An Online Load Balancing Algorithm for Virtual Machine Allocation with Fixed Process Intervals

Article *in* Journal of Information and Computational Science · February 2014

DOI: 10.12733/jics20102932

Minxian Xu
University of Melbourne
34 PUBLICATIONS **660** CITATIONS

Journal of Information & Computational Science 11:3 (2014) 989–1001 February 10, 2014
Available at http://www.joics.com

An Online Load Balancing Algorithm for Virtual Machine Allocation with Fixed Process Intervals [⋆]

Minxian Xu, Wenhong Tian [*]

University of Electronic Science and Technology of China, Chengdu 611731, China

Abstract

To improve the performance and reliability of resource in Cloud data centers, it is significant to apply load balancing strategy for resource scheduling. In a dynamic changing traffic environment, workload of a Cloud data center has real-time and fixed process time characteristic. One of the challenging scheduling problems in Cloud datacenters is to take the real-time allocation and migration of reconfigurable virtual machines as well as the integrated features of hosting physical machines into consideration. In general, load-balance scheduling is NP-hard problem as proved in many open literatures. We introduce an Online Load-balance Resource Scheduling Algorithm (OLRSA) for Cloud datacenters considering real-time and multi-dimensional resources. Unlike traditional load balance scheduling algorithms which often do not consider lifecycle and fixed interval constraints, OLRSA treats life cycles and fixed intervals for both physical machines and virtual machines. We develop and apply integrated measurement for each server and a Cloud datacenter. New metrics such as integrated imbalance level, capacity_makespan, capacity_skew are defined for Cloud data centers. Simulation results show that OLRSA has better performance than a few related load balancing algorithms with regard to total imbalance level, makespan, overall load efficiency as well as capacity_makespan.

Keywords: Cloud Computing; Load Balance; Online Resource Scheduling Algorithm; Cloud Data Centers

1 Introduction

Cloud datacenters can be a distributed network in structure, containing many compute nodes (such as servers), storage nodes, and network devices. Each node is formed by a series of resources such as CPU, memory, network bandwidth and so on, which are called multi-dimensional resources, each has its corresponding properties in this paper. The definition and model defined by this paper are aimed to be general enough to be used by a variety of Cloud providers and focus on the Infrastructure as a Service (IaaS). In a traditional data centers, applications are tied to specific physical servers that are often over-provisioned to deal with upper-bound workload. Such configuration makes data centers expensive to maintain with wasted energy and floor space, low

[⋆]Project supported by the National Nature Science Foundation of China (No. 61150110486).
[*]Corresponding author.
Email address: tian_wenhong@uestc.edu.cn (Wenhong Tian).

resource utilization and significant management overhead. With virtualization technology, today's Cloud data centers become more flexible, secure and provide better support for on-demand allocating. Under virtualization situation, Cloud data centers should have ability to migrate an application from one set of resources to another in a non-disruptive manner. Such ability is essential in modern cloud computing infrastructure that aims to efficiently share and manage extremely large data centers. One key technology playing an important role in Cloud data centers is resource scheduling. There are quite many load balance scheduling algorithms. Most of them are for load balancing of traditional web servers or server farms. One of the challenging scheduling problems in Cloud data centers is to consider allocation and migration of reconfigurable virtual machines and integrated features of hosting physical machines. Unlike traditional load balance scheduling algorithms which consider only physical servers with one factor such as CPU, OLRSA treats CPU, memory and network bandwidth integrated for both physical machines (PMs) and virtual machines (VMs). The major contributions of this paper are:

- Providing a modeling approach to virtual machine scheduling problem with capacity sharing by modifying traditional interval scheduling problem and considering life cycles and multi-dimensional characteristics of both VMs and PMs.
- Designing and implementing an online load balancing scheduling algorithm with computational complexity and competitive analysis.
- Providing performance evaluation of multiple metrics such as makespan, load efficiency, imbalance value, capacity_makespan, capacity-skew by simulating different algorithms.

The remaining parts of this paper are organized as follows. Section 2 discusses the related work on load balance algorithms. Section 3 introduces problem formulation. Section 4 presents OLRSA algorithm in details. Performance evaluation of different scheduling algorithms is shown in Section 5. Finally in Section 6, a conclusion is given.

2 Related Work

A great mount of work has been devoted to the schedule algorithms and can be mainly divided into two types: online load balance algorithms and offline ones. The major difference lies in that online schedulers only know current request and status of all PMs but offline schedulers know all the requests and status of all PMs. Andre et al. [1] discussed the detailed design of a data center. Armbrust et al. [2] summarized the key issues and solutions in Cloud computing. Foster et al. [6] provided detailed comparison between Cloud computing and Grid computing. Buyya et al. [4] introduced a way to model and simulated Cloud computing environments. Wickremasinghe et al. [17] introduced three general scheduling algorithms for Cloud computing and provided simulation results. Wood et al. [18] introduced techniques for virtual machine migration and proposed some migration algorithms. Zhang [19] compared major load balance scheduling algorithms for traditional Web servers. Singh et al. [13] proposed a novel load balance algorithm called VectorDot which deals with hierarchical and multi-dimensional resources constraints by considering both servers and storage in a Cloud. Arzuaga et al. [3] proposed a quantifying measure of load imbalance on virtualized enterprise servers. Tian et al. [15] provided a comparative study of major existing scheduling strategies and algorithms for Cloud data centers. Sun et al. [14] presented a novel heuristic algorithm to improve integrated utilization considering multi-dimensional resource. Tian et al. [16] introduced a dynamic load balance scheduling algorithm considering only current allocation period and multi-dimensional resource but without considering life-cycles of both VMs and PMs. Li et al. [12] proposed a cloud task scheduling policy based on ant colony optimization algorithm to balance the entire system and minimize the makespan of a given task set. Galloway in [7] introduced an online greedy algorithm, in which PMs can be dynamic turned on and off but the life -cycle of a VM is not considered. Hu et al. [10] stated a algorithm named Genetic, which calculates the history data and current states to choose an allocation. Most of existing research does not consider real-time and fixed interval constraints of virtual machine allocation. We will address this issue in this paper.

2.1 Problem Description and Formulation

In this paper we model the VM allocations as a Modified Interval Scheduling Problem (MISP) with fixed processing time. More explanation and analysis about traditional interval scheduling problems with fixed processing time can be found in [11] and references there in. We present a general formulation of modified interval-scheduling problem and evaluate its results compared to well-known existing algorithms. A set of requests 1, 2, . . . , n where the $i-th$ request corresponds to an interval of time starting at s_i and finishing at f_i associated with a capacity requirement c_i. There are several following assumptions:

1) All data are deterministic and unless otherwise specified, the time is formatted in slotted windows. As shown in Fig. 1, we partition the total time period [0, T] into slots with equal length (s_0), the total number of slots is $k = T/s_0$. The start time s_i and finish time f_i are integer numbers of one slot. Then the interval of a request can be represented in slot format with (start-time, finish-time). For example, if $s_0 = 5$ minutes, an interval (3, 10) means that it has start time and finish time at the 3rd-slot and 10th-slot respectively. The actual duration of this request is $(10 - 3) \times 5 = 35$ minutes.

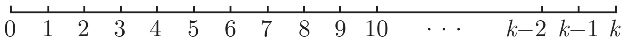


Fig. 1: Slots format

2) All tasks are independent. There are no precedence constraints other than those implied by the start and finish time.

3) The required capacity of each request is a positive real number between (0, 1]. Notice that the capacity of a single physical machine is normalized to be 1.

4) Assuming that, when processed, each VM request is assigned to a single physical machine, thus interrupting a request and resuming it on another machine is not allowed, unless explicitly stated otherwise.

5) Each physical machine is always available, i.e., each machine is continuously available in $[0, \infty)$.

Traditional interval scheduling problem (ISP) with fixed processing time: A set of requests 1, 2, . . . , n where the $i-th$ request corresponds to an interval of time starting at s_i and finishing at f_i, each request needs a capacity of 1, i.e. occupying the whole capacity of a machine during fixed processing time.

Interval scheduling with capacity sharing (ISWCS): The only difference from traditional interval scheduling is that a resource (to be concrete, a physical machine) can be shared by different requests if the total capacity of all requests allocated on the single resource at any time does not surpass the total capacity that the resource can provide.

Sharing compatible intervals for ISWCS: A subset of intervals with total required capacity not surpass the total capacity of a physical machine at any time, therefore they can share the capacity of a PM.

The formulation of ISWCS can be described as follows. Given a set of m identical machines (PMs) $PM_1, PM_2, \ldots, PM_m$ and a set of n requests (VMs), each request has a processing time (consider only CPU processing for example), the objective of load balance is to assign each request to one of PMs so that the loads placed on all machines are balanced. The on-line scheduler knows only current request and status of all PMs. In the literature, the makespan is used to measure the load balance, which is simply the maximum total load (processing time) on any machine. Traditionally, the makespan is the total length of the schedule (that is, when all the jobs have finished processing where each job occupies the whole capacity of a machine during processing).

Theorem 1 The offline scheduling problem of finding an allocation of minimizing makespan in general case is NP-complete

Proof: We sketch a brief proof as follows, and the detailed proof is referred to [11]. We show that this scheduling problem (called Load-balance Scheduling Problem, LSP) is polynomial time reducible to a well-know NP complete problem, Subset Sum Problem (SSP). Thus consider an instance of Subset Sum with numbers $w_1, w_2, \ldots, w_n$, which corresponding to the CPU load of n VM requests and have total CPU load W. To be load-balance, in an ideal case, it is to let all m machines have same share of total CPU load, i.e., W/m. This needs all allocations on all PMs to be satisfied. Suppose there are j VMs on PM_i, this requires that $L_i = W/m$. It is reduced to Subset Sum Problem. This completes the proof.

Remarks: Notice that Theorem 2.1 is considering offline load-balancing scheduling for single resource CPU on identical machines. When there are multiple resources to be considered and in heterogeneous case (like in this paper), the problem is more difficult and can be proved in the similar way that it is NP-complete too (a detailed proof is provided in [5] by transforming the problem to 3-Dimensional matching problem or a multi-dimensional vector bin packing problem). The load balance of ISWCS is different from load balance of traditional mutli-processor scheduling: firstly each request may have different capapcity demand in ISWCS while each job occupies the whole capacity of one machine in traditional mutli-processor scheduling; secondly, ISWCS has fixed process interval while the job can be delayed in traditional mutli-processor scheduling without considering start-time or end-time. Traditional metric such as makespan may not reflect the real load for ISWCS problem. For example, consider there are n=7 jobs, m=3 machines and each machine has capacity C=3, the first six jobs all have start-time at zero and end-time at 1, with capacity 1; the last job has start-time zero and end-time 3 with capacity 1. Traditional List Scheduling algorithm [8] allocates two jobs of first six jobs to each machine and the last job to the first machine, and has makespan 3; the optimal solution is to allocate three jobs of the first six jobs to the first two machines and the last job to the third machine, this will also have makespan equal to 3. However, this does not reflect the real load of each machine, actually List Scheduling will have the maximum load 5 in all machines while max load is 3 for all machines in optimal solution. The reason is that both capacity sharing and fixed processing interval constraint should be considered for ISWCS problem.

In view this issue, we redefine the makespan as capacity_makespan.

Capacity_makespan: In any allocation of VM requests to PMs, we can let $A(i)$ denote the set of VM requests allocated to machine PM_i, under this allocation, machine PM_i will have total loads,

$$L_i = \sum_{j \in A(i)} c_j t_j \tag{1}$$

where c_j is the capacity (for example CPU) requests of VM_j and t_j is the span of request j (i.e., the length of processing time of request j). The goal of load balancing is to minimize the maximum load (capacity_makespan) on any PM. Some other related metrics such as imbalance value and load efficiency are also considered and will be explained in the following section.

2.1.1 Metrics for Real-time Load Balancing Scheduling Algorithms

In this section, a few existing metrics and new metrics for load balancing scheduling will be presented. Zheng et al. in [20] introduced an integrated load balancing index and load balancing algorithm:

$$B = a \times \frac{N_{1i} \times C_i}{N_{1m} \times C_m} + b \times \frac{N_{2i} \times M_i}{N_{2m} \times M_m} + c \times \frac{N_{3i} \times D_i}{N_{3m} \times D_m} + d \times \frac{Net_i}{Net_m} \tag{2}$$

where i is the index of PM and m is the ID of referred PM, N_1 is the capability of CPU, N_2 is the parameter of memory, N_3 refers to the parameter of bandwidth, C and M are the utilization of CPU and memory, D is the transferring rate of hard disk, Net is the network throughput, a, b, c, d are the compared weighted value of CPU, memory, hard disk and network respectively and initialized as 1. The optimization goal is finding the PM with the smallest B value to allocate requests. For OLRSA algorithm, we take the following parameters into consideration:

1) PM resource: $PM_i(i, PCPU_i, PMem_i, PStorage_i)$, i is the index number of PM, $PCPU_i$, $PMem_i$, $PStorage_i$ are the CPU, memory, storage capacity of that a PM can provide.

2) VM resource: $VM_j(j, VCPU_j, VMem_j, VStorage_j, T_j^{start}, T_j^{end})$, j is the VM type ID, $VCPU_j$, $VMem_j$, $VStorage_j$ are the CPU, memory, storage requirements of VM_j, T_j^{start}, T_j^{end} are the start time and end time, which are used to represent the life cycle of a VM.

3) Time slot: we consider a time span from 0 to T be divided into parts with same length. Then n parts can be defined as $[(t_1 - t_0), (t_2 - t_1), \ldots, (t_n - t_{n-1})]$, each time slot T_k means the time span $(t_k - t_{k-1})$.

4) Average CPU utilization of PM_i during slot 0 and T_n:

$$PCPU_i^U = \frac{\sum_{k=0}^{n} (PCPU_i^{T_k} \times T_k)}{\sum_{k=0}^{n} T_k} \tag{3}$$

And memory $PMem_i^U$ and storage $PStorage_i^U$ utilization of both PMs and VMs can be computed in the same way. Similarly, average CPU utilization of a VM can be computed.

5) Integrated load imbalance value ILB_i of PM_i. The variance is widely used as a measure of how far a set of values are spread out from each other in statistics. Using variance, an integrated

load imbalancing value ILB_i of server i is defined

$$ILB_i = \frac{(Avg_i - CPU_u^A)^2}{3} + \frac{(Avg_i - Mem_u^A)^2}{3} + \frac{(Avg_i - Storage_u^A)^2}{3} \tag{4}$$

where

$$Avg_i = \frac{PCPU_i^U + PMem_i^U + PStoarge_i^U}{3} \tag{5}$$

and $CPU_u^A, Mem_u^A, Storage_u^A$ are respectively the average utilization of CPU, memory and storage in a Cloud data center. ILB_i is applied to indicate load imbalance level comparing utilization of CPU, memory and network bandwidth of a single server itself. This metric is very similar to VMware DRS load balance metric, standard deviation, as presented in [9].

6) Makespan is the same as traditional definition, and therefore the capacity_makespan of all PMs can be formulated as below:

$$capacity\_makespan = \max_i (L_i) \tag{6}$$

7) Load efficiency (skew of makespan) is defined as the minimal average load divided by the maximal average load on all machines:

$$skew(makespan) = \frac{\min_i(L_i)}{\max_i(L_i)} \tag{7}$$

where L_i is the load of PM i. Skew shows the load balance efficiency to some degree.

8) Imbalance Level (IBL) of CPU is defined as:

$$IBL_{cpu} = \frac{\sum_{i=0}^{n}(PCPU_i^U - PCPU_{avg})^2}{n} \tag{8}$$

where $PCPU_{avg}$ is the average utilization of all CPUs in a data center. The imbalance level of memory IBL_{mem} and imbalance level of storage $IBL_{storage}$ can be obtained in the same way. Then total imbalance level of a data center is:

$$IBL_{total} = IBL_{cpu} + IBL_{mem} + IBL_{storage} \tag{9}$$

Based on the above definitions and equations, we have developed another metric, capacity_skew on load balancing algorithm for the new situation as follow:

9) Skew of capacity_makespan is defined as the minimal capacity_makespan over maximal capacity_makespan on all machines (referring to Eq. (1)):

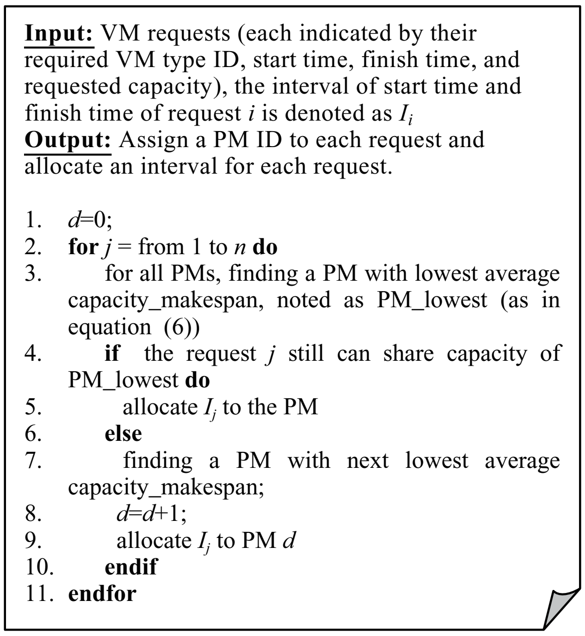
$$skew(capacity\_makespan) = \frac{\min \sum_{j \in A(i)} c_j t_j}{\max \sum_{j \in A(i)} c_j t_j} \tag{10}$$

The higher value shows a better load balance to some degree.

From these equations, we notice that life cycle and capacity sharing are two major differences from traditional metrics such as makespan and skew. Traditionally List Scheduling [8] is widely used for load balance of online multi-processor scheduling. By considering both fixed process intervals and capacity sharing properties in Cloud data center, we propose a new online algorithm as follows.

3 OLRSA Algorithm

Fig. 2 shows the core process of OLRSA algorithm. For each request it firstly find the PM with lowest average capacity_makespan, and a PM with next lowest average capacity_makespan would be turn-on only if there is no enough resource left on the first PM, so that all requests are allocated without rejection.



Input: VM requests (each indicated by their required VM type ID, start time, finish time, and requested capacity), the interval of start time and finish time of request i is denoted as I_i
Output: Assign a PM ID to each request and allocate an interval for each request.

```
1.  d=0;
2.  for j = from 1 to n do
3.      for all PMs, finding a PM with lowest average capacity_makespan, noted as PM_lowest (as in equation (6))
4.      if the request j still can share capacity of PM_lowest do
5.          allocate I_j to the PM
6.      else
7.          finding a PM with next lowest average capacity_makespan;
8.          d=d+1;
9.          allocate I_j to PM d
10.     endif
11. endfor
```

Fig. 2: Pseudo code of OLRSA algorithm

Theorem 2: The computational complexity of OLRSA algorithm is $O(nlogm)$ using priority queue data structure where n is the number of VM requests and m is 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 highest priority (the smaller average capacity_makespan value is, the higher priority it is). Sorting a set of n number in a priority queue takes $O(n)$ time and a priority queue performs insertion and the extraction of minima in $O(logn)$ steps (detailed proof of the priority queue is shown in [11]). Therefore, by using priority queue or related data structure, the algorithm can find a PM with lowest average capacity_makespan in $O(logm)$ time. Altogether, for n requests, OLRSA algorithm has time complexity $O(nlogm)$.

Theorem 3: The competitive ratio of OLRSA algorithm is $(2 - 1/m)$ where m is the total number of machines.

Proof: Considering m machines and n requests:

$$n > m, m > 2 \tag{11}$$

CM_i is the capacity_makespan of VM i, c_i is the resource capacity VM i needed, which could be CPU, memory, storage or integrated resource:

$$CM_i = c_i * (T_i^{end} - T_i^{start}) \tag{12}$$

Let OPT and $OLRSA$ represent the scheduling results of optimal solution and OLRSA solution respectively. Let L_i denote the load of machine M_i and let M^* be the most heavily loaded machine in the schedule by OLRSA. Let j_k be the last job assigned to M^*. We can easily deduce the following two equations:

$$OPT \geq \frac{1}{m}\sum_{i=1}^{n} CM_i \tag{13}$$

$$OPT \geq \max_i CM_i \tag{14}$$

All PMs must be loaded at least (OLRSA-CM_k) at the time of allocating CM_k because OLRSA alreays allocates a VM to a PM with the lowest capacity_makespan. Since $OLRSA - CM_k$ represents the PM with the lowest capacity_makespan, Then we have:

$$\sum_{i=1}^{i=n} CM_i - CM_k \geq m(OLRSA - CM_k) \tag{15}$$

The above equations can be transformed as:

$$\begin{aligned} OLRSA &\leq \frac{\sum_{i=1}^{i=n} CM_i - CM_k}{m} + CM_k \\ &= \frac{\sum_{i=1}^{i=n} CM_i}{m} + \Big(1 - \frac{1}{m}\Big)CM_k \\ &\leq OPT + \Big(1 - \frac{1}{m}\Big)OPT \\ &= (2 - \frac{1}{m})OPT \end{aligned} \tag{16}$$

Observation 1: The upper bound for OLRSA algorithm is tight.

Remarks: We have exemplified a general case to show that the upper bound holds. Considering m machine are providing resources and each machine can be allocated VMs with total capacity g (total capacity of a machine is g). Suppose there are $(m-1)\times g+1$ requests in total, the first $(m-1)\times g$ requests all start at time slot 0 and finish at time slot 1, while the last request starts at 0 and ends at g. In this case, for OPT algorithm, the capacity_makespan is:

$$OPT = \frac{(m-1)g + g}{m} = g$$

As for $OLRSA$, the first $(m-1)g$ would be allocated to m machines evenly by the allocation rule (let $(m-1)g$ divide m), and the last one would also be allocated to a PM with lowest capacity_makespan value (in this case any PM will be fine). So

$$OLRSA = \frac{(m-1)g}{m} + g = g\Big(1 - \frac{1}{m}\Big) + g$$

Then, the competitive ratio of $OLRSA$ over OPT is:

$$\frac{OLRSA}{OPT} = 1 - \frac{1}{m} + 1 = 2 - \frac{1}{m}$$

4 Performance Evaluation

In this section, we compare simulation results of different scheduling algorithms regarding load-balance.

4.1 Mythology and Simulation Setting

In this part, we will show the simulation results between the OLRSA algorithm and other existing algorithms. A Java discrete simulator is developed for this purpose. All simulations are conducted on a Pentium dual-core computer with 3.2GHz CPU and 2GB memory. We compare the simulation results of our proposed algorithm with four existing algorithms:

1) Random Algorithm (Random): a general scheduling algorithm by randomly allocating the VM requests to the PM that can provide resource required.

2) Round-Robin (Round): a traditional load balancing scheduling algorithm by allocating the VM request one by one to each PM in turn that can provide resource required.

3) ZHJZ algorithm: as defined in [20], it selects a reference physical machine, and calculates the value and chooses the physical machines with the lowest B value (as defined in Eq. (2)) and available resource to allocate virtual machines.

4) List Scheduling (LS) algorithm [11]: One of the best-known online traditional load-balancing algorithm, it selects the available PM with the lowest current load to allocate virtual machine.

For simulation, to be realistic, we adopt the log data at Lawrence Livermore National Lab (LLNL) [21]. That log contains months of records collected by a large Linux cluster. Each line of data in that log file includes 18 elements, while in our simulation, we only need the requestID, start time, duration, needed processor. To enable those data be fit with our simulation, some conversions are needed, like we convert the units from seconds in LLNL log file into minutes, because we set a minute as a time slot length mentioned in previous section. Another conversion is that processor number needed in LLNL log file has been changed into 8 types of VM requests. To simplify the simulation, three types of heterogeneous PMs and eight types of VMs are considered (can be dynamic configured and extended). We do the simulation with enough PM that would satisfy all the VM requests (for example, in the situation with VMs 200 and duration larger than 30, the number of PMs is 18 type-1, 20 type-2, 12 type-3 respectively) and VM numbers vary from 200 to 800 (each type approximately 1/8). The simulations for different algorithms are based on the same environment with same VM requests. The only difference lies in the scheduling process of each algorithm, including that all PMs are turned on at the beginning in other algorithms while in OLRSA PMs are turned on one by one according to the VM requests. To be fair, if the actual total number of turn-on PMs are not the same for different algorithms, all metrics such as capacity_makespan, skew and imbalance value are adjusted by timing a co-efficiency (the actual total number of turn-on PMs divides the max number of PMs used by all algorithms).

4.2 Simulation Results and Analysis

4.2.1 Divisible Capacity Configuration of VMs and PMs

Strongly divisible capacity of jobs and machines: the capacity of all jobs form a divisible sequence, i.e., the sequence of distinct capacities $c_1 \geq c_2 \geq ... \geq c_i \geq c_{i+1} \geq ...$ taken on by jobs (the number of jobs of each capacity is arbitrary) is such that for all $i > 1$, c_{i+1} exactly divides c_i, the largest item capacity c_1 in L exactly divides the capacity C. See paper [5] for detailed discussion.

Table 1: Eight types of VMs in Amazon EC2

CPU Units	MEM	Storage	VM Type
1 units	1.7GB	160GB	1-1(1)
4 units	7.5GB	850GB	1-2(2)
8 units	15GB	1690GB	1-3(3)
6.5 units	17.1GB	420GB	2-1(4)
13 units	34.2GB	850GB	2-2(5)
26 units	68.4GB	1690GB	2-3(6)
5 units	1.7GB	350GB	3-1(7)
20 units	7GB	1690GB	3-2(8)

Table 2: Three types of PMs suggested

PM Pool Type	CPU Units	MEM	Storage
Type 1	16 units	30GB	3380 GB
Type 2	52 units	136GB	3380 GB
Type 3	40 units	14GB	3380 GB

In this paper, we also adopt the following divisible capacity configuration of VMs and PMs as shown in Table 1 and 2. Note that one Compute Unit (CU) has equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor [22].

To simplify the corresponding relationship mentioned in Section 3, we use VM type1, 2, 3 for PM Type1, Vm type 4, 5, 6 for PM Type2 and VM 7, 8 for PM Type 3.

Table 3: Eight types of Virtual Machines (VMs) in Amazon EC2

Compute Units	Memory	Storage	VM Type
1 units	1.7GB	160GB	1-1(1)
4 units	7.5GB	850GB	1-2(2)
8 units	15GB	1690GB	1-3(3)
6.5 units	17.1GB	420GB	2-1(4)
13 units	34.2GB	850GB	2-2(5)
26 units	68.4GB	1690GB	2-3(6)
5 units	1.7GB	350GB	3-1(7)
20 units	7GB	1690GB	3-2(8)

Table 4: Three types of Physical Machines (PMs) suggested

PM Pool Type	Compute Units	Memory	Storage
Type 1	16 units	30GB	3380 GB
Type 2	52 units	136GB	3380 GB
Type 3	40 units	14GB	3380 GB

Fig. 3 to 7 show the imbalance level, makespan and skew of makespan, capacity_makespan and skew of capacity_makespan results respectively when fixing total number of VM requests as 200 but varying the max duration of VMs. The results are the average value of five times simulation of the same inputs (data is from LLNL log file). From these figures, we can notice that OLRSA algorithm shows the best performance in IBL, makespan, capacity_makespan, skew of capacity_makespan except for skew of makespan compared with other four algorithms. Notice that the skew of makespan is a traditional index to measure the load balance scheduling without considering capacity sharing and fixed interval constraints.

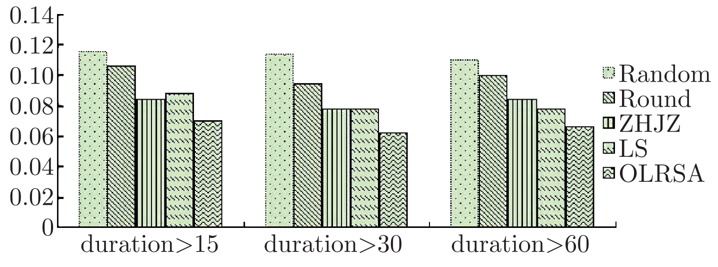


Fig. 3: Imbalance level comparison when varying duration of VMs

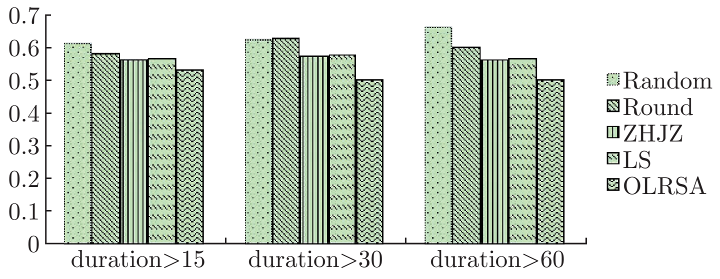


Fig. 4: Makespan comparison when varying duration of VMs

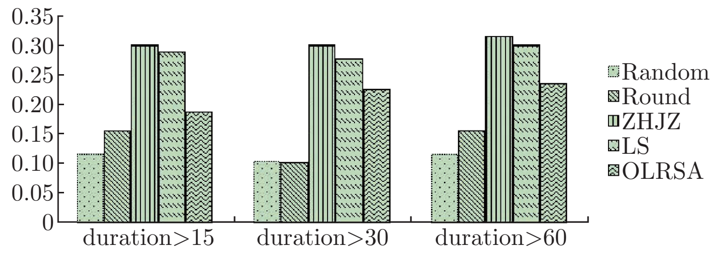


Fig. 5: Skew of makespan comparison when varying duration of VMs

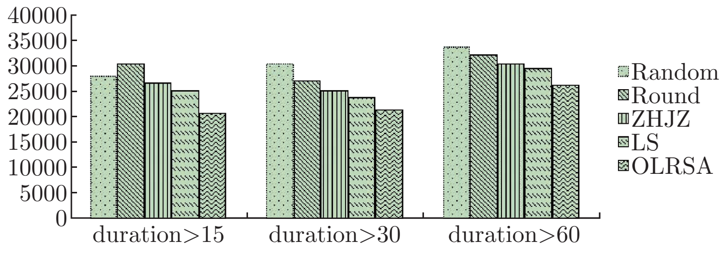


Fig. 6: Capacity_makespan comparison when varying duration of VMs

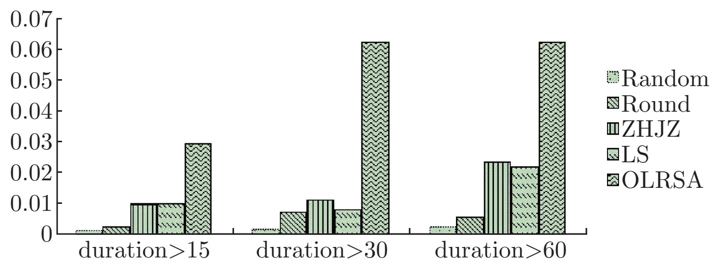


Fig. 7: Skew of capacity_makespan comparison when varying duration of VMs

5 Discussion and Conclusions

In this paper, to reflect capacity sharing property and fixed interval constraint in Cloud data centers, we propose an Online Load balancing Resource Scheduling Algorithm (OLRSA) with new metrics such as capacity_makespan and skew of capacity_makespan. Simulations have shown that OLRSA has better performance than a few existing algorithms at imbalance level, capacity_makespan as well as skew of capacity_makespan. A theoretical competitive ratio upper bound ($2-\frac{1}{m}$) is provided and the proof is also given where m is the number of physical machines. There are still a few research issues can be considered: heterogeneous configuration of physical machines, Offline load balancing scheduling for ISWCS problem and Combining with energy-efficient and other scheduling algorithms.

6 Acknowledgement

This research is supported by China National Science Foundation (CNSF) with project ID 61150110486.

References

[1] L. Andre et al., The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Ebook, 2009

[2] M. Armbrust et al., Above the Coulds: A Berkeley View of Cloud Computing, Technical Report, 2009

[3] E. Arzuaga, D. R. Kaeli, Quantifying load imbalance on virtualized enterprise servers, Proceedings of WOSP/SIPEW 10, San Jose, California, USA, Jan. 28-30, 2010

[4] R. Buyya, R. Ranjan, R. N. Calheiros, Modeling and simulation of scalable cloud computing environments and the CloudSim toolkit: Challenges and Opportunities, Proceedings of the 7th High Performance Computing and Simulation Conference, HPCS 2009, Leipzig, Germany, Jun. 21-24, 2009

[5] E. G. Coffman Jr., M. R. Garey, D. S. Johnson, Bin-packing with divisible item sizes, J. Complexity, 3 (1987), 406-428

[6] I. Foster, Y. Zhao, I. Raicu, S. Lu, Cloud computing and grid computing 360-degree compared, IEEE International Workshop on Grid Computing Environments (GCE) 2008, Co-located with IEEE/ACM Supercomputing 2008

[7] J. M. Galloway, K. L. Smith, S. S. Vrbsky, Power aware load balancing for cloud computing, Proceedings of the World Congress on Engineering and Computer Science 2011, WCECS 2011, Vol I, Oct. 19-21, 2011

[8] R. L. Graham, Bounds on multiprocessing timing anomalies, SIAM Journal on Applied Mathematics, Vol. 17, No. 2, Mar. 1969, 416-429

[9] A. Gulati, G. Shanmuganathan, A. Holler, I. Ahmad, Cloud-scale resource management: Challenges and techniques, VMware Technical Journal, 2011

[10] J. Hu, J. Gu, G. Sun et al., A scheduling strategy on load balancing of virtual machine resources in cloud computing environment, 2010 Third International Symposium on Parallel Architectures, Algorithms and Programming (PAAP), Dec. 18-20, 2010, 89-96

[11] J. Kleinberg, E. Tardos, Algorithm Design, Pearson Education Inc., 2005

[12] K. Li, G. Xu, G. Zhao et al., Cloud task scheduling based on load balancing ant colony optimization, 2011 Sixth Annual ChinaGrid Conference, 2011, 3-9

[13] A. Singh, M. Korupolu, D. Mohapatra, Server-storage virtualization: Integration and load balancing in data centers, International Conference for High Performance Computing, Networking, Storage and Analysis, 2008

[14] X. Sun, P. Xu, K. Shuang et al., Multi-dimensional aware scheduling for co-optimizing utilization in data center, China Communications, 8(6), 2011, 19-27

[15] W. Tian, Adaptive dimensioning of cloud data centers, Proceeding of the 8th IEEE International Conference on Dependable, Automatic and Secure Computing, DACS 2009, 2009

[16] W. Tian, C. Jing, J. Hu, Analysis of resource allocation and scheduling policies in Cloud datacenter, Proceedings of the IEEE 3rd International Conference on Networks Security Wireless Communications and Trusted Computing, Mar. 2011

[17] B. Wickremasinghe et al., CloudAnalyst: A CloudSim-based tool for modelling and analysis of large scale cloud computing environments, Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA 2010), Perth, Australia, Apr. 20-23, 2010

[18] T. Wood et. al., Black-box and gray-box strategies for virtual machine migration, Proceedings of Symp. on Networked Systems Design and Implementation (NSDI), 2007

[19] W. Zhang, Research and Implementation of Elastic Network Service, PhD Dissertation, National University of Defense Technology, China, 2000 (in Chinese)

[20] H. Zheng, L. Zhou, J. Wu, Design and implementation of load balancing in web server cluster system, Journal of Nanjing University of Aeronautics & Astronautics, Vol. 38, No. 3, Jun. 2006

[21] Hebrew University, Experimental Systems Lab, www.cs.huji.ac.il/labs/parallel/workload, 2007

[22] Amazon, Amazon Elastic Compute Cloud, http://aws.amazon.com/ec2/, 2012