An Efficient Method for Maximizing Total Weights in Virtual Machines Allocation

Wenhong Tian, Jun Cao, Xingyang Wang, Minxian Xu, Yu Chen
University of Electronic Science and Technology of China Chengdu, China 610054 Email: tian_wenhong@uestc.edu.cn

***Abstract*—When allocating virtual machines (VMs) in data centers, weights such as profits and other benefits are associated with all VMs. This paper considers maximizing the total weight in VMs allocation. As virtualization widely adopted in Cloud computing, requests may only consume part of the total capacity of a single hardware resource (for example a physical machine), this requires a new model for maximizing the total weight. In this paper, for the first time we model this problem as shared interval scheduling for capacity proportional weight and propose an exact efficient algorithm for it with computational complexity O(n^2) where n is the number of jobs. The proposed method has good scalability and can be applied to maximize the total weight or related metrics in cloud computing.**

Keywords*—*Keywords*—*cloud computing; virtual machine allocation; maximize the total weight; weighted interval scheduling; weighted interval scheduling with capacity sharing

I. Introduction

Cloud computing is developing based on various recent advancements in virtualization, Grid computing, Web computing, utility computing and related technologies. Cloud computing provides both platforms and applications on demand through the Internet or intranet. Cloud computing allows the sharing, allocation and aggregation of software, computational and storage network resources on demand. Some of the key benefits of Cloud computing include the hiding and abstraction of complexity, virtualized resources and efficient use of distributed resources. Cloud computing is still considered in its infancy as there are many challenging issues to be resolved. In this paper, we focus on Infrastructure as a service (IaaS) in Cloud data centers. With large-scale application of Cloud computing, maximizing total weights becomes one of key factors for many service providers to be considered. We consider maximizing the total weights (or profits) of virtual machines allocation in Cloud data centers. For example, Amazon [8] offers two different types of services: on-demand and spot instances. On-demand instances are more expensive but have a fixed price. Spot instances are cheaper than on-demand instances. However the provider may terminate the spot instance prematurely depending on how the spot price changes. A third pricing option, called timed instances, is proposed in [7]. Timed instances have an a priori specified reservation time of fixed length. The scheduling algorithm uses the reservation time to colocate instances with similar expiration times. We adopt timed instances with reservation in the following discussion. As for real-time virtual machine scheduling, Kim et al. [4] provide a detailed discussion. Tian et al. [9] introduce dynamic load-balance of virtual machine allocation, Fig.1 shows an example of virtual machine (VM) requests with weights and timed instances where ID, s_i, f_i, c_i, w_i are the ID number, start-time, end-time, capacity request, and weight of the request respectively. Let us consider a physical machine (PM) with 2×68.4GB memory, 16 cores×3.25 units(where each CPU unit is equal to 1Ghz 2007 Intel Pentium processor [20]), 2×1690GB storage. There are virtual machines (VMs) with capacities 1/8, 1/4 and 1/2 of the total capacity of the given PM. As an example, a set of 6 VM requests are considered, they are vm1(1, 0, 6, 0.25,1), vm2(1,1, 4, 0.125,2), vm3(1,3, 6, 0.25,3), vm4(2, 3, 8, 0.5,4), vm5(2,4, 8, 0.25,5), vm6(2,5, 9, 0.25,6), where vm1(1,0, 6, 1, 0.25,1) means vm1 starts at time 0, finishes at the end of the slot 6th, has weight 1 and capacity requirement of 0.25 of the total capacity of the given PM (i.e., PM♯1). Other requests are in the similar fashion. This maximizing total weights problem can be modeled by

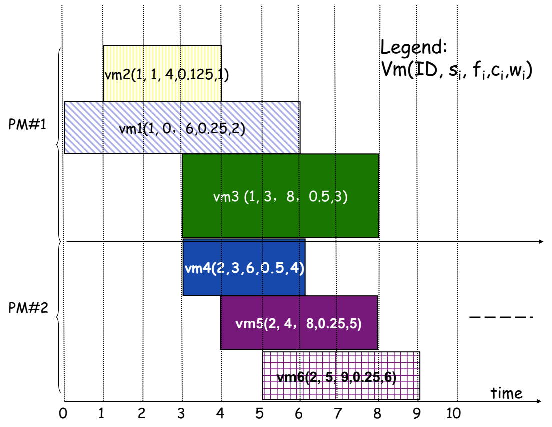


Fig. 1. An Example of VM requests

our proposed new model, weighted interval scheduling with capacity sharing (WISWCS). Interval scheduling problem (ISP) has been studied extensively for a long time, see reference [1] and references therein. It is a tradition to state in terms of machines and jobs for scheduling problems. The machines represent resources and the jobs (requests) represent tasks that need to be carried out using resources. The interval scheduling with fixed processing time is that each request has a fixed start and end time [2]. Similar to the paper [1], basic interval scheduling problem can be stated as follows. Given n intervals of the form $[s_j, f_j]$ with start-time $s_j < f_j$ (finish-time), for j=1, $\dots, n$. These intervals are the jobs that require uninterrupted processing during that interval. The objective of basic interval scheduling problem is to process all jobs using a minimum number of machines. In other words, finding an assignment of jobs to machines such that no two jobs assigned to the same machine overlap while using a minimum number of machines. Weighted interval scheduling problem (WISP) is that each request is associated with a weight, with the goal to find a subset of mutually compatible intervals with maximal total weight. In this paper, we consider scheduling algorithm for weighted interval scheduling with capacity sharing (WISWCS). The difference from WISP is that all intervals may require part of the total capacity of a single resource so that they can share the capacity if their total required capacity at any time does not surpass the total capacity a machine can provides. To the best of our knowledge, this problem is not studied in the open literature. The major contributions of this paper are:

- formulating a model for the weighted interval scheduling with capacity sharing for the first time.
- providing an exact scheduling algorithm and its complexity analysis for WISWCS problem.

II. Problem Formulation: Weighted Interval Scheduling with Capacity Sharing

A. Traditional Weighted Interval Scheduling Problem

A set of requests $1, 2, \ldots, n$ where the $i-th$ request corresponds to an interval of time starting at s_i and finishing at f_i, each request is associated with a weight w_i. The goal is to find a subset of mutually compatible intervals, as to maximize the sum of the values of the selected intervals. There are following assumptions:

1) all data are deterministic and unless otherwise specified,

0
1
2
3
4
5
6
7
8
9
10
...
k-2
k-1
k

Fig. 2. Time in slotted windows

the time is formatted in slotted windows as shown in Fig.2, we partition total time period $[0, T]$ into slots with equal length (s_0), the total slots is k=T/s_0, all are integer numbers. The starting time s_i and finishing time f_i are integer. 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 respectively at the 3rd-slot and 10th-slot, the actual duration of this request is (10-3)×5=35 minutes.

2) there are no precedence constraints other than those implied by the start and finishing time.

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

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

5) the weight of a job (denoted as w_i) is proportional to its required capacity (c_i) and its duration (f_i-s_i), i.e., w_i=$\alpha c_i(f_i - s_i)$.

Definition II.1. *compatible intervals for WIS: a subset of intervals is compatible if no two of them overlap in time, that is, either request i is for an earlier time interval than request $j(f_i < s_j)$, or request i is for a later time than request $j(f_j < s_i)$. More generally, a subset A of requested intervals is compatible if all pairs of requests $(i, j\ in\ A, i \neq j)$ are compatible.*

Definition II.2. *Weighted Interval Scheduling(WIS): In the weighted interval scheduling problem, we want to find the maximum-weight subset of non-overlapping jobs, given a set J of jobs that have weights associated with them. Job i in J has a start time s_i, a finish time f_i, and a weight w_i. Suppose we have a set of weighted intervals J=$\{I_1, I_2, I_3, \ldots, I_n\}$ and w_j is the weight of interval I_j. We seek to find an optimal schedule–a subset O of non-overlapping jobs in J with the maximum possible sum of weights. In other words, the goal is to choose intervals from J that don't overlap in time that gives the highest possible total weight.*

Note that when the weights are all 1, this problem is identical to basic interval scheduling problem (ISP), and for that, we know that a greedy algorithm that chooses jobs in order of earliest finish time first gives an optimal schedule [5]. For traditional WIS problem, classic dynamic programming (DP) approach provides efficient solution to find both optimal total weight and subset of intervals which are compatible, see [5] for example. The basic optimization model in dynamic programming is as the follows. After sorting all intervals in the non-decreasing finish time, consider the optimal total weight using following recursive formula for $j-th$ interval:

$$OPT(j) = max(w_j + OPT(p(j)), OPT(j-1)) \quad (1)$$

where $p(j)$ is the largest index $i < j$ such that intervals i and j are disjoint for an interval j, $OPT(j)$ is the optimal total weight for j intervals.

B. Weighted Interval Scheduling With Capacity Sharing (WISWCS)

Definition II.3. *weighted interval scheduling with capacity sharing (WISWCS): The only difference from traditional WIS is that a resource (to be concrete, a machine or a processor) can be shared by different jobs if the total capacity of all jobs allocated on the single source at any time does not surpass the total capacity of a resource can provides. A request can be represented in a victor $[ID,\ s_i,\ f_i,\ c_i,\ w_i]$ where $ID,\ s_i,\ f_i,\ c_i,\ w_i$ are the ID number, start-time, end-time, capacity request, and weight of the request respectively. The objective of WISWCS is to maximize total weight by accepting a subset of requests for a given number of machines.*

Definition II.4. *sharing compatible intervals for WISWCS: a subset of intervals which can maximally share the total capacity of a machine at any time.*

Definition II.5. *Divisible Capacity for WISWCS: The capacity of different requests (jobs) have follows feature:*
$c_1 > c_2 > \ldots > c_i > c_{i+1} > \ldots$
Such that for all $i \leq 1, c_{i+1}$ exactly divides c_i. There are a list L of requests (each can have arbitrary number), the capacity of requests in L form a divisible capacity. If L is a list of requests and C is the total capacity of a machine

(considering homogeneous case here), we say that the pair (L, C) is strongly divisible if in addition the largest item capacity c_1 in L exactly divides the total capacity C.

See paper [3] for more detailed discussion about divisible size bin-packing. We observe that popular providers, such as Amazon and Google, have a small and finite set of instance sizes following divisible capacity pattern.Knauth et al.[7] also introduce the similar idea. In [7], virtual machines, as rented out to customers, have fractional sizes of the original hardware, e.g., 1/8, 1/4, 1/2, or 1. Individual resources of a VM, such as CPU, RAM, and local disk, double between VM sizes. For example, a small instance may have one CPU, 1 GB RAM, and 100 GB local disk. The next instance size has 2 CPUs, 2 GB RAM, and 200 GB local disk, so on. These settings are justified by real providers such as Amazon and Google. Note that, if all requests demand one-unit capacity (unit-size) from the total capacity of a PM, then it satisfies the divisible capacity requirement.

Definition II.6. *Capacity-duration proportional weight for WISWCS: We assume that the weight of a request is proportional to the product of its capacity and duration.*

Definition II.6 is a reasonable assumption in many literatures, and important assumption for our main results. For WISWCS with divisible capacity and capacity proportional weight, we seek to find an optimal schedule–a subset O of sharing compatible intervals (jobs) with the maximum possible sum of weights. Notice that WIS is the special case of WISWCS when all weights are equal to the total capacity of a machine. Therefore WISWCS problem is more difficult. An example for WISWCS problem is shown in Fig 3. Unfortunately, in this case of WISWCS, we cannot

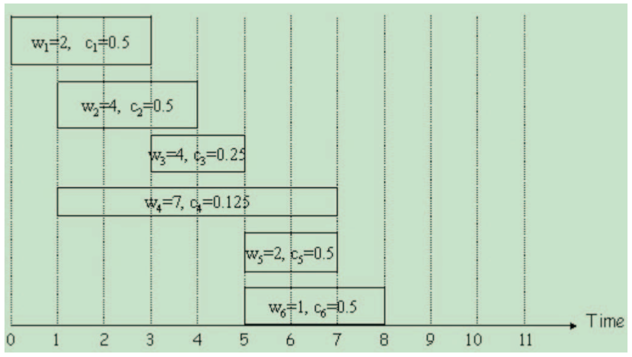


Fig. 3. An Example of WISWCS

use dynamic programming technique any more. The reason is that the situation of sharing compatible intervals makes WISWCS different from WIS, we cannot use compatible intervals defined by WIS to recursively apply memorization technique in dynamic programming to find optimal solution.

C. Problem Formation of Maximizing Profits for WISWCS

For each VM request, a weight (profit) is associated with it. For example it can be expressed as follows:

$$w_i = price(i) - cost(i) = \alpha c_i (f_i - s_i), \tag{2}$$

which can be further simplified as a value proportional to the product of its capacity and duration, also called capacity-duration. Then the maximizing weight problem (MWP) can be formulated as follows:

$$\textbf{Maximize} \quad \sum w_i \tag{3}$$

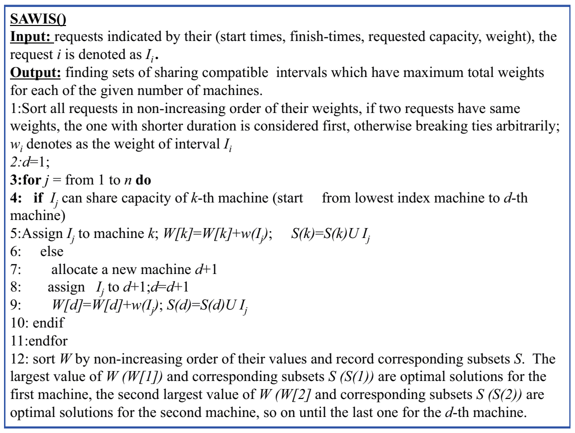
$$\textbf{subject to 1).} \ \forall \ slot \ j, \sum_{VM_j \in PM_i} d_j \leq g_i \tag{4}$$

$$\textbf{2).} \ \forall VM_j, \ s_j \ and \ f_j \ is \ given \ by \ reservation. \tag{5}$$

where g_i is the total capacity of PM i; Eqn (4) states the capacity constraint and (5) is for fixed interval constraint of each request.

D. An Exact Scheduling Algorithm for Weighted Interval Scheduling With Capacity Sharing

In the following, we introduce an exact scheduling algorithm for WISWCS. The algorithm is shown in Fig.4.



```
SAWIS()
Input: requests indicated by their (start times, finish-times, requested capacity, weight), the request $i$ is denoted as $I_i$.
Output: finding sets of sharing compatible intervals which have maximum total weights for each of the given number of machines.
1:Sort all requests in non-increasing order of their weights, if two requests have same weights, the one with shorter duration is considered first, otherwise breaking ties arbitrarily; $w_i$ denotes as the weight of interval $I_i$
2:$d$=1;
3:for $j$ = from 1 to $n$ do
4:  if $I_j$ can share capacity of $k$-th machine (start from lowest index machine to $d$-th machine)
5:Assign $I_j$ to machine $k$; $W[k]=W[k]+w(I_j)$;   $S(k)=S(k) U I_j$
6:    else
7:       allocate a new machine $d$+1
8:      assign  $I_j$ to $d$+1;$d$=$d$+1
9:       $W[d]=W[d]+w(I_j)$; $S(d)=S(d) U I_j$
10: endif
11:endfor
12: sort $W$ by non-increasing order of their values and record corresponding subsets $S$. The largest value of $W$ ($W[1]$) and corresponding subsets $S$ ($S(1)$) are optimal solutions for the first machine, the second largest value of $W$ ($W[2]$ and corresponding subsets $S$ ($S(2)$) are optimal solutions for the second machine, so on until the last one for the $d$-th machine.
```

Fig. 4. Our proposed scheduling algorithm SAWIS

Theorem II.7. *Algorithm SAWIS correctly finds the optimal solution for a subset of sharing compatible intervals (jobs) with the maximum possible sum of weights.*

Proof: The algorithm SAWIS as shown in Fig.4 firstly sorts all requests in non-increasing order of their weights, this guarantees that requests with larger weights are considered first; then the algorithm applies sharing compatible rule (Definition II.3) for all requests as shown in line 1 of Fig.4, this makes assure that all possible requests are included in the optimal solution if they are sharing compatible as shown in line 2-11; finally, the algorithm finds optimal results by comparing total weights of each machine as shown in line 12. By the definition of the objective of WISWCS, the algorithm finds the optimal solution for a subset of sharing compatible intervals (jobs) with the maximum possible sum of weights. ■

For better understanding, let us take the example shown in Fig.3 to see how algorithm SAWIS works for WISWCS:

1). Sort all requests in non-increasing order of their weights, we have $I_4(w_4$=7), $I_3(w_3$=4), $I_2(w_2$=4), $I_1(w_1$=2), $I_5(w_5$=2), $I_6(w_6$=1);

2). for j=1, I_4 with weight w_4=7 and capacity c_4=0.125 is considered, it is allocated to the first (d=1) machine;W[1]=w_4=7, S(1)={I_4};

3). for j=2, I_3 with w_3=4 (shorter duration than I_2) and capacity c_3=0.25 is selected, it is allocated to the first machine since it is sharable compatible with I_4, W[1]=w_4+w_3=I_1, S(1)={I_4, I_3};

4). for j=3, I_2 with w_2=4 and capacity c_2=0.5 is selected, it is allocated to the first machine since it can share the capacity, so W[1]=w_3+w_4+w_2=i_5, S(1)={I_3, I_4, I_2};

5). for j=4, I_5 with w_5=2 (shorter duration than I_1) and capacity c_5=0.5 is selected, it can share the capacity of machine 1 with existing intervals, so it is allocated to machine 1, W[1]=w_3+w_4+w_2+w_5=17, S(1)={I_3, I_4, I_2, I_5};

6). for j=5, I_1 with w_1=2 and capacity c_1=0.5 is selected, it cannot share the capacity of machine 1 with other existing intervals, so d=1+1=2 is allocated for it, W[2]=w_1=2, S(2)={I_1};

7). for j=6, I_6 with w_1=1 and capacity c_6=0.5 is selected, it cannot share capacity with machine 1 but can share capacity of machine 2, so it is allocated to machine 2, W[2]=w_1+w_6=3, S(2)={I_1, I_6}. From above steps, it is shown that the optimal subset is {I_3, I_4, I_2, I_5}, with total weight 17.

Theorem II.8. *the time complexity of SAWIS algorithm as shown in Fig.4 is $O(nd)$ where n is the number of requests (jobs) and d is the number of machines.*

Proof: As shown in Fig.4, the algorithm firstly sorts all intervals in non-increasing order of their weights (if two requests have same weights, the one with shorter duration is considered first, otherwise breaking ties arbitrarily), this takes $O(nlogn)$ time where n is the number of intervals (requests). Then the algorithm finds sharing compatible intervals for all intervals as shown in line 6 to line 12, this takes $O(nd)$ steps in worst case. The worst case is that all intervals have largest required capacity, same start-time and finish-time. Then all intervals are not sharing compatible, therefore finding a machine for a job to allocate needs $O(d)$ steps, n intervals need $O(nd)$ steps. Finally the algorithm find optimal solution using a simple comparison with costs $O(dlogd)$ time. So all together the algorithm for WISWCS takes $O(nd)$ time where normally $n > d$. ■

For implementation of SAWIS, interval tree data structure can be used. An interval tree is an ordered tree data structure to hold intervals. It allows one to efficiently find all intervals that overlap with any given interval or point. The trivial or traditional solution (for example using arrays) is to visit each interval and test whether it intersects the given point or interval, which requires $O(n^2)$ time or higher, where n is the number of intervals in the collection. Interval trees are dynamic, i.e., they allow insertion and deletion of intervals. They obtain a query time of $O(logn)$ while the preprocessing time to construct the data structure has tight bound $O(nlogn)$, see for example Cormen et al. [6].

Remarks: from algorithm SAWIS, Theorem II.7 and Theorem II.8 we knows that:

- 1) if the total number of machines needed is d and d=m (m is the total available resources), then algorithm SAWIS can find optimal solutions for all requests by using d resources;
- 2) if there are $m < d$ available resources, algorithm SAWIS also can find optimal solutions for all requests, it sorts W by non-increasing order of their values and records corresponding subsets S. The largest value of W and corresponding subsets are the optimal solutions for the first machine, the second largest value of W and corresponding subsets are the optimal solutions for the second machine, so on until the last one for the d-th machine.
- 3) if there is only single resource, SAWIS in the line 12 of Fig.4 uses a simple comparison to find the largest value of W and corresponding subsets S, which are optimal solutions for the single resource (machine).

III. Conclusion and Future Work

In this paper a new algorithm for weighted interval scheduling with capacity sharing (WISWCS) is proposed for the first time by considering divisible capacity and Capacity-duration proportional weight. It is interesting to notice that the proposed algorithm works for both single machine and multiple machines cases. Our future work will investigate scheduling problem where certain time of delay is allowed for a number of requests, also other cases than divisible capacity, Capacity-duration proportional weight, will be extended.

Acknowledgment

This research is partially supported by the National Natural Science Foundation of China (NSFC) Grant 61150110486 and Central University Fund (No.ZYGX2012J073).

References

[1] KOLEN,A.W.J., LENSTRA, J. K., PAPADIMITRIOU, C. H., SPIEKSMA F.C.R., Interval Scheduling: A Survey, Published online 16 March 2007 in Wiley InterScience (www.interscience.wiley.com).

[2] ARKIN E.M., and SILVERGERG, E.B., Scheduling with fixed start and end times, Discrete Appl Math 18

[3] COFFMAN E.G.Jr., GRAREY, M.R. , and JOHNSON, D.S., Bin-packing with Divisible Item Sizes, J. Complexity 3 (1987), 406-428.

[4] KIM, KH., BELOGLAZOV, A., BUYYA, R., Power-aware provisioning of virtual machines for real-time cloud services.

[5] KLEINBERG, J., TARDOS,É., Algorithm Design, Pearson Education Inc., 2005.

[6] CORMEN,T. H., LEISERSON, C. E., RIVERST, R. L., STEIN, C., Introduction to Algorithms (2nd ed.), MIT Press and McGraw-Hill, ISBN 0-262-03293-7. 2001.

[7] KNAUTH, T., FETZER, C., Energy-aware Scheduling for Infras- tructure Clouds, In the proceedings of CloudCom 2012.

[8] Amazon EC2, http://aws.amazon.com/ec2/, 2013.

[9] TIAN, WH., ZHAO, Y., ZHONG, YL., XU, MX., JING, C., Dynamic and Integrated Load-Balancing Scheduling Algorithm for Cloud Data Centers, China Communications, 2011, Vol.8, Issue 6,117-126.

[10] TIAN WH., SUN, XS., JIANG, YQ., WANG, HY., CRESS: A Platform of Infrastructure Resource Sharing for Educational Cloud Computing, China Communications, 2013, Vol.10, Issue 9,43-52.