Scheduling Distributed Clusters of Parallel Machines : Primal-Dual and LP-based Approximation Algorithms [Full Version] *

Riley Murray[1], Megan Chao[2], and Samir Khuller[3]

1 **Department of Industrial Engineering & Operations Research, University of California, Berkeley**
Berkeley, CA 94709, USA
rjmurray@berkeley.edu

2 **Department of Electrical Engineering & Computer Science, Massachusetts Institute of Technology**
50 Vassar St, Cambridge, MA 02142, USA
megchao@mit.edu

3 **Department of Computer Science, University of Maryland, College Park**
College Park, MD 20742, USA
samir@cs.umd.edu

Abstract

The Map-Reduce computing framework rose to prominence with datasets of such size that dozens of machines on a single cluster were needed for individual jobs. As datasets approach the exabyte scale, a single job may need distributed processing not only on multiple machines, but on multiple *clusters*. We consider a scheduling problem to minimize weighted average completion time of n jobs on m distributed clusters of parallel machines. In keeping with the scale of the problems motivating this work, we assume that (1) each job is divided into m "subjobs" and (2) distinct subjobs of a given job may be processed concurrently.

When each cluster is a single machine, this is the NP-Hard *concurrent open shop* problem. A clear limitation of such a model is that a serial processing assumption sidesteps the issue of how different tasks of a given subjob might be processed in parallel. Our algorithms explicitly model clusters as pools of resources and effectively overcome this issue.

Under a variety of parameter settings, we develop two constant factor approximation algorithms for this problem. The first algorithm uses an LP relaxation tailored to this problem from prior work. This LP-based algorithm provides strong performance guarantees. Our second algorithm exploits a surprisingly simple mapping to the special case of one machine per cluster. This mapping-based algorithm is combinatorial and extremely fast. These are the first constant factor approximations for this problem.

Remark - A shorter version of this paper (one that omitted several proofs) appeared in the proceedings of the 2016 European Symposium on Algorithms. Any citations of this work should reference the version appearing in the conference proceedings.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases approximation algorithms, distributed computing, machine scheduling, LP relaxations, primal-dual algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.234

* All authors conducted this work at the University of Maryland, College Park. This work was made possible by the National Science Foundation, REU Grant CNS 1262805, and the Winkler Foundation. This work was also partially supported by NSF Grant CCF 1217890.

© Riley Murray, Samir Khuller, Megan Chao;
licensed under Creative Commons License CC-BY

1 Introduction

It is becoming increasingly impractical to store full copies of large datasets on more than one data center [7]. As a result, the data for a single job may be located not on multiple machines, but on multiple *clusters* of machines. To maintain fast response-times and avoid excessive network traffic, it is advantageous to perform computation for such jobs in a completely distributed fashion [8]. In addition, commercial platforms such as AWS Lambda and Microsoft's Azure Service Fabric are demonstrating a trend of centralized cloud computing frameworks in which the user manages neither data flow nor server allocation [1, 11]. In view of these converging issues, the following scheduling problem arises:

If computation is done locally to avoid excessive network traffic, how can individual clusters on the broader grid coordinate schedules for maximum throughput?

This was precisely the motivation for Hung, Golubchik, and Yu in their 2015 ACM Symposium on Cloud Computing paper [8]. Hung et al. modeled each cluster as having an arbitrary number of identical parallel machines, and choose an objective of average job completion time. As such a problem generalizes the NP-Hard concurrent open shop problem, they proposed a heuristic approach. Their heuristic (called "SWAG") runs in $O(n^2m)$ time and performed well on a variety of data sets. Unfortunately, SWAG offers poor worst-case performance, as we show in Section 5.

Our contributions to this problem are to extend the model considered by Hung et al. and to introduce the first constant-factor approximation algorithms for this general problem. Our extensions of Hung et al.'s model are (1) to allow different machines within the same cluster to operate at different speeds, (2) to incorporate pre-specified "release times" (times before which a subjob cannot be processed), and (3) to support *weighted* average job completion time. We present two algorithms for the resulting problem. Our combinatorial algorithm exploits a surprisingly simple mapping to the special case of one machine per cluster, where the problem can be approximated in $O(n^2 + nm)$ time. We also present an LP-rounding approach with strong performance guarantees. E.g., a 2-approximation when machines are of unit speed and subjobs are divided into equally sized (but not necessary *unit*) tasks.

1.1 Formal Problem Statement

▶ **Definition 1** (Concurrent Cluster Scheduling)**.**

- There is a set M of m clusters, and a set N of n jobs. For each job $j \in N$, there is a set of m "subjobs" (one for each cluster).
- Cluster $i \in M$ has m_i parallel machines, and machine ℓ in cluster i has speed $v_{\ell i}$. Without loss of generality, assume $v_{\ell i}$ is decreasing in ℓ. [1]
- The i^{th} subjob for job j is specified by a set of tasks to be performed by machines in cluster i, denote this set of tasks T_{ji}. For each task $t \in T_{ji}$, we have an associated processing time p_{jit} (again w.l.o.g., assume p_{jit} is decreasing in t). We will frequently refer to "the subjob of job j at cluster i" as "subjob (j, i)."
- Different subjobs of the same job may be processed concurrently on different clusters.
- Different tasks of the same subjob may be processed concurrently on different machines within the same cluster.

[1] Where we write "decreasing", we mean "non-increasing." Where we write "increasing", we mean "non-decreasing".

- A subjob is complete when all of its tasks are complete, and a job is complete when all of its subjobs are complete. We denote a job's completion time by "C_j".
- The objective is to minimize weighted average job completion time (job j has weight w_j).
- For the purposes of computing approximation ratios, it is equivalent to minimize $\sum w_j C_j$. We work with this equivalent objective throughout this paper.

A machine is said to operate at *unit speed* it if can complete a task with processing requirement "p" in p units of time. More generally, a machine with speed "v" ($v \geq 1$) processes the same task in p/v units of time. Machines are said to be *identical* if they are all of unit speed, and *uniform* if they differ only in speed.

In accordance with Graham et al.'s $\alpha|\beta|\gamma$ taxonomy for scheduling problems [6] we take $\alpha = CC$ to refer to the concurrent cluster environment, and denote our problem by $CC||\sum w_j C_j$.[2] Optionally, we may associate a release time r_{ji} to every subjob. If any subjobs are released after time zero, we write $CC|r|\sum w_j C_j$.

1.1.1 Example Problem Instances

We now illustrate our model with several examples (see Figures 1 and 2). The tables at left have rows labeled to identify jobs, and columns labeled to identify clusters; each entry in these tables specifies the processing requirements for the corresponding subjob. The diagrams to the right of these tables show how the given jobs might be scheduled on clusters with the indicated number of machines.

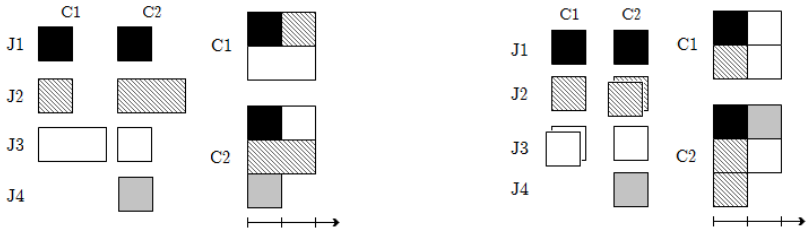

Figure 1 Two examples of our scheduling model. **Left**: Our baseline example. There are 4 jobs and 2 clusters. Cluster 1 has 2 identical machines, and cluster 2 has 3 identical machines. Note that job 4 has no subjob for cluster 1 (this is permitted within our framework). In this case every subjob has at most one task. **Right**: Our baseline example with a more general subjob framework : subjob (2,2) and subjob (3,1) both have two tasks. The tasks shown are unit length, but our framework *does not* require that subjobs be divided into equally sized tasks.

1.2 Related Work

Concurrent cluster scheduling subsumes many fundamental machine scheduling problems. For example, if we restrict ourselves to a single cluster (i.e. $m = 1$) we can schedule a set of jobs on a bank of identical parallel machines to minimize makespan ($C_{\max}$) or total weighted completion time ($\sum w_j C_j$). With a more clever reduction, we can even minimize *total weighted lateness* ($\sum w_j L_j$) on a bank of identical parallel machines (see Section 6). Alternatively, with $m > 1$ but $\forall i \in M, m_i = 1$, our problem reduces to the well-studied "concurrent open shop" problem.

[2] A problem $\alpha|\beta|\gamma$ implies a particular environment α, objective function γ, and optional constraints β.

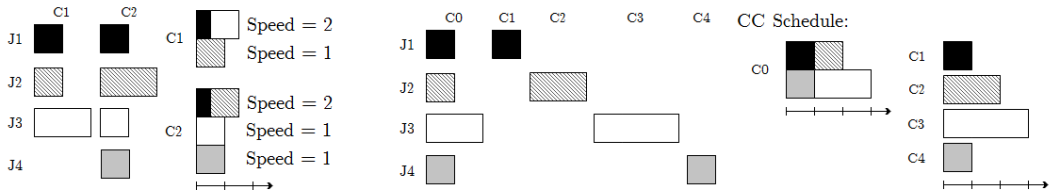

Figure 2 Two additional examples of our model. **Left**: Our baseline example, with variable machine speeds. Note that the benefit of high machine speeds is only realized for tasks assigned to those machines in the final schedule. **Right**: A problem with the peculiar structure that (1) all clusters but one have a single machine, and (2) most clusters have non-zero processing requirements for only a single job. We will use such a device for the total weighted lateness reduction in Section 6.

Using Graham et al.'s taxonomy, the concurrent open shop problem is written as $PD||\sum w_jC_j$. Three groups [3, 4, 9] independently discovered an LP-based 2-approximation for $PD||\sum w_jC_j$ using the work of Queyranne [12]. The linear program in question has an exponential number of constraints, but can still be solved in polynomial time with a variant of the Ellipsoid method. Our "strong" algorithm for concurrent cluster scheduling refines the techniques contained therein, as well as those of Schulz [14, 15] (see Section 4).

Mastrolilli et al. [10] developed a primal-dual algorithm for $PD||\sum w_jC_j$ that does not use LP solvers. "MUSSQ"[3] is significant for both its speed and the strength of its performance guarantee : it achieves an approximation ratio of 2 in only $O(n^2 + nm)$ time. Although MUSSQ does not require an LP solver, its proof of correctness is based on the fact that it finds a feasible solution to the dual a particular linear program. Our "fast" algorithm for concurrent cluster scheduling uses MUSSQ as a subroutine (see Section 5).

Hung, Golubchik, and Yu [8] presented a framework designed to improve scheduling across geographically distributed data centers. The scheduling framework had a centralized scheduler (which determined a job ordering) and local dispatchers which carried out a schedule consistent with the controllers job ordering. Hung et al. proposed a particular algorithm for the controller called "SWAG." SWAG performed well in a wide variety of simulations where each data center was assumed to have the same number of identical parallel machines. We adopt a similar framework to Hung et al., but we show in Section 5.1 that SWAG has no constant-factor performance guarantee.

1.3 Paper Outline & Algorithmic Results

Although only one of our algorithms requires *solving* a linear program, both algorithms use the same linear program in their proofs of correctness; we introduce this linear program in Section 2 before discussing either algorithm. Section 3 establishes how an ordering of jobs can be processed to completely specify a schedule. This is important because the complex work in both of our algorithms is to generate an ordering of jobs for each cluster.

Section 4 introduces our "strong" algorithm: CC-LP. CC-LP can be applied to any instance of concurrent cluster scheduling, including those with non-zero release times r_{ji}. A key in CC-LP's strong performance guarantees lay in the fact that it allows different permutations of subjobs for different clusters. By providing additional structure to the problem (but while maintaining a generalization of concurrent open shop) CC-LP becomes a

[3] A permutation of the author's names: Mastrolilli, Queyranne, Schulz, Svensson, and Uhan.

2-approximation. This is significant because it is NP-Hard to approximate concurrent open shop (and by extension, our problem) with ratio $2 - \epsilon$ for any $\epsilon > 0$ [13].

Our combinatorial algorithm ("CC-TSPT") is presented in Section 5. The algorithm is fast, provably accurate, and has the interesting property that it can schedule all clusters using the same permutation of jobs.[4] After considering CC-TSPT in the general case, we show how fine-grained approximation ratios can be obtained in the "fully parallelizable" setting of Zhang et al. [17]. We conclude with an extension of CC-TSPT that maintains performance guarantees while offering improved empirical performance.

The following table summarizes our results for approximation ratios. For compactness, condition Id refers to identical machines (i.e. $v_{\ell i}$ constant over ℓ), condition A refers to $r_{ji} \equiv 0$, and condition B refers to p_{jit} constant over $t \in T_{ji}$.

	(Id, A, B)	$(Id, \neg A, B)$	$(Id, A, \neg B)$	$(Id, \neg A, \neg B)$	$(\neg Id, A)$	$(\neg Id, \neg A)$
CC-LP	2	3	3	4	$2 + R$	$3 + R$
CC-TSPT	3	-	3	-	$2 + R$	-

The term R is the maximum over i of R_i, where R_i is the ratio of fastest machine to *average* machine speed at cluster i.

The most surprising of all of these results is that our scheduling algorithms are remarkably simple. The first algorithm solves an LP, and then the scheduling can be done easily on each cluster. The second algorithm is again a rather surprising simple reduction to the case of one machine per cluster (the well understood concurrent open shop problem) and yields a simple combinatorial algorithm. The proof of the approximation guarantee is somewhat involved however.

In addition to algorithmic results, we demonstrate how our problem subsumes that of minimizing total weighted lateness on a bank of identical parallel machines (see Section 6). Section 7 provides additional discussion and highlights our more novel technical contributions.

2 The Core Linear Program

Our linear program has an unusual form. Rather than introduce it immediately, we conduct a brief review of prior work on similar LP's. All the LP's we discuss in this paper have objective function $\sum w_j C_j$, where C_j is a decision variable corresponding to the completion time of job j, and w_j is a weight associated with job j.

For the following discussion only, we adopt the notation in which job j has processing time p_j. In addition, if multiple machine problems are discussed, we will say that there are m such machines (possibly with speeds $s_i, i \in \{1, \ldots, \mathsf{m}\}$).

The earliest appearance of a similar linear program comes from Queyranne [12]. In his paper, Queyranne presents an LP relaxation for sequencing n jobs on a single machine where all constraints are of the form $\sum_{j \in S} p_j C_j \geq \frac{1}{2}\left[\left(\sum_{j \in S} p_j\right)^2 + \sum_{j \in S} p_j^2\right]$ where S is an arbitrary subset of jobs. Once a set of optimal $\{C_j^\star\}$ is found, the jobs are scheduled in increasing order of $\{C_j^\star\}$. These results were primarily theoretical, as it was known at his time of writing that sequencing n jobs on a single machine to minimize $\sum w_j C_j$ can be done optimally in $O(n \log n)$ time.

[4] We call such schedules "single-σ schedules." As we will see later on, CC-TSPT serves as a constructive proof of existence of near-optimal single-σ schedules for all instances of $CC||\sum w_j C_j$, *including* those instances for which single-σ schedules are strictly sub-optimal. This is addressed in Section 7.

Queyranne's constraint set became particularly useful for problems with *coupling* across distinct machines (as occurs in concurrent open shop). Four separate groups [3, 4, 9, 10] saw this and used the following LP in a 2-approximation for concurrent open shop scheduling.

$$\text{(LP0)} \quad \min \sum_{j \in N} w_j C_j \quad \text{s.t.} \quad \textstyle\sum_{j \in S} p_{ji} C_j \geq \frac{1}{2} \left[\left(\sum_{j \in S} p_{ji} \right)^2 + \left(\sum_{j \in S} p_{ji}^2 \right) \right] \ \forall \ {\scriptstyle S \subseteq N \atop i \in M}$$

In view of its tremendous popularity, we sometimes refer to the linear program above as the *canonical relaxation* for concurrent open shop.

Andreas Schulz's Ph.D. thesis developed Queyranne's constraint set in greater depth [14]. As part of his thesis, Schulz considered scheduling n jobs on m identical parallel machines with constraints of the form $\sum_{j \in S} p_j C_j \geq \frac{1}{2\mathsf{m}} \left(\sum_{j \in S} p_j \right)^2 + \frac{1}{2} \sum_{j \in S} p_j^2$. In addition, Schulz showed that the constraints $\sum_{j \in S} p_j C_j \geq \left[2 \sum_{i=1}^{\mathsf{m}} s_i \right]^{-1} \left[\left(\sum_{j \in S} p_j \right)^2 + \sum_{j \in S} p_j^2 \right]$ are satisfied by any schedule of n jobs on m uniform machines. In 2012, Schulz refined the analysis for several of these problems [15]. For constructing a schedule from the optimal $\{C_j^\star\}$, Schulz considered scheduling jobs by increasing order of $\{C_j^\star\}$, $\{C_j^\star - p_j/2\}$, and $\{C_j^\star - p_j/(2\mathsf{m})\}$.

2.1 Statement of LP1

The model we consider allows for more fine-grained control of the job structure than is indicated by the LP relaxations above. Inevitably, this comes at some expense of simplicity in LP formulations. In an effort to simplify notation, we define the following constants, and give verbal interpretations for each.

$$\mu_i \doteq \textstyle\sum_{\ell=1}^{m_i} v_{\ell i} \qquad q_{ji} \doteq \min\{|T_{ji}|, m_i\} \qquad \mu_{ji} \doteq \textstyle\sum_{\ell=1}^{q_{ji}} v_{\ell i} \qquad p_{ji} \doteq \textstyle\sum_{t \in T_{ji}} p_{jit} \tag{1}$$

From these definitions, μ_i is the processing power of cluster i. For subjob (j,i), q_{ji} is the maximum number of machines that could process the subjob, and μ_{ji} is the maximum processing power than can be brought to bear on the same. Lastly, p_{ji} is the total processing requirement of subjob (j,i). In these terms, the core linear program, LP1, is as follows.

$$\begin{aligned}
&\text{(LP1)} \ \min \textstyle\sum_{j \in N} w_j C_j \\
s.t. \quad &(1A) \quad \textstyle\sum_{j \in S} p_{ji} C_j \geq \frac{1}{2} \left[\left(\sum_{j \in S} p_{ji} \right)^2 / \mu_i + \sum_{j \in S} p_{ji}^2 / \mu_{ji} \right] \qquad \forall S \subseteq N, i \in M \\
&(1B) \quad C_j \geq p_{jit}/v_{1i} + r_{ji} \qquad \forall i \in M,\ j \in N,\ t \in T_{ji} \\
&(1C) \quad C_j \geq p_{ji}/\mu_{ji} + r_{ji} \qquad \forall j \in N,\ i \in M
\end{aligned}$$

Constraints $(1A)$ are more carefully formulated versions of the polyhedral constraints introduced by Queyranne [12] and developed by Schulz [14]. The use of μ_{ji} term is new and allows us to provide stronger performance guarantees for our framework where subjobs are composed of *sets* of tasks. As we will see, this term is one of the primary factors that allows us to parametrize results under varying machine speeds in terms of maximum to *average* machine speed, rather than maximum to *minimum* machine speed. Constraints $(1B)$ and $(1C)$ are simple lower bounds on job completion time.

The majority of this section is dedicated to proving that LP1 is a valid relaxation of $CC|r|\sum w_j C_j$. Once this is established, we prove the that LP1 can be solved in polynomial time by providing a separation oracle with use in the Ellipsoid method. Both of these proofs use techniques established in Schulz's Ph.D. thesis [14].

2.2 Proof of LP1's Validity

The lemmas below establish the basis for both of our algorithms. Lemma 2 generalizes an inequality used by Schulz [14]. Lemma 3 relies on Lemma 2 and cites an inequality mentioned in the preceding section (and proven by Queyranne [12]).

▶ **Lemma 2.** *Let $\{a_1, \dots a_z\}$ be a set of non-negative real numbers. We assume that $k \leq z$ of them are positive. Let b_i be a set of decreasing positive real numbers. Then*

$$\sum_{i=1}^{z} a_i^2/b_i \geq \left(\sum_{i=1}^{z} a_i\right)^2 / \left(\sum_{i=1}^{k} b_i\right).$$

Proof. [5] We only show the case where $k = z$. Define $\mathbf{a} = [a_1, \dots, a_k] \in \mathbb{R}_+^k$, $\mathbf{b} = [b_1, \dots, b_k] \in \mathbb{R}_{++}^k$, and $\mathbb{1}$ as the vector of k ones. Now, set $\mathbf{u} = \mathbf{a}/\sqrt{\mathbf{b}}$ and $\mathbf{w} = \sqrt{\mathbf{b}}$ (element-wise), and note that $\langle \mathbf{a}, \mathbb{1} \rangle = \langle \mathbf{u}, \mathbf{w} \rangle$. In these terms, it is clear that $(\sum_{i=1}^{k} a_i)^2 = \langle \mathbf{u}, \mathbf{w} \rangle^2$.

Given this, one need only cite Cauchy-Schwarz (namely, $\langle \mathbf{u}, \mathbf{w} \rangle^2 \leq \langle \mathbf{u}, \mathbf{u} \rangle \cdot \langle \mathbf{w}, \mathbf{w} \rangle$) and plug in the definitions of $\mathbf{u}$ and $\mathbf{w}$ to see the desired result. ◀

▶ **Lemma 3** (Validity Lemma). *Every feasible schedule for an instance I of $CC|r|\sum w_j C_j$ has completion times that define a feasible solution to LP1(I).*

Proof. As constraints $(1B)$ and $(1C)$ are clear lower bounds on job completion time, it suffices to show the validity of constraint $(1A)$. Thus, let S be a non-empty subset of N, and fix an arbitrary but feasible schedule "F" for I.

Define C_{ji}^F as the completion time of subjob (j, i) under schedule F. Similarly, define $C_{ji\ell}^F$ as the first time at which tasks of subjob (j, i) scheduled on machine ℓ of cluster i are finished. Lastly, define p_{ji}^ℓ as the total processing requirement of job j scheduled on machine ℓ of cluster i. Note that by construction, we have $C_{ji}^F = \max_{\ell \in \{1,\dots,m_i\}} C_{ji\ell}^F$ and $C_j^F = \max_{i \in M} C_{ji}^F$. Since $p_{ji} = \sum_{\ell=1}^{m_i} p_{ji}^\ell$, we can rather innocuously write

$$\sum_{j \in S} p_{ji} C_{ji}^F = \sum_{j \in S} \left[\sum_{\ell=1}^{m_i} p_{ji}^\ell\right] C_{ji}^F. \tag{2}$$

But using $C_{ji}^F \geq C_{ji\ell}^F$, we can lower-bound $\sum_{j \in S} p_{ji} C_{ji}^F$. Namely,

$$\sum_{j \in S} p_{ji} C_{ji}^F \geq \sum_{j \in S} \sum_{\ell=1}^{m_i} p_{ji}^\ell C_{ji\ell}^F = \sum_{\ell=1}^{m_i} v_{\ell i} \sum_{j \in S} \left[p_{ji}^\ell / v_{\ell i}\right] C_{ji\ell}^F \tag{3}$$

The next inequality uses a bound on $\sum_{j \in S} \left[p_{ji}^\ell / v_{\ell i}\right] C_{ji\ell}^F$ proven by Queyranne [12] for any subset S of N jobs with processing times $\left[p_{ji}^\ell / v_{\ell i}\right]$ to be scheduled on a single machine.[6]

$$\sum_{j \in S} \left[p_{ji}^\ell / v_{\ell i}\right] C_{ji\ell}^F \geq \tfrac{1}{2} \left[\left(\sum_{j \in S} \left[p_{ji}^\ell / v_{\ell i}\right]\right)^2 + \sum_{j \in S} \left(\left[p_{ji}^\ell / v_{\ell i}\right]\right)^2\right] \tag{4}$$

Combining inequalities (3) and (4), we have the following.

$$\sum_{j \in S} p_{ji} C_{ji}^F \geq \frac{1}{2} \sum_{\ell=1}^{m_i} v_{\ell i} \left[\left(\sum_{j \in S} \left[p_{ji}^\ell / v_{\ell i}\right]\right)^2 + \sum_{j \in S} \left(\left[p_{ji}^\ell / v_{\ell i}\right]\right)^2\right] \tag{5}$$

$$\geq \frac{1}{2} \left[\sum_{\ell=1}^{m_i} \left(\sum_{j \in S} p_{ji}^\ell\right)^2 / v_{\ell i} + \sum_{j \in S} \sum_{\ell=1}^{m_i} \left(p_{ji}^\ell\right)^2 / v_{\ell i}\right] \tag{6}$$

[5] The proceedings version of this paper stated that the proof cites the AM-GM inequality and proceeds by induction from $z = k = 2$. We have opted here to demonstrate a different (simpler) proof that we discovered only after the proceedings version was finalized.

[6] Here, our machine is machine ℓ on cluster i.

Next, we apply Lemma 2 to the right hand side of inequality (6) a total of $|S|+1$ times.

$$\sum_{\ell=1}^{m_i}\left(\sum_{j\in S}p_{ji}^{\ell}\right)^2/v_{\ell i} \geq \left(\sum_{\ell=1}^{m_i}\sum_{j\in S}p_{ji}^{\ell}\right)^2/\sum_{\ell=1}^{m_i}v_{\ell i} = \left(\sum_{j\in S}p_{ji}\right)^2/\mu_i \tag{7}$$

$$\sum_{\ell=1}^{m_i}\left(p_{ji}^{\ell}\right)^2/v_{\ell i} \geq \left(\sum_{\ell=1}^{m_i}p_{ji}^{\ell}\right)^2/\sum_{\ell=1}^{q_{ji}}v_{\ell i} = p_{ji}^2/\mu_{ji} \quad \forall\ j\in S \tag{8}$$

Citing $C_j^F \geq C_{ji}^F$, we arrive at the desired result.

$$\sum_{j\in S}p_{ji}C_j^F \geq \frac{1}{2}\left[\left(\sum_{j\in S}p_{ji}\right)^2/\mu_i + \sum_{j\in S}p_{ji}^2/\mu_{ji}\right] \qquad \text{“constraint } (1A)\text{”} \tag{9}$$

◀

2.3 Theoretical Complexity of LP1

As the first of our two algorithms requires solving LP1 directly, we need to address the fact that LP1 has $m\cdot(2^n-1)+n$ constraints. Luckily, it is still possible to such solve linear programs in polynomial time with the Ellipsoid method; we introduce the following separation oracle for this purpose.

▶ **Definition 4** (Oracle LP1)**.** Define the *violation*

$$V(S,i) = \frac{1}{2}\left[\left(\sum_{j\in S}p_{ji}\right)^2/\mu_i + \sum_{j\in S}p_{ji}^2/\mu_{ji}\right] - \sum_{j\in S}p_{ji}C_j \tag{10}$$

Let $\{C_j\}\in\mathbb{R}^n$ be a *potentially* feasible solution to LP1. Let σ_i denote the ordering when jobs are sorted in increasing order of $C_j - p_{ji}/(2\mu_{ji})$. Find the most violated constraint in $(1A)$ for $i\in M$ by searching over $V(S_i,i)$ for S_i of the form $\{\sigma_i(1),\ldots,\sigma_i(j-1),\sigma_i(j)\},\ j\in\{1,\ldots,n\}$. If any of maximal $V(S_i^*,i)>0$, then return (S_i^*,i) as a violated constraint for $(1A)$. Otherwise, check the remaining n constraints ($(1B)$ and $(1C)$) directly in linear time.

For fixed i, Oracle-LP1 finds the subset of jobs that maximizes “violation” for cluster i. That is, Oracle-LP1 finds S_i^* such that $V(S_i^*,i)=\max_{S\subset N}V(S,i)$. We prove the correctness of Oracle-LP1 by establishing a necessary and sufficient condition for a job j to be in S_i^*.

▶ **Lemma 5.** *For $\mathbb{P}_i(A)\doteq\sum_{j\in A}p_{ji}$, we have $x\in S_i^* \Leftrightarrow C_x - p_{xi}/(2\mu_{xi}) \leq \mathbb{P}_i(S_i^*)/\mu_i$.*

Proof. For given S (not necessarily equal to S_i^*), it is useful to express $V(S,i)$ in terms of $V(S\cup x,i)$ or $V(S\setminus x,i)$ (depending on whether $x\in S$ or $x\in N\setminus S$). Without loss of generality, we restrict our search to $S: x\in S\Rightarrow p_{x,i}>0$.

Suppose $x\in S$. By writing $\mathbb{P}_i(S)=\mathbb{P}_i(S\setminus x)+\mathbb{P}_i(x)$, and similarly decomposing the sum $\sum_{j\in S}p_{ji}^2/(2\mu_{ji})$, one can show the following.

$$V(S,i) = V(S\setminus x,i) + p_{xi}\left(\frac{1}{2}\left(\frac{2\mathbb{P}_i(S)-p_{xi}}{\mu_i}+\frac{p_{xi}}{\mu_{xi}}\right)-C_x\right) \tag{11}$$

Now suppose $x\in N\setminus S$. In the same strategy as above (this time writing $\mathbb{P}_i(S) = \mathbb{P}_i(S\cup x)-\mathbb{P}_i(x)$), one can show that

$$V(S,i) = V(S\cup x,i) + p_{xi}\left(C_x - \frac{1}{2}\left(\frac{2\mathbb{P}_i(S)+p_{xi}}{\mu_i}+\frac{p_{xi}}{\mu_{xi}}\right)\right). \tag{12}$$

Note that Equations (11) and (12) hold for all S, including $S=S_i^*$. Turning our attention to S_i^*, we see that $x\in S_i^*$ implies that the second term in Equation (11) is non-negative, i.e.

$$C_x - p_{xi}/(2\mu_{xi}) \leq \left(2\mathbb{P}_i(S_i^*)-p_{xi}\right)/(2\mu_i) < \mathbb{P}_i(S_i^*)/\mu_i. \tag{13}$$

Similarly, $x \in N \setminus S_i^*$ implies the second term in Equation (12) is non-negative.

$$C_x - p_{xi}/(2\mu_{xi}) \geq \left(2\mathbb{P}_i(S_i^*) + p_{xi}\right)/(2\mu_i) \geq \mathbb{P}_i(S_i^*)/\mu_i \tag{14}$$

It follows that $x \in S_i^*$ iff $C_x - p_{xi}/(2\mu_{xi}) < \mathbb{P}_i(S_i^*)/\mu_i$. ◀

Given Lemma 5, It is easy to verify that sorting jobs in increasing order of $C_x - p_{xi}/(2\mu_{xi})$ to define a permutation σ_i guarantees that S_i^* is of the form $\{\sigma_i(1), \ldots, \sigma_i(j-1), \sigma_i(j)\}$ for some $j \in N$. This implies that for fixed i, Oracle-LP1 finds S_i^* in $O(n\log(n))$ time. This procedure is executed once for each cluster, leaving the remaining n constraints in $(1B)$ and $(1C)$ to be verified in linear time. Thus Oracle-LP1 runs in $O(mn\log(n))$ time.

By the equivalence of separation and optimization, we have proven the following theorem:

▶ **Theorem 6.** *LP1(I) is a valid relaxation of $I \in \Omega_{CC}$, and is solvable in polynomial time.*

As was explained in the beginning of this section, linear programs such as those in [3, 4, 9, 12, 14, 15] are processed with an appropriate sorting of the optimal decision variables $\{C_j^*\}$. It is important then to have bounds on job completion times for a particular ordering of jobs. We address this next in Section 3, and reserve our first algorithm for Section 4.

3 List Scheduling from Permutations

The complex work in both of our proposed algorithms is to generate a *permutation* of jobs. The procedure below takes such a permutation and uses it to determine start times, end times, and machine assignments for every task of every subjob.

List-LPT : Given a single cluster with m_i machines and a permutation of jobs σ, introduce $\text{List}(a,i) \doteq (p_{ai1}, p_{ai2}, \ldots, p_{ai|T_{ai}|})$ as an ordered set of tasks belonging to subjob (a,i), ordered by longest processing time first. Now define $\text{List}(\sigma) \doteq \text{List}(\sigma(1),i) \oplus \text{List}(\sigma(2),i) \oplus \cdots \oplus \text{List}(\sigma(n),i)$, where $\oplus$ is the concatenation operator.

Place the tasks of $\text{List}(\sigma)$ in order- from the largest task of subjob $(\sigma(1),i)$, to the smallest task of subjob $(\sigma(n),i)$. When placing a particular task, assign it whichever machine and start time results in the task being completed as early as possible (without moving any tasks which have already been placed). Insert idle time (on all m_i machines) as necessary if this procedure would otherwise start a job before its release time.

The following Lemma is essential to bound the completion time of a set of jobs processed by List-LPT. The proof is adapted from Gonzalez et al. [5].

▶ **Lemma 7.** *Suppose n jobs are scheduled on cluster i according to List-LPT(σ). Then for $\bar{v_i} \doteq \mu_i/m_i$, the completion time of subjob $(\sigma(j),i)$ (denoted $C_{\sigma(j)i}$) satisfies*

$$C_{\sigma(j)i} \leq \max_{1\leq k\leq j} r_{\sigma(k)i} + p_{\sigma(j)i1}/\bar{v_i} + \left(\textstyle\sum_{k=1}^{j} p_{\sigma(k)i} - p_{\sigma(j)i1}\right)/\mu_i \tag{15}$$

Proof. For now, assume all jobs are released at time zero. Let the task of subjob $(\sigma(j),i)$ to finish last be denoted t^*. If t^* is not the task in $T_{\sigma(j)i}$ with least processing time, then construct a new set $T'_{\sigma(j)i} = \{t : p_{\sigma(j)it^*} \leq p_{\sigma(j)it}\} \subset T_{\sigma(j)i}$. Because the tasks of subjob $(\sigma(j),i)$ were scheduled by List-LPT (i.e. longest-processing-time-first), the sets of potential start times and machines for task t^* (and hence the set of potential completion times for

task t^*) are the same regardless of whether subjob $(\sigma(j), i)$ consisted of tasks $T_{\sigma(j)i}$ or the subset $T'_{\sigma(j)i}$. Accordingly, reassign $T_{\sigma(j)i} \leftarrow T'_{\sigma(j)i}$ without loss of generality.

Let D_ℓ^j denote the total demand for machine ℓ (on cluster i) once all tasks of subjobs $(\sigma(1), i)$ through $(\sigma(j-1), i)$ and all tasks in the set $T_{\sigma(j)i} \setminus \{t^*\}$ are scheduled. Using the fact that $C_{\sigma(j)i} v_{\ell i} \le (D_\ell^j + p_{\sigma(j)it^*}) \forall \ell \in \{1, \ldots, m_i\}$, sum the left and right and sides over ℓ. This implies $C_{\sigma(j)i} \left(\sum_{\ell=1}^{m_i} v_{\ell i}\right) \le\ m_i p_{\sigma(j)it^*} + \sum_{\ell=1}^{m_i} D_\ell^j$. Dividing by the sum of machine speeds and using the definition of μ_i yields

$$C_{\sigma(j)i} \ \le\ m_i p_{\sigma(j)it^*}/\mu_i + \textstyle\sum_{\ell=1}^{m_i} D_\ell^j/\mu_i \ \le\ p_{\sigma(j)i1}/\bar{v_i} + \left(\sum_{k=1}^{j} p_{\sigma(k)i} - p_{\sigma(j)i1}\right)/\mu_i \quad (16)$$

where we estimated $p_{\sigma(j)it^*}$ upward by $p_{\sigma(j)i1}$. Inequality (16) completes our proof in the case when $r_{ji} \equiv 0$.

Now suppose that some $r_{ji} > 0$. We take our policy to the extreme and suppose that all machines are left idle until every one of jobs $\sigma(1)$ through $\sigma(j)$ are released; note that this occurs precisely at time $\max_{1 \le k \le j} r_{\sigma(k)i}$. It is clear that beyond this point in time, we are effectively in the case where all jobs are released at time zero, hence we can bound the remaining time to completion by the right hand side of Inequality 16. As Inequality 15 simply adds these two terms, the result follows. ◀

Lemma 7 is cited directly in the proof of Theorem 8 and Lemma 13. Lemma 7 is used implicitly in the proofs of Theorems 9, 10, and 15.

4 An LP-based Algorithm

In this section we show how LP1 can be used to construct near optimal schedules for concurrent cluster scheduling both when $r_{ji} \equiv 0$ and when some $r_{ji} > 0$. Although solving LP1 is somewhat involved, the algorithm itself is quite simple:

Algorithm CC-LP : Let $I = (T, r, w, v)$ denote an instance of $CC|r|\sum w_j C_j$. Use the optimal solution $\{C_j^\star\}$ of LP1(I) to define m permutations $\{\sigma_i : i \in M\}$ which sort jobs in increasing order of $C_j^\star - p_{ji}/(2\mu_{ji})$. For each cluster i, execute List-LPT(σ_i).

Each theorem in this section can be characterized by how various assumptions help us cancel an additive term[7] in an upper bound for the completion time of an arbitrary subjob (x, i). Theorem 8 is the most general, while Theorem 10 is perhaps the most surprising.

4.1 CC-LP for Uniform Machines

▶ **Theorem 8.** *Let $\hat{C}_j$ be the completion time of job j using algorithm CC-LP, and let R be as in Section 1.3. If $r_{ji} \equiv 0$, then $\sum_{j \in N} w_j \hat{C}_j \le (2 + R)\, OPT$. Otherwise, $\sum_{j \in N} w_j \hat{C}_j \le (3 + R)\, OPT$.*

Proof. For $y \in \mathbb{R}$, define $y^+ = \max\{y, 0\}$. Now let $x \in N$ be arbitrary, and let $i \in M$ be such that $p_{xi} > 0$ (but otherwise arbitrary). Define t^* as the last task of job x to complete on cluster i, and let j_i be such that $\sigma_i(j_i) = x$. Lastly, denote the optimal LP solution $\{C_j\}$.[8] Because $\{C_j\}$ is a feasible solution to LP1, constraint $(1A)$ implies the following (set

[7] "$+p_{xit^*}$"; see associated proofs.
[8] We omit the customary $\star$ to avoid clutter in notation.

$S_i = \{\sigma_i(1), \ldots, \sigma_i(j_i - 1), x\}$)

$$\frac{\left(\sum_{k=1}^{j_i} p_{\sigma_i(k)i}\right)^2}{2\mu_i} \leq \sum_{k=1}^{j_i} p_{\sigma_i(k)i}\left(C_{\sigma_i(k)} - \frac{p_{\sigma_i(k)i}}{2\mu_{\sigma_i(k)i}}\right) \leq \left(C_x - \frac{p_{xi}}{2\mu_{xi}}\right)\sum_{k=1}^{j_i} p_{\sigma_i(k)i} \tag{17}$$

which in turn implies $\sum_{k=1}^{j_i} p_{\sigma_i(k)i}/\mu_i \leq 2C_x - p_{xi}/\mu_{xi}$.

If all subjobs are released at time zero, then we can combine this with Lemma 7 and the fact that $p_{xit^*} \leq p_{xi} = \sum_{t \in T_{xi}} p_{xit}$ to see the following (the transition from the first inequality the second inequality uses $C_x \geq p_{xit^*}/v_{1i}$ and $R_i = v_{1i}/\bar{v}_i$).

$$\hat{C}_{xi} \leq 2C_x - \frac{p_{xi}}{\mu_{xi}} + \frac{p_{xit^*}}{\bar{v}_i} - \frac{p_{xit^*}}{\mu_i} \leq C_x(2 + [R_i(1 - 2/m_i)]^+) \tag{18}$$

When one or more subjobs are released after time zero, Lemma 7 implies that it is sufficient to bound $\max\limits_{1 \leq k \leq j_i} \left\{r_{\sigma_i(k)i}\right\}$ by some constant multiple of C_x. Since σ_i is defined by increasing $L_{ji} \doteq C_j - p_{ji}/(2\mu_{ji})$, $L_{\sigma_i(a)i} \leq L_{\sigma_i(b)i}$ implies

$$r_{\sigma_i(a)i} + \frac{p_{\sigma_i(a)i}}{2\mu_{\sigma_i(a)i}} + \frac{p_{\sigma_i(b)i}}{2\mu_{\sigma_i(b)i}} \leq C_{\sigma_i(a)} - \frac{p_{\sigma_i(a)i}}{2\mu_{\sigma_i(a)i}} + \frac{p_{\sigma_i(b)i}}{2\mu_{\sigma_i(b)i}} \leq C_{\sigma_i(b)} \;\forall\; a \leq b \tag{19}$$

and so $\max_{1 \leq k \leq j_i}\left\{r_{\sigma_i(k)i}\right\} + p_{xi}/(2\mu_{xi}) \leq C_x$. As before, combine this with Lemma 7 and the fact that $p_{xit^*} \leq p_{xi} = \sum_{t \in T_{xi}} p_{xit}$ to yield the following inequalities

$$\hat{C}_{xi} \leq 3C_x - \frac{3p_{xi}}{2\mu_{xi}} + \frac{p_{xit^*}}{\bar{v}_i} - \frac{p_{xit^*}}{\mu_i} \leq C_x(3 + [R_i(1 - 5/(2m_i))]^+) \tag{20}$$

-which complete our proof. ◀

4.2 CC-LP for Identical Machines

▶ **Theorem 9.** *If machines are of unit speed, then CC-LP yields an objective that is...*

	$r_{ji} \equiv 0$	*some* $r_{ji} > 0$
single-task subjobs	$\leq$ *2 OPT*	$\leq$ *3 OPT*
multi-task subjobs	$\leq$ *3 OPT*	$\leq$ *4 OPT*

Proof. Define $[\cdot]^+$, x, C_x, $\hat{C}_x$, i, σ_i, and t^* as in Theorem 8. When $r_{ji} \equiv 0$, one need only give a more careful treatment of the first inequality in (18) (using $\mu_{ji} = q_{ji}$).

$$\hat{C}_{x,i} \leq 2C_x + p_{xit^*} - p_{xit^*}/m_i - p_{xi}/q_{xi} \leq C_x(2 + [1 - 1/m_i - 1/q_{xi}]^+) \tag{21}$$

Similarly, when some $r_{ji} > 0$, the first inequality in (20) implies the following.

$$\hat{C}_{x,i} \leq 3C_x + p_{xit^*} - p_{xit^*}/m_i - 3p_{xi}/(2q_{xi}) \leq C_x(3 + [1 - 1/m_i - 3/(2q_{xi})]^+) \tag{22}$$

◀

The key in the refined analysis of Theorem 9 lay in how $-p_{xi}/q_{xi}$ is used to annihilate $+p_{xit^*}$. While $q_{xi} = 1$ (i.e. single-task subjobs) is sufficient to accomplish this, it is not strictly *necessary*. The theorem below shows that we can annihilate the $+p_{xit^*}$ term whenever all tasks of a given subjob are of the same length. Note that the tasks need not be *unit*, as the lengths of tasks across different subjobs can differ.

▶ **Theorem 10.** *Suppose $v_{\ell i} \equiv 1$. If p_{jit} is constant over $t \in T_{ji}$ for all $j \in N$ and $i \in M$, then algorithm CC-LP is a 2-approximation when $r_{ji} \equiv 0$, and a 3-approximation otherwise.*

Proof. The definition of p_{xi} gives $p_{xi}/q_{xi} = \sum_{t \in T_{xi}} p_{xit}/q_{xi}$. Using the assumption that p_{jit} is constant over $t \in T_{ji}$, we see that $p_{xi}/q_{xi} = (q_{xi} + |T_{xi}| - q_{xi})p_{xit^*}/q_{xi}$, where $|T_{xi}| \geq q_{xi}$. Apply this to Inequality (21) from the proof of Theorem 9; some algebra yields

$$\hat{C}_{xi} \leq 2C_x - p_{xit^*}/m_i - p_{xit^*}\left(|T_{xi}| - q_{xi}\right)/q_{xi} \leq 2C_x. \tag{23}$$

The case with some $r_{ji} > 0$ uses the same identity for p_{xi}/q_{xi}. ◀

Sachdeva and Saket [13] showed that it is NP-Hard to approximate $CC|m_i \equiv 1|\sum w_j C_j$ with a constant factor less than 2. Theorem 10 is significant because it shows that CC-LP can attain the same guarantee for *arbitrary* m_i, provided $v_{\ell i} \equiv 1$ and p_{jit} is constant over t.

5 Combinatorial Algorithms

In this section, we introduce an extremely fast combinatorial algorithm with performance guarantees similar to CC-LP for "unstructured" inputs (i.e. those for which some $v_{\ell i} > 1$, or some T_{ji} have p_{jit} non-constant over t). We call this algorithm *CC-TSPT*. CC-TSPT uses the MUSSQ algorithm for concurrent open shop (from [10]) as a subroutine. As SWAG (from [8]) motivated development of CC-TSPT, we first address SWAG's worst-case performance.

5.1 A Degenerate Case for SWAG

As a prerequisite for addressing worst-case performance of an existing algorithm, we provide psuedocode and an accompanying verbal description for SWAG.

SWAG computes queue positions for every subjob of every job, supposing that each job was scheduled next. A job's potential makespan ("mkspn") is the largest of the potential finish times of all of its subjobs (considering current queue lengths q_i and each subjob's processing time p_{ji}). Once potential makespans have been determined, the job with smallest potential makespan is selected for scheduling. At this point, all queues are updated. Because queues are updated, potential makespans will need to be re-calculated at the next iteration. Iterations continue until the very last job is scheduled. Note that SWAG runs in $O(n^2 m)$ time.

```
1: procedure SWAG(N, M, p_ji)
2:     J ← ∅
3:     q_i ← 0, ∀i ∈ M
4:     while |J| ≠ |N| do
5:         mkspn_j ← max_{i∈M} ((q_i + p_ji)/m_i)
               ∀j ∈ N \ J
6:         nextJob ← argmin_{j∈N\J} mkspn_j
7:         J.append(nextJob)
8:         q_i ← q_i + p_ji
9:     end while
10:    return J
11: end procedure
```

▶ **Theorem 11.** *For an instance I of $PD||\sum C_j$, let $SWAG(I)$ denote the objective function value of SWAG applied to I, and let $OPT(I)$ denote the objective function value of an optimal solution to I. Then for all $L \geq 1$, there exists an $I \in \Omega_{PD||\sum C_j}$ such that $SWAG(I)/OPT(I) > L$.*

Proof. Let $L \in \mathbb{N}^+$ be a fixed but arbitrary constant. Construct a problem instance I_L^m as follows:

$N = N_1 \cup N_2$ where N_1 is a set of m jobs, and N_2 is a set of L jobs. Job $j \in N_1$ has processing time p on cluster j and zero all other clusters. Job $j \in N_2$ has processing time $p(1-\epsilon)$ on all m clusters. ϵ is chosen so that $\epsilon < 1/L$ (see Figure 3).

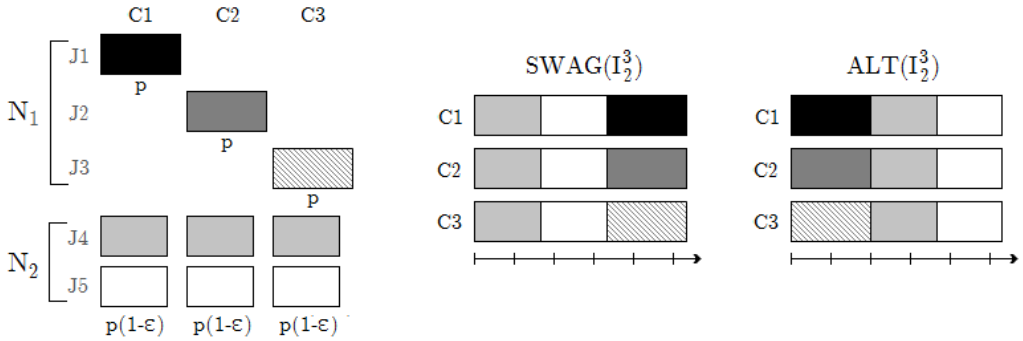

Figure 3 At left, an input for SWAG example with $m = 3$ and $L = 2$. At right, SWAG's resulting schedule, and an alternative schedule.

It is easy to verify that SWAG will generate a schedule where all jobs in N_2 precede all jobs in N_1 (due to the savings of $p\epsilon$ for jobs in N_2). We propose an *alternative* solution in which all jobs in N_1 preceed all jobs in N_2. Denote the objective value for this alternative solution $ALT(I_L^m)$, noting $ALT(I_L^m) \geq OPT(I_L^m)$.

By symmetry, and the fact that all clusters have a single machine, we can see that $SWAG(I_L^m)$ and $ALT(I_L^m)$ are given by the following

$$SWAG(I_L^m) = p(1-\epsilon)L(L+1)/2 + p(1-\epsilon)Lm + pm \tag{24}$$

$$ALT(I_L^m) = p(1-\epsilon)L(L+1)/2 + pL + pm \tag{25}$$

Since L is fixed, we can take the limit with respect to m.

$$\lim_{m\to\infty} \frac{SWAG(I_L^m)}{ALT(I_L^m)} = \lim_{m\to\infty} \frac{p(1-\epsilon)Lm + pm}{pm} = L(1-\epsilon) + 1 > L \tag{26}$$

The above implies the existence of a sufficiently large number of clusters $\overline{m}$, such that $m \geq \overline{m}$ implies $SWAG(I_L^m)/OPT(I_L^m) > L$. This completes our proof. ◀

Theorem 11 demonstrates that that although SWAG performed well in simulations, it may not be reliable. The rest of this section introduces an algorithm not only with superior runtime to SWAG (generating a permutation of jobs in $O(n^2+nm)$ time, rather than $O(n^2m)$ time), but also a constant-factor performance guarantee.

5.2 CC-TSPT : A Fast 2 + R Approximation

Our combinatorial algorithm for concurrent cluster scheduling exploits an elegant transformation to concurrent open shop. Once we consider this simpler problem, it can be handled with MUSSQ [10] and List-LPT. Our contributions are twofold: (1) we prove that this intuitive technique yields an approximation algorithm for a decidedly more general problem, and (2) we show that a *non-intuitive* modification can be made that maintains theoretical bounds while improving empirical performance. We begin by defining our transformation.

▶ **Definition 12** (The Total Scaled Processing Time (TSPT) Transformation)**.** Let Ω_{CC} be the set of all instances of $CC||\sum w_jC_j$, and let Ω_{PD} be the set of all instances of $PD||\sum w_jC_j$. Note that $\Omega_{PD} \subset \Omega_{CC}$. Then the Total Scaled Processing Time Transformation is a mapping

$$TSPT: \ \Omega_{CC} \to \Omega_{PD} \quad \text{with} \quad (T, v, w) \mapsto (X, w) \ : \ x_{ji} = \textstyle\sum_{t \in T_{ji}} p_{jit}/\mu_i$$

i.e., x_{ji} is the total processing time required by subjob (j, i), scaled by the sum of machine speeds at cluster i. Throughout this section, we will use $I = (T, v, w)$ to denote an arbitrary instance of $CC||\sum w_jC_j$, and $I' = (X, w)$ as the image of I under TSPT. Figure 4 shows the result of TSPT applied to our baseline example.

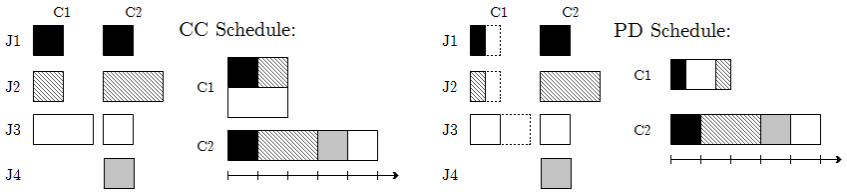

Figure 4 An instance I of $CC||\sum w_jC_j$, and its image $I' = TSPT(I)$. The schedules were constructed with List-LPT using the same permutation for I and I'.

We take the time to emphasize the simplicity of our reduction. Indeed, the TSPT transformation is perhaps the first thing one would think of given knowledge of the concurrent open shop problem. What is surprising is how one can attain constant-factor performance guarantees even after such a simple transformation.

Algorithm CC-TSPT : Execute MUSSQ on $I' = TSPT(I)$ to generate a permutation of jobs σ. List schedule instance I by σ on each cluster according to List-LPT.

Towards proving the approximation ratio for CC-TSPT, we will establish a critical inequality in Lemma 13. The intuition behind Lemma 13 requires thinking of every job j in I as having a corresponding representation in j' in I'. Job j in I will be scheduled in the CC environment, while job j' in I' will be scheduled in the PD environment. We consider what results when the same permutation σ is used for scheduling in both environments.

Now the definitions for the lemma: let $C^{CC}_{\sigma(j)}$ be the completion time of job $\sigma(j)$ resulting from List-LPT on an arbitrary permutation σ. Define $C^{CC\star}_{\sigma(j)}$ as the completion time of job $\sigma(j)$ in the CC environment in the optimal solution. Lastly, define $C^{PD,I'}_{\sigma(j')}$ as the completion time of job $\sigma(j')$ in I' when scheduling by List-LPT(σ) in the PD environment.

▶ **Lemma 13.** *For $I' = TSPT(I)$, let j' be the job in I' corresponding to job j in I. For an arbitrary permutation of jobs σ, we have $C^{CC}_{\sigma(j)} \le C^{PD,I'}_{\sigma(j')} + R \cdot C^{CC\star}_{\sigma(j)}$.*

Proof. After list scheduling has been carried out in the CC environment, we may determine $C^{CC}_{\sigma(j)i}$ - the completion time of subjob $(\sigma(j), i)$. We can bound $C^{CC}_{\sigma(j)i}$ using Lemma 7 (which implies (27)), and the serial-processing nature of the PD environment (which implies (28)).

$$C^{CC}_{\sigma(j)i} \le p_{\sigma(j)i1}\,(1/\bar{v} - 1/\mu_i) + \textstyle\sum_{\ell=1}^{j} p_{\sigma(\ell)i}/\mu_i \tag{27}$$

$$\textstyle\sum_{\ell=1}^{j} p_{\sigma(\ell)i}/\mu_i \le C^{PD,I'}_{\sigma(j')} \quad \forall\ i \in M \tag{28}$$

If we relax the bound given in Inequality (27) and combine it with Inequality (28), we see that $C^{CC}_{\sigma(j)i} \le C^{PD,I'}_{\sigma(j')} + p_{\sigma(j)i1}/\bar{v}$. The last step is to replace the final term with something

more meaningful. Using $p_{\sigma(j)1}/\bar{v} \leq R \cdot C^{CC\star}_{\sigma(j)}$ (which is immediate from the definition of R) the desired result follows. ◀

While Lemma 13 is true for arbitrary σ, now we consider $\sigma = MUSSQ(X, w)$. The proof of MUSSQ's correctness established the first inequality in the chain of inequalities below. The second inequality can be seen by substituting p_{ji}/μ_i for x_{ji} in LP0(I') (this shows that the constraints in LP0(I') are weaker than those in LP1(I)). The third inequality follows from the Validity Lemma.

$$\sum_{j\in N} w_{\sigma(j)} C^{PD,I'}_{\sigma(j)} \leq 2\sum_{j\in N} w_j C^{\text{LP0}(I')}_j \leq 2\sum_{j\in N} w_j C^{\text{LP1}(I)}_j \leq 2OPT(I) \tag{29}$$

Combining Inequality (29) with Lemma 13 allows us to bound the objective in a way that does not make reference to I'.

$$\sum_{j\in N} w_{\sigma(j)} C^{CC}_{\sigma(j)} \leq \sum_{j\in N} w_{\sigma(j)} \left[C^{PD,I'}_{\sigma(j)} + R \cdot C^{CC\star}_{\sigma(j)}\right] \leq \ 2 \cdot OPT(I) + R \cdot OPT(I) \tag{30}$$

Inequality (30) completes our proof of the following theorem.

▶ **Theorem 14.** *Algorithm CC-TSPT is a* $2 + R$ *approximation for* $CC||\sum w_j C_j$.

5.3 CC-TSPT with Unit Tasks and Identical Machines

Consider concurrent cluster scheduling with $v_{\ell i} = p_{jit} = 1$ (i.e., all processing times are unit, although the size of the collections T_{ji} are unrestricted). In keeping with the work of Zhang, Wu, and Li [17] (who studied this problem in the single-cluster case), we call instances with these parameters "fully parallelizable," and write $\beta = fps$ for Graham's $\alpha|\beta|\gamma$ taxonomy.

Zhang et al. showed that scheduling jobs greedily by "Largest Ratio First" (decreasing w_j/p_j) results in a 2-approximation, where 2 is a tight bound. This comes as something of a surprise since the Largest Ratio First policy is *optimal* for $1||\sum w_j C_j$ - which their problem very closely resembles. We now formalize the extent to which $P|fps|\sum w_j C_j$ resembles $1||\sum w_j C_j$: define the *time resolution* of an instance I of $CC|fps|\sum w_j C_j$ as $\rho_I = \min_{j\in N, i\in M} \lceil p_{ji}/m_i \rceil$. Indeed, one can show that as the time resolution increases, the performance guarantee for LRF on $P|fps|\sum w_j C_j$ approaches that of LRF on $1||\sum w_j C_j$. We prove the analogous result for our problem.

▶ **Theorem 15.** *CC-TSPT for* $CC|fps|\sum w_j C_j$ *is a* $(2 + 1/\rho_I)-$*approximation.*

Proof. Applying techniques from the proof of Lemma 13 under the hypothesis of this theorem, we have $C^{CC}_{\sigma(j),i} \leq C^{PD,I'}_{\sigma(j)} + 1$. Next, use the fact that for all $j \in N$, $C^{CC,OPT}_{\sigma(j)} \geq \rho_I$ by the definition of ρ_I. These facts together imply $C^{CC}_{\sigma(j),i} \leq C^{PD,I'}_{\sigma(j)} + C^{CC,OPT}/\rho_I$. Thus

$$\sum_{j\in N} w_j C^{CC}_{\sigma(j)} \leq \sum_{j\in N} w_j \left[C^{PD,I'}_{\sigma(j)} + C^{CC,OPT}/\rho_I\right] \leq 2 \cdot OPT + OPT/\rho_I. \tag{31}$$

◀

5.4 CC-ATSPT : Augmenting the LP Relaxation

The proof of Theorem 14 appeals to a trivial lower bound on $C^{CC\star}_{\sigma(j)}$, namely $p_{\sigma(j)1}/\bar{v} \leq R \cdot C^{CC\star}_{\sigma(j)}$. We attain constant-factor performance guarantees in spite of this, but it is natural to wonder how the *need* for such a bound might come hand-in-hand with empirical weaknesses. Indeed, TSPT can make subjobs consisting of many small tasks look the same as subjobs consisting of a single very long task. Additionally, a cluster hosting a subjob with a single

extremely long task might be identified as a bottleneck by MUSSQ, even if that cluster has more machines than it does tasks to process.

We would like to mitigate these issues by introducing the simple lower bounds on C_j as seen in constraints $(1B)$ and $(1C)$. This is complicated by the fact that MUSSQ's proof of correctness only allows constraints of the form in $(1A)$. For $I \in \Omega_{PD}$ this is without loss of generality, since $|S| = 1$ in LP0 implies $C_j \geq p_{ji}$, but since we apply LP0 to $I' = TSPT(I)$, $C_j \geq x_{ji}$ is equivalent to $C_j \geq p_{ji}/\mu_i$ (a much weaker bound than we desire).

Nevertheless, we can bypass this issue by introducing additional clusters and appropriately defined subjobs. We formalize this with the "Augmented Total Scaled Processing Time" (ATSPT) transformation. Conceptually, ATSPT creates n "imaginary clusters", where each imaginary cluster has nonzero processing time for exactly one job.

▶ **Definition 16** (The Augmented TSPT Transformation)**.** Let Ω_{CC} and Ω_{PD} be as in the definition for TSPT. Then the Augmented TSPT Transformation is likewise a mapping

$$ATSPT:\ \Omega_{CC} \to \Omega_{PD} \quad \text{with} \quad (T, v, w) \mapsto (X, w)\ :\ X = \left[\ X_{TSPT(I)}\ \middle|\ D\ \right].$$

Where $D \in \mathbb{R}^{n \times n}$ is a diagonal matrix with d_{jj} as any valid lower bound on the completion time of job j (such as the right hand sides of constraints $(1B)$ and $(1C)$ of LP1).

Given that d_{jj} is a valid lower bound on the completion time of job j, it is easy to verify that for $I' = ATSPT(I)$, LP1(I') is a valid relaxation of I. Because MUSSQ returns a permutation of jobs for use in list scheduling by List-LPT, these "imaginary clusters" needn't be accounted for beyond the computations in MUSSQ.

6 A Reduction for Minimizing Total Weighted Lateness on Identical Parallel Machines

The problem of minimizing total weighted lateness on a bank of identical parallel machines is typically denoted $P||\sum w_j L_j$, where the lateness of a job with deadline d_j is $L_j \doteq \max\{C_j - d_j, 0\}$. The reduction we offer below shows that $P||\sum w_j L_j$ can be stated in terms of $CC||\sum w_j C_j$ *at optimality*. Thus while a Δ approximation to $CC||\sum w_j C_j$ does not imply a Δ approximation to $P||\sum w_j L_j$, the reduction below nevertheless provides new insights on the structure of $P||\sum w_j L_j$.

▶ **Definition 17** (Total Weighted Lateness Reduction)**.** Let $I = (p, d, w, m)$ denote an instance of $P||\sum w_j L_j$. p is the set of processing times, d is the set of deadlines, w is the set of weights, and m is the number of identical parallel machines. Given these inputs, we transform $I \in \Omega_{P||\sum w_j L_j}$ to $I' \in \Omega_{CC}$ in the following way.

Create a total of $n + 1$ clusters. Cluster 0 has m machines. Job j has processing time p_j on this cluster, and $|T_{j0}| = 1$. Clusters 1 through n each consist of a single machine. Job j has processing time d_j on cluster j, and zero on all clusters other than cluster 0 and cluster j. Denote this problem I'.

We refer the reader to Figure 2 for an example output of this reduction.

▶ **Theorem 18.** *Let I be an instance of $P||\sum w_j L_j$. Let I' be an instance of $CC||\sum w_j C_j$ resulting from the transformation described above. Any list schedule σ that is optimal for I' is also optimal for I.*

Proof. If we restrict the solution space of I' to single permutations (which we may do without loss of generality), then any schedule σ for I or I' produces the same value of

$\sum_{j\in N} w_j(C_j - d_j)^+$ for I and I'. The additional clusters we added for I' ensure that $C_j \geq d_j$. Given this, the objective for I can be written as $\sum_{j\in N} w_j d_j + w_j(C_j - d_j)^+$. Because $w_j d_j$ is a constant, any permutation to solve I' optimally also solves $\sum_{j\in N} w_j(C_j - d_j)^+$ optimally. Since $\sum_{j\in N} w_j(C_j - d_j)^+ = \sum_{j\in N} w_j L_j$, we have the desired result. ◀

7 Closing Remarks

We now take a moment to address a subtle issue in the concurrent cluster problem: what price do we pay for using the same permutation on all clusters (i.e. single-σ schedules)? For concurrent open shop, it has been shown ([16, 10]) that single-σ schedules may be assumed without loss of optimality. As is shown in Figure 5, this does *not* hold for concurrent cluster scheduling in the general case. In fact, that is precisely why the strong performance guarantees for algorithm CC-LP rely on clusters having possibly unique permutations.

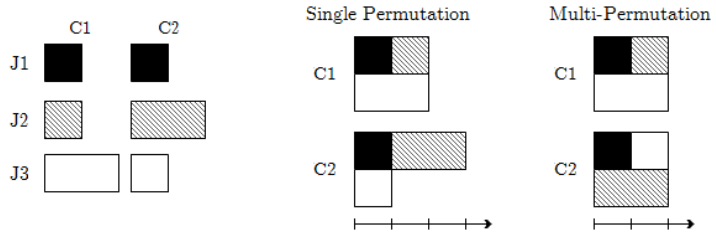

Figure 5 An instance of $CC||\sum C_j$ (i.e. $w_j \equiv 1$) for which there does not exist a single-σ schedule which attains the optimal objective value. In the single-σ case, one of the jobs necessarily becomes delayed by one time unit compared to the multi-σ case. As a result, we see a 20% optimality gap even when $v_{\ell i} \equiv 1$.

Our more novel contributions came in our analysis for CC-TSPT and CC-ATSPT. First, we could not rely on the processing time of the last task for a job to be bounded above by the job's completion time variable C_j in LP0(I'), and so we appealed to a lower bound on C_j that was not stated in the LP itself. The need to incorporate this second bound is critical in realizing the strength of algorithm CC-TSPT, and uncommon in LP rounding schemes. Second, CC-ATSPT is novel in that it introduces constraints that would be redundant for LP0(I) when $I \in \Omega_{PD}$, but become relevant when viewing $LP0(I')$ as a relaxation for $I \in \Omega_{CC}$. This approach has potential for more broad applications since it represented effective use of a limited constraint set supported by a known primal-dual algorithm.

We now take a moment to state some open problems in this area. One topic of ongoing research is developing a factor 2 purely combinatorial algorithm for the special case of concurrent cluster scheduling considered in Theorem 10. In addition, it would be of broad interest to determine the worst-case loss to optimality incurred by assuming single-permutation schedules for $CC|v \equiv 1|\sum w_j C_j$. The simple example above shows that an optimal single-σ schedule can have objective 1.2 times the globally optimal objective. Meanwhile, Theorem 14 shows that there always exists a single-σ schedule with objective no more than 3 times the globally optimal objective. Thus, we know that the worst-case performance ratio is in the interval $[1.2, 3]$, but we do not know its precise value. As a matter outside of scheduling theory, it would be valuable to survey primal-dual algorithms with roots in LP relaxations to determine which have constraint sets that are amenable to implicit modification, as in the fashion of CC-ATSPT.

Acknowledgments. Special thanks to Andreas Schulz for sharing some of his recent work with us [15]. His thorough analysis of a linear program for $P||\sum w_j C_j$ drives the LP-based results in this paper. Thanks also to Chien-Chung Hung and Leana Golubchik for sharing [8] while it was under review, and to Ioana Bercea and Manish Purohit for their insights on SWAG's performance. Lastly, our sincere thanks to William Gasarch for organizing the REU which led to this work, and to the 2015 CAAR-REU cohort for making the experience an unforgettable one; in the words of Rick Sanchez *wubalubadubdub!*

References

1 Inc Amazon Web Services. *AWS Lambda - Serverless Compute*, 2016 (accessed April 3, 2016). URL: `https://aws.amazon.com/lambda/`.

2 Nikhil Bansal and Subhash Khot. Inapproximability of Hypergraph Vertex Cover and Applications to Scheduling Problems. *Automata, Languages and Programming*, 6198:250–261, 2010. URL: `http://link.springer.com/10.1007/978-3-642-14165-2`, `doi:10.1007/978-3-642-14165-2`.

3 Zhi-Long Chen and Nicholas G. Hall. Supply chain scheduling: Assembly systems. Working paper., 2000. `doi:10.1007/978-3-8349-8667-2`.

4 Naveen Garg, Amit Kumar, and Vinayaka Pandit. Order Scheduling Models: Hardness and Algorithms. *FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science*, 4855:96–107, 2007. `doi:10.1007/978-3-540-77050-3\_8`.

5 Teofilo Gonzalez, Oscar Ibarra, and Sartaj Sahni. Bounds for LPT Schedules on Uniform Processors. *SIAM Journal on Computing*, 6(1):155–166, 1977.

6 Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5:287–326, 1979.

7 Mohammad Hajjat, Shankaranarayanan P N, David Maltz, Sanjay Rao, and Kunwadee Sripanidkulchai. Dealer : Application-aware Request Splitting for Interactive Cloud Applications. *CoNEXT 2012*, pages 157–168, 2012.

8 Chien-Chun Hung, Leana Golubchik, and Minlan Yu. Scheduling jobs across geo-distributed datacenters. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, pages 111–124. ACM, 2015.

9 J. Y T Leung, Haibing Li, and Michael Pinedo. Scheduling orders for multiple product types to minimize total weighted completion time. *Discrete Applied Mathematics*, 155(8):945–970, 2007. `doi:10.1016/j.dam.2006.09.012`.

10 Monaldo Mastrolilli, Maurice Queyranne, Andreas S. Schulz, Ola Svensson, and Nelson A. Uhan. Minimizing the sum of weighted completion times in a concurrent open shop. *Operations Research Letters*, 38(5):390–395, 2010. `doi:10.1016/j.orl.2010.04.011`.

11 Microsoft. *Azure Service Fabric*, 2016 (accessed April 3, 2016). URL: `https://azure.microsoft.com/en-us/services/service-fabric/`.

12 Maurice Queyranne. Structure of a simple scheduling polyhedron. *Mathematical Programming*, 58(1-3):263–285, 1993. `doi:10.1007/BF01581271`.

13 Sushant Sachdeva and Rishi Saket. Optimal inapproximability for scheduling problems via structural hardness for hypergraph vertex cover. In *2013 IEEE Conference on Computational Complexity*, pages 219–229. IEEE, 2013.

14 Andreas S. Schulz. Polytopes and scheduling. *PhD Thesis*, 1996.

15 Andreas S Schulz. From linear programming relaxations to approximation algorithms for scheduling problems : A tour d ' horizon. Working paper; available upon request., 2012.

16 C. Sriskandarajah and E. Wagneur. Openshops with jobs overlap. *European Journal of Operations Research*, 71:366–378, 1993.

17 Qiang Zhang, Weiwei Wu, and Minming Li. Resource Scheduling with Supply Constraint and Linear Cost. *COCOA 2012 Conference*, 2012. arXiv:9780201398298, doi:10.1007/3-540-68339-9_34.