J. Parallel Distrib. Comput. 74 (2014) 2304–2309

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Research note

Online vector scheduling and generalized load balancing

Xiaojun Zhu [a,*], Qun Li [b], Weizhen Mao [b], Guihai Chen [a,c]

[a] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China
[b] Department of Computer Science, College of William and Mary, Williamsburg, Virginia 23187, USA
[c] Shanghai Key Laboratory of Scalable Computing and Systems, Shanghai Jiao Tong University, Shanghai 200240, China

HIGHLIGHTS

- We prove that vector scheduling is a special case of generalized load balancing.
- We give the first non-trivial online algorithm for vector scheduling based on the proof.
- We show that generalized load balancing does not admit constant approximation algorithms unless $P = NP$.

ARTICLE INFO

Article history:
Received 16 March 2013
Received in revised form
25 September 2013
Accepted 5 December 2013
Available online 16 December 2013

Keywords:
Online algorithm
Vector scheduling
Load balancing
Approximation algorithm

ABSTRACT

We give a polynomial time reduction from the vector scheduling problem (VS) to the generalized load balancing problem (GLB). This reduction gives the first non-trivial online algorithm for VS where vectors come in an online fashion. The online algorithm is very simple in that each vector only needs to minimize the $L_{\ln(md)}$ norm of the resulting load when it comes, where m is the number of partitions and d is the dimension of vectors. It has an approximation bound of $e\log(md)$, which is in $O(\ln(md))$, so it also improves the $O(\ln^2 d)$ bound of the existing polynomial time algorithm for VS. Additionally, the reduction shows that GLB does not have constant approximation algorithms that run in polynomial time unless $P = NP$.

1. Introduction

Scheduling with costs is a very well studied problem in combinatorial optimization. The traditional paradigm assumes a single-cost scenario: each job incurs a single cost to the machine that it is assigned to. The *load* of a machine is the total cost incurred by the jobs it serves. The objective is to minimize the *makespan*, the maximum machine load. Vector scheduling and generalized load balancing extend the scenario in different directions.

Vector scheduling assumes that each job incurs a vector cost to the machine that it is assigned to. The load of a machine is defined as the maximum cost among all dimensions. The objective is to minimize the makespan. Vector scheduling is a multi-dimensional generalization of the traditional paradigm. It finds application in multi-dimensional resource scheduling in parallel query optimization [2]. For example, a task may have requirements for CPU, memory and network at the same time, and this requirement is best described by a vector of CPU, memory and network, instead of an aggregate measure. In this scenario, the load of a server is also described by a vector. To solve vector scheduling, there are three approximation solutions [2]. Two of them are deterministic algorithms based on derandomization of a randomized algorithm, with one providing an $O(\ln^2 d)$ approximation,[1] where d is the dimension of vectors, and the other providing an $O(\ln d)$ approximation with running time polynomial in n^d, where n is the number of vectors. The third algorithm is a randomized algorithm, which assigns each vector to a uniformly and randomly chosen partition. It gives an $O(\ln dm/\ln\ln dm)$ approximation with high probability, where m is the number of partitions (servers). For fixed d, there exists a polynomial time approximation scheme (PTAS) [2]. A PTAS has also been proposed for a wide class of cost functions (rather than max) [3].

Generalized load balancing was recently introduced to model the effect of wireless interference [5,6]. Each job incurs costs to all machines, no matter which machine it is assigned to. The exact cost incurred by a job to a specific machine is dependent on which machine the job is assigned to. The load of a machine

* Corresponding author.
E-mail addresses: gxjzhu@gmail.com (X. Zhu), liqun@cs.wm.edu (Q. Li), wm@cs.wm.edu (W. Mao), gchen@nju.edu.cn (G. Chen).

[1] In this paper, e denotes the natural number, $\ln(\cdot)$ denotes the natural logarithm, and $\log(\cdot)$ denotes the logarithm base 2.

0743-7315/$ – see front matter
http://dx.doi.org/10.1016/j.jpdc.2013.12.006

is the total cost incurred by all the jobs, instead of just the jobs it serves. This model is well suited for wireless transmission, since, in a wireless network, a user may influence all APs in its transmission range due to the broadcast nature of the wireless signal. To solve the generalized load balancing problem, the current solution is an online algorithm, adapted from recent progress in online scheduling on traditional models [1]. The solution, though it provides a good approximation, is rather simple: each job selects the machine to minimize the L_τ norm of the resulting loads at all machines, where τ is a constant parameter to be optimized. To avoid confusion, we keep the two terms *job* and *machine* unchanged for generalized load balancing, while we refer to job and machine in the vector scheduling model as *vector* and *partition* respectively.

We make two contributions. First, we present an approach to encode any vector scheduling instance by an instance of the generalized load balancing problem (Section 2). This encoding method directly shows that the generalized load balancing problem does not admit constant approximation algorithms unless $P = \text{NP}$. Second, we design the first non-trivial online algorithm for vector scheduling based on the encoding method (Section 3). Directly applying the encoding method does not necessarily lead to a polynomial time algorithm, because it needs to compute the $L_{\ln l}$ norm function (l is the number of machines), and it is unclear whether this norm can be computed in polynomial time. We eliminate this uncertainty by rounding $\ln l$ to the next integer, guaranteeing polynomial running time. In addition, we prove that the approximation loss due to rounding is small. We conclude this section by giving the following two definitions.

1.1. Vector scheduling

We are given positive integers n, d, m. There are a set $\mathcal{V}$ of n rational and d-dimensional vectors $p_1, p_2, \ldots, p_n$ from $[0, \infty)^d$. Denote vector $p_i = (p_{i1}, \ldots, p_{id})$. We need to partition the vectors in $\mathcal{V}$ into m sets $A_1, \ldots, A_m$. The problem is to find a partition to minimize $\max_{1\le i\le m} \|\overline{A_i}\|_\infty$, where $\overline{A_i} = \sum_{j\in A_i} p_j$ is the sum of the vectors in A_i, and $\|\overline{A_i}\|_\infty$ is the infinity norm defined as the maximum element in the vector $\overline{A_i}$. For the case $m \ge n$, there is a trivial optimal solution that assigns vectors to distinct partitions. Therefore, we consider only the case $m < n$.

For ease of presentation, we give an equivalent integer program formulation. Let x_{ij} be the indicator variable such that $x_{ij} = 1$ if and only if vector p_i is assigned to partition A_j. Then

$$\|\overline{A_j}\|_\infty = \max_{1\le k\le d} \sum_i x_{ij} p_{ik}. \tag{1}$$

The vector scheduling problem can be rewritten as

$$\begin{aligned} \min_{\mathbf{x}} \max_{j,k} \quad & \sum_i x_{ij} p_{ik} \\ \text{subject to} \quad & \\ & \sum_j x_{ij} = 1, \quad \forall i \\ & x_{ij} \in \{0, 1\}, \quad \forall i, j. \end{aligned} \tag{VS}$$

1.2. Generalized load balancing

This formulation first appears in [6]. We reformulate it with slightly different notations. There are a set $\mathcal{M}$ of independent machines, and a set $\mathcal{J}$ of jobs. If job i is assigned to machine j, there is non-negative cost c_{ijk} to machine k. The *load* of a machine is defined as its total cost. The problem is to find an assignment (or schedule) to minimize the *makespan*, the maximum load of all the machines. This problem can be formally defined as follows.

$$\begin{aligned} \min_{\mathbf{x}} \max_{k} \quad & \sum_{i,j} x_{ij} c_{ijk} \\ \text{subject to} \quad & \\ & \sum_j x_{ij} = 1, \quad \forall i \in \mathcal{J} \\ & x_{ij} \in \{0, 1\}, \quad \forall i \in \mathcal{J},\ j \in \mathcal{M} \end{aligned} \tag{GLB}$$

where $\mathbf{x} \in \{0, 1\}^{|\mathcal{J}|\times|\mathcal{M}|}$ is the assignment matrix with elements $x_{ij} = 1$ if and only if job i is assigned to machine j. The two constraints require each job to be assigned to one machine.

2. Encoding vector scheduling by generalized load balancing

We first create a GLB instance for any VS instance, then prove their equivalence. At last, we discuss the hardness of GLB and extend the VS model.

2.1. Creating GLB instances

Comparing VS to GLB, we can find that they mainly differ in the subscripts of max and $\sum$. Our construction is inspired by this observation.

Given as input to VS the vector set $\mathcal{V}$ and m partitions, we construct the GLB instance as follows. We set the jobs $\mathcal{J} = \mathcal{V}$. For each partition A_j and its k-th dimension, we construct a machine with a label of (j, k). Thus, the constructed machine set $\mathcal{M}$ is $\{(j, k) \mid j = 1, 2, \ldots, m \text{ and } k = 1, 2, \ldots, d\}$. Here we refer to a machine as a pair of indices so that we can map the machine back to its corresponding partition and dimension easily. For a machine $t = (j, k)$, where $t \in \mathcal{M}$, we refer to the partition j as t_1, and the dimension k as t_2, i.e., $t = (t_1, t_2)$. We can see that there are totally d machines (t included) corresponding to the same partition as the machine t. We denote $[t]$ as the set of these machines, i.e., $[t] = \{(j, 1), (j, 2), \ldots, (j, d)\}$, where $j = t_1$. Among these d machines, we select the first one $(j, 1)$ as the *anchor machine*, denoted by $\bar{t}$, such that a vector chooses partition A_j in VS if and only if the corresponding job chooses $\bar{t}$ in the new GLB problem.

The incurred cost c_{ist} of job i to machine t if i chooses machine s is defined as

$$c_{ist} = \begin{cases} p_{it_2} & \text{if } s = \bar{t} \quad \text{(a)} \\ \infty & \text{if } s \in [t] \wedge s \ne \bar{t} \quad \text{(b)} \\ 0 & \text{if } s \notin [t] \quad \text{(c)} \end{cases} \tag{2}$$

where (2)(a), (2)(b) are for the situation where s and t correspond to the same partition. They force a job to select only the anchor machines. (2)(c) is for the situation where s and t correspond to different partitions. In this case, there is no load increase.

The resulting GLB instance is defined as VS–GLB:

$$\begin{aligned} \min_{\mathbf{x}'} \max_{t} \quad & \sum_{i,s} x'_{is} c_{ist} \\ \text{subject to} \quad & \\ & \sum_s x'_{is} = 1, \quad \forall i \in \mathcal{J} \\ & x'_{is} \in \{0, 1\}, \quad \forall i \in \mathcal{J}, s \in \mathcal{M}. \end{aligned} \tag{VS–GLB}$$

To avoid confusion with the general GLB problem, we intentionally use different notations $\mathbf{x}'$, s and t. The notation i is kept since it is in 1–1 correspondence with the vectors in VS.

As an example, consider the case when $d = 1$. All vectors in VS have only one element, and there is only one machine in VS–GLB representing a partition in VS. The objective of VS becomes $\max_j \sum_i x_{ij} p_{i1}$. On the other hand, the objective of VS–GLB is $\max_t \sum_i x'_{it} c_{itt} = \max_t \sum_i x'_{it} p_{i1}$. Since any machine t corresponds to a distinct partition A_j, simply changing subscripts shows that the two problems are equivalent. For the case when $d > 1$, the proof is much involved, which we delay to Section 2.2.

Theorem 1. *The construction of VS–GLB can be done in polynomial time.*

Proof. An instance of VS needs $\Omega(nd)$ bits. The constructed VS–GLB instance has n jobs, md machines and $n(md)^2$ costs. Since $m < n$, all three terms are polynomials in n and d. The theorem follows immediately. □

The following theorem shows that the constructed VS–GLB problem is equivalent to its corresponding VS problem. Let T be a positive constant.

Theorem 2. *There is a feasible solution* $\mathbf{x}$ *to VS with objective value* T *if and only if there is a feasible solution* $\mathbf{x}'$ *to VS–GLB with the same objective value* T.

This theorem shows that VS and its corresponding VS–GLB have the same optimal value. In addition, any c-approximation solution to VS–GLB, after transformation, is also a c-approximation solution to VS, and vice versa.

2.2. Proof of equivalence

We first study the properties of feasible solutions to VS–GLB in Lemmas 1 and 2, and then prove Theorem 2.

Lemma 1. *Given a feasible solution* $\mathbf{x}'$ *to VS–GLB yielding objective value* T, *for any* $i \in \mathcal{I}$, *we have*

1. $\forall s \neq \bar{s}, x'_{is} = 0$;
2. $\exists j$ *such that for* $s = (j, 1), x'_{is} = 1$.

Proof. For (1), suppose $x'_{is} = 1$ for some s with $s \neq \bar{s}$. Then $x'_{is} c_{is\bar{s}} = \infty > T$, contradicting with $\max_t \sum_{i,s} x_{is} c_{ist} = T$. For (2), since $\sum_s x'_{is} = 1$, there exists some s such that $x'_{is} = 1$. Due to (1), we must have $s = \bar{s}$. □

Lemma 2. *Given a machine* t, *a job* i, *and a feasible solution* $\mathbf{x}'$ *to VS–GLB yielding objective value* T, *we have* $\sum_s x'_{is} c_{ist} = x'_{i\bar{t}} p_{it_2}$.

Proof. Recall that $[t] = \{(t_1, 1), (t_1, 2), \ldots, (t_1, d)\}$. We have

$$\sum_s x'_{is} c_{ist} = \sum_{s\in[t]} x'_{is} c_{ist} + \sum_{s\notin[t]} x'_{is} c_{ist} = \sum_{s\in[t]} x'_{is} c_{ist} = x'_{i\bar{t}} c_{i\bar{t}t} = x'_{i\bar{t}} p_{it_2} \quad (3)$$

where the second equality is due to (2)(c), the third is due to Lemma 1, and the fourth is due to (2)(a). □

With the two lemmas, we can now prove the equivalence.

Proof of Theorem 2. "$\Longrightarrow$" Given a feasible solution $\mathbf{x}$ to VS, construct a feasible solution $\mathbf{x}'$ to VS–GLB as follows. Set $x'_{i\bar{s}} = x_{is_1}$ and all others to be 0. We first show that $\mathbf{x}'$ is a feasible solution to VS–GLB. Obviously, $\mathbf{x}'$ is an integer assignment. We will check that $\sum_s x'_{is} = 1$. Observe that $x'_{is} = 0$ if $s \neq \bar{s}$. We only need to consider m machines $(1, 1), (2, 1), \ldots, (m, 1)$. Since $\mathbf{x}$ is a feasible solution to VS, then for any $i \in \mathcal{V}$, there exists one and only one partition A_j such that $x_{ij} = 1$. Our transformation sets $x'_{is} = 1$, where $s = (j, 1)$. So $\sum_s x'_{is} = 1$.

Second, the objective values of the two feasible solutions are equal.

$$\max_t \sum_{i,s} x'_{is} c_{ist} = \max_t \sum_{\substack{i\\ s\in[t]}} x'_{is} c_{ist} = \max_t \sum_i x'_{i\bar{t}} c_{i\bar{t}t} = \max_t \sum_i x_{it_1} p_{it_2} = \max_{j,k} \sum_i x_{ij} p_{ik} \quad (4)$$

where the first equality is due to c_{ist} being equal to 0 if $s \notin [t]$, and the second is due to our assignment of $\mathbf{x}'$ that $x'_{is} = 0$ if $s \neq \bar{s}$.

"$\Longleftarrow$" Given $\mathbf{x}'$ for VS–GLB, construct $\mathbf{x}$ for VS as follows. Set $x_{ij} = x'_{i\bar{s}}$ where $s_1 = j$. We show that $\mathbf{x}$ is a feasible solution to VS. Due to Lemma 1, for any i, there exists one s such that $x'_{is} = 1$ and $s = \bar{s}$. Therefore, there exists one j such that $x_{ij} = 1$. On the other hand, there cannot be two js both with $x_{ij} = 1$, otherwise $\mathbf{x}'$ is not a feasible solution to VS–GLB.

For the objective value, we have

$$\max_{j,k} \sum_i x_{ij} p_{ik} = \max_{t=(j,k)} \sum_i x_{it_1} p_{it_2} = \max_t \sum_i x'_{i\bar{t}} p_{it_2} = \max_t \sum_{i,s} x'_{is} c_{ist} \quad (5)$$

where the last equality is due to Lemma 2. This completes our proof. □

2.3. Inapproximability result for GLB

The encoding method implies the following hardness result for GLB.

Theorem 3. *For any constant* $c > 1$, *there does not exist a polynomial time* c*-approximation algorithm for GLB, unless* $P = \text{NP}$.

Proof. It has been proved that no polynomial time algorithm can give a c-approximation solution to VS for any $c > 1$ unless NP = ZPP [2]. The result also holds if we replace "NP = ZPP" by "P = NP". Because, the inapproximability proof in Ref. [2] relies on the result that no polynomial time algorithm can approximate the chromatic number problem to within $n^{1-\epsilon}$ for any $\epsilon > 0$ if NP ≠ ZPP, but recently, this assumption of NP ≠ ZPP has been relaxed to $P \neq$ NP [7].

In addition, by Theorem 2, any c-approximation algorithm for GLB can be transformed to a c-approximation algorithm for VS. Combining the two results proves the theorem. □

2.4. Extending to generalized VS

Our construction of VS–GLB and its proof can be easily extended to a general version of vector scheduling. In the current VS definition, all machines (partitions) are identical, so that any job (vector) incurs the same vector cost to all machines. The machines can be generalized to be heterogeneous, so that each job incurs a different vector cost to different machines. Formally, job i incurs vector cost $p_i^{(j)}$ to machine j if i is assigned to machine j. The formulation and transformation can be slightly changed as follows. In the integer program formulation of VS, change the objective to $\max_{j,k} \sum_i x_{ij} p_{ik}^{(j)}$. Change p_{it_2} in Eq. (2)(a) to be $p_{it_2}^{(t_1)}$. For Lemma 2, change $x'_{i\bar{t}} p_{it_2}$ to $x'_{i\bar{t}} p_{it_2}^{(t_1)}$. It can be verified that the proof of Theorem 2 is still valid with minor modifications. The online algorithm adopted later is also valid for this general version of vector scheduling. For simplicity, we mainly focus on the original VS model.

3. Online algorithm for VS

Based on Theorem 2, we can solve VS by its corresponding VS–GLB. We review the approximation algorithm [5] for GLB, and then modify it to solve VS.

Given a GLB instance and a positive number τ, the algorithm [5] considers jobs one by one (in an arbitrary order) and assigns the current job to a machine to minimize the L_τ norm[2] of the resulting

[2] The L_τ norm of a vector $x = (x_1, x_2, \ldots, x_t)$ is defined as $(\sum_i x_i^\tau)^{1/\tau}$.

load of all machines. Specifically, suppose jobs are numbered as $1, 2, \ldots, n$, the same as the considered order. Suppose the load of machine k after jobs $1, 2, \ldots, i-1$ are assigned is l_k^{i-1}. Then job i is assigned to the machine

$$\arg\min_j \left(\sum_k (l_k^{i-1} + c_{ijk})^\tau \right)^{1/\tau}. \tag{6}$$

The above optimization problem can be solved by trying each possible machine. During the optimization, the computation of the last step of the L_τ norm, $(\cdot)^{1/\tau}$, can be omitted. In addition, because the algorithm does not require the order of jobs and each job is assigned once, it can be implemented in an online fashion. This algorithm was originally proposed for the traditional load balancing problem [1], and recently extended to the GLB problem [5]. The parameter τ controls the approximation ratio of the algorithm, as shown in the following lemma.

Lemma 3 (*[1,5]*). *Minimizing the L_τ norm gives a $\frac{\tau}{\ln(2)} l^{1/\tau}$ approximation ratio to solve GLB, where l is the number of machines.*

Setting $\tau = \ln l$ yields the best approximation ratio $e \log l$. However, it is unclear whether the computation of $L_{\ln l}$ can be done in polynomial time. We consider this issue later.

3.1. Adapting to VS

To apply the above algorithm to VS, we can first solve VS–GLB and transform the solution to VS. This process can be simplified by omitting the transformation between VS and VS–GLB.

Recall that the algorithm is to assign vectors one by one. Consider a vector p_i in VS. To solve VS–GLB, this vector should choose a machine to minimize the L_τ norm of the resulting load. Due to the construction of VS–GLB, this vector can only choose from the *anchor machines*, otherwise, the resulting L_τ norm would be infinite (definitely not the optimal choice). Thus, this is equivalent to picking from the corresponding partitions in VS. After the assignment of any number of vectors that leads to partitions $A_1, A_2, \ldots, A_m$, the L_τ norm of the load of machines in VS–GLB is, in fact, equal to

$$f^{(\tau)}(A_1, \ldots, A_m) = \left(\|\bar{A}_1\|_\tau^\tau + \cdots + \|\bar{A}_m\|_\tau^\tau \right)^{1/\tau} \tag{7}$$

where

$$\|\bar{A}_j\|_\tau^\tau = \sum_k \left(\sum_{i \in A_j} p_{ik} \right)^\tau. \tag{8}$$

Suppose the assignment of vectors $p_1, p_2, \ldots, p_{i-1}$ leads to partitions $A_1, A_2, \ldots, A_m$. Let $f_{i,j}^{(\tau)}$ be the L_τ norm of the resulting load if vector p_i chooses partition A_j, i.e.,

$$f_{i,j}^{(\tau)} = f^{(\tau)}(A_1, \ldots, A_j \cup \{p_i\}, \ldots, A_m). \tag{9}$$

Then, according to the algorithm, vector p_i should be assigned to the partition

$$\arg\min_j f_{i,j}^{(\tau)}. \tag{10}$$

The procedure is described in Algorithm 1. For each incoming vector, it only needs to execute Lines 5–9.

Algorithm 1 with $\tau = \ln(md)$ is an $e \log(md)$ approximation algorithm to solve the corresponding VS–GLB. Thus, we have the following result due to Theorem 2.

Lemma 4. *Algorithm 1 with $\tau = \ln(md)$ is an $e \log(md)$ approximation algorithm to solve VS.*

Algorithm 1: Vector Scheduling

Input: m, the number of partitions; d, the dimension of each vector; $p_1, p_2, \ldots, p_n$, the n vectors to be scheduled; τ, the norm
Output: $A_1, \ldots, A_m$, the m partitions

```
1  begin
2      for j from 1 to m do
3          A_j ⟵ ∅;
4      for i from 1 to n do
5          if ∃A_j, A_j = ∅ then
6              A_j ⟵ A_j ⋃ {p_i};
7          else
8              find j to minimize f_{i,j}^(τ);
9              A_j ⟵ A_j ⋃ {p_i};
10 end
```

However, it is unclear whether Algorithm 1 with $\tau = \ln(md)$ can terminate within polynomial time. The algorithm requires the computation of $x^{\ln(md)}$ for some x. First, the number $\ln(md)$ is irrational, thus cannot be represented by polynomial bits to achieve arbitrary resolution. Second, even though we can approximate it by a rational number with acceptable resolution, the number $x^{\tilde{\tau}}$ may still be irrational, where $\tilde{\tau}$ is the rational approximation to τ. For example, when $\tilde{\tau} = 1.5$, there are lots of values of x such that $x^{1.5}$ are irrational. Though we can still approximate it by a rational number, it is complicated to theoretically analyze whether the approximation ratio still holds and how the running time increases with respect to rational number approximation accuracy. This problem has not been addressed in the literature.

Our solution is to round $\ln(md)$ to the next integer $\lceil \ln(md) \rceil$ and compute the $L_{\lceil \ln(md) \rceil}$ norm. This guarantees polynomial running time, but causes the loss of the approximation ratio. We show in the following that the loss is very small.

3.2. Guaranteeing polynomial running time

To deal with the irrational number issue, we round $\ln(md)$ to the next integer $\lceil \ln(md) \rceil$. In the following, we analyze the resulting approximation ratio.

Theorem 4. *Let l be the number of machines. Minimizing the $L_{\lceil \ln l \rceil}$ norm gives an $e \log(l) + \frac{e \log(e)}{\ln l + 1}$ approximation ratio to solve GLB.*

Proof. This result is obtained from Lemma 3 by performing calculus analysis. Let $g(x) = \frac{x}{\ln(2)} l^{1/x}$. Consider the derivative of g, i.e., $g'(x) = \frac{l^{1/x}}{\ln 2} \left(1 - \frac{\ln l}{x}\right)$. For $x \geq \ln l$, it holds that $g'(x) \geq 0$ so that the function $g(x)$ is monotonically increasing. Since $\ln(l) \leq (\lceil \ln(l) \rceil) \leq \ln(l) + 1$, we have

$$g(\lceil \ln(l) \rceil) - g(\ln l) \leq g(\ln l + 1) - g(\ln l). \tag{11}$$

In addition, consider the two points $(\ln l, g(\ln l))$ and $(\ln l + 1, g(\ln l + 1))$. Due to Lagrange's mean value theorem in calculus, there exists $\xi \in [\ln l, \ln l + 1]$ such that

$$g(\ln l + 1) - g(\ln l) = g'(\xi). \tag{12}$$

Since $\xi \geq \ln l$, we have $l^{1/\xi} \leq e$. Additionally, $\xi \leq \ln l + 1$, so $1 - \frac{\ln l}{\xi} \leq \frac{1}{\ln l + 1}$. Therefore, $g'(\xi) \leq \frac{e}{\ln(2)} \left(1 - \frac{\ln l}{\xi}\right) \leq \frac{e \log(e)}{\ln l + 1}$. We have

$$g(\lceil \ln l \rceil) - g(\ln l) \leq g(\ln l + 1) - g(\ln l) \leq \frac{e \log(e)}{\ln l + 1}. \tag{13}$$

Note that $g(\ln l) = e \log(l)$. This completes our proof. □

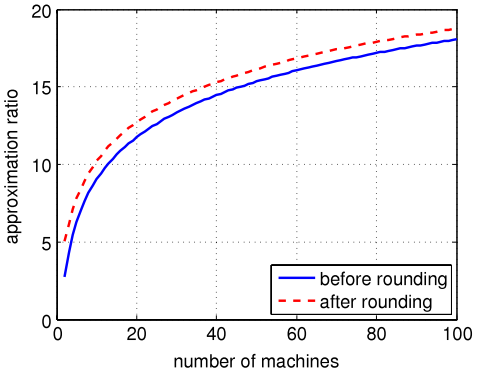

Fig. 1. Approximation ratio loss due to rounding. Before rounding, the approximation ratio is $y = e\log(x)$ and it becomes $y = e\log(x) + \frac{e\log(e)}{\ln(x)+1}$ after rounding.

This theorem holds for the general GLB problem, such as the one considered in [5,1,6]. Of course, it holds for VS–GLB as well. To have an intuition on the loss, we plot the two approximation ratios with respect to the number of machines in Fig. 1. We can see that the loss is small.

We have the following corollary due to Theorem 4.

Corollary 1. *With $\tau = \lceil \ln(md) \rceil$, Algorithm 1 is an $e\log(md) + \frac{e\log(e)}{\ln(md)+1}$ approximation algorithm to VS, and it runs in polynomial time.*

The polynomial running time can be shown by the following analysis. We assume $\tau = \lceil \ln(md) \rceil$ if not specified. The main time consuming step is to minimize $f_{i,j}^{(\tau)}$ over j for given i. We can omit the computation of the outer $1/\tau$ power since function x^y is monotonic for $x \geq 0$ and $y > 0$. One basic operation is the computation of the integer power of a number, i.e., a^τ, where a is an element in any vector $\overline{A}_j$. This requires $\lfloor \log(\tau) \rfloor + \nu(\tau) - 1$ multiplications, where $\nu(\tau)$ is the number of 1s in the binary representation of τ (Chapter 4.6.3 in [4]). For simplicity, we put an upper bound of $2\log\tau$ to the number of multiplications needed to compute a^τ.

To compute $f_{i,j}^{(\tau)}$ for given i and j, it needs $d + m - 1$ additions (adding p_i to $\overline{A}_j$, suppose $\overline{A}_j$ is maintained in each iteration) and $2md\log(\tau)$ multiplications (md numbers, each needs to compute its τ power). To find the optimal j for given i, we need to compute $f_{i,j}^{(\tau)}$ for all j, and select the optimal one by comparison. This procedure needs $m(d + m - 1)$ additions, $2d\log(\tau)m^2$ multiplications, and $m - 1$ comparisons. In summary, it takes $O(d\log(\tau)m^2)$ time for one vector. For the overall algorithm, it takes $O(d\log(\tau)nm^2)$ time. The computations can be sped up by exploiting the problem structure. The complexity can be reduced to $O(d\log(\tau)mn)$, dropping one m factor, as shown in the following.

3.3. Computation speedup

Towards VS–GLB, we have the following lemma. Note that this lemma does not apply to the general GLB problem.

Lemma 5. *For any j_1, j_2, it holds that $f_{i,j_1}^{(\tau)} > f_{i,j_2}^{(\tau)}$ if and only if*

$$\left\|\overline{A_{j_1} \cup \{p_i\}}\right\|_\tau^\tau - \left\|\overline{A_{j_1}}\right\|_\tau^\tau > \left\|\overline{A_{j_2} \cup \{p_i\}}\right\|_\tau^\tau - \left\|\overline{A_{j_2}}\right\|_\tau^\tau. \tag{14}$$

Proof. Adding $\|\overline{A}_1\|_\tau^\tau + \cdots + \|\overline{A}_m\|_\tau^\tau$ to both sides proves the lemma. □

Algorithm 2 shows the final design. For each partition A_j, the algorithm maintains two variables, the vector $\overline{A}_j$ (μ_j in the algorithm) and its norm $\|\overline{A}_j\|_\tau^\tau$ (δ_j in the algorithm). If there is no empty partition, then each incoming vector searches over all partitions to find the j to minimize $\left\|\overline{A_j \cup \{p_i\}}\right\|_\tau^\tau - \left\|\overline{A_j}\right\|_\tau^\tau$ (Lines 12–24). As Lemma 5 shows, this is equivalent to minimizing $f_{i,j}^{(\tau)}$.

For the running time, consider a new vector that cannot find an empty partition. There are md additions (Lines 13,16), $2md\log(\tau)$ multiplications (Lines 14,17), $2(m-1)$ subtractions and $m-1$ comparisons (Line 18). The dominating factor is $md\log(\tau)$. This is for one vector. For all n vectors, the running time is $O(mnd\log(\tau))$, compared to $O(m^2nd\log\tau)$ before speedup. Substituting $\tau = \lceil \ln(md) \rceil$ into the formula yields $O(nmd\ln\ln(md))$ running time, polynomial in the input length (note $m < n$). This analysis, together with Corollary 1 and Lemma 5, gives the following theorem.

Theorem 5. *Algorithm 2 is an $e\log(md) + \frac{e\log(e)}{\ln(md)+1}$ approximation algorithm to VS. It runs in $O(nmd\ln\ln(md))$ time.*

Algorithm 2: Sped-up Vector Scheduling with $\tau = \lceil \ln(md) \rceil$

Input: m, the number of partitions; d, the dimension of each vector; $p_1, p_2, \ldots, p_n$, the n vectors to be scheduled
Output: $A_1, \ldots, A_m$, the m partitions

```
1  begin
2    for j from 1 to m do
3      A_j ⟵ ∅;
4      μ_j ⟵ 0;                          // vector Ā_j
5      δ_j ⟵ 0;                          // scalar ‖Ā_j‖_τ^τ
6    for i from 1 to n do
7      if ∃A_j, A_j = ∅ then
8        A_j ⟵ A_j ⋃ {p_i};
9        μ_j ⟵ p_i;
10       δ_j ⟵ ‖p_i‖_τ^τ;
11     else
12       j_min ⟵ 1;                      // partition index
13       μ_min ⟵ μ_1 + p_1;
14       δ_min ⟵ ‖μ_min‖_τ^τ;
15       for j from 2 to m do
16         μ̃ ⟵ μ_j + p_i;                // vector addition
17         δ̃ ⟵ ‖μ̃‖_τ^τ;
18         if δ_min − δ_{j_min} > δ̃ − δ_j then
19           j_min ⟵ j;
20           μ_min ⟵ μ̃;
21           δ_min ⟵ δ̃;
22       μ_{j_min} ⟵ μ_min;
23       δ_{j_min} ⟵ δ_min;
24       A_{j_min} ⟵ A_{j_min} ⋃ {p_i};
25 end
```

3.4. Simulations

We implement three approaches for comparison: Algorithm 1 with $\tau = \ln(md)$, Algorithm 2 with $\tau = \lceil \ln(md) \rceil$, and a list scheduling algorithm mentioned in [2]. The list scheduling algorithm is a $(d+1)$ approximation algorithm for vector scheduling. It ignores the multi-dimension property of vectors, and treats vectors as scalars equal to the summation of elements. We did not implement the $O(\ln^2(d))$ approximation algorithm in [2] due to complicated implementation.

We consider two scenarios. In the first scenario, we study the approximation ratio of each algorithm. This requires the computation of the optimal solution, which is done by enumerating all solutions and is time consuming, so we only consider small problem

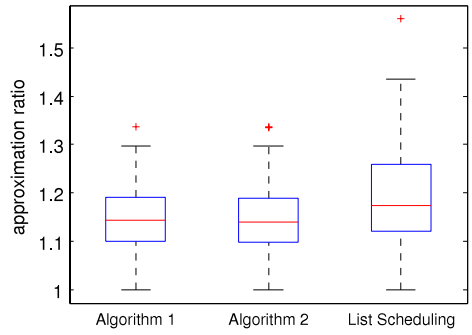

Fig. 2. Approximation ratio of three vector scheduling approaches. One hundred problem instances are generated to plot this figure.

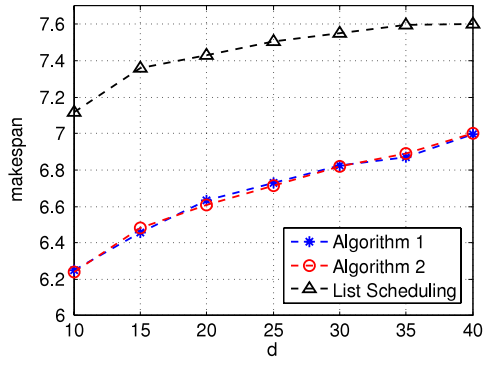

Fig. 3. Compare three vector scheduling approaches in terms of makespan. Each point in the figure is averaged over 100 problem instances.

instances. Specifically, we consider problem instances with 3 machines ($m = 3$), 10 jobs ($n = 10$) and a dimension of 20 ($d = 20$). For each job, its elements are drawn independently from the uniform distribution in the range of [0, 1]. Under such settings, the theoretical worst-case approximation ratios for Algorithm 1, Algorithm 2 and the list scheduling algorithm are 16.0566, 16.8264 and 21 respectively. We generate 100 problem instances and Fig. 2 shows the box plot of the approximation ratio of each algorithm. We can see that the empirical performance of every algorithm is much better than that suggested by the worst-case analysis, and Algorithms 1 and 2 outperform the list scheduling algorithm.

In the second scenario, we compare the three algorithms on larger problem instances. There are 10 machines and 100 jobs. The elements of a job are drawn from a uniform distribution as before. We vary the dimension d from 10 to 40 with increments of 5. For each dimension, we generate 100 problem instances and compute the average makespan of the three approaches. Fig. 3 shows that Algorithms 1 and 2 perform similarly, and both of them greatly outperform the list scheduling algorithm. Note that with the increase of dimension, the makespan of all approaches increases. This is because the probability of an imbalanced dimension increases in this case.

4. Conclusion

In this work, we connect the vector scheduling problem with the generalized load balancing problem, and obtain new results by applying existing results to each other. Besides showing that generalized load balancing does not admit constant approximation algorithms unless $P = \text{NP}$, we give the first non-trivial online algorithm for vector scheduling. This online algorithm also provides a better approximation bound to solve VS than the existing offline polynomial time algorithm.

Acknowledgments

The authors would like to thank all the reviewers for their helpful comments. The work is partly supported by the program B for Outstanding PhD Candidate of Nanjing University (201301B014), US National Science Foundation grants CNS-1320453, CNS-1117412, CAREER Award CNS-0747108, China NSF grants (61133006, 61021062) and China 973 projects (2014CB340300, 2012CB316200).

References

[1] I. Caragiannis, Better bounds for online load balancing on unrelated machines, in: Proceedings of SODA, 2008.
[2] C. Chekuri, S. Khanna, On multidimensional packing problems, SIAM J. Comput. 33 (4) (2004) 837–851.
[3] L. Epstein, T. Tassa, Vector assignment problems: a general framework, J. Algorithms 48 (2) (2003) 360–384.
[4] D.E. Knuth, The Art of Computer Programming, Volume 2 (second ed.): Seminumerical Algorithms, Addison-Wesley Longman Publishing Co., 1981.
[5] F. Xu, C.C. Tan, Q. Li, G. Yan, J. Wu, Designing a practical access point association protocol, in: Proceedings of INFOCOM, 2010.
[6] F. Xu, X. Zhu, C.C. Tan, Q. Li, G. Yan, W. Jie, Smartassoc: decentralized access point selection algorithm to improve throughput, IEEE Trans. Parallel Distrib. Syst. 24 (12) (2013) 2482–2491.
[7] D. Zuckerman, Linear degree extractors and the inapproximability of max clique and chromatic number, in: Proceedings of STOC, 2006.

Xiaojun Zhu received the BS degree in Computer Science from Nanjing University in 2008, where he is now a PhD candidate. His research interests include RFID systems, smartphone systems, wireless sensor networks and vehicular networks. He was a visiting scholar in the College of William and Mary during August 2011 to August 2012.

Qun Li is an associate professor in the Department of Computer Science at the College of William and Mary. He holds a PhD degree in computer science from Dartmouth College. His research interests include wireless networks, sensor networks, RFID, and pervasive computing systems. He received the NSF Career award in 2008.

Weizhen Mao received her PhD degree in Computer Science from Princeton University under the supervision of Andrew Yao. Her research interests include the design and analysis of online and approximation algorithms, especially for problem arising from application areas, such as parallel and multi-core job scheduling and wireless/sensor networks. She is currently a professor in the Department of Computer Science at the College of William and Mary.

Guihai Chen obtained his BS degree from Nanjing University, M. Engineering from Southeast University, and PhD from University of Hong Kong. He visited Kyushu Institute of Technology, Japan in 1998 as a research fellow, and University of Queensland, Australia in 2000 as a visiting Professor. During September 2001 to August 2003, he was a visiting Professor in Wayne State University. He is a Distinguished Professor with the Department of Computer Science, Shanghai Jiao Tong University. Prof. Chen has published more than 280 papers in peer-reviewed journals and refereed conference proceedings in the areas of wireless sensor networks, high-performance computer architecture, peer-to-peer computing and performance evaluation. He has also served on technical program committees of numerous international conferences.