Algorithms for Scheduling Data Centers & Assembly Systems

The first provably correct approximation algorithms for multi-machine concurrent open shop environments.

Riley Murray (rjmurray@berkeley.edu) I Megan Chao (megchao@mit.edu) I Samir Khuller (samir@cs.umd.edu)

Motivation

Data is distributed, but practical considerations prevent us from moving everything to one place for computation.

Distributed computing frameworks such as MapReduce provide a means to divide datasets, perform distributed computation, and combine the results into a final product. If a single subset of the data lags behind, the entire query is affected. For this reason, data centers must collaborate to provide fast service.

The associated scheduling problem is NP-hard even if we model each datacenter as a single machine. In this special case, approximation algorithms have existed for several years. But what if we don't want to make this simplification?

1. How do we schedule queries "optimally" when specifically considering how many machines each datacenter has?
2. What if each job had a *collection of parallelizable tasks* at each datacenter, rather than a single task?
3. What if the machines at a given data center vary in speed?

Our goal was to develop provably correct algorithms for more realistic models of datacenter scheduling.

Key Results

Our objective function was to minimize the total weighted completion time of given collection of jobs.

$$\text{minimize} \quad \sum_{j \in N} w_j C_j$$

If all machines within each datacenter are of the same speed, then we demonstrated a 2-approximation LP-based algorithm, and a 3-approximation primal-dual algorithm (additional results for machines that vary in speed are covered in our paper).

If any of the jobs have collections of parallelizable tasks on some datacenter, then both algorithms have a performance guarantee of at least 3. Under certain circumstances, the LP remains a 2-approximation even with parallelizable tasks for each sub-job.

Problem Statement

Note: We phrase the problem in terms of "workcenters" rather than "datacenters" to reflect the problem's generality. The case of one-machine-per-workcenter is known in the literature as "concurrent open shop".

We are given a set of workcenters, and a set of jobs. Each workcenter has some fixed number of machines, and each machine has a given speed. Each job an associated weight, and a set of tasks (a "sub-job") to be completed at each workcenter.

Each task of each sub-job has an associated processing time. Different sub-jobs of the same job, and different tasks of the same sub-job may be processed **concurrently**.

The completion time of a job is defined as the earliest time at which all sub-jobs have completed processing; produce a schedule that minimizes the total weighted completion time of the jobs.

A Toy Example

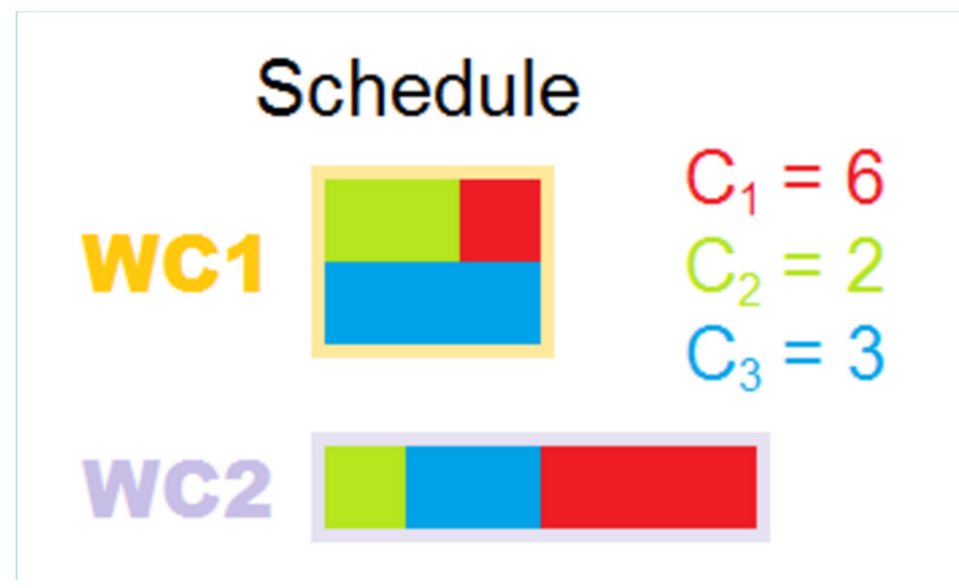
Suppose we are given the following information:

There are 3 jobs, where each sub-job consists of only a single task. Job 1 takes 1 unit of time on workcenter 1, and 3 units of time on workcenter 2. Job 2 takes 2 units on workcenter 1 and 1 unit on workcenter 2. Job 3 takes 3 and 2 units on workcenters 1 and 2 respectively.

Workcenter 1 has 2 machines, and workcenter 2 has 1 machine.

	WC1	WC2	
J1	1	3	P
J2	2	1	
J3	3	2	

WC1	WC2	
2	1	m



One valid schedule is given at left. If all jobs have equal weight, the objective value for this schedule is 11.

An L.P. Approach

We combine and extend two existing LP approaches to scheduling. This constraint was first proven valid by Andreas Schulz [4] for scheduling jobs on a bank of identical parallel machines.

$$\sum_{j \in S} p_j C_j \geq \frac{1}{2}\left[\frac{\left(\sum_{j \in S} p_j\right)^2}{m} + \sum_{j \in S} p_j^2\right] \quad \forall\, S \subseteq N$$

This constraint was used for concurrent open shop scheduling by several researchers over the 2000's

$$\sum_{j \in S} p_{j,i} C_j \geq \frac{1}{2}\left[\left(\sum_{j \in S} p_{j,i}\right)^2 + \sum_{j \in S} p_{j,i}^2\right] \quad \forall\, {S \subseteq N, \atop i \in M}$$

Both of these introduce an exponential number of constraints, but can still be solved in weakly polynomial time with the "Ellipsoid Method" for linear programming (when used with an appropriate separation oracle).

Over the course of the summer, we proved the validity of all the constraints in the following linear program, and developed a separation oracle to solve it.

$$\text{minimize} \quad \sum_{j \in N} w_j C_j \qquad \text{subject to:}$$

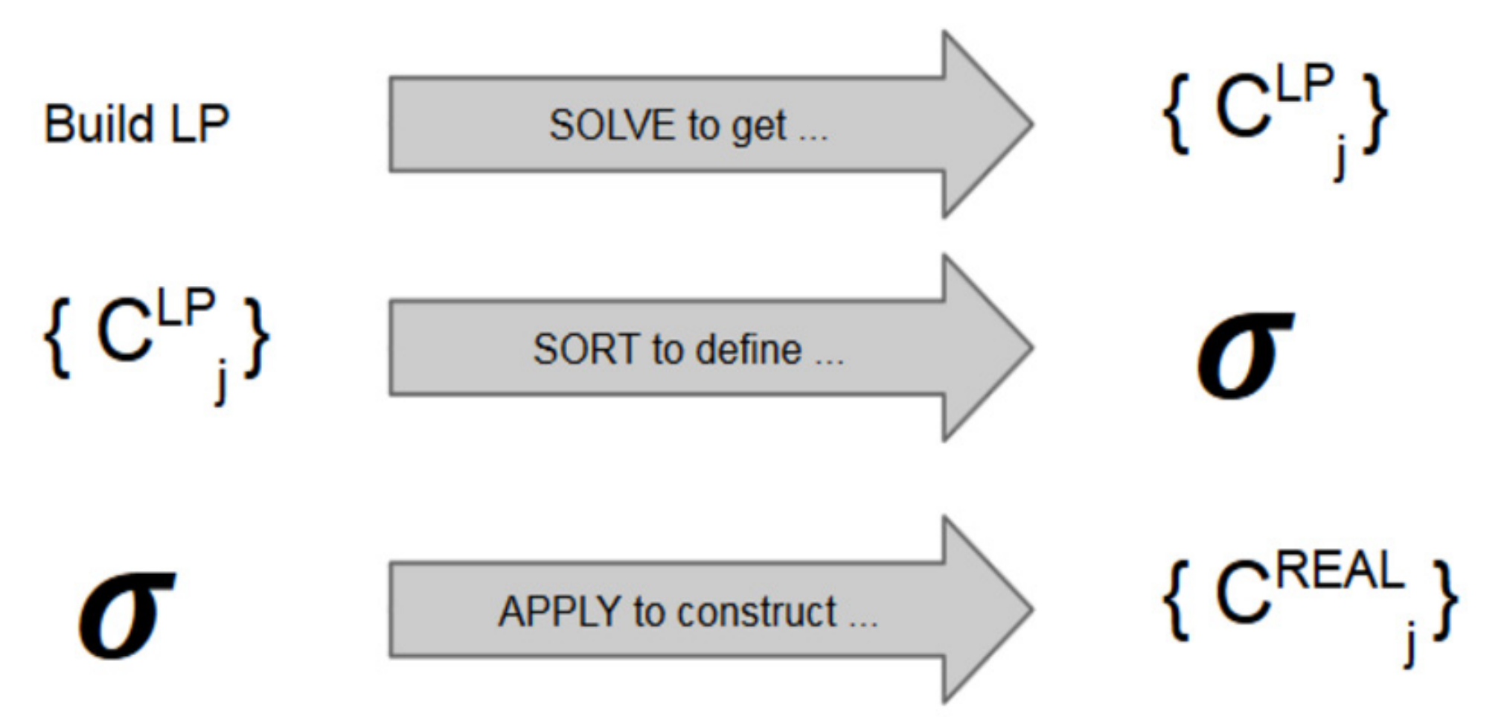
$$(1A) \quad \sum_{j \in S} p_{j,i} C_j \geq \frac{1}{2}\left[\frac{\left(\sum_{j \in S} p_{j,i}\right)^2}{m_i} + \sum_{j \in S} \frac{p_{j,i}^2}{q_{j,i}}\right] \quad \forall\, {S \subseteq N, \atop i \in M}$$

$$(1B) \quad C_j \geq p_{j,i,t} \qquad \forall i \in M,\ j \in N,\ t \in T_{j,i}$$

$$(1C) \quad C_j \geq p_{j,i}/q_{j,i} \qquad \forall j \in N,\ i \in M$$

$$\text{where } q_{j,i} = \min\{|T_{j,i}|, m_i\} \quad \text{and} \quad p_{j,i} = \sum_{t \in T_{j,i}} p_{j,i,t}$$

Given an input, we can solve the LP to get a super-optimal set of completion times $\{C^{LP}_j\}$ for each job. Given these, translate completion times by 1/2 processing time on each workcenter, then sort the result. This defines the order of jobs (sigma) that each workcenter will use in scheduling. Once each workcenter carries out list scheduling, we have a new set of completion times $\{C^{REAL}_j\}$ which defines our near-optimal schedule.

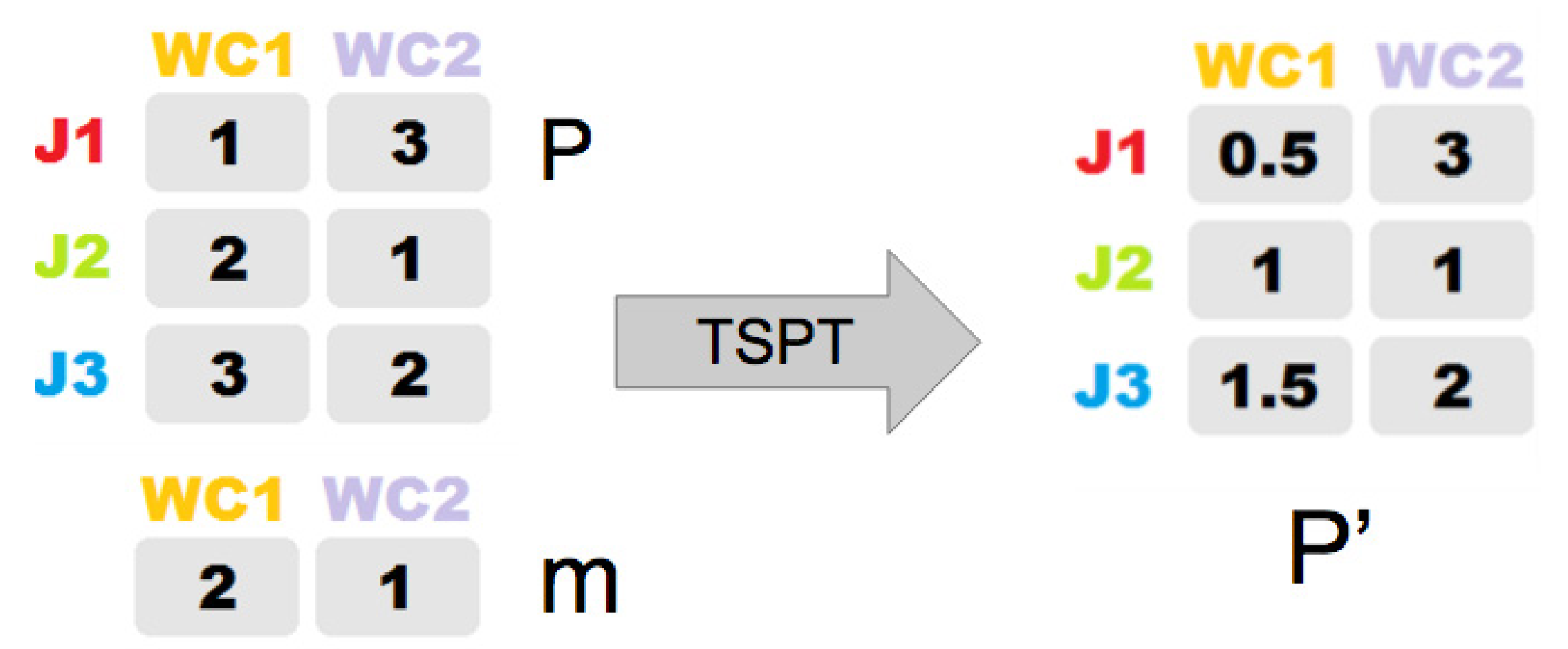


Each completion time in the set $\{C^{REAL}_j\}$ is a constant factor away from the corresponding completion time in $\{C^{LP}_j\}$ (this is the source of the approximation factor for our algorithm). If all machines on each workcenter run at the same speed, this factor is 2, meaning that this algorithm is 2-approximate.

A Primal-Dual Algorithm

A primal-dual algorithm we call "MUSSQ" [8] was developed for concurrent open shop scheduling in 2011. It takes processing times and weights as input, and returns an ordering in which all workcenters will process the jobs. It is much faster than LP approaches, but only supports one machine per workcenter.

We can transform a given multi-machine instance into a single machine instance so that MUSSQ is able to solve it. The process is simple: for each job, divide the *total* processing time that it takes on each workcenter (sum over all tasks) by the number of machines on that workcenter. We call this the **"Total Scaled Processing Time" (TSPT)** transformation.



	WC1	WC2	
J1	1	3	P
J2	2	1	
J3	3	2	

WC1	WC2	
2	1	m

	WC1	WC2
J1	0.5	3
J2	1	1
J3	1.5	2

P'

At first glance the transformation seems ad-hoc, but it's grounded in the following observation- dividing total processing times by the number of machines is equivalent to the following *valid* constraint:

$$\sum_{j \in S} p_{j,i} C_j \geq \frac{1}{2}\left[\frac{\left(\sum_{j \in S} p_{j,i}\right)^2}{m_i} + \sum_{j \in S} \frac{p_{j,i}^2}{m_i}\right] \quad \forall\, {S \subseteq N, \atop i \in M}$$

Nevertheless, TSPT has an issue: It can make very long tasks look very short if the number of machines is large. For instance, a task with duration 1000 on a workcenter with 500 machines seems as though it could complete in 2 units of time. In reality, the job could not complete any earlier than time 1000. To address this, we developed "Augmented Total Scaled Processing Time (ATSPT)."

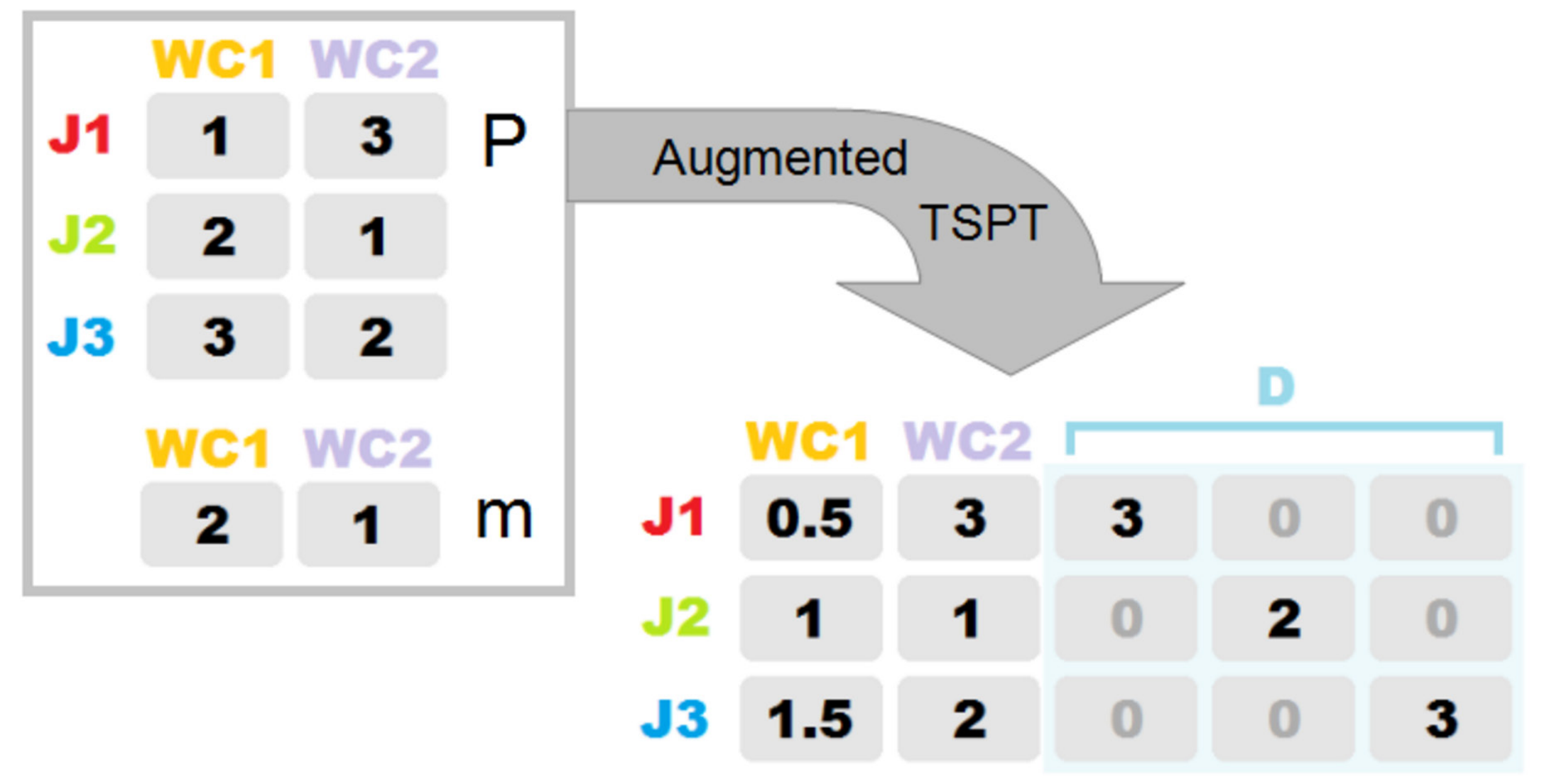
The premise of ATSPT is simple: the completion time of a job is trivially lower bounded by its longest processing time. That is, we would like constraints of the form:

$$C_j \geq p_j^* \quad \text{where} \quad p_j^* \doteq \left(\max_{i,t} p_{j,i,t}\right)$$

Paying special attention to the constraint defined over all subsets of jobs, we see that *if* S *contains a single job,* then it is equivalent to write $C_j \geq p_{j,i} / m_i$. This motivates

$$p_j^* C_j \geq \frac{1}{2}\left[\left(p_j^*\right)^2 + {p_j^*}^2\right] \quad \Leftrightarrow \quad C_j \geq p_j^*$$

So we define a new set of workcenters, where one and only one job has nonzero processing time on each new workcenter. Then we set this processing time equal to p_j^*, and get the desired constraint! This process is illustrated below.



	WC1	WC2	
J1	1	3	P
J2	2	1	
J3	3	2	

WC1	WC2	
2	1	m

	WC1	WC2	D		
J1	0.5	3	3	0	0
J2	1	1	0	2	0
J3	1.5	2	0	0	3

Conclusion

Concurrent open shop scheduling has been a promising model for some instances of distributed computing. Prior to this past summer, any implementations of published approximation algorithms would have relied on an inaccurate abstraction of an entire datacenter as a single resource. By the end of the summer, we had developed provably good algorithms that directly addressed this simplification, and the implications that followed.

Our paper ("Non-Homogeneous Concurrent Open Shop") is under review, and covers a variety of results not discussed in this poster.

Prior Work

[1] Teolo Gonzalez, Oscar Ibarra, and Sartaj Sahni. "Bounds for LPT Schedules on Uniform Processors".
[2] Queyranne. "Structure of a simple scheduling polyhedron"
[3]Sriskandarajah & Wagneur. "Openshops with jobs overlap".
[4] Andreas Schulz. "Polytopes and Scheduling".
[5] Zhi-Long Chen and Nicholas G. Hall. "Supply Chain Scheduling: Assembly Systems". 2000.
[6] Naveen Garg, Amit Kumar, and Vinayaka Pandit. "Order Scheduling Models: Hardness and Algorithms".
[7] Leung, Li, and Pinedo. "Scheduling orders for multiple product types to minimize total weighted completion time".
[8] Monaldo Mastrolilli et al. "Minimizing the sum of weighted completion times in a concurrent open shop".
[9] Andreas Schulz. "From Linear Programming Relaxations to Approximation Algorithms for Scheduling Problems : A Tour D'Horizon". 2012.
[10] Qiang Zhang, Weiwei Wu, and Minming Li. "Resource Scheduling with Supply Constraint and Linear Cost".
[11] Chien-chun Hung, Leana Golubchik, and Minlan Yu. "Scheduling Jobs Across Geo-Distributed Datacenters".