**College of**
**William & Mary**
Department of Computer Science

WM-CS-2012-01

# Online Vector Scheduling and Generalized Load Balancing

Xiaojun Zhu, Qun Li, Weizhen Mao, Guihai Chen

March 5, 2012

# Online Vector Scheduling and Generalized Load Balancing

Xiaojun Zhu*†, Qun Li†, Weizhen Mao† and Guihai Chen*
* State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, P. R. China
† Department of Computer Science, the College of William and Mary, Williamsburg, VA, USA
Email: {njuxjzhu,liqun,wm}@cs.wm.edu, gchen@nju.edu.cn

***Abstract*—We give an approximation-preserving reduction from vector scheduling problem (VS) to generalized load balancing problem (GLB). The reduction bridges existing results of the two problems. Specifically, the hardness result for VS holds also for GLB and any algorithm for GLB can be used to solve VS. Based on this, we get two new results. First, GLB does not have constant approximation algorithms that run in polynomial time unless $P = NP$. Second, there is an online algorithm (vectors coming in an online fashion) that solves VS with approximation bound $e \log(md)$, where $e$ is the natrual number, $m$ is the number of partitions and $d$ is the dimension of vectors. The algorithm is borrowed from GLB literature and is very simple in that each vector only needs to minimize the $L_{\ln(md)}$ norm of the resulting load. However, it is unclear whether this algorithm runs in polynomial time, due to the irrational and non-integer nature of $\ln(md)$. We address this issue by rounding $\ln(md)$ to the next integer. We prove that the resulting algorithm runs in polynomial time with approximation bound $e \log(md) + \frac{e \log(e)}{\ln(md)+1}$ which is in $O(\ln(md))$. This improves the $O(\ln^2 d)$ bound of the existing polynomial time algorithm for VS.**

## I. Introduction

Scheduling with costs is a very well studied problem in combinatorial optimization. The traditional paradigm assumes 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 [1], where it is not suitable to represent different requests (such as CPU, memory, network resource, etc.) as a single scalar. To solve vector scheduling, there are three approximation solutions [1]. Two of them are deterministic algorithms based on derandomization of a randomized algorithm, with one providing $O(\ln^2 d)$ approximation[1], where $d$ is the dimension of vectors, and the other providing $O(\ln d)$ 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 $O(\ln dm/ \ln \ln dm)$ approximation with high probability, where $m$ is the number of partitions (servers). For fixed $d$, there exists a polynomial time approximation scheme (PTAS) [1]. A PTAS has also been proposed for a wide class of cost functions (rather than max) [2].

Generalized load balancing is recently introduced to model the effect of wireless interference [3][4]. 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 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 wireless network, a user may influence all APs in its transmission range due to the broadcast nature of wireless signal. To solve the generalized load balancing problem, the current solution is an online algorithm, adapted from the recent progress in online scheduling on traditional model [5]. The solution, though provides good approximation, is rather simple: each job selects the machine to minimize the $L_\tau$ norm of the resulting loads at all machines where $\tau$ is a parameter.

To avoid confusion, we keep the two terms *job* and *machine* unchanged for generalized load balancing, while 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 generalized load balancing problem. This encoding method brings the recent progress in generalized load balancing into the vector scheduling domain, and takes the hardness result of vector scheduling to generalized load balancing problem. Second, the existing solution to GLB needs to compute the $L_{\ln l}$ norm ($l$ is the number of machines), but 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 the following two definitions.

The work was done when the first author was visiting the College of William and Mary.

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

### A. 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 only consider 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

$$\|\overline{A}_j\|_\infty = \max_{1 \le k \le d} \sum_i x_{ij} p_{ik}$$

The vector scheduling problem can be rewritten as

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

### B. Generalized Load Balancing

This formulation first appears in [4]. 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.

## II. 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.

### A. 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, denoted by the pair $(j, k)$. Thus, the constructed machine set $\mathcal{M}$ is $\{(j,k) \mid j = 1, 2, \ldots, m \text{ and } k = 1, 2, \ldots, d\}$. 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

$$c_{ist} = \begin{cases} p_{it_2} & \text{if } s = \bar{t} & (1) \\ \infty & \text{if } s \in [t] \wedge s \ne \bar{t} & (2) \\ 0 & \text{if } s \notin [t] & (3) \end{cases}$$

where (1)(2) are for the situation where $s$ and $t$ correspond to the same partition. They force a job to select only the anchor machines. (3) 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 the 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 II-B.

**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, vice versa. We prove this theorem in Section II-B.

It is worth mentioning that VS-GLB is a special instance of GLB. Since VS-GLB is converted from VS, VS is a special instance of GLB, which implies that VS should have approximation algorithms at least as good as GLB. Unfortunately, on the contrary, the literature shows better approximation algorithm for GLB than that for VS. Hence, it is worth applying algorithms of GLB to VS.

### B. Proof of Equivalence

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

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

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

*Proof:* For 1), suppose $x'_{is} = 1$ for some $s$ with $s \neq \overline{s}$. Then $x'_{is} c_{is\overline{s}} = \infty > T$, 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 $s = \overline{s}$. ∎

**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

$$\begin{aligned}
\sum_s x'_{is} c_{ist} &= \sum_{s \in [t]} x'_{is} c_{ist} + \sum_{s \notin [t]} x'_{is} c_{ist} \\
&= \sum_{s \in [t]} x'_{is} c_{ist} && (4) \\
&= x'_{i\bar{t}} c_{i\bar{t}t} && (5) \\
&= x'_{i\bar{t}} p_{it_2} && (6)
\end{aligned}$$

where (4) is due to (3), (5) is due to Lemma 1, and (6) is due to (1). ∎

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 $x'_{i\overline{s}} = x_{is_1}$ 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 $s \neq \overline{s}$. 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 $x_{i,j} = 1$. Our transformation sets $x'_{is} = 1$ where $s = (j, 1)$. So $\sum_s x'_{is} = 1$.

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

$$\begin{aligned}
\max_t \sum_{i,s} x'_{is} c_{ist} &= \max_t \sum_{\substack{i \\ s \in [t]}} x'_{is} c_{ist} && (7) \\
&= \max_t \sum_i x'_{i\bar{t}} c_{i\bar{t}t} && (8) \\
&= \max_t \sum_i x_{it_1} p_{it_2} \\
&= \max_{j,k} \sum_i x_{ij} p_{ik},
\end{aligned}$$

where (7) is due to that $c_{ist} = 0$ if $s \notin [t]$, and (8) is due to our assignment of $\mathbf{x}'$ that $x'_{is} = 0$ if $s \neq \overline{s}$.

"$\Longleftarrow$" Given $\mathbf{x}'$ for VS-GLB, construct $\mathbf{x}$ for VS as follows. Set $x_{ij} = x'_{i\overline{s}}$ 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 $s = \overline{s}$. Therefore, there exists one $j$ such that $x_{ij} = 1$. On the other hand, there cannot be two $j$s both with $x_{ij} = 1$, otherwise $\mathbf{x}'$ is not a feasible solution to VS-GLB.

For the objective value, we have

$$\begin{aligned}
\max_{j,k} \sum_i x_{ij} p_{ik} &= \max_{t=(j,k)} \sum_i x_{it_1} p_{it_2} \\
&= \max_t \sum_i x'_{i\bar{t}} p_{it_2} \\
&= \max_t \sum_{i,s} x'_{is} c_{ist}, && (9)
\end{aligned}$$

where (9) is due to Lemma 2. This completes our proof. ∎

### C. Inapproximability for GLB

It has been proved that no polynomial time algorithm can give $c$-approximation solution to VS for any $c > 1$ unless $NP = ZPP$ [1]. Combining this result with Theorem 2, we have the following theorem.

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

*Proof:* Since VS-GLB is a special instance of GLB, any $c$-approximation algorithm for GLB can be used to obtain $c$-approximation solution to VS-GLB. By Theorem 2, any $c$-approximation solution to VS-GLB is also a $c$-approximation solution to the corresponding VS. Therefore, the approximation algorithm is also a $c$-approximation algorithm for VS, a contradiction. ∎

We can obtain a stronger result by relaxing the assumption $NP \neq ZPP$ to $P \neq NP$. (It is a relaxation because $P \subseteq ZPP \subseteq NP$.) This can be done by examining the inapproximability proof for VS [1]. The inapproximability proof relies on the result that no polynomial time algorithm can approximate chromatic number to within $n^{1-\epsilon}$ for any $\epsilon > 0$ unless $NP = ZPP$. Recently, it has been proved that no polynomial time algorithm can approximate chromatic number to within $n^{1-\epsilon}$ for any $\epsilon > 0$ unless $P = NP$ [6]. Thus, we can change the assumption $NP \neq ZPP$ to $P \neq NP$ safely.

**Theorem 4.** *For any constant $c > 1$, there does not exist a polynomial time $c$-approximation algorithm for GLB, unless $P = NP$.*

### D. Extending to generalized VS

Our construction of VS-GLB and 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 transformations 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 equation (1) 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.

## III. Online Algorithm for VS

Based on Theorem 2, we can solve VS by its corresponding VS-GLB. We review the approximation algorithm [3] for GLB, and then modify it to solve VS.

Given a GLB instance and a positive number $\tau$, the algorithm [3] considers jobs one by one (in an arbitrary order) and assigns the current job to a machine to minimize the $L_\tau$ norm[2] of the resulting 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

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

The above optimization problem can be solved by trying each possible machine. During the optimization, the computation of the last step of $L_\tau$ 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 [5], and recently extended to the GLB problem [3]. The parameter $\tau$ controls the approximation ratio of the algorithm, as shown in the following lemma.

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

**Lemma 3** ([5], [3]). *Minimizing $L_\tau$ norm gives $\frac{\tau}{\ln(2)} l^{1/\tau}$ 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.

### A. 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 $L_\tau$ norm of the resulting load. Due to the construction of VS-GLB, this vector can only choose from the *anchor machines*, otherwise, the resulting $L_\tau$ 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 $L_\tau$ norm of the load of machines in VS-GLB is, in fact, equal to

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

where

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

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 $L_\tau$ norm of the resulting load if vector $p_i$ chooses partition $A_j$, i.e.,

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

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

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

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.*

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

**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; $\tau$, the norm

**Output**: $A_1, \ldots, A_m$, the $m$ partitions

**begin**
  **for** $j$ *from* 1 *to* $m$ **do**
    $A_j \longleftarrow \emptyset$;
  **for** $i$ *from* 1 *to* $n$ **do**
    **if** $\exists A_j, A_j = \emptyset$ **then**
      $A_j \longleftarrow A_j \bigcup \{p_i\}$;
    **else**
      find $j$ to minimize $f_{i,j}^{(\tau)}$;
      $A_j \longleftarrow A_j \bigcup \{p_i\}$;

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 $\tau$. 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 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 approximation ratio. We show in the following that the loss is very small.

### *B. 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 5.** *Let $l$ be the number of machines. Minimizing $L_{\lceil \ln l \rceil}$ norm gives $e\log(l) + \frac{e\log(e)}{\ln l + 1}$ 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 $g$,

$$g'(x) = \frac{l^{1/x}}{\ln 2}\left(1 - \frac{\ln l}{x}\right).$$

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

$$g(\lceil \ln(l) \rceil) - g(\ln l) \leq g(\ln l + 1) - g(\ln l).$$

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

$$g(\ln l + 1) - g(\ln l) = g'(\xi).$$

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

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

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

This theorem holds for general GLB problem, such as the one considered in [3] [5] and [4]. 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 Figure 1. We can see that the loss is small.

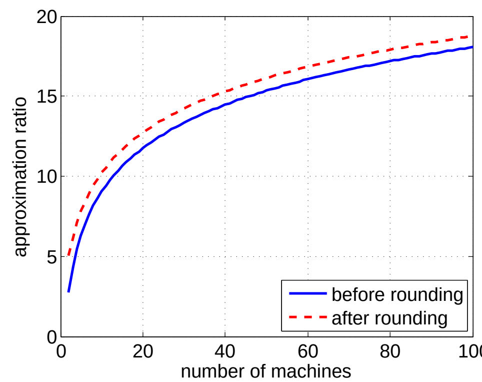


Fig. 1. Approximation ratio loss due to rounding. Before rounding, the approximation ratio is $y = e\log(x)$ and it becomes $y = e\log(x) + \frac{e\log(e)}{\ln(x)+1}$ after rounding.

We have the following corollary due to Theorem 5.

**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 $y > 0$. In computing $L_\tau$ norm, there is a basic operation, the integer power of a number, $a^\tau$, where $a$ is an element in any vector $\overline{A}_j$. The naive approach, which multiplies $a$ iteratively, involves $\tau - 1$ multiplications. This can be improved by utilizing partial multiplication results. For example, computing $a^8$ as $((a^2)^2)^2$ only needs 3 multiplications. Generally, computing $a^\tau$ requires $\lfloor \log(\tau) \rfloor + \nu(\tau) - 1$ multiplications, where $\nu(\tau)$ is the number of 1s in the binary representation of $\tau$ (Chapter 4.6.3 in [7]). In the following, we put an upper bound $2\log\tau$ to the number of multiplications needed to compute $a^\tau$.

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 $\tau$ 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.

### C. Computation Speedup

Towards VS-GLB, we have the following lemma. Note that this lemma does not hold for 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*

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

*Proof:* Adding $\|\overline{A}_1\|_\tau^\tau + \ldots + \|\overline{A}_m\|_\tau^\tau$ to both sides proves the lemma. ■

Algorithm 2 shows the final design. For each partition $A_j$, the algorithm maintains two variables, the vector $\overline{A}_j$ ($\mu_j$ in the algorithm) and its norm $\|\overline{A}_j\|_\tau^\tau$ ($\delta_j$ 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 minimize $f_{i,j}^{(\tau)}$.

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 6.** *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.*

It should be noted that we treat multiplications as basic operations in the above running time analysis. The running time will be different if we further consider the complexity of computing multiplications. Multiplying two $n$-bit integers takes time $O(n^{1.59})$ for a recursive algorithm (Chapter 5.5 in [8]). Applying such analysis to Algorithm 2, however, requires the consideration of the length of the binary representation of each numeric value in the vectors, which may be complicated. Nevertheless, it is clear that multiplications run in polynomial time in the input length. Thus Algorithm 2 terminates certainly in polynomial time.

## IV. 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. On one hand, we show that generalized load balancing does not admit constant approximation algorithms unless $P = NP$. On the other hand,

**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     $A_j \longleftarrow \emptyset$;
4     $\mu_j \longleftarrow \mathbf{0}$; // vector $\overline{A}_j$
5     $\delta_j \longleftarrow 0$ ; // scalar $\|\overline{A}_j\|_\tau^\tau$
6   **for** $i$ *from* $1$ *to* $n$ **do**
7     **if** $\exists A_j, A_j = \emptyset$ **then**
8       $A_j \longleftarrow A_j \bigcup \{p_i\}$;
9       $\mu_j \longleftarrow p_i$;
10       $\delta_j \longleftarrow \|p_i\|_\tau^\tau$;
11     **else**
12       $j_{\min} \longleftarrow 1$; // partition index
13       $\mu_{\min} \longleftarrow \mu_1 + p_1$;
14       $\delta_{\min} \longleftarrow \|\mu_{\min}\|_\tau^\tau$;
15       **for** $j$ *from* $2$ *to* $m$ **do**
16         $\tilde{\mu} \longleftarrow \mu_j + p_i$; // vector addition
17         $\tilde{\delta} \longleftarrow \|\tilde{\mu}\|_\tau^\tau$;
18         **if** $\delta_{\min} - \delta_{j_{\min}} > \tilde{\delta} - \delta_j$ **then**
19           $j_{\min} \longleftarrow j$;
20           $\mu_{\min} \longleftarrow \tilde{\mu}$;
21           $\delta_{\min} \longleftarrow \tilde{\delta}$;
22       $\mu_{j_{\min}} \longleftarrow \mu_{\min}$;
23       $\delta_{j_{\min}} \longleftarrow \delta_{\min}$;
24       $A_{j_{\min}} \longleftarrow A_{j_{\min}} \bigcup \{p_i\}$;

we design a polynomial time algorithm for vector scheduling by using an existing algorithm for GLB and modifying it to guarantee polynomial running time. The resulting algorithm has two advantages. It can be performed in an online fashion, and it provides better approximation bound to solve VS than existing offline polynomial time algorithm. Note that we round $\ln(md)$ to an integer to guarantee polynomial running time, which causes a small approximation ratio loss. It is unclear whether computing $L_{\ln(md)}$ norm directly, without rounding, can be done in polynomial time.

## References


[1] C. Chekuri and S. Khanna, "On multidimensional packing problems," *SIAM J. Comput.*, vol. 33, pp. 837–851, April 2004.

[2] L. Epstein and T. Tassa, "Vector assignment problems: a general framework," *J. Algorithms*, September 2003.

[3] F. Xu, C. C. Tan, Q. Li, G. Yan, and J. Wu, "Designing a practical access point association protocol," in *Proceedings of INFOCOM'10*.

[4] F. Xu, X. Zhu, C. C. Tan, Q. Li, G. Yan, and W. Jie, "Smartassoc: Decentralized access point selection algorithm to improve throughput," *under submission*.

[5] I. Caragiannis, "Better bounds for online load balancing on unrelated machines," in *Proceedings of SODA '08*.

[6] D. Zuckerman, "Linear degree extractors and the inapproximability of max clique and chromatic number," in *Proceedings of STOC'06*.
[7] D. E. Knuth, *The art of computer programming, volume 2 (2nd ed.): seminumerical algorithms*. Addison-Wesley Longman Publishing Co., Inc., 1981.
[8] J. Kleinberg and E. Tardos, *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., 2005.