A Competitive Flow Time Algorithm for Heterogeneous Clusters Under Polytope Constraints

Sungjin Im[*1], **Janardhan Kulkarni**[2], **Benjamin Moseley**[†3], **and Kamesh Munagala**[‡4]

1 **EECS, University of California at Merced, Merced, CA, USA**
`sim3@ucmerced.edu`

2 **Microsoft Research, Redmond, WA, USA**
`jakul@microsoft.com`

3 **Department of Computer Science and Engineering, Washington University, St. Louis, MO, USA**
`bmoseley@wustl.edu`

4 **Department of Computer Science, Duke University, Durham, NC, USA**
`kamesh@cs.duke.edu`

Abstract

Modern data centers consist of a large number of heterogeneous resources such as CPU, memory, network bandwidth, etc. The resources are pooled into clusters for various reasons such as scalability, resource consolidation, and privacy. Clusters are often heterogeneous so that they can better serve jobs with different characteristics submitted from clients. Each job benefits differently depending on how much resource is allocated to the job, which in turn translates to how quickly the job gets completed.

In this paper, we formulate this setting, which we term Multi-Cluster Polytope Scheduling (MCPS). In MCPS, a set of n jobs arrive over time to be executed on m *clusters*. Each cluster i is associated with a polytope $\mathcal{P}_i$, which constrains how fast one can process jobs assigned to the cluster. For MCPS, we seek to optimize the popular objective of minimizing average weighted flow time of jobs in the online setting. We give a constant competitive algorithm with small constant resource augmentation for a large class of polytopes, which capture many interesting problems that arise in practice. Further, our algorithm is non-clairvoyant. Our algorithm and analysis combine and generalize techniques developed in the recent results for the classical unrelated machines scheduling and the polytope scheduling problem [10, 12, 11].

1998 ACM Subject Classification F.2.2 [Nonnumerical Algorithms and Problem]: Sequencing and scheduling

Keywords and phrases Polytope constraints, average flow time, multi-clusters, online scheduling, and competitive analysis.

Digital Object Identifier 10.4230/LIPIcs.APPROX-RANDOM.2016.10

[*] S. Im was supported in part by NSF grants CCF-1409130 and CCF-1617653.
[†] B. Moseley was supported in part by NSF grant CCF-1617724, a Yahoo faculty award and a Google faculty award.
[‡] K. Munagala was supported by NSF grants CCF-1408784, IIS- 1447554, and CCF-1348696.

1 Introduction

Modern data centers consist of a large number of machines, each with its own resources such as CPU, memory, network etc, that are organized into hundreds of clusters. Typically, jobs are data intensive and require a lot of resources for running. Examples of such jobs can be found in MapReduce systems. At such large scales, the resources are assumed to be continuously divisible, thus can be compactly represented as a vector consisting of CPU, memory, network bandwidth, etc. Efficiently partitioning these resources among jobs is a major challenge in system design. Complicating the scheduling decisions further is the fact that jobs have different characteristics, and may get different benefits/utilities even when assigned the same resources – some jobs are CPU intensive while others require more memory. In fact, this *multi-dimensional* nature is a key factor that differentiates data center scheduling from other well-studied scheduling settings. Due to the explosive growth of data centers and the associated operating costs, the multi-dimensional scheduling problems have gained a lot of attention in the systems literature recently; see [8] and follow-up work [4, 18, 9, 1, 2, 17, 15].

Various scheduling problems in the presence of multiple resources can be modeled by a polytope which constrains the rate at which jobs can be processed. This abstraction of the data center scheduling problems was introduced under the name POLYTOPE SCHEDULING PROBLEM (PSP) in a recent work by Im et al. [10, 11].

Unfortunately, despite its generality, PSP model fails to accurately capture the system architectures where machines are grouped into clusters. By grouping machines into clusters, the system can restrict the set of jobs that share a given cluster. Moreover, clusters in data centers are often heterogeneous in that some clusters are more suitable for certain types of jobs than others. For example, jobs may get processed more effectively on some clusters due to their proximity to data. Further, the location of data can also change the resource requirements of jobs – if a job is assigned to the cluster on which its data is located then it does not need access to the network when executing. Data centers also have special purpose hardware (such as FPGA/GPUs) for faster execution of certain jobs. Thus, the processing times of jobs can also depend on the clusters that they are assigned to.

In this paper, we introduce a more realistic scheduling setting where each cluster is associated with a distinct polytope that determines the rates at which jobs get processed. At a high level, this not only captures the *multi-dimensional* nature found in PSP, but also captures the *unrelated* aspect reminiscent of the classical scheduling literature [3]. In the classical unrelated machines model, each job can get processed at a completely different rate depending on its machine assignment. Thus, our model lifts the unrelated nature from machines to clusters while staying faithful to the multi-dimensional aspect of data center scheduling problems.

We focus on minimizing the average (weighted) flow time of jobs. A job's flow time measures how long the job waits from its arrival until its completion, thus the average (or equivalently total) flow time measures the average delay experienced by clients. For this popular objective, competitive algorithms are known for some special cases of PSP and also for the unrelated machines setting. However, their analyses use two very different methods – dual fitting for the unrelated machines setting [12], and potential function argument for PSP [11]. In this paper, we combine the two different algorithms used in [12, 11] and develop a new potential function analysis that *unifies* the two disparate analyses from the previous works.

1.1 Problem Definition

The Multi-Cluster Polytope Scheduling (MCPS), which is a generalization of the Polytope Scheduling Problem (PSP) [10], is defined as follows. A set of n jobs arrive over time. Each job j has a weight w_j, size (or processing length) p_j, and arrival time r_j, and needs to be processed using a set of m clusters. Each cluster i is associated with a convex polytope $\mathcal{P}_i$ that constrains the feasible space of rates of jobs assigned to the cluster. Each polytope $\mathcal{P}_i$ is defined over the entire set of jobs and is assumed to be downward-closed, meaning that if $\vec{y} \in \mathcal{P}_i$ and $\vec{z} \leq \vec{y}$, then any $\vec{z} \in \mathcal{P}_i$. The scheduler has to assign rates to jobs, $\{y_j\}_j$, subject to the polytope constraints and a natural requirement that a job can be scheduled only on a single cluster at any given instant of time. Applications of MCPS will be discussed in Section 1.2.

Note that we allow a job to be processed on more than one cluster over the course of its execution. However, at any given time a job can be processed on only one cluster. This is exactly the PSP problem [10] when there is only one cluster.

In this paper we seek to design *online* scheduling algorithms for MCPS, which have to make scheduling decisions only based on the jobs that have arrived. In other words, the online scheduler learns about a job j along with its properties when it arrives. Our goal is to minimize the total weighted flow time of jobs in the setting of MCPS. Let $y_{jt}^{\mathcal{A}}$ denote the rate at which job j is processed at time t by a scheduler $\mathcal{A}$. Then, job j's completion time $C_j^{\mathcal{A}}$ under the schedule of $\mathcal{A}$ is defined to be the first time t' such that $\int_{t=r_j}^{t'} y_{jt}^{\mathcal{A}} \mathsf{d}t \geq p_j$. Job j's flow time is the length of time job j waits to be completed since its arrival and is defined as $F_j^{\mathcal{A}} = C_j^{\mathcal{A}} - r_j$. Similarly, job j's weighted flow time is defined as $w_j F_j^{\mathcal{A}}$ factoring in the job's weight. When the algorithm $\mathcal{A}$ and time t are clear from the context, we may drop them from the notation. The goal is to minimize $\sum_j w_j F_j^{\mathcal{A}}$.

We will use the standard notion of *competitive ratio* for analyzing our algorithms. An online algorithm is α-competitive if for every finite input instance, the cost incurred by the algorithm is at most α times the cost of some optimal offline solution to the instance. Unfortunately, the standard competitive analysis turns out to be too pessimistic in analyzing flow time related objectives: there are no online algorithms with bounded competitive ratios even in much simpler single dimensional settings [7]. We therefore appeal to the standard speed augmentation analysis [14], where we assume the online algorithm can perform $c > 1$ allocations per time step, while OPT is restricted to allocate at the rate of 1. Our goal is to design algorithms that achieve constant competitive ratios on the total flow time objective using the smallest possible extra speed c.

1.2 Our Results

Our main result is a non-trivial generalization of the result shown for 'monotone' PSP in [11] to the multi-cluster setting.

Proportional Fairness and Monotone Polytope.

The proportional fairness (PF) algorithm, at each instant of time t, assigns rates $\{y_{jt}\}_j$ to jobs by solving the following convex program over the polytope $\mathcal{P}$.

$$\max \sum_j w_j \log y_{jt} \qquad \text{s.t.} \qquad \vec{y}_t \in \mathcal{P} \tag{1}$$

Note that $\mathcal{P}$ is in the definition of PSP. The PF algorithm generalizes the weighted round robin (WRR) algorithm to the multidimensional case. The study of PF even dates back to

Nash's seminal work [16]. The algorithm PF has a very nice market clearing interpretation – It finds prices for resources so that every resource (with a positive price) is completely sold out when each player (job) j with money w_j buys resources to maximize its utility under the prices. Further, the algorithm PF is known to have many desirable fairness properties such as sharing incentiveness, and envy-freeness [8].

▶ **Definition 1** **([11]).** For a subset of jobs S, let $y_j(S)$ denote the rate allocated by PF to job $j \in S$, as given by the equation (1). The PF allocation is said to be *monotone* if for any S and $j' \notin S$, we have the following condition: For all $j \in S$, $y_j(S) \geq y_j(S \cup \{j'\})$. The class Monotone PSP is the sub-class of PSP for which the PF algorithm leads to monotone allocation.

We call MCPS as Monotone-MCPS when the PF algorithm is monotone for every polytope $\mathcal{P}_i$. When there is only one cluster, Im et al. [11] showed that the PF algorithm is $(e+\epsilon)$-speed $O(1/\epsilon^2)$-competitive for the Monotone PSP case. Our first main result is a generalization of this result to the case with an arbitrary number of clusters.

▶ **Theorem 2.** *For Monotone-*MCPS*, there is a $(e+\epsilon)$-speed, $O\left(1/\epsilon^2\right)$-competitive algorithm for minimizing the total weighted flow-time of jobs. Further, our algorithm is non-clairvoyant as it does not make use of the processing lengths of jobs.*

Monotone-PSP captures many important problems such as flow routing to a single sink, routing multicast trees (video-on-demand) etc. We refer the readers to [11] for more details on applications of Monotone-PSP. For completeness, here we give one important class of problems captured by Monotone-PSP that is very relevant to data center scheduling.

Resource Allocation with Substitutes [11]. Consider the multi-dimensional resource allocation problem that arises in scheduling jobs within a cluster. Formally, there are D divisible resources (or dimensions), numbered $1, 2, \dots, D$. By scaling we can assume w.l.o.g. that each resource is available in a unit supply. If job j is assigned a non-negative vector of resources $\vec{x} = \{x_1, x_2, \dots, x_D\}$, then the rate at which the job executes is determined by $y_j = u_j(\vec{x})$, where u_j is a concave *utility function* that is known to the scheduler. The constraints $\mathcal{P}$ simply capture that each resource can be allocated to unit amount, i.e. $\sum_j x_{jd} \leq 1$ for all $d \in \{1, 2, \dots, D\}$. A well-studied special class of utilities in the resource allocation literature are the Constant Elasticity of Scale (CES) utilities, given by:

$$u_j(\vec{x}_j) = \left(\sum_{d=1}^{D} c_{jd} x_{jd}^{\rho_j}\right)^{1/\rho_j}.$$

When $\rho \in (0, 1]$ the utility function captures resources that are *imperfect substitutes* of each other, and the parameter ρ captures the extent of substitutability. A special case as $\rho \to 0$ is termed *Cobb-Douglas* utilities: $u_j(\vec{x}_j) = \prod_{d=1}^{D} x_{jd}^{\alpha_{jd}}$, where $\sum_d \alpha_{jd} \leq 1$ and $\alpha_{jd} \geq 0$ for all j, d. These utilities can be used to model task rates in heterogeneous microprocessor architectures [19]; further, these are widely studied in economics. When $\rho = 1$, CES utilities reduce to *linear utilities.*

It was shown that prove that CES is a special case of Monotone PSP in [11]. Thus, our result immediately gives a competitive algorithm for these problems in the multiple cluster setting.

1.3 High-level Description of the Algorithm

Our algorithm for the Monotone-MCPS consists of two parts: Within a cluster, jobs are assigned rates using the PF algorithm. To decide the assignment of jobs to clusters, we use the Selfish Migrate framework introduced in [12]. In the Selfish Migrate framework, at every time instant, a job behaves like a selfish agent and moves to the cluster that maximizes its own virtual utility function. More precisely, a virtual ordering is maintained among jobs assigned to each cluster i, and a job j's virtual utility is calculated as the speed the job would get on the cluster i under the PF algorithm as if other jobs behind in the ordering were not present. If the job moves to another cluster i', it is placed the last in the virtual ordering of cluster i'. These selfish moves lead to a Nash equilibrium where each job has the best virtual utility on the cluster it is currently assigned to. The monotonicity property of polytopes is crucially used in establishing the existence of such an equilibrium. Thus our algorithm is a *mixture of two interesting equilibria* – a Nash equilibrium across machines under virtual utilities and a market clearing equilibrium on each individual cluster under the algorithm PF. Note that our algorithm does not use job sizes in scheduling decisions, thus is non-clairvoyant.

1.4 Justification for not Encapsulating Clusters into One Polytope

We take a sidestep to clarify some questions that may arise from our definitions of PSP and MCPS. It is fair to ask why one needs to define MCPS when PSP is general enough to model the multiple cluster setting. More precisely, one can define a giant polytope $\mathcal{P}$ that is the union of all polytope constraints $\mathcal{P}_i$, with an extra constraint that at any give time no job can be processed on more than one cluster. Indeed, such a formulation captures MCPS. However, this way of looking at the problem leads to a major technical difficulty: *The giant polytope $\mathcal{P}$ may not inherit the properties satisfied by the individual polytopes $\mathcal{P}_i$.* In particular, the new polytope may not be monotone even if all individual polytopes $\mathcal{P}_i$ are monotone, meaning that we no longer have nice properties that lead to constant-speed constant competitiveness of the PF algorithm. To see this, consider the classical unrelated machines setting. Although it is a special case of PSP, thus our problem, it is not clear if it is a special case of Monotone-PSP. No known techniques can be directly applicable to prove unrelated machines fall into a category of Monotone-PSP. In fact, we conjecture that it is not.

1.5 Our Techniques

Our problem MCPS extends the unrelated machines scheduling and multidimensional scheduling in a natural way. Expectedly, our algorithm for the problem uses a combination of the algorithms developed for the unrelated machines scheduling and the PSP setting. More precisely, within a cluster, our algorithm allocates rates using the Proportional Fairness algorithm similar to the Monotone-PSP case [11] while assigning jobs to clusters using the Selfish Migrate framework as done in the unrelated machine setting [12]. The main technical contribution of this paper lies in unifying the two different analyses of these algorithms – the former uses a potential function argument and the latter a dual-fitting argument.

To unify the two different analyses, one could attempt to use potential function or dual fitting. If one wants to try a dual fitting argument for our problem, the first thing to do would be proving competitiveness of the Monotone-PSP using dual fitting. There are two math programmings involved here: the convex programming (CP) we solve at any instantaneous moment to implement the algorithm PF, and the linear programming (LP) we use to establish the competitiveness of PF for the total weighted flow time objective. One could try to use the values of CP dual variables derived from the KKT conditions of the CP to set the LP dual variables. However, as discussed in [11], the CP dual variables can have highly unstructured values even for monotone-PSP, and this is why [11] used only a CP optimality condition repeatedly, without looking at the dual.

Hence we use a different route by giving an alternative potential function based analysis of the Selfish Migration rule for unrelated machines, and generalizing it to our problem, Monotone-MCPS. Surprisingly, we use an unexpectedly simple potential function, which is just the sum of potential functions for each cluster. The potential for each cluster is defined over the sets of jobs assigned to the cluster by our algorithm and the optimal solution, and there are no terms in the potential connecting different clusters. This is surprising since it has been believed that more sophisticated potential functions should be needed to factor in the changes of the projected objective based on the current assignment of jobs that occur when jobs migrate across different machines.

The reader familiar with potential functions may wonder how we can bound the change of the potential since the jobs the optimal solution assigns to each machine can be vastly different from those our algorithm does. Hence, the credits we get from our algorithm's processing might be completely offset by the debits due to the optimal processing. This is where we use the Selfish Migration rule crucially. At high level, using the fact that each job currently resides on the best machine maximizing its virtual utility, we can safely assume that jobs are assigned following the optimal scheduler since it only gives less credits, which effectively reduces the analysis to each individual machine. However, the extension of this analysis from unrelated machines to Monotone-PSP has another issue. In this thought process of pretending that jobs are assigned to clusters following the optimal solution, we face the challenge of measuring how fast jobs get processed within a cluster under the PF where some of them come from our algorithm's assignment and the other from the optimal solution. We bound such rates using the polytope monotonicity (Proposition 3) and a CP optimality condition (Proposition 4). See Section 2.2 for formal statement of the two properties and how they are used in our analysis.

To summarize, our algorithm, which is a mixture of two equilibria from the Proportional Fairness algorithm and the Selfish Migration rule, is analyzed delicately using the two respective monotonicity properties resulting from the two algorithms: (i) jobs get lower processing rates within a cluster when competing with more jobs; and (ii) each job's migration only increases its virtual utility without hurting other jobs.

Finally, as a byproduct we obtain an alternate analysis of the unrelated machine scheduling to minimize the weighted flow-time in the non-clairvoyant setting. As mentioned before, the classical unrelated machine setting is a special case of the Monotone-MCPS, where machines correspond to clusters, and the polytope constraints simply enforce that only one unit of CPU is allocated at any given time instant. In this single dimensional setting, PF is same as Weighted Round Robin. Using these facts, we obtain the alternate analysis as a corollary of Theorem 2. The sketch of this analysis can be found in Section 3. The original analysis in [12] relied on a dual-fitting argument.

2 Monotone Multi-cluster Polytope Scheduling

In this section we prove Theorem 2. First, we set up some notation. Recall that at every time instant, each alive job is assigned to exactly one cluster. Let A_{it} denote the set of alive jobs at time t that are assigned to cluster i in our algorithm's schedule. We often drop the subscripts t or i when it is clear from the context. Similarly, define A_t to be the set of all jobs alive at time t; that is, $A_t := \bigcup_i A_{it} = \{j \mid t \in [r_j, C_j]\}$. Let $\vec{y}_t$ denote the vector of processing rates of jobs. We use y_{jt} to denote the processing rate job j gets at time t. For any subset of jobs S, define $\vec{y}_t(S)$ as the projection of $\vec{y}$ into S; so, $y_{jt}(S) = y_{jt}$ if $j \in S$, and 0 otherwise.

2.1 Algorithm

Our algorithm for Monotone-MCPS consists of two components: Rate allocation using Proportional Fairness (PF) and job assignment using Selfish Migrate [12]

1. Proportional Fairness (PF): Each cluster i assigns rates to the set of jobs assigned to i that are alive at time t using the Proportional Fairness (PF) algorithm. The PF allocation can be obtained by solving the following convex program.

$$\text{Maximize} \quad \sum_{j \in A_{it}} w_j \log y_{jt} \qquad \text{s.t.} \qquad \vec{y}_t(A_{it}) \in \mathcal{P}_i \,.$$

   The rates assigned to jobs remain unchanged unless a new job arrives to the cluster or a job departs.

2. Selfish Job Migration: This rule is applied only when a job completes or arrives – at other times, no job changes its assignment. Selfish Migrate algorithm is best viewed as a game where each job tries to maximize its own utility (defined later). A key property of our assignment policy is that at each time instant, a job is assigned to the cluster that maximizes its utility. In other words, jobs are in Nash equilibrium with respect to their utility functions.

   To define the utility of a job, we need the notion of virtual queues. Each cluster has a complete *virtual* ordering of jobs assigned to the cluster, and the utility of a job depends on its position in this virtual ordering. We emphasize that this ordering is used only for the job assignment, and the PF algorithm itself is oblivious to this ordering. If job j is ahead of j' on cluster i, we denote it as $j \leq_i j'$. For a subset of jobs S and a job $j \in S$, let $y^*_{ij}(S)$ denote the rate PF assigns to job j if the set of jobs assigned to cluster i is S. Suppose a job j is on cluster i at time t. Then its utility on i is defined as $y^*_{ij}(A^{\leq j}_{it})$. The job's utility on any other cluster i' is defined as $y^*_{i'j}(A_{i't} \cup j)$. Now we describe how the virtual ordering of jobs are built on each cluster.

   - When a job j arrives at time t, it is assigned to the cluster i for which $y^*_{ij}(A_{it} \cup \{j\})$ is maximized. Further, the newly arrived job goes to the *tail of the virtual ordering.*
   - When a job completes, it simply disappears without affecting the relative ordering of the other jobs. However, this may start a chain of jobs migrations, as jobs may increase their utilities by switching to the cluster from which the job departed. Fix a job j. If job j is currently residing on cluster i, its utility is $y^*_{ij}(A^{\leq j}_{it})$. Here, $A^{\leq j}_{it}$ refers to all the jobs in A_{it} ahead of j in the virtual ordering including j itself. If job j moves to another cluster $i' \neq i$, the job is placed *behind* all jobs in $A_{i't}$ in the virtual ordering on i'. Hence, its utility will be $y^*_{i'j}(A_{i't} \cup j)$. A job j is free to move to any cluster i' as long as it's utility improves. If two jobs try to move simultaneously, then we break ties arbitrarily. This process is repeated until no jobs can improve their utilities. A priori, it is not clear if this process will terminate. For now we assume that it terminates, and in Section 2.4 we show that each jobs migrates at most $O(\frac{n}{\epsilon} \log n)$ times in total.

   This completes the description of our algorithm.

2.2 Key Properties Used in the Analysis

In this section, we summarize two key properties we will crucially use in our analysis. Recall that we assume in our problem Monotone-MCPS that all polytopes $\mathcal{P}_i$ are monotone. The following proposition is a restatement of Definition 1.

▶ **Proposition 3** (Polytope Monotonicity [11]). *Let $y_j^*(S)$ denote job j's processing rate under the PF algorithm for an arbitrary fixed monotone polytope $\mathcal{P}$. Then, for all $j \in S$ and $j' \notin S$, we have $y_j^*(S) \geq y_j^*(S \cup \{j'\})$.*

The next proposition, which we call the optimality condition, immediately follows from the convexity of the polytope and the PF algorithm's objective. We include the proof for completeness.

▶ **Proposition 4** (Optimality Condition [11]). *Let $\vec{y} \in \mathcal{P}$ denote any feasible rate vector for the jobs in S. If the space of feasible rates $\mathcal{P}$ is convex, then*

$$\sum_{j \in S} w_j \frac{y_j}{y_j^*(S)} \leq \sum_{j \in S} w_j \, .$$

Proof. For notational simplicity, let $y_j^* := y_j^*(S)$. Let $f(\vec{y}) = \sum_{j \in S} w_j \log y_j$. We have $\frac{\partial f(\vec{y}^*)}{\partial y_j} = \frac{w_j}{y_j^*}$. The optimality of $\vec{y^*}$ implies $\nabla f(\vec{y^*}) \cdot (\vec{y} - \vec{y^*}) \leq 0$ for all $\vec{y} \in \mathcal{P}$. The proposition now follows by elementary algebra. ◀

These two conditions will be repeatedly used in our analysis. As mentioned earlier, our analysis is based on a potential function which depends on jobs arrival, and processing of our algorithm and the optimal scheduler. The potential changes will be categorized into two: discontinuous changes and continuous changes. As we will discuss soon in detail, discontinuous changes occur when jobs arrive or complete, and all other changes are continuous. Our analysis differs from the previous work in bounding discontinuous changes, and continuous changes due to the optimal scheduler's processing. In particular, the latter, which is formalized in Lemma 6, is the most interesting part in our analysis.

Before we move to the detailed analysis, we discuss at high level how we bound the continuous changes of the potential, particularly Lemma 6. As mentioned, the potential adds up the potential defined over each cluster, which only depends on the jobs assigned to the cluster by the algorithm and the optimal scheduler, which we denote A_i and O_i, respectively. If $A_i = O_i$ for all i, then the analysis is essentially equivalent to that of the single cluster case, which was done in [11]. Otherwise, using the greedy nature of the Selfish Migration rule and the Polytope monotonicity, we can w.l.o.g. proceed assuming that all jobs in O_i are also added to A_i for each cluster i. As a result, we are left with the task of upper bounding $\sum_{j \in O_i} w_j \cdot \frac{y_j^O}{y_j^*(A_i \cup O_i)}$; see Eqn. (4). At first sight, it is not clear how to bound this. The numerators are the processing rates of jobs in O_i due to the adversary, and the denominators are those under PF with extra jobs A_i added. This is where we apply the optimality condition, Proposition 4 assuming that PF is run on the jobs $A_i \cup O_i$ and setting y_j following the adversary for jobs $j \in O_i$; $y_j = 0$ for other jobs. Thus, the polytope monotonicity and the optimality condition are nicely combined to prove the key lemma.

2.3 Competitive Analysis: Proof of Theorem 2

We use amortized local competitiveness to prove the theorem. Our potential function $\Phi(t)$ is inspired by [6, 11]. The potential function adds up potential functions defined for individual clusters that are essentially identical to the potential in [11]. Define O_t and O_{it} for the optimal scheduler analogously as we did for A_t and A_{it} for our algorithm. For a set of jobs S, let $W(S)$ denote the total weight of jobs in the set. Assuming that our algorithm is given $(e + \epsilon)$-speed, we show that the following conditions are satisfied that will imply Theorem 2. These are standard conditions which are verified for most potential functions. See [13] for a tutorial on the framework.

1. (Boundary condition) $\Phi(0) = \Phi(\infty) = 0$;
2. (Discontinuous changes) Φ can only decrease when a job arrives into or departs from the system; and
3. (Continuous changes) At the other times t, $W(A_t) + \frac{d}{dt}\Phi(t) \le \frac{3}{\epsilon^2} W(O_t)$.

It is an easy exercise to verify that conditions are sufficient to establish our algorithm's competitiveness by integrating the last inequality [13]. We give a brief explanation. Suppose all jobs are completed by our algorithm and OPT by time T. The first two conditions imply that $\int_{t=0}^{T} \frac{d}{dt}\Phi(t)dt \ge 0$. Then integrating the above inequality over time, we have:

$$\int_{t=0}^{T} W(A_t)dt + \int_{t=0}^{T} \frac{d}{dt}\Phi(t)dt \le \frac{3}{\epsilon^2} \int_{t=0}^{T} W(O_t)dt\,.$$

This implies Theorem 2 since the first term above is the weighted flow time of our algorithm, and the RHS is that of OPT.

To define the potential function formally, we need to set up more notation. Fix a time instant t. For job j, let p_{jt} denote the remaining size of the job in the PF's schedule, and let p^O_{jt} denote the remaining size of the job in OPT's schedule. Define a job j's *lag* as $\tilde{p}_{jt} = \max(0, p_{jt} - p^O_{jt})$. The quantity $\tilde{p}_{jt}$ indicates how much our algorithm is behind the optimal schedule in terms of job j's processing. Let $L_t = \{j \in A_t \mid \tilde{p}_{jt} > 0\}$. Note that $A_t \setminus L_t \subseteq O_t$. Recall that $y^*_j(S)$ denote the optimal rate the PF algorithm allocates to job $j \in S$ when working on the set S. We define the following potential function:

$$\Phi(t) := \sum_i \Phi_i(t) \tag{2}$$

$$\text{where } \Phi_i(t) := \frac{1}{\epsilon} \sum_{j \in A_{it}} w_j \frac{\tilde{p}_{jt}}{y^*_{ij}(A^{\le j}_{it})} \tag{3}$$

It now remains to verify all the above conditions(1-3) are satisfied. The boundary condition trivially holds since at times $t = 0$ and $t = \infty$, the algorithm has no alive jobs. In the following sections, we show the last two conditions hold true.

2.3.1 Discontinuous Changes

First we show that $\Phi(t)$ can only decrease when a job arrives or completes in our algorithm's schedule.

▶ **Lemma 5.** *The discontinuous changes in the potential function (2) due to a job arrival or departure is at most zero.*

Proof. Suppose a job j arrives at time t; for notational convenience, we assume that $j \notin A_t$. For the job j, $\tilde{p}_{jt} = 0$. Suppose j is assigned to cluster i. Since job j is behind any other jobs on the cluster in the virtual ordering of jobs, no existing terms in Φ_i change. A new term, $\left(w_j \cdot \frac{\tilde{p}_{jt}}{y^*_j(A_{it} \cup \{j\})}\right)$ is added for the job j, and the value of this term is 0 since $\tilde{p}_{jt} = 0$. Therefore, the lemma is true for job arrivals.

We now focus on job completions. It is easy to see that the optimal scheduler completing a job does not lead to any discontinuous changes in the potential function. Hence, we only consider changes in the potential due to our algorithm completing a job. Suppose a job j completes. If j was on cluster i just before it completed, the term $(w_j \cdot \frac{\tilde{p}_{jt}}{y_j^*(A_{it}^{\leq j})})$ drops from $\Phi_i(t)$ as $\tilde{p}_{jt}$ becomes zero.

Due to this the value of other terms for jobs k such that $j \leq_i k$ (that is, jobs that were behind the job j in the virtual order) may change from $(w_k \cdot \frac{\tilde{p}_{kt}}{y_j^*(A_{it}^{\leq k})})$ to $(w_k \cdot \frac{\tilde{p}_{kt}}{y_j^*(A_{it}^{\leq k} \setminus \{j\})})$. Such changes are non-positive due to the monotonicity of $\mathcal{P}_i$.

But completion of a job may result in a sequence of jobs migrations as other jobs may get a higher utility on the cluster that a job departed from. We show that the potential can only decrease when jobs migrate. Say a job j migrates from cluster i to cluster i'. Note that the term $(w_j \cdot \frac{\tilde{p}_{jt}}{y_j^*(A_{it}^{\leq j})})$ drops from $\Phi_i(t)$, and a new term $(w_j \cdot \frac{\tilde{p}_{jt}}{y_j^*(A_{i't} \cup \{j\})})$ is added to $\Phi_{i'}(t)$.

When a job migrates from cluster i to i', the value of other terms for jobs k such that $j \leq_i k$ change from $(w_k \cdot \frac{\tilde{p}_{kt}}{y_j^*(A_{it}^{\leq k})})$ to $(w_k \cdot \frac{\tilde{p}_{kt}}{y_j^*(A_{it}^{\leq k} \setminus \{j\})})$. This change in the value is non-positive due to the monotonicity of $\mathcal{P}_i$. Therefore, we have

$$\Delta\Phi_i(t) \leq -\frac{1}{\epsilon} \cdot w_j \cdot \frac{\tilde{p}_{jt}}{y_j^*(A_{it}^{\leq j})}.$$

Since j moves to cluster i', $(w_j \cdot \frac{\tilde{p}_{jt}}{y_j^*(A_{i't} \cup \{j\})})$ is added to $\Phi_{i'}(t)$. However, no terms in the summation of $\Phi_{i'}$ change since j is placed at the end in the ordering of jobs on i'. Hence we have

$$\Delta\Phi_{i'}(t) = \frac{1}{\epsilon} \cdot w_j \cdot \frac{\tilde{p}_{jt}}{y_j^*(A_{i't} \cup \{j\})}.$$

Since $\Phi(t)$ does not change on other cluster, we have $\Delta\Phi(t) = \Delta\Phi_i(t) + \Delta\Phi_{i'}(t) \leq 0$, as desired. The inequality follows from the fact that job j having moved to i' means that $y_j^*(A_{it}^{\leq j}) \leq y_j^*(A_{i't} \cup \{j\})$. ◀

2.3.2 Continuous Changes

Fix a time instant t when no jobs arrive or depart. To simplify notation, we omit the subscript t from rest of the proof. Let $\Phi'|_O$ and $\Phi'|_A$ denote the potential changes due to OPT's processing and our algorithm's processing respectively. We note that $\frac{d}{dt}\Phi(t) = \Phi'|_A + \Phi'|_O$.

▶ **Lemma 6.** $\Phi'|_O \leq \frac{1}{\epsilon}(W(A) + W(O))$.

Proof. Fix a cluster i and consider each job $j \in O_i$. We consider two cases. Let y_j^O denote the rate the optimal scheduler processes job j. Consider the case when j is also assigned to i by our algorithm at time t; that is, $j \in A_i$. Then, $\Phi'|_O$ due to job j is

$$w_j \cdot \frac{y_j^O}{y_j^*(A_i^{\leq j})} \leq w_j \cdot \frac{y_j^O}{y_j^*(A_i \cup O_i)},$$

where the inequality follows due to PF being monotone for $\mathcal{P}_i$.

Next, we consider the other case where the algorithm processes job j on a different cluster i' from i. The reason the job j decided to stay on cluster i' instead of moving to cluster i is because it can't get a better utility when it is added to the end of the ordering of jobs on cluster i, i.e. $y_j^*(A_{i'}^{\leq j}) \geq y_j^*(A_i \cup \{j\}) \geq y_j^*(A_i \cup O_i)$. The last inequality is due to the monotonicity of $\mathcal{P}_i$ and the fact that $j \in O_i$. Hence $\Phi'|_O$ due to job j is

$$\Phi'|_O \leq w_j \cdot \frac{y_j^O}{y_j^*(A_{i'}^{\leq j})} \leq w_j \cdot \frac{y_j^O}{y_j^*(A_i \cup O_i)}.$$

Summing over all jobs O_i on each cluster i, we have

$$\Phi'|_O \leq \sum_i \sum_{j \in O_i} w_j \cdot \frac{y_j^O}{y_j^*(A_i \cup O_i)}. \tag{4}$$

Finally, we show

$$\sum_{j \in O_i} w_j \cdot \frac{y_j^O}{y_j^*(A_i \cup O_i)} \leq W(A_i) + W(O_i). \tag{5}$$

The above two equations 4 and 5 will yield $\Phi'|_O \leq \sum_i W(A_i) + W(O_i)$, proving the lemma.

To show equation (5), we appeal to the optimality condition stated in Proposition 4. Say we process jobs $A_i \cup O_i$ on cluster i using PF. Then each job j gets processed at a rate of $y_j^*(A_i \cup O_i)$. Then, set $y_j = y_j^O$ for all $j \in O_i$ and $y_j = 0$ for all other jobs. Note that this setting of $\{y_j\}$ is a feasible allocation of rates to jobs $A_i \cup O_i$. Hence, equation (5) follows from Proposition 4. ◀

We now bound Φ's continuous changes due to our algorithm's processing. Recall that $L = A \setminus O$ denote the set of jobs that the optimal scheduler has finished but are alive in PF schedule. Similarly, let $L_i = A_i \setminus O$.

We consider two cases.

Case 1: $\boldsymbol{W(L) \leq (1-\epsilon)W(A)}$. Since $A \setminus L \subseteq O$, we have $W(O) \geq \epsilon W(A)$. Since $\Phi'|_A \leq 0$, we have:

$$W(A) + \Phi' \leq W(A) + \Phi'|_O \leq \frac{2}{\epsilon}(W(A) + W(O)) \leq \frac{3}{\epsilon^2} W(O)$$

where the second inequality follows from Lemma 6.

Case 2: $\boldsymbol{W(L) \geq (1-\epsilon)W(A)}$. This is a more interesting case. If job j is on cluster i , then PF processes job j at a rate of $y_j^*(A_i)$. For every job $j \in L$, PF decreases $\tilde{p}_{jt}$ at the rate of $y_j^*(A_i)$. For all other jobs, PF can only decrease the potential. Hence we have,

$$\Phi'|_A \leq -\frac{1}{\epsilon} \sum_i \sum_{j \in L_i} w_j \frac{y_j^*(A_i)}{y_j^*(A_i^{\leq j})} \tag{6}$$

To bound this quantity, we use the following inequality which was shown in [11] using the polytope monotonicity and the optimality condition. For the sake of completeness, we repeat the proof in [11].

▶ **Lemma 7** ([11]). *Let S be an arbitrary ordered set of jobs. Let $S^{\leq j}$ denote jobs ahead of j, including j. For any subset $S' \subseteq S$, we have,*

$$\sum_{j \in S'} w_j \cdot \frac{y_j^*(S)}{y_j^*(S^{\leq j})} \geq W(S') \cdot \exp\left(-\frac{W(S)}{W(S')}\right).$$

Proof. For notational convenience, let $|S| = \kappa$, and number the jobs in S in increasing order of arrival time as $1, 2, \ldots, \kappa$. For $k > j$ and $k \le \kappa$, let $\alpha_{jk} = \frac{y_j^*(S^{\le k-1})}{y_j^*(S^{\le k})}$. By the monotonicity of PF, we have $\alpha_{jk} \ge 1$. Define $\delta_{jk} = \alpha_{jk} - 1$. Note that $\delta_{jk} \ge 0$.

We now apply Proposition 4 to the set $\{1, 2, \ldots, k\}$ as follows: For jobs $j \in \{1, 2, \ldots, k\}$, the rate assigned by PF when executed on this set is $y_j^*(S^{\le k})$, and this goes into the denominator in Proposition 4. We consider $y_j^*(S^{\le k-1})$ for $j < k$, and $y_k^*(S^{\le k-1}) = 0$ as a different set of rates that go into the numerator in Proposition 4. This yields:

$$\sum_{j=1}^{k-1} w_j \frac{y_j^*(S^{\le k-1})}{y_j^*(S^{\le k})} \le \sum_{j=1}^{k} w_j \, .$$

Observing that $\frac{y_j^*(S^{\le k-1})}{y_j^*(S^{\le k})} = 1 + \delta_{jk}$, we obtain $\sum_{j=1}^{k-1} w_j \delta_{jk} \le w_k$ for $k = 1, 2, \ldots, \kappa$.

Adding these inequalities for $k = 1, 2, \ldots, \kappa$ and changing the order of summations, we obtain:

$$\sum_{k=1}^{\kappa} \sum_{j=1}^{k-1} w_j \delta_{jk} = \sum_{j=1}^{\kappa} w_j \left(\sum_{k=j+1}^{\kappa} \delta_{jk} \right) \le W(S) \, .$$

Hence,

$$\sum_{j \in S'} w_j \left(\sum_{k=j+1}^{\kappa} \delta_{jk} \right) \le W(S) \, .$$

Let $\Delta_j = \sum_{k=j+1}^{\kappa} \delta_{jk}$, so that the above inequality becomes $\sum_{j \in S'} w_j \Delta_j \le W(A)$. Now observe that

$$\frac{y_j^*(S)}{y_j^*(S^{\le j})} = \prod_{k=j+1}^{\kappa} \frac{1}{\alpha_{jk}} = \prod_{k=j+1}^{\kappa} \frac{1}{1 + \delta_{jk}} \ge \exp\left(- \sum_{k=j+1}^{\kappa} \delta_{jk} \right) = \exp(-\Delta_j) \, .$$

We used the fact that $\delta_{jk} \ge 0$ for all j, k. Therefore,

$$\sum_{j \in S'} w_j \frac{y_j^*(S)}{y_j^*(S^{\le j})} \ge \sum_{j \in S'} w_j \exp(-\Delta_j) \, .$$

Since $\sum_{j \in S'} w_j \Delta_j \le W(S)$, the RHS is maximized when $\Delta_j = W(S)/W(S')$. Therefore,

$$\sum_{j \in S'} w_j \frac{y_j^*(S)}{y_j^*(S^{\le j})} \ge \exp(-W(S)/W(S')) \sum_{j \in S'} w_j = W(S') \cdot \exp(-W(S)/W(S')) \, . \qquad \blacktriangleleft$$

By applying this lemma with $S' = L_i$ and $S = A_i$ for each machine i, we have,

$$\begin{aligned}
\epsilon\Phi'|_A &\leq -\sum_i \sum_{j\in L_i} w_j \frac{y_j(A)}{y_j(A^{\leq j})} && \text{[Equation 6]}\\
&\leq -\sum_i W(L_i)\exp\left(-\frac{W(A_i)}{W(L_i)}\right) && \text{[Lemma 7]}\\
&= -W(L)\sum_i \frac{W(L_i)}{W(L)}\exp\left(-\frac{W(A_i)}{W(L_i)}\right)\\
&\leq -W(L)\exp\left(-\sum_i \frac{W(L_i)}{W(L)}\cdot\frac{W(A_i)}{W(L_i)}\right) && \text{[Due to convexity of } \exp(-x)\text{]}\\
&= -W(L)\exp\left(-\frac{W(A)}{W(L)}\right)\\
&\leq -(1-\epsilon)W(A)\cdot\exp(-1/(1-\epsilon)) && \text{[Since } W(L)\geq(1-\epsilon)W(A)\text{]}\\
&\leq -\frac{1-2\epsilon}{e}\cdot W(A)
\end{aligned}$$

for $(0 \leq \epsilon < 1/2)$. Thus, when PF is given $(e + 3\epsilon)$ speed , we have $\Phi'|_A \leq -(1 + 1/\epsilon)W(A)$. This together with Lemma 6 gives,

$$W(A) + \Phi'|_A + \Phi'|_O \leq 1/\epsilon \cdot W(O)\,.$$

Thus we conclude that in both cases $W(A) + \Phi' \leq 3/\epsilon^2 W(O)$, completing the proof of Theorem 2.

2.4 Bounding the Number of Migrations

First, observe that jobs can't migrate forever. This is because the total utility of jobs strictly increases when a job migrates and there are only a finite number of configurations regarding where each job can be residing. To ensure the migration process ends in polynomial time, we allow each job moves to another cluster only when its utility increases by a factor of at least $(1 + \epsilon)$. Also we force job j to move to the cluster where its utility is maximized. For any polytope $\mathcal{P}_i$, it is well-known that a job gets a rate at least $1/n$ times the rate it would get when it is the only job on cluster i in the PF allocation. Let s_j be the maximum processing rate when j is the only job in the system. Then, it is easy to see that job j's processing rate/utility is at least s_j/n and at most s_j. Therefore, each job can migrate at most $O(\log_{1+\epsilon} n)$ times. Hence the total number of jobs migrations is at most $O(\frac{n}{\epsilon}\log n)$.

3 Non-Clairvoyant Scheduling On Unrelated Machines

As already mentioned, the unrelated machines model is a special case of Monotone-MCPS, where each machine corresponds to a cluster i. Recall that in the unrelated machine setting, if a job is assigned to machine i it takes p_j/s_{ij} time to complete. The term $0 \leq s_{ij} \leq 1$ is the machine dependent *slow-down* factor of job j. It is easy to verify that this can be captured using polytope constraints of the form $\sum_j x_{jt} \leq 1$ and $y_{jt} \leq s_{ij} \cdot x_{jt}$. Therefore, Theorem 2 gives an $(e + \epsilon)$-speed $O\left(1/\epsilon^2\right)$-competitive algorithm for minimizing the weighted flow-time in the non-clairvoyant setting. However, PF algorithm is the same as Weighted Round Robin (WRR) in the unrelated machine setting, hence by a more careful analysis of Lemma 7 we can reduce the speed augmentation to $2 + \epsilon$. The entire analysis of Theorem 2 also goes through if we replace WRR by the Latest Arrival Processor Sharing algorithm (LAPS) [5], and we obtain $(1+\epsilon)$-speed $O\left(1/\epsilon^2\right)$-competitive algorithm, matching the best known result [12]. This also gives the first analysis of LAPS for unrelated machine scheduling. The omitted details are simple, and we defer the complete proof to the full version of the paper.

References

1 Faraz Ahmad, Srimat T. Chakradhar, Anand Raghunathan, and T. N. Vijaykumar. Tarazu: optimizing mapreduce on heterogeneous clusters. In *ASPLOS*, pages 61–74. ACM, 2012. doi:10.1145/2150976.2150984.

2 Amazon EC2-Spot-Instances. URL: http://aws.amazon.com/ec2/spot-instances/.

3 Jivitej S. Chadha, Naveen Garg, Amit Kumar, and V. N. Muralidhara. A competitive algorithm for minimizing weighted flow time on unrelated machines with speed augmentation. In *STOC*, pages 679–684, 2009.

4 R. Cole, V. Gkatzelis, and G. Goel. Mechanism design for fair division: allocating divisible items without payments. In *ACM EC*, pages 251–268, 2013.

5 Jeff Edmonds and Kirk Pruhs. Scalably scheduling processes with arbitrary speedup curves. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 685–692, 2009.

6 Kyle Fox, Sungjin Im, and Benjamin Moseley. Energy efficient scheduling of parallelizable jobs. In *SODA*, pages 948–957, 2013.

7 N. Garg and A. Kumar. Better algorithms for minimizing average flow-time on related machines. In *ICALP (1)*, 2006.

8 A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, I. Stoica, and S. Shenker. Dominant resource fairness: Fair allocation of multiple resource types. In *NSDI*, 2011.

9 Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. Multi-resource packing for cluster schedulers. In *ACM SIGCOMM 2014 Conference, SIGCOMM'14, Chicago, IL, USA, August 17-22, 2014*, pages 455–466, 2014. doi:10.1145/2619239.2626334.

10 Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. Competitive algorithms from competitive equilibria: Non-clairvoyant scheduling under polyhedral constraints. In *STOC*, 2014.

11 Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. Competitive flow time algorithms for polyhedral scheduling. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 506–524, 2015. doi:10.1109/FOCS.2015.38.

12 Sungjin Im, Janardhan Kulkarni, Kamesh Munagala, and Kirk Pruhs. Selfishmigrate: A scalable algorithm for non-clairvoyantly scheduling heterogeneous processors. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 531–540, 2014. doi:10.1109/FOCS.2014.63.

13 Sungjin Im, Benjamin Moseley, and Kirk Pruhs. A tutorial on amortized local competitiveness in online scheduling. *SIGACT News*, 42(2):83–97, 2011. doi:10.1145/1998037.1998058.

14 Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *JACM*, 47(4):617–643, 2000.

15 Gunho Lee, Byung-Gon Chun, and Randy H Katz. Heterogeneity-aware resource allocation and scheduling in the cloud. In *Proceedings of the 3rd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud*, volume 11, 2011.

16 J. Nash. The bargaining problem. *Econometrica*, 18(2):155–162, 1950.

17 Lucian Popa, Gautam Kumar, Mosharaf Chowdhury, Arvind Krishnamurthy, Sylvia Ratnasamy, and Ion Stoica. Faircloud: sharing the network in cloud computing. In *ACM SIGCOMM*, pages 187–198, 2012.

18 Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz, and Ion Stoica. Improving mapreduce performance in heterogeneous environments. In *OSDI*, pages 29–42, Berkeley, CA, USA, 2008. USENIX Association. URL: `http://dl.acm.org/citation.cfm?id=1855741.1855744`.

19 S. M. Zahedi and B. C. Lee. REF: resource elasticity fairness with sharing incentives for multiprocessors. In *ASPLOS*, pages 145–160, 2014.