and t_j for the span of demand j, being the length of processing time of demand j. Similarly, the Capacity_makespan of a given VM request is simply the product of the requested capacity and its duration.

2.3. Optimization Objective

Then, the objective of load balancing is to achieve the minimization of the maximum load (Capacity_makespan) on all PMs as noted in Eq. (2). A few related metrics such as average utilization and makespan are also applied

in the followings. Considering m PMs are in the data center, we can form the problem as:

$$\underset{1\leq i\leq m}{Min}\ CM_i \tag{2}$$

$$\text{subject to } \forall\ slot\ s,\ \sum_{j\in A(i)} d_j \leq 1 \tag{3}$$

$$\forall\ i, s_i < f_i \tag{4}$$

in which d_j is the capacity demand of VM j and the total capacity of a PM i is normalized to 1. The condition (3) shows the sharing capacity constraint, (4) is the constraint that finished time f_i should be late than the start time s_i.

From this form, we see that lifecycle and capacity sharing are two major differences from traditional metrics such as makespan which only considers process time. Traditionally Longest Process Time first (LPT) [9] is widely applied to load balance of offline multi-processor scheduling. Reactive (post) migration of VMs is another way for load-balancing. *However, reactive migration is difficult to obtain predefined load balance goals and may bring interruption and instability to services and other associated costs [10].* So we present a new paradigm called Prepartition for load balance in Cloud data centers.

2.4. Metrics for ISWCS load balancing

A few key metrics for ISWCS load balance problem will be given in the followings. Other metrics are the same as given in [11].

1) PM resources:

$PM_i(i, PCPU_i,\ PMem_i,\ PSto_i)$, i is the index number of PM, $PCPU_i$, $PMem_i$, $PSto_i$ are the CPU, memory, storage capacity of that a PM can offer.

2) VM resources:

$VM_j(j, VCPU_j, VMem_j, VSto_j, T_j^{start}, T_j^{end})$, j is the VM type ID, $VCPU_j$, $VMem_j$, $VSto_j$ are the CPU, memory, storage demands of VM_j, T_j^{start}, T_j^{end} are the start time and finish time, showing the lifecycle of a VM.

3) Time slots: Considering a time span from 0 to T be divided into equal length of slots. The s slots can be considered as $[(t_0, t_1), (t_1, t_2), \ldots, (t_{s-1}, t_s)]$, each time slot T_r represents the time span (t_{r-1}, t_r).

4) Average CPU utilization of PM_i during slot 0 and T_s is defined as:

$$PCPU_i^U = \frac{\sum_{r=0}^{s}(PCPU_i^{T_r} \times T_r)}{\sum_{r=0}^{s} T_r} \tag{5}$$

where $PCPU_i^{T_r}$ is the average CPU utilization monitored and computed in slot T_r which may be a few minutes long, and $PCPU_i^{T_r}$ can be obtained by monitoring CPU utilization in slot T_r. Average memory utilization ($PMem_i^U$) and storage utilization ($PSto_i^U$) of PMs can be calculated similarly. In the same way, average CPU (memory and storage) utilization of a VM can be calculated.
5) Makespan: represents the whole length of a schedule for a set of VM reservations, i. e., the difference between the start-time of the first job [2] and the finishing time of the last job.
6) The maximum Capacity_makespan (CM^p) of all PMs: is calculated as :

$$CM^p = \max_i (CM_i) \tag{6}$$

where CPU, memory and storage utilization can be applied.
7) Imbalance degree (IMD): The variance is widely used as a measure of how far a set values are spread out from each other in statistics. IMD is the normalized variance (regarding to its average) of CPU, memory and storage utilization for all PMs and it measures load imbalance effect and is defined as:

$$\frac{\sum_{i=0}^{m}(\frac{(Avg_i - CPU_u)^2}{3} + \frac{(Avg_i - Mem_u)^2}{3} + \frac{(Avg_i - Sto_u)^2}{3})}{m} \tag{7}$$

where

$$Avg_i = \frac{PCPU_i^U + PMem_i^U + PSto_i^U}{3} \tag{8}$$

and CPU_u, Mem_u,Sto_u are respectively the average utilization of CPU, memory and storage in a Cloud data center during consideration and can be computed using utilization of all PMs in a Cloud data center.

[2] in this paper, we interchange demands and requests, both of them are referred to VM requests

Theorem 1. The offline scheduling problem of minimizing the makespan is NP-hard.

The proof was provided in our previous work [11] and is omitted here. Our model in this paper differs from the the previous one in several perspectives: 1). we model that the multiple VM requests are allowed to be executed on the same host simultaneously rather than a single VM request; 2) Our objective is minimizing the Capacity_makespan rather than the longest processing time; 3) we extend our model to be suitable for both offline and online scenarios.

Combining the properties of both fixed process time intervals and capacity sharing, we present new offline and online algorithms in the following section.

3. Prepartition Algorithm

In this section, we propose one algorithm for the offline scenario and two algorithms for the online scenario.

3.1. PrepartitionOff Algorithm

Considering a set of VM reservations, m PMs are in a data center and OPT is used as the optimal solution with regard to minimizing the Capacity_makespan. Firstly define

$$P_0 = \frac{1}{m}\sum_{j=1}^{J} CM_j \leq OPT \tag{9}$$

where P_0 denotes the lower bound for the OPT. Algorithm 1 gives the pseudocodes of PrepartitionOff algorithm which measures the ideal load balancing among m PMs. The algorithm firstly calculates balance value by formula (10), sets a partition value (k) and computes the length of each partition (i.e. $\lceil P_0/k \rceil$, representing the max CM a VM can continuously be allocated on a PM). For every demand, PrepartitionOff divides it into multiple $\lceil P_0/k \rceil$ subintervals when its CM is equal or larger than P_0, and then finds a PM with the lowest average Capacity_makespan and available capacity to allocate, and updates the load on each PM. The algorithm computes the Capacity_makespan of each PM when all requests are assigned and finds total partition (migration) numbers. For practice, the scheduler needs to record all possible subintervals and their hosting PMs of each request so that migrations of VMs can be prepared in advance to reduce overheads.

Theorem 2. The computational complexity of PrepartitionOff algorithm is $O(nlogm)$ using priority queue data structure where n is the number

Input: Total number of PMs m, total number of VM requests n, requests are indicated by their (demanded VM IDs, start times, finishing times, demanded capacity), CM_j is the Capacity_makespan of request j, CM_i for the Capacity_makespan of PM i

Output: Assign PM IDs to all requests and their partitions

Initialization: computing the bound value P_0 and partition value k (e.g. 1, 2, ...);
forall i from 1 to m **do**
  if $CM_i \geq P_0$ **then**
    divide it by $\lceil P_0/k \rceil$ subintervals equally and treat each subinterval as a new request
  end
end
All intervals are sorted in decreasing order of CM, break ties arbitrarily;
Let $I_1, I_2, \ldots, I_t$ represent the intervals;
forall j from 1 to n **do**
  Pick up the VM with the earliest start time in the VM queue for execution;
  Allocate j to the PM with the smallest load and enough capacity;
  Update load (CM) of the PM;
end
Calculate CM of every PM and total number of partitions

Algorithm 1: The pseudo codes of PrepartitionOff algorithm

of VM requests after pre-partition and m is the total number of PMs used.
Proof: The priority queue is designed such that each element (PM) has a priority value (average Capacity_makespan), and each time the algorithm needs to select an element from it, the algorithm takes the one with the highest priority. It costs $O(n)$ time to sort n elements, and $O(logn)$ steps for insertion and the extraction of minima in a priority queue (detailed proof of the priority queue is given in [6]). Then, by using priority queue, the algorithm picks a PM with the lowest average Capacity_makespan in $O(logm)$ time. In total, PrepartitionOff algorithm has time complexity $O(nlogm)$ for n demands.

Theorem 3. The approximation ratio of PrepartitionOff algorithm is

$(1+\epsilon)$ regarding the capacity_makespan where ϵ=$\frac{1}{k}$ and k is the partition value (a preset constant).

Proof: It can be seen that every demand has bounded Capacity_makespan by Preparition applying the lower bound P_0. Every request has start-time s_i, finish-time f_i and process time p_i=f_i-s_i. Think the last job to finish (after scheduling all other jobs) and assume this job starts at time T_0. All other servers are fully utilized and denote the maximum Capacity_makespan as CM_m, this means $CM_m \leq$OPT. Since, for all requests $i \in J$, we have $CM_i \leq \epsilon$OPT (by the setting of PrepartitionOff algorithm in formula (9)), this job finishes with load (CM_m+ϵOPT). Therefore, the schedule with Capacity_makespan (CM_m+ϵOPT) $\leq$ (1+ϵ)OPT, this ends the proof.

3.2. PrepartitionOn1 Algorithm

For online VM allocations, scheduling decisions must be made without complete information about the entire job instances because jobs arrive one by one. We firstly extend the offline Prepartition algorithm to the online scenario as PrepartitionOn1.

Given m PMs and L VMs (including the one just came) in a data center. Firstly define

$$B_d = min(\max_{1\leq j\leq L}(CM_j)/2, \sum_{j=1}^{L}(CM_j)/m) \tag{10}$$

B_d is called dynamic balance value, which is one half of the max Capacity_makespan of all current PMs or the ideal load balance value of all current PMs in the system, where L is the number of VMs requests already arrived. Notice that the reason to set B_d as one half of the max Capacity_makespan of all current PMs is to avoid that large number of migrations may cause instability in some cases.

Algorithm 3.2 shows the pseudo codes of PrepartitionOn1 algorithm. Since in online algorithm, the requests come one by one, the system can only capture the information of arrived requests. When a new request comes into the system, the algorithm computes dynamic balance value by equation (10). To be noticed, L represents the number of requests already arrived, and m represents the number of PMs in use. After the dynamically balanced value (B_d) is computed, then the initial request is partitioned into

Input: VM requests come one by one given by (demanded VM IDs, start times, finishing times, demanded capacity), CM_j is the Capacity_makespan of demand j

Output: Allocate a PM ID to every demand and its partitions

set the partition value k, total partition number P=0;
for each arrived job j **do**
    Pick up the VM with the earliest start time in the VM queue to schedule;
    Compute CM_j of VM j, and B_d using Eq. (10);
    if $CM_j > \lceil (B_d/k) \rceil$ **then**
        partition VM_j into multiple $\lceil (B_d/k) \rceil$ equal subintervals, treat each subinterval as a new demand and add them into VM queue, $P = P + \lceil \frac{CM_j}{B_d/k} \rceil$, Update load ($CM$) of the PM;
    end
    else
        Allocate j to PM with the smallest load and enough capacity;
        Update load (CM) of the PM;
    end
end
Output total number of partitions P

Algorithm 2: PrepartitionOn1 Algorithm

several requests (segments) based on the partition value k. In these partitioned requests, the first one would be executed when its start-time begins, which will be allocated to the PM with the lowest Capacity_makespan, while others would be put back into the queue waiting to be executed. Then the algorithm picks up the next arrived request to follow the same partition and allocation process. Once all demands are allocated, PrepartitionOn1 calculates the Capacity_makespan of every PM and outputs the total partition numbers for n demands. Since the number of partitions and segments of each VM request are known at the moment of allocation, the system can prepare VM migration in advance so that the process time and instability of migration can be reduced.

Theorem 4. PrepartitionOn1 has a competitive ratio of $(1 + \frac{1}{k} - \frac{1}{mk})$ with regarding to the Capacity_makespan.

Proof: Without loss of generality, we label PMs in order of non-decreasing

final loads (CM) in PrepartitionOn1. Denote OPT and $PrepartitionOn1(I)$ as the optimal load balance value of corresponding offline scheduling and load balance value of PrepartitionOn1 for a given set of jobs I, respectively. Then the load of PM_m defines the Capacity_makespan. The first (m-1) PMs each process a subset of the jobs and then experience an (possibly none) idle period. All PMs together finish a total Capacity_makespan $\sum_{i=1}^{n} CM_i$ during their busy periods. Consider the allocation of the last job j to PM_m. By the scheduling rule of PrepartitionOn1, PM_m had the lowest load at the time of allocation. Hence, any idle period on the first (m-1) PMs cannot be bigger than the Capacity_makespan of the last job allocated on PM_m and hence cannot exceed the maximum Capacity_makespan divided by k (partition value), i.e., $\frac{\max_{1\leq i\leq n} CM_i}{k}$. Based on Equation (10) , then we have

$$m \times PrepartitionOn1(I) \leq \sum_{i=1}^{n}(CM_i) + (m-1)\frac{\max(CM_i)}{k} \tag{11}$$

which is equivalent to

$$PrepartitionOn1(I) \leq \sum_{i=1}^{n}(\frac{CM_i}{m}) + (m-1)\frac{\max(CM_i)}{mk} \tag{12}$$

which is

$$PrepartitionOn1(I) \leq (OPT + (\frac{1}{k} - \frac{1}{mk})OPT) \tag{13}$$

Note that $\frac{\sum_{i=1}^{n} CM_i}{m}$ is the lower bound on $OPT(I)$ because the optimum Capacity_makespan cannot be smaller than the average Capacity_makespan on all PMs. And $OPT(I) \geq \max_{1\leq i\leq n} CM_i$ since the largest job must be processed on a PM. We therefore have $PrepartitionOn1(I) \leq (1 + \frac{1}{k} - \frac{1}{mk})OPT$.

Theorem 5. The computational complexity of PrepartitionOn1 is $O(nlogm)$ using priority queue data structure, where n is the number of VM requests after pre-partition and m is the total number of PMs used.

Proof: The proof is similar to the proof for Theorem 2, therefore, we omit it here.

3.3. PrepartitionOn2 Algorithm

Observing the PrepartitionOn1 may bring too many migrations in some cases by extensive tests, we present PrepartitionOn2 algorithm in this section,

the differences from PrepartitionOn1 are the followings:
1) To avoid a large number of partitions (migrations), we set a constant value f (for instance 0.125,0.25 etc.) for measuring load balancing;
2) Setting a CM bound for each PM, for instance, each PM has a CM=1×24 in each day within 24 hours, but we consider a PM can at most run with 100% CPU utilization in 16 hours, i.e., we set a CM bound for each PM for each day as CM_B=16.

If overloading happens to a PM according to predefined thresholds $(1+f)$ and CM_B, then a new request should be partitioned into multiple x (the number of PMs turned on) subintervals equally and the scheduler allocates each subintervals to every PM.

Theorem 6. PrepartitionOn2 has a computational complexity of $O(nlogm)$ by applying a priority queue, where n is the number of VM requests after pre-partition and m is the total number of PMs used.
Proof: The proof is also similar to the proof for Theorem 2, we therefore omit it.

Theorem 7. The competitive ratio of PrepartitionOn2 is at most $(1+f)$ and each PM has CM at most CM_B with regard to the Capacity_makespan.
Proof: According to Algorithm 3, whenever a PM has CM larger than CM_B or the competitive ratio of the algorithm is larger than $(1+f)$; the allocating VMs will be pre-partitioned into multiple sub-instances and migrated. Therefore the competitive ratio of PrepartitionOn2 is at most $(1+f)$. This completes the proof.

4. Performance Evaluation

Notice that there are three types of PMs in TABLE 2 and 8 types VMs in TABLE 1, where each type of VM occupies 1/16 or 1/8 or 1/4 or 1/2 of the whole capacity of corresponding PM considering all three dimension resources of CPU, memory and storage, therefore the three dimension resources become one dimension in this case. In the future, we will extend to other cases.
In the following, the simulation results of Prepartition algorithms and a few existing algorithms are provided. To conduct simulation, a Java simulator called CloudSched (refer to Tian et al. [12]) is used. Data from both Normal distribution and Hebrew University Experimental Systems Lab (ESL)

Input: VM requests come one by one indicated by their information (required VM type IDs, start times, ending times, requested capacity), CM_j is the Capacity_makespan of request j, CM_B is predefined the CM bound for each PM

Output: Assign a PM ID to each request and its partitions

Initialization: set the CM bound B for each PM, a constant value $f(\leq 0.125)$, total partition number P=0;
for each arrived job j **do**
    Pick up the VM with the earliest start time in the VM queue to schedule;
    Compute CM_j of VM j, and CM of each PM;
    Choose the minimum value CM of PM named CM_{oldmin};
    Suppose to allocate the VM j to the PM which has CM_{oldmin} and compute its' new value of CM named CM_{oldmin^+};
    Get the new minimum value of CM of PM named CM_{newmin};
    if $(CM_{oldmin^+}/CM_{newmin}) > (1+f)$ or $CM_{oldmin^+} > CM_B$ **then**
        partition VM j into multiple x(the number of PM turned on) subintervals equally, allocate each subintervals to every PM, $P = P + x$;
        else
            Allocate j to PM with the lowest load and available capacity;
        end
    end
end
Output total number of partitions P

Algorithm 3: PrepartitionOn2 Algorithm

trace (see [13]) are adopted for simulation, the detailed parameter setting is explained in the following subsection.

All simulations ran on a computer configured with Intel i5 processor at 2.5GHz and 4GB memory. All VM requests are generated by following Normal distribution. In offline algorithm comparisons, Round-Robin (RR) algorithm, Longest Process Time (LPT) algorithm and Post Migration Algorithm (PMG) are implemented.

1) **Round-Robin Algorithm (R-R)**: a load balancing scheduling algorithm

Table 1: 8 types of virtual machines (VMs) in Amazon EC2

Compute Units	Memory	Storage	VM Type
1 units	1.875GB	211.25GB	1-1(1)
4 units	7.5GB	845GB	1-2(2)
8 units	15GB	1690GB	1-3(3)
6.5 units	17.1GB	422.5GB	2-1(4)
13 units	34.2GB	845GB	2-2(5)
26 units	68.4GB	1690GB	2-3(6)
5 units	1.875GB	422.5GB	3-1(7)
20 units	7GB	1690GB	3-2(8)

Table 2: 3 types of physical machines (PMs) suggested

PM Pool Type	Compute Units	Memory	Storage
Type 1	16 units	30GB	3380GB
Type 2	52 units	136.8GB	3380GB
Type 3	40 units	14GB	3380GB

by allocating the VM demands in turn to each PM that can offer demanded resources.

2) **Longest Processing Time first (LPT)**: LPT is one of the best practices for offline scheduling algorithms without migration, which has approximate ratio as 4/3. All the VM demands are sorted by processing time in decreasing order firstly. Then demands are allocated in that order to the PM with the smallest load. The smallest load indicates the smallest Capacity_makespan of all PMs.

3) **Post Migration algorithm (PMG)**: PMG algorithm comes from the VMware DRS algorithm [14], which adopts migrations to achieve load balance regarding to makespan. In the beginning, it allocates the demands the same way as LPT does. Here we replace makespan by Capacity_makespan. Then the average Capacity_makespan of all demands is computed. The up-threshold and low-threshold of the Capacity_makespan for the post migration are computed using the average Capacity_makespan multiplied by a factor (here we set the factor as 0.1, so the up-threshold is average Capacity_makespan multiplied by 1.1 and the low-threshold is multiplied by 0.9). The factor can

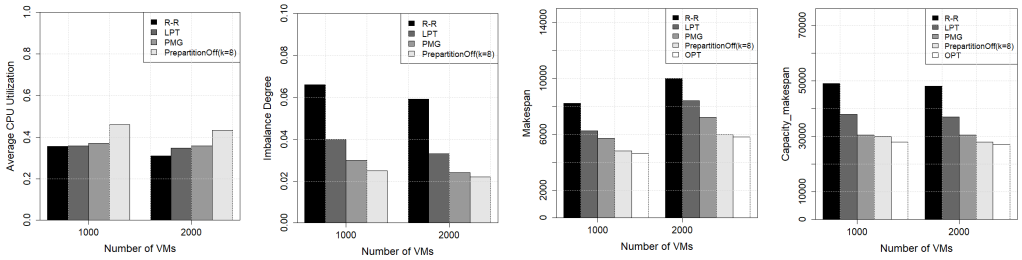


Figure 2: The offline algorithm comparison of (a) average utilization with ESL trace; (b) imbalance degree with ESL trace; (c) makespan with ESL trace; (d) Capacity_makespan with ESL trace

be set dynamically to meet different applications. A migration list is built by having the VMs taken from PMs with Capacity_makespan higher than the low-threshold. The VMs would be migrated from a PM only if the operation would not lead the Capacity_makespan of the PM to be smaller than the low threshold. After that, the VMs in the migration list would be re-allocated to a PM with Capacity_makespan smaller than the up-threshold. The VMs would be migrated to a new PM only if the operation would not cause the Capacity_makespan of the PM to be larger than the up-threshold. There may be still some VMs left in the list, finally the algorithm distributes the left VMs to the PMs with the smallest Capacity_makespan until the list is empty.

VMs and PMs have the same configuration with Amazon EC2. The configurations are shown in Table 1 and Table 2. Note that one compute unit (CU) is equivalent to 1.0 - 1.2 GHz 2007 Opteron or 2007 Xeon processors [7].

Remarks: We adopted the typical recommended VM types suggested by Amazon EC2. EC2 has a variety of VM types, and it classifies them as General Purpose, Compute Optimized (computational intensive VMs), Memory Optimized (memory intensive VMs), Storage Optimized (storage intensive VMs). Although we adopted EC2 classification, our approach can still be extended to other classifications.

Observation 1. Although PMG is one of the best heuristic methods for load balancing, it does not assure a bounded or predefined load balance target. This will be validated in the following performance evaluation section.

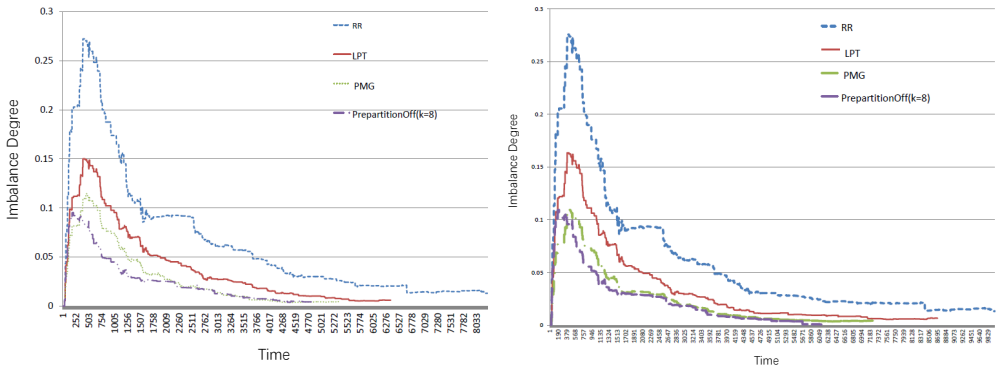
4.0.1. ***Replay with ESL Data Trace***

To reflect realistic data generation, we use the log data at Experimental System Lab(ESL) [13] which has been used frequently for realistic data. The log with monthly records collected by Linux cluster has characteristics consistent with our model. In the log file, each line includes 18 elements where only the requested-ID, start-time, duration and the number of processors (capacity demands) are needed in our simulation. Because the time slot length mentioned previously is set to 5 minutes, we convert the units from seconds to minutes in ESL log file.

Fig. 2 shows the comparison of different algorithms in average utilization, imbalance degree, makespan and Capacity_makespan. From these figures, we know that PrepartitionOff algorithm has better performance than other algorithms on four aspects. For average utilization, PrepartitionOff algorithm is 10%-20% higher than PMG, LPT, and Random-Robin (RR). The reason for different algorithms to have different average CPU utilization lies in that we consider heterogeneous PMs and different algorithms may use the different number of total PMs. For makespan and Capacity_makespan, PrepartitionOff algorithm is 10%-20% lower than PMG and LPI, 30%-40% lower than Random-Robin (RR). And for imbalance degree, it is 30%-40% lower than LPT.

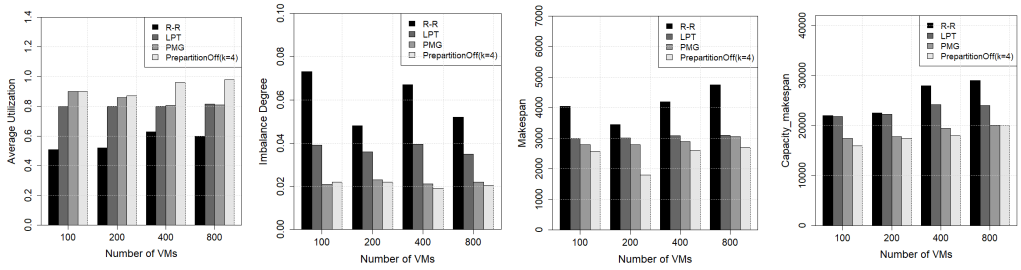
Observation 2. Post migration algorithm (PMG) cannot achieve the same level of average utilization, makespan and Capacity_makespan as PrepartitionOff does, no matter what numbers of migration have been taken.

This is because that PrepartitionOff works in a much more refined and desired scale by pre-partition based on reservation data while PMG is just a best-effort trial by migration. The reason is that PrepartitionOff is much more precise and desired with the aid of pre-partition while PMG is just a trial to balance load as much as possible. To compare imbalance degree (IMD) change as time goes, we also did the test about consecutive imbalance degree using 1000 and 2000 VMs among 4 different offline algorithms. In Fig 3, we provide the consecutive imbalance degree comparison for four algorithms in offline scheduling with 1000 VMs and 2000VMs respectively. In these two figures, the X-axis is for makespan and Y-axis is for imbalance degree. We can see that PrepartitionOff (with k=8) has the smallest makespan and smallest imbalance degree in most of the time during tests, except for the initial period. Notice that the value of k can be set differently, here we



(a) (b)

Figure 3: The consecutive imbalance degree under 1000 VMs among 4 different offline algorithms, where x-axis is for time and y-axis for imbalance degrees (a) 1000 VMs; (b) 2000 VMs.



(a) (b) (c) (d)

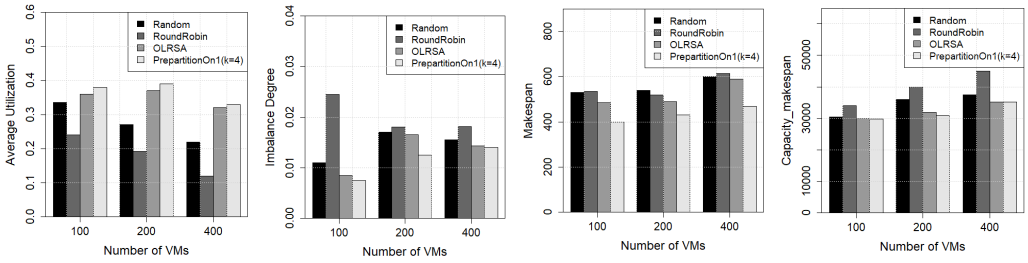
Figure 4: The offline algorithm comparison of (a) average utilization; (b) imbalance degree; (c) makespan; (d) capacity_makespan with Normal distribution

just present the results for k=8.

4.1. Offline Algorithm Performance Evaluation

4.1.1. ***Results Comparison by Synthetic Data***

We set a time slot to be 5 minutes as mentioned before, so there are 12 slots for an hour, 288 slots for a day. All requests are subject to Normal distribution with mean μ and standard deviation δ as 864 (three days) and 288 (one day) respectively. After requests are generated in this way, we start the simulator to simulate the scheduling effects of different algorithms and comparison results are collected. For gathering data, first we set k of PartitionOff algorithm as 4, different types of VM with equal probabilities. Then we vary the VMs number from 100, 200, 400 and 800 to analyze the trend. Each data set is the average of 10-runs.



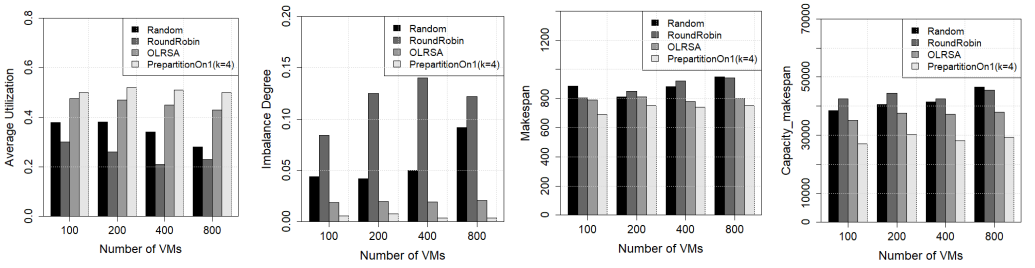
(a) (b) (c) (d)

Figure 5: The online algorithm comparison of (a) average utilization; (b) imbalance degree; (c) makespan; (d) capacity_makespan with ESL trace

Fig. 4 displays the comparison of different algorithms in average utilization, makespan and Capacity_makespan. From these figures, we can know that PartitionOff algorithm is 10%-20% higher than PMG and LPT for average utilization, 40%-50% higher than Random-Robin (RR). As for makespan and Capacity_Makespan, PartitionOff algorithm is 8%-13% lower than PMG and LPT, 40%-50% lower than Round-Robin (RR). We can also notice that the PM algorithm can improve the performance of LPT algorithm. LPT algorithm is better than R-R algorithm. Similar results are observed for the comparison of makespan.

4.2. PrepartitionOn1 Algorithm

In this section, we present the simulation results of PrepartitionOn1 algorithm and the other three algorithms. All VM requests are generated by following Normal distribution, and Random, Round-Robin, Online Resource Scheduling Algorithm (OLRSA) [15] which has best competitive ratio for online algorithm, and PrepartitionOn1 Algorithm are implemented to compare:
1) **Random Algorithm:** a scheduling algorithm which randomly allocates VM requests to PMs satisfying the VMs requirements.
2) **Round-Robin Algorithm(R-R):** a traditional load balancing algorithm which allocates the VM requests in turn to PMs which have enough resources.
3) **OLRSA algorithm**: an online scheduling problem which calculates the Capacity_makespan of each PM and sorts PM by Capacity_makespan in descending order. This algorithm always allocates the request to the PM with the lowest Capacity_makespan and required resources.



(a) (b) (c) (d)

Figure 6: The online algorithm comparison of (a) average utilization; (b) imbalance degree; (c) makespan; (d) capacity_makespan with normal distribution

4.2.1. ***Replay with ESL Data Trace***

For realistic data, we utilize the log data at Hebrew University, Experimental Systems Lab (ESL)[13] because it is derived from the real trace and suitable for our scenario. Fig. 5 illustrates the comparisons of the average utilization, imbalance degree, makespan, Capacity_makespan. From these figures, we can notice that PrepartitionOn1 shows the highest average utilization, the lowest imbalance degree, and the lowest makespan. As for Capacity_makespan, OLRSA has been proved much better performance compared with random and round-robin algorithms, and PrepartitionOn1 still improves 10%-15% in average utilization, 20%-30% in imbalance degree, and 5% to 20% in makespan than OLRSA.

4.2.2. ***Results Comparison by Synthetic Data***

The requests are configured as same as in Section IV.A based on Normal distribution. We set that different types of VMs have equal probabilities, then we change the requests generation approach to produce different size of requests to trace the tendency. From Fig. 6, we can see that PrepartitionOn1 has better performance in average utilization, imbalance degree, makespan and Capacity_makespan. Comparing to OLRSA, PrepartitionOn1 still improves about 10% in average utilization, 30%-40% in imbalance degree, 10%-20% in makespan, as well as 10%-20% in Capacity_makespan.

It is apparent that the large k values may bring side effects since it will need more partitions. In Fig. 7, we compare the time costs (simulated with ESL data and the time unit is millisecond) under different partition value k, PrepartitionOn1 algorithm with $k = 3$ takes about 10% less running time than that with $k = 4$, and $k = 2$ takes 15% less running time than that

with $k = 4$. It is easy to understand that a larger k value will produce a better load balance with longer process time. We also observe that larger k value will induce a lower Capacity_makespan value. Similarly, with a larger k value, larger average utilization, lower imbalance degree and makespan are obtained.

LPT is one kind of the best methods for offline load balance algorithms without migration, which has the approximate ratio as 4/3. So we suggest setting the k value as 4, which can obtain an approximate ratio as 1+ $1/k$= 5/4. Under this configuration, the better approximate ratio could be obtained. With higher k value, better load balancing effects could be achieved. While there exist tradeoffs between load balancing effect and time cost. For online load balance algorithms, we also suggest setting the k value as 4 and cloud service providers could reconfigure that value to be higher as suitable as the load balancing effects they desired.

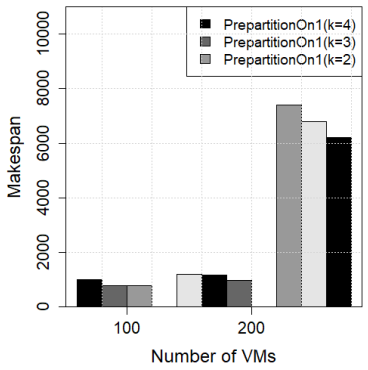


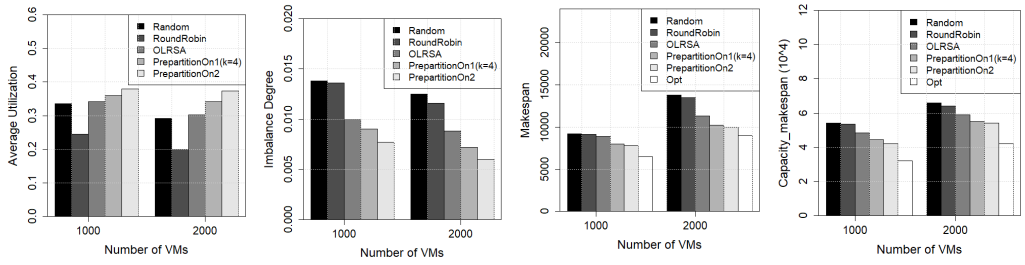
Figure 7: The comparison of time costs for PrepartitionOn1 by varying k values

4.3. PrepartitionOn2 Algorithm

In this section, we provide the simulation results of PrepartitionOn2 algorithm and the other three algorithms. Random, Round-Robin, Online Resource Scheduling Algorithm (OLRSA) [15] and PrepartitionOn2 Algorithm are implemented to compare.

4.3.1. Replay with ESL Data Trace

We still use the log data at Hebrew University, Experimental Systems Lab (ESL)[13] for realistic data. Fig. 8 illustrates the comparisons of the average



(a) (b) (c) (d)

Figure 8: The online algorithms and Prepartition algorithm comparison of (a) average utilization; (b) imbalance degree; (c) makespan; (d) capacity_makespan with normal distribution

utilization, imbalance degree, makespan, Capacity_makespan. From these figures, we can notice that PrepartitionOn2 shows the highest average utilization, lowest imbalance degree, lowest makespan and lowest Capacity_makespan.

In Fig. 9, we provide the consecutive imbalance degree comparison for four algorithms in online scheduling with 1000 VMs and 2000VMs respectively. In these two figures, the X-axis is for makespan and Y-axis is for imbalance degree. We can see that PrepartitionOn2 has the smallest makespan and smallest imbalance degree in most of the time during tests.

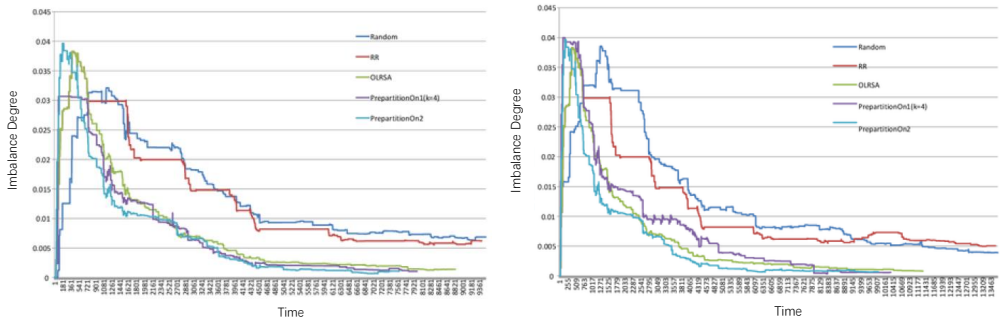
Table 3 shows the number of migrations with different algorithms during our testes. Since PrepartitionOff algorithm is offline, so the number is much smaller than the online algorithms. And the PrepartitionOn2 reduces the number of migrations than PrepartitionOn1.

Table 3: Number of Migrations in different Algorithms

Algorithm	Number of Migrations	
	1000 VMs	2000 VMs
PrepartitionOff	64	109
PrepartitionOn1	159	361
PrepartitionOn2	115	293

5. Related work

A large number of literature has devoted to resource scheduling and load balancing algorithms in cloud computing, as introduced in several popular surveys. Xu et al. [25] had a survey for the state-of-art VM placement algorithms. Ghomi et al. [26] recently made a comprehensive survey on load



(a) (b)

Figure 9: The consecutive imbalance degree under 1000 VMs among 5 different online algorithms, where x-axis is for time and y-axis for imbalance degrees (a) 1000 VMs; (b) 2000 VMs.

Table 4: The comparison of closely related work

Approach	Algorithm Type		VM Type		Resource Type		Theoretical Analysis	Metrics					
	Online	Offline	Heterogeneous	Homogeneous	Single	Multiple		Utilization	Imblance Degree	Makespan	Time Cost	Capacity_makespan	SLA violations
Song et al. [16]	✓			✓	✓			✓					
Thiruvenkadam et al. [17]	✓		✓			✓	✓	✓					
Wen et al. [18]		✓	✓			✓							✓
Cho et al. [19]		✓	✓			✓		✓					
Tian et al. [20]		✓	✓			✓	✓		✓	✓			
Chhabra et al. [21]	✓		✓		✓			✓	✓				
Bala et al. [22]		✓	✓			✓		✓	✓				✓
Ebadfard et al. [23]		✓	✓			✓	✓	✓		✓			
Ray et al. [24]		✓		✓		✓			✓				
Our Approach Prepartition	✓	✓	✓			✓	✓	✓	✓	✓	✓	✓	

balancing algorithms in cloud computing. Thakur et al. [27] introduced a taxonomic survey on load balancing in cloud. Noshy et al. [28] reviewed the latest optimization technology dedicated to developing live VM migration. They also emphasized the open research that needs further investigation to optimize the process for migrating virtual machines. Kumar et al. [29] conducted a survey to discuss the issues and challenges associated with existing load balancing techniques. In general, approaches for VM load balancing can be categorized into two categories: online and offline. The online ones assume that only the current requests and PMs status are known, while the offline ones assume all the information is known in advance.

Online approach for loading balancing: Song et al. [16] proposed a VM migration approach to dynamically balance VM loads for high-level application federations. Thiruvenkadam et al. [17] presented a hybrid genetic algorithm for scheduling and optimizing VMs, which aims to minimize the number of migrations when balancing VMs. Cho et al. [19] combined ant colony and particle swarm optimization to balance the loads in Cloud

computing, which aims to maximize the balance of resource utilization.

Offline approach for VM load balancing: Tian et al. [20] presented an algorithm for offline VM allocation within the reservation mode, in which the VM information are known before placement. Wen et al. [18] proposed a distributed VM load balancing strategy based on ant colony optimization, with the goals of utilizing resources in a balanced manager and minimizing the number of migrations. By estimating resource usage, Chhabra et al. [21] proposed an VM placement approach for loading balancing based on maximum likelihood estimation for parallel and distributed applications. Bala et al. [22] presented an approach to improving proactive load balancing by predicting multiple resource types in cloud. Ebadifard et al. [23] proposed a static task scheduling approach based on particle swarm optimization algorithm that the tasks are considered as non-preemptive and independent. Ray et al. [24] proposed a genetic algorithm based load balancing approach to distribute VM requests uniformly among the physical machines.

Different from all the above methods, 1) we investigated the reservation model that makespan and VM capacity are considered together for optimization rather than only considering the makespan or capacity separately. 2) Our approach can be applied to both online and offline scenarios rather than for single scenario. 3) We also performed theoretical analysis for the proposed approach, and 4) evaluated more performance metrics. A qualitative comparison between our method and others is listed in Table 4.

6. Conclusion and Future Work

We presented a novel load balancing paradigm for cloud administrators to balance the VM loads for PMs in data centers. Through preparation operations before allocation for VMs, better load balancing effects could be achieved compared to well-known load balancing algorithms. In this paper, to reflect the feature of capacity sharing and fixed interval constraint of VM scheduling in cloud data centers, we propose new offline and online load balancing algorithms. Theoretically, we prove that PrepartitionOff is a $(1+\epsilon)$-approximation where $\epsilon=\frac{1}{k}$ and k is a positive integer. By increasing k it is possible to be very close to the optimal solution, i.e., through setting k, it is also possible to achieve predefined load balance goal as desired because PrepartitionOff is a $(1+\frac{1}{k})$-approximation, PrepartitionOn1 has competitive ratio $(1+\frac{1}{k}-\frac{1}{mk})$ and PrepartitionOn2 has competitive ratio $(1+f)$ where f is a constant below 0.5. Both synthetic and trace driven simulations have val-

idated theoretical observations and shown Prepartition algorithm has better performance than a few existing algorithms at average utilization, imbalance degree, makespan, and Capacity_makespan both for offline and online algorithms. There are still a few research issues can be considered:

- Making a suitable choice between total partition numbers and load balance objective. Prepartition algorithm can achieve desired load balance objective by setting suitable k value. It may need a large number of partitions so that the number of migrations can be large depending on the characteristics of VM requests. For example in EC2 [7], the duration of VM reservations varies from a few hours to a few months, we can classify different types of VMs based on their durations (Capacity_makespans) firstly, then applying Prepartition will not have a large partition number for each type. In practice, we need analyzing traffic patterns to make the number of partitions (pre-migrations) reasonable so that the total costs, including running time and the number migration can be reduced.

- Considering the heterogeneous configuration of PMs and VMs. We mainly consider that a VM requires a portion of total capacity from a PM. This is also applied in EC2 and Knauth et al. [8]. When this is not true, multi-dimensional resources such as CPU, memory and bandwidth etc. have to be considered together or separately in the load balance, see [30] and [31] for a detailed discussion about considering multi-dimensional resources.

- Considering precedence constraints among different VM requests. In reality, some VMs may be more important than others depending on applications running on them, we should extend current algorithm to consider this case.

Acknowledgments

This research is sponsored by the National Natural Science Foundation of China (NSFC) (Grand Number:61672136, 61828202), Key-Area Research and Development Program of Guangdong Province (NO. 2020B010164003) and SIAT Innovation Program for Excellent Young Researchers.

References

[1] M. Xu, R. Buyya, Brownout approach for adaptive management of resources and applications in cloud computing systems: a taxonomy and future directions, ACM Computing Surveys (CSUR) 52 (2019) 1–27.

[2] S. S. Gill, S. Tuli, A. N. Toosi, F. Cuadrado, P. Garraghan, R. Bahsoon, H. Lutfiyya, R. Sakellariou, O. Rana, S. Dustdar, R. Buyya, Thermosim: Deep learning based framework for modeling and simulation of thermal-aware resource management for cloud computing environments, Journal of Systems and Software 166 (2020) 110596.

[3] M. Xu, R. Buyya, Brownoutcon: A software system based on brownout and containers for energy-efficient cloud computing, Journal of Systems and Software 155 (2019) 91 – 103.

[4] J. Zhang, F. R. Yu, S. Wang, T. Huang, Z. Liu, Y. Liu, Load balancing in data center networks: A survey, IEEE Communications Surveys & Tutorials 20 (2018) 2324–2352.

[5] M. Rahman, S. Iqbal, J. Gao, Load balancer as a service in cloud computing, in: 2014 IEEE 8th International Symposium on Service Oriented System Engineering, IEEE, 2014, pp. 204–211.

[6] E. T. J. Kleinberg, Algorithm Design, 2005.

[7] Amazon, Amazon elastic compute cloud, 2013. URL: `http://aws.amazon.com/ec2/`.

[8] T. Knauth, C. Fetzer, Energy-aware scheduling for infrastructure clouds, in: 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings, IEEE, 2012, pp. 58–65.

[9] R. L. Graham, Bounds on multiprocessing timing anomalies, SIAM journal on Applied Mathematics 17 (1969) 416–429.

[10] H. Shen, L. Chen, Distributed autonomous virtual resource management in datacenters using finite-markov decision process, IEEE/ACM Transactions on Networking 25 (2017) 3836–3849.

[11] W. Tian, Y. Zhao, Y. Zhong, M. Xu, C. Jing, A dynamic and integrated load-balancing scheduling algorithm for cloud datacenters, in: 2011 IEEE International Conference on Cloud Computing and Intelligence Systems, IEEE, 2011, pp. 311–315.

[12] W. Tian, Y. Zhao, M. Xu, Y. Zhong, X. Sun, A toolkit for modeling and simulation of real-time virtual machine allocation in a cloud data center, IEEE Transactions on Automation Science and Engineering 12 (2013) 153–161.

[13] H. University, 2013. URL: `www.cs.huji.ac.il/labs/parallel/workload`.

[14] A. Gulati, G. Shanmuganathan, A. M. Holler, I. Ahmad, Cloud scale resource management: Challenges and techniques., HotCloud 11 (2011) 3–3.

[15] M. Xu, W. Tian, An online load balancing scheduling algorithm for cloud data centers considering real-time multi-dimensional resource, in: 2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems, volume 1, IEEE, 2012, pp. 264–268.

[16] X. Song, Y. Ma, D. Teng, A load balancing scheme using federate migration based on virtual machines for cloud simulations, Mathematical Problems in Engineering 2015 (2015).

[17] T. Thiruvenkadam, P. Kamalakkannan, Energy efficient multi dimensional host load aware algorithm for virtual machine placement and optimization in cloud environment, Indian journal of science and technology 8 (2015) 1–11.

[18] W.-T. Wen, C.-D. Wang, D.-S. Wu, Y.-Y. Xie, An aco-based scheduling strategy on load balancing in cloud computing environment, in: 2015 Ninth International Conference on Frontier of Computer Science and Technology, IEEE, 2015, pp. 364–369.

[19] K.-M. Cho, P.-W. Tsai, C.-W. Tsai, C.-S. Yang, A hybrid meta-heuristic algorithm for vm scheduling with load balancing in cloud computing, Neural Computing and Applications 26 (2015) 1297–1309.

[20] W. Tian, M. Xu, Y. Chen, Y. Zhao, Prepartition: A new paradigm for the load balance of virtual machine reservations in data centers, in:

2014 IEEE International Conference on Communications (ICC), IEEE, 2014, pp. 4017–4022.

[21] S. Chhabra, A. K. Singh, Optimal vm placement model for load balancing in cloud data centers, in: 2019 7th International Conference on Smart Computing & Communications (ICSCC), IEEE, 2019, pp. 1–5.

[22] A. Bala, I. Chana, Prediction-based proactive load balancing approach through vm migration, Engineering with Computers 32 (2016) 581–592.

[23] F. Ebadifard, S. M. Babamir, A pso-based task scheduling algorithm improved using a load-balancing technique for the cloud computing environment, Concurrency and Computation: Practice and Experience 30 (2018) e4368.

[24] K. Ray, S. Bose, N. Mukherjee, A load balancing approach to resource provisioning in cloud infrastructure with a grouping genetic algorithm, in: 2018 International Conference on Current Trends towards Converging Technologies (ICCTCT), 2018, pp. 1–6.

[25] M. Xu, W. Tian, R. Buyya, A survey on load balancing algorithms for virtual machines placement in cloud computing, Concurrency and Computation: Practice and Experience 29 (2017) e4123.

[26] E. J. Ghomi, A. M. Rahmani, N. N. Qader, Load-balancing algorithms in cloud computing: A survey, Journal of Network and Computer Applications 88 (2017) 50–71.

[27] A. Thakur, M. S. Goraya, A taxonomic survey on load balancing in cloud, Journal of Network and Computer Applications 98 (2017) 43–57.

[28] M. Noshy, A. Ibrahim, H. A. Ali, Optimization of live virtual machine migration in cloud computing: A survey and future directions, Journal of Network and Computer Applications 110 (2018) 1–10.

[29] P. Kumar, R. Kumar, Issues and challenges of load balancing techniques in cloud computing: A survey, ACM Comput. Surv. 51 (2019). URL: https://doi.org/10.1145/3281010. doi:10.1145/3281010.

[30] A. Singh, M. Korupolu, D. Mohapatra, Server-storage virtualization: integration and load balancing in data centers, in: SC'08: Proceedings

of the 2008 ACM/IEEE conference on Supercomputing, IEEE, 2008, pp. 1–12.

[31] S. X, X. P, S. Kai, Multi-dimensional aware scheduling for co-optimizing utilization in data center, China Communications (2011) 19–27.